POLITECNICO DI MILANO

Corso di Laurea in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

Una metodologia empirica per analizzare l'impatto della struttura del codice sul consumo energetico indotto dalle applicazioni software

Relatore: Prof.ssa Chiara FRANCALANCI

Correlatore: Prof. Eugenio CAPRA

Tesi di Laurea di:

Antonio AMALFI, matricola 734874

Tomas AURORA VAGNATO, matricola 725427

Anno Accademico 2009-2010

Alle nostre famiglie

Sommario

La tecnologia, negli ultimi anni, sta crescendo notevolmente e diventando indispensabile. Essa occupa un ruolo fondamentale nella nostra vita, poiché ci porta notevoli benefici e semplifica molti compiti. Purtroppo la rapida crescita e il notevole sviluppo dei sistemi IT porta ad un incremento dei consumi energetici avendo quindi un rilevante impatto sull'ambiente. Quest'ultimo non può più essere trascurato, infatti secondo recenti ricerche, l'IT ha un impatto ambientale pari a quello dell'industria aeronautica, producendo più del 2% delle emissioni di CO_2.

I consumi energetici, invece, rappresentano uno dei problemi principali di scalabilità dei *data center*, in quanto i fornitori non sono sempre in grado di soddisfare i fabbisogni energetici richiesti.

Tutti questi problemi hanno fatto movimentare i produttori di tecnologia, i responsabili dei sistemi informativi delle aziende e le comunità scientifiche, ed è così nata una nuova disciplina, chiamata *Green IT*, la quale si occupa di migliorare l'efficienza energetica e l'impatto ambientale dei sistemi informativi.

In particolare il *Green IT* studia:

- come migliorare l'impatto ambientale dell'IT
- come diminuire i costi dovuti all'IT, in termini energetici

- come risolvere il limite di scalabilità dell'IT provocato da un insostenibile richiesta energetica

Oggigiorno il consumo energetico per alimentare e raffreddare un sistema IT copre una significativa parte del Total Cost of Ownership, infatti si può notare che il costo di acquisto dell'hardware negli ultimi anni è cresciuto molto debolmente, se non addirittura diminuito, mentre il costo per alimentare e raffreddare i sistemi IT è quadruplicato.

I problemi studiati dal *Green IT* devono essere necessariamente affrontati a tutti i vari livelli infrastrutturali che compongono un *data center* (utilizzo del processore, alimentazione, raffreddamento, ecc.) in quanto tutti i livelli contribuiscono al consumo totale.

Le ricerche si sono sempre concentrare sul miglioramento dell'efficienza energetica dell'hardware, mentre sono state tralasciate per quanto riguarda il software applicativo. Infatti il mondo dell'impresa considera soltanto i costi, le prestazioni e la qualità delle applicazioni, non facendo alcuno studio per valutare l'efficienza energetica del software, sia perché mancano metodologie e strumenti per poterla misurare e sia perché manca la cultura aziendale necessaria per tenerne conto.

Il nostro lavoro di tesi estende precedenti ricerche effettuate dal Politecnico di Milano al fine di definire metriche green che possono dare un indice del grado di efficienza energetica di una applicazione. In particolare questo progetto di ricerca ha lo scopo di identificare metriche nuove che siano in grado di valutare l'impatto sul consumo energetico dovuto all'uso delle librerie esterne nell'implementazione del software.

Nella tesi si è mostrato che l'uso delle librerie esterne in un'appli-

cazione impatta in due modi differenti: per applicazioni complesse un intensivo uso di librerie esterne aumenta il consumo energetico contrariamente a quello che avviene per le applicazioni meno complesse, nelle quali un eccessivo uso di librerie esterne aumenta l'efficienza energetica.

Ringraziamenti

Giunti alla fine di questo lavoro pensiamo sia giusto ringraziare tutte le persone che ci hanno aiutato e sostenuto nel nostro lungo e faticoso percorso di studi.

In primo luogo desideriamo ringraziare la Prof.ssa Chiara Francalanci e i Prof. Eugenio Capra e Francesco Merlo per i consigli, la pazienza e la disponibilità mostrata nei nostri confronti per aver reso possibile la stesura di questa tesi.

Un ringraziamento speciale per la fiducia e il grande affetto va ai nostri genitori, Giovanni & Domenica Grazia e Santo & Rosa, senza i quali non saremmo mai potuti arrivare fino a questo punto. Inoltre ringraziamo i fratelli Giuseppe e Luca, sorelle Katia e Natasha, il cognato Filippo, i nonni e gli zii vari.

Ringrazio la mia ragazza Hande che è riuscita a darmi una spinta in più nell'ultima parte del mio percorso di studi e ha condiviso con me le ansie pre-esame e gioie e dolori post-esame.

Ringraziamo tutti gli amici di sempre che ci hanno fatto vivere mo-

menti speciali e indimenticabili, ci hanno dimostrato che le vere amicizie durano nel tempo e ci hanno dato supporto morale nei momenti più difficili:
in particolare Antonio ringrazia: Gianni, Brigida, Cinzia, Carmelisa, Quintino, Francesco, Pasquale, gli amici-coinquilini Franco, Massimo e Daniele;
mentre Tomas ringrazia: Igor, Denis, Miran, Ciccio, Anna, Faio, Macci, Cass, Elle, Giovanna, Mais, Paola, Marta, Stefania, Gioele, Giovanni, Alice.

Tomas desidera ringraziare rapidamente anche gli amici: Diego, Fofo, Barza, Laura, Sciù, Porta, Paride, Piro, Silvia, Peacy, Ema, Antonio, Alessia...;
mentre Antonio ringrazia: tutti gli amici di Policoro e di Milano e gli amici del calcetto

E infine, ultimi ma tutt'altro che ultimi per importanza, i nostri compagni di studio con i quali abbiamo condiviso traguardi e fatiche: Arlix, Mbacicc, Agnoletto, Filippo, Dany, Pito, Claudio, Fabri, Bulma.

Sicuramente ci siamo dimenticati di qualcuno, quindi chiediamo scusa ma ogni tanto la memoria ci abbandona.

Grazie a tutti.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi dieci anni i sistemi IT sono cresciuti molto velocemente, con un conseguente incremento dei consumi energetici.

Questa crescente richiesta di energia porta con se una considerevole moltitudine di problemi. Primo fra tutti i costi di energia sono drammaticamente aumentati e il loro impatto su tutti i costi infrastrutturali dell'IT sono diventati sempre più significativi. In secondo luogo la maggior richiesta di energia rappresenta uno dei problemi per la scalabilità dei *data center*, da quando i fornitori di energia non possono supportare la grande richiesta di energia per i *data centers*. Inoltre l' IT contribuisce fortemente alle emissioni dei gas serra CO_2.

Sebbene il software non consuma direttamente energia, influisce profondamente sul consumo energetico dei componenti IT. Le applicazioni

software, che vanno dai sistemi operativi e i drivers per i dispositivi hardware ai sistemi di supporto alle decisioni e suites ERP, indicano come le informazioni devono essere elaborate e in una certa misura guidano il funzionamento dell'hardware. Di conseguenza, il software è indirettamente responsabile del consumo di energia.

L'efficienza energetica non è legata all'ingegneria del software, dove l'attenzione è focalizzata sulle performance e sulle metriche classiche di qualità del software e dove il consumo di energia non è preso in considerazione. Questa ha influenzato anche il mondo aziendale, in cui il trade-off tra costi, prestazioni e qualità del software non prende in considerazione l'efficienza energetica del software.

Un recente filone di ricerca in atto al Politecnico di Milano, si dedica ad affrontare il tema dell'efficienza energetica del software, fornendo una road map per lo sviluppo di software green, metriche e relativi tools [9]. Parte dell'attività di ricerca è stata già svolta ed è stato dimostrato in un precedente progetto di Galli [27] che il software ha un'impatto importante sui consumi energetici.

Questo lavoro di tesi propone una metodologia per valutare, analizzare e quantificare in che modo la struttura di un'applicazione software ha un impatto sul consumo energetico. In altre parole, si andrà a studiare se un'applicazione, implementata mediante l'uso di librerie, consuma energeticamente di più o di meno rispetto ad una stessa applicazione che non ne usa.

Il progetto di tesi è strutturato nel seguente modo:

- **Capitolo 2:** presenta un'introduzione ai concetti principali su cui si basa questa tesi. In primo luogo è introdotto e descritto il green IT. In seguito si ha un overview sullo stato attuale dell'arte per la misura del consumo di energia e sulle metodologie di misurazione del consumo energetico del software.

- **Capitolo 3:** in questo capitolo andremo a spiegare l'ipotesi dalla quale ha avuto inizio il nostro lavoro di tesi.

- **Capitolo 4:** descrive la metodologia seguita durante il nostro lavoro di tesi. All'inizio è illustrato il progetto al quale la tesi si riferisce. In seguito sono presentati gli step necessari per capire che impatto ha l'uso di librerie esterne nell'implementazione di applicazioni software.

- **Capitolo 5:** illustra l'implementazione delle metriche e del software che sono stati necessari ai fini del nostro lavoro di tesi. In particolare, e ai fini pratici, è stato fatto l'upgrate di un tool ad-hoc per il calcolo automatico delle metriche create.

- **Capitolo 6:** illustra i risultati ottenuti e la loro validazione con un approccio statistico.

- **Capitolo 7:** è dedicato alla presentazione delle conclusioni del lavoro di tesi e dei possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 Il Green IT

Oggi la tecnologia è diventata una parte fondamentale della nostra vita. Nel corso degli anni l'uso della tecnologia è cresciuto in vari settori, migliorando la nostra vita, il lavoro e offrendo diversi tipi di benefici. Usiamo la tecnologia quasi per tutto e talvolta anche a nostra insaputa ed è quasi impossibile concepire un mondo senza di essa: ad esempio senza l'uso della mail, delle chats, dello shopping online e di tutti i vantaggi che la tecnologia ha portato nella nostra vita nel corso degli anni [1].

L' Information Technology (IT), purtroppo, consuma una significativa quantità di energia, fa crescere notevolmente il costo della bolletta elettrica e contribuisce fortemente alle emissioni di CO_2. Inoltre l'hardware introduce altri problemi sia durante la sua produzione sia durante il suo smaltimento.

L' Information Technology(IT) ha un forte impatto sull'ambiente ma allo stesso tempo sta cercando di porre soluzioni al problema del-

l'inquinamento ambientale provocato dallo stesso e si pone come un strumento sia per monitorare il consumo energetico sia per ottimizzare l'efficienza energetica del processo di produzione; ha quindi un ruolo predominante per la riduzione del consumo energetico.

L'IT, secondo recenti ricerche [1] [7] [8], è responsabile di una quota tra il 2% e il 3% delle emissioni a livello mondiale di CO_2, con un'impatto equivalente a quello dell'industria aereonautica.

Possiamo stimare l'emissione annua di CO_2 di ogni PC pari a una tonnellata, un server genera la stessa quantità di CO_2 che produce un SUV che percorre 25km[10].

La veloce evoluzione negli ultimi decenni dei processori, i quali sono diventati sempre più piccoli e veloci, ha indotto un forte aumento della potenza dissipata per il loro normale funzionamento, infatti mentre un ormai storico 486 dissipava circa 10 W, un Pentium IV ne dissipa 120 W, quindi un consumo energetico dieci volte più grande. Per avere un'idea ancora più precisa del consumo di alcuni componenti di un sistema IT possiamo stimare il consumo di un moderno *server blade* pari a 1 KW, cioè il consumo di un frigorifero di casa, un *rack di un server blade*, composto da 5 scaffali con 8 unità ciascuna, consuma 40W, ciò che consuma una palazzina; continuando con gli esempi, si stima il consumo di un *data center* di medie dimensioni pari a 250W, come un quartiere e infine un *data center* di grandi dimensioni può arrivare a consumare 10MW, come una cittadina.

La veloce crescita dei consumi energetici dell'IT spinge i produttori di tecnologia, i responsabili dei sistemi informativi aziendali a studiare e a mettere in pratica tecniche di progettazione e realizzazione di componenti hardware e software efficienti, con impatti ambientali limitati o

nulli.

Nasce il termine *GREEN IT*, che mette in relazione la parte IT con l'efficienza energetica, e indica lo studio e l'utilizzo di tecnologie informatiche in modo efficiente[1].

Con il termine *Green IT* si possono distinguere tre differenti aree tematiche. Le elencheremo di seguito con le specifiche soluzioni da adottare che verranno successivamente argomentate:

1. l'efficienza energetica dell' IT;
    - efficienza energetica
    - progettazione 'data center' innovativi e power management
    - comportamenti 'green'
2. la gestione eco-compatibile del ciclo di vita dell' IT;
    - produzione 'green'
    - gestione dello smaltimento e riciclo di componenti vecchi e inutilizzabili
    - eco-labelling
    - ottimizzazione del package
3. l'utilizzo dell'IT come strumento di Governance 'green'
    - strumenti di gestione e di business intelligence per monitorare i parametri 'green' per ogni processo di business
    - osservanza di nuove regole

Analizzando la prima area, l'efficienza energetica si può aumentare progettando e gestendo le infrastrutture e i *data center* in modo migliore, ma agendo per prima cosa sulla cultura aziendale e le pratiche di utilizzo.

Dobbiamo pur evidenziare un fatto positivo: in realtà, negli ultimi anni, l'efficienza energetica dell' IT, quindi le prestazioni rapportare al consumo energetico, è cresciuta, in pratica le prestazioni sono migliorate di molto rispetto alla crescita della potenza richiesta. Questo è confermato considerando i dati ottenuti dal *benchmark* TPC-C[16], utilizzato per misurare le prestazioni dei processori, dove si nota che il valore dell'efficienza energetica, misurata in migliaia di transazioni per watt assorbito (Ktpm-c/watt), è aumentato di un fattore di 2,5 nell'ultimo decennio, indicando quindi un miglioramento dell'efficienza energetica.

Nonostante ciò, la crescente domanda di capacità di calcolo e l'aumento del consumo energetico dell'IT impongono di migliorare ulteriormente e in modo radicale l'efficienza energetica.

L'industria IT non è responsabile soltanto dell'inquinamento dovuto al consumo di energia, ma anche alle sostanze tossiche disperse nell'ambiente per la produzione dei vari componenti IT.

E' proprio questa la seconda area tematica della quale si occupa il *Green IT*, che considera l'impatto ambientale del ciclo produttivo dei componenti IT e del loro smaltimento. Secondo alcune ricerche[8], infatti, una significativa percentuale dell'inquinamento del suolo da piombo, cadmio e mercurio deriva dall' IT. I rifiuti di apparecchiature elettriche ed elettroniche (talvolta citati semplicemente con l'acronimo *RAEE* e in inglese Waste of electric and electronic equipment (*WEEE*) o *e-waste*) sono rifiuti di tipo particolare; consistono in apparecchiature elettriche o elettroniche di cui il possessore intenda disfarsi in quanto guasta, inutilizzata, o obsoleta e dunque destinata all'abbandono.

Questo tipo di rifiuti, non biodegradabili, creano seri problemi che

derivano dalla presenza di sostanze tossiche per l'ambiente. Inoltre la continua diffusione di apparecchi elettronici determina un sempre maggior rischio di abbandono nell'ambiente con conseguente inquinamento del suolo, dell'aria e con ripercussioni sulla salute dell'uomo.

Questi prodotti devono essere trattati in modo corretto e devono essere destinati al recupero differenziato, in modo tale che le risorse possano essere riutilizzate per costruire nuove apparecchiature.

La ricerca Green IT in quest'area si occupa di ridurre le sostanze inquinanti presenti nelle componenti IT a partire dal processo produttivo, ottimizzare il packaging, eco-etichettare le varie unità, e studia le diverse tecniche di ricondizionamento e di recupero dei componenti dismessi.

Questa area è molto vasta e non la approfondiremo ulteriormente.

La terza area del Green IT pone l' IT come uno strumento per controllare, quindi misurare e monitorare i parametri 'green' (ad esempio il consumo energetico, la temperatura, consumo di carta e materiali di consumo, rifiuti tossici prodotti) di tutti i processi di business, IT e non IT.

Per il monitoraggio e quindi per poter controllare e migliorare l'efficienza energetica di un processo industriale ci vengono in aiuto i Key Performance Indicator (*KPI*) 'green'. Essi possono essere memorizzati e analizzati tramite cruscotti direzionali e di business intelligence.

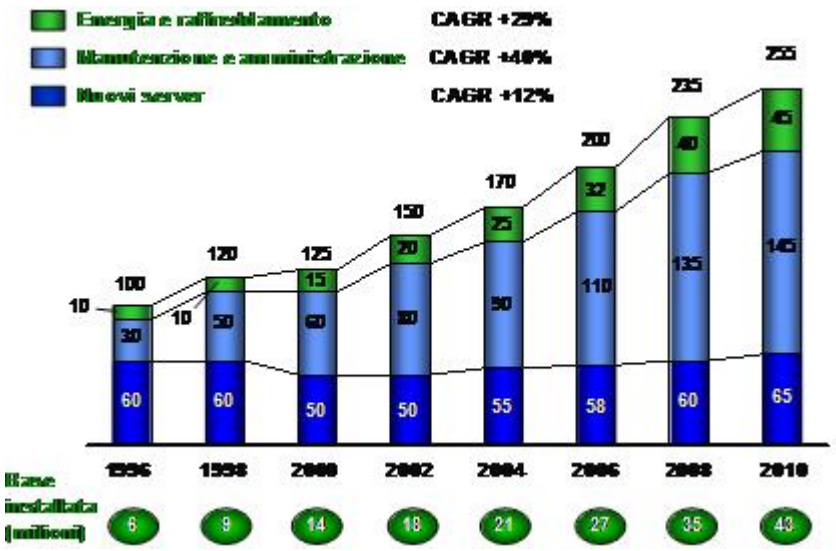


Figura 2.1: Spesa mondiale per i server (Miliardi di dollari)

2.1.1 L' importanza del Green IT

Come discusso nella precedente sezione l'IT ha un impatto ambientale significativo sia per il consumo di energia, quindi emissioni di gas serra, causa del surriscaldamento terrestre, sia per la dispersione di sostanze inquinanti nell'ambiente. Affrontare le problematiche del Green IT è quindi prima di tutto una responsabilità sociale.

Il costo dell'energia consumata dai sistemi IT copre un significativa parte del Total Cost of Ownership (TCO) dei sistemi ed è in continua crescita, in questo modo cresce l'interesse per il Green IT da parte dei CIO e dei responsabili IT.

La Fig.2.1 mostra i dati relativi alla spesa mondiale per i server negli ultimi anni e alcune stime per il futuro. Possiamo notare che il costo di acquisto dell'hardware negli ultimi dodici anni è cresciuto molto debolmente, mentre il costo per alimentare e raffreddare i sistemi è quadruplicato.

Dalla Fig.2.2 si evince che il costo di energia e raffreddamento rappresenta circa il 60% della spesa in nuove infrastutture, con un significativo impatto sul Total Cost of Ownership (TCO) e i costi sono destinati a crescere come conseguenza del continuo aumento del costo unitario dell'energia.

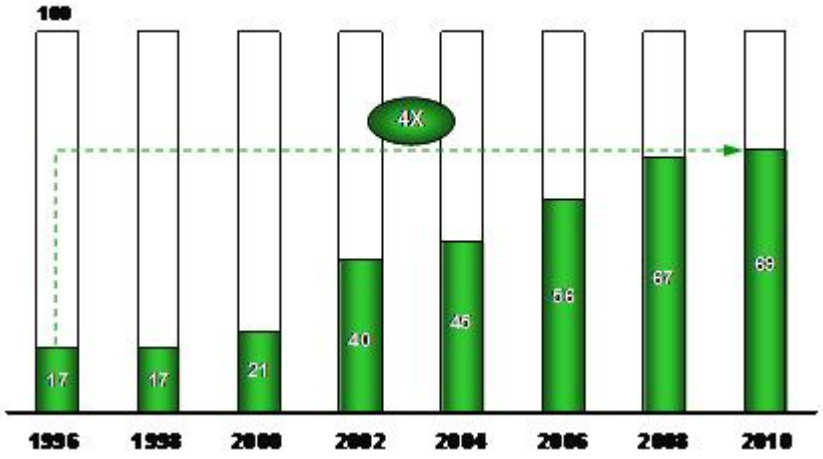


Figura 2.2: Rapporto spesa per energia raffreddamento e spesa per l'acquisto di nuovi server

Questi costi molto spesso sono nascosti, o meglio, vengono ignorati, poichè da una parte mancano gli strumenti e le metodologie per misurarli con esattezza, dall'altra parte non sono contabilizzati nel budget dei sistemi IT, ma vengono considerati nei consumi di tutta l'azienda, e questo non evidenzia gli elevati consumi energetici della parte IT.

L' alto consumo di energia delle apparecchiature informatiche diviene un limite alla scalabilità dei *data center* di medie e grandi dimensioni, posti in aree ad alta densità di popolazione. La potenza elettrica richiesta cresce dell' 8%-10% l'anno e i gestori della rete elettrica potrebbero essere non capaci di convogliare tanta energia in un'area ristretta

di un centro urbano. La potenza assorbita dai nuovi server è spesso incompatibile con le caratteristiche elettriche degli attuali *data center*.

Per aumentare la capacità di calcolo potrebbe essere necessario edificare nuove strutture in aree a più bassa densità abitativa, con un ulteriore impatto ambientale e di costo.

Figura 2.3: Costo in $ cent/ KWh dell'energia per uso industriale

Nei prossimi anni si arriverà ad avere il 60% dei *data center* limitati dal consumo di energia, dalle esigenze di raffreddamento e dallo spazio.

Possiamo, quindi, riassumere quanto detto prima e sottolineare l'importanza del Green IT in tre punti fondamentali:

1. L'IT ha un impatto ambientale significativo;
2. il consumo energetico dell'IT costa;

3. il fabbisogno energetico è un limite alla scalabilità dell'IT.

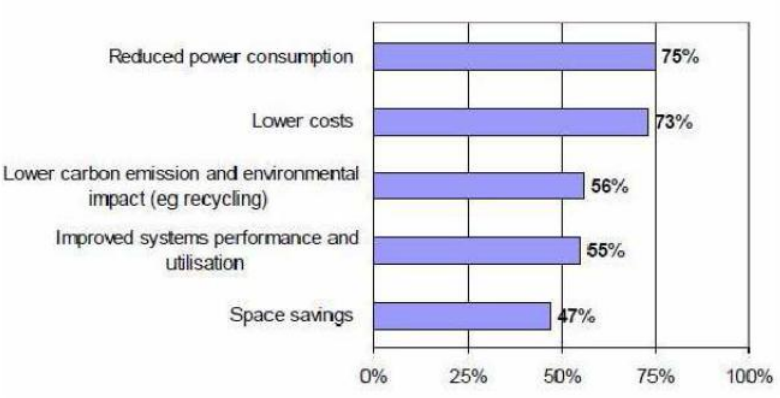


Figura 2.4: Ragioni per usare un IT Green

2.1.2 Iniziative Green IT

Le origini del Green IT devono essere trovate nel 1992, quando l' 'U.S. Environmental Protection Agency' iniziò il programma 'Energy Star'[2].

Questo programma è stato creato per promuovere e riconoscere l'efficienza energetica nei monitors, le apparecchiature di controllo del clima e altre tecnologie. Grazie a questo, le modalità 'sleep' e 'power saving' iniziarono ad essere adottate.

Allo stesso tempo, la 'TCO Development', una organizzazione svedese, iniziò un altro programma, il 'TCO Certification'. Lo scopo del programma era quello di promuovere le basse emissioni magnetiche ed elettriche nei monitor CRT dei computer; successivamente il programma 'TCO Certification' è stato allargato in modo che coinvolgesse altri componenti che consumano energia.

Negli ultimi anni molti enti governativi hanno creato in continuità norme e regolamenti al fine di promuovere il Green IT. Nell'Ottobre 2006, il programma 'Energy Star' è stato rivisitato per poter includere requisiti di efficienza più severi per tutti i computer e inserire i prodotti in un ranking.

Nel 2002 l'Unione Europea avanzò una lista di direttive, 2002/95/EC, chiamate RoHS [5], che sta per Reduction of Hazardous Substances. Queste direttive limitano l'uso di sostanze pericolose nella fabbricazione di vari tipi di apparecchiature elettriche ed elettroniche. Nello stesso anno l'Unione Europea avanzò un'altra direttiva, la 2002/96/EC, chiamata WEEE [6], che sta per Waste Electrical and Electronic Equipement. Questa direttiva impone la sostituzione dei materiali pesanti e dei ritardanti di fiamma come PBBs e PBDEs in tutte le apparecchiature elettroniche. Essa ha anche assegnato la responsabilità ai produttori di apparecchiature di raccogliere e riciclare i vecchi componenti. Queste società sono responsabili nel fondare un'infrastruttura per la raccolta dei WEEE in modo tale che l'utente possa avere la possibilità di restituire le apparecchiature elettriche ed elettroniche gratuitamente.

Al fine di sottolineare quanto sia importante questa direttiva, la Royal Society di arte nel Regno Unito ha costruito una scultura, chiamata 'WEEE Man' la quale è completamente costruita con 3,3 tonnellate di materiale elettrico, che è la quantità media di WEEE generata da una persona nella sua vita.

Accanto alle direttive governative, ci sono molte altre organizzazioni che hanno le loro attenzioni sul Green IT.

Una di queste è la Climate Savers Computing Initiative. Questa organizzazione, non profit, promuove nuove tecnologie capaci di migliorare

l'efficienza energetica e ridurre il consumo di energia delle apparecchiature IT. Ci sono anche produttori che partecipano all' organizzazione e hanno introdotto nuovi prodotti che soddisfano i requisiti di efficienza energetica. L'organizzazione vuole ridurre i consumi di potenza del 50% entro il 2010, in modo da ridurre le emissioni di CO_2 di 54 milioni di tonnellate l'anno.

Un'altra organizzazione non profit è la Green Computing Impact Organization, che mira a guidare gli utenti ad un comportamento amichevole verso l'ambiente tramite i prodotti elettrici ed elettronici. GCIO organizza eventi educativi al fine di realizzare i propri obiettivi. La parte più importante dell'organizzazione è la GCIO Cooperative, che è una comunità di leader IT che si occupano di questioni di Green IT e usano il proprio tempo e le proprie risorse per promuovere ed educare all'uso di prodotti Green It.

La terza organizzazione orientata al 'green' è la Green Electronics Council, che ha gli obiettivi di ispirare e supportare la progettazione efficace, la produzione, l'uso e il recupero di prodotti elettronici affinché si abbia un mondo sano e pulito.

L'ultima organizzazione non profit di cui parleremo è la Green Grid; il principale scopo di questa organizzazione è quello di aumentare l'efficienza energetica dei prodotti IT. Questa aiuta a creare e a promuovere l'adozione di metriche e standards al fine di misurare l'efficienza energetica dei 'data center', e ridurre i consumi.

Le più importanti aziende IT sono già legate a questa organizzazione (Microsoft, AMD, Dell, HP, IBM, Intel, Sun Microsystems e VMware). L' organizzazione ha ora centinaia di membri, inclusi gli utenti finali e gli enti governativi.

2.1.3 L' IT per un business 'green'

I sistemi IT, come abbiamo detto precedentemente, sono responsabili del 2% delle emissioni mondiali di CO_2, ma possono giocare un ruolo fondamentale per ridurre il restante 98% delle emissioni [4].

Molti sono i settori in cui l'IT ha ridotto l'impatto ambientale rendendo essi più 'green'. Ha fatto questo migliorando l'efficienza energetica, utilizzando in modo razionale i materiali e mediante la riduzione dei trasporti.

Le centraline elettroniche hanno aumentato notevolmente l'efficienza dei veicoli, così come il controllo digitale ha permesso di ridurre i consumi energetici degli elettrodomestici. L'IT ha razionalizzato i consumi.

Ad esempio i sistemi centralizzati per l'accensione e lo spegnimento delle luci possono ridurre in modo notevole il consumo di energia di un edificio.

Possiamo stilare una lunga lista di strumenti che permettono di ridurre il consumo di carta di un'azienda, prima tra tutte la posta elettronica, i sistemi di archiviazione digitale, la fatturazione elettronica.

Molti sistemi, inoltre, hanno anche un impatto sui processi, che diventano più veloci ed efficienti e, quindi, di conseguenza hanno bisogno di meno energia.

I supporti digitali riscrivibili per archiviare documenti, immagini permettono di razionalizzare l'utilizzo dei materiali di consumo. I sis-

temi IT applicati ai processi industriali in molti casi permettono di ottimizzare la produzione e ridurre gli scarti di materia prima.

Dal punto di vista dei trasporti, non si possono non considerare i viaggi resi evitabili dalle moderne tecnologie di telecomunicazione, come la video conferenza e il telelavoro. Gli strumenti di e-commerce, ad esempio, riducono gli spostamenti fisici della merce trattata a beneficio dell'ambiente.

Si potrebbero elencare ancora altre cose ma ci fermiamo qui.

In conclusione, la relazione tra IT e ambiente sta diventando sempre più complessa e sta acquisendo rilevanza, sia perché l'IT di per sè consuma energia, sia perché può avere un ruolo cruciale nel monitorare e ridurre l'impatto ambientale di altri processi.

Nel prossimo paragrafo vedremo come possiamo agire su un sistema IT per renderlo 'green'.

2.1.4 Rendere "green" l' IT

I componenti IT consumano energia per la natura della trasmissione di informazione che richiede energia. L' unità minima di informazione, il bit, è associato allo stato di un sistema fisico e per poterlo commutare occorre cambiare lo stato del sistema e in questo modo consumare energia. Ma questi valori di energia sono piuttosto bassi rispetto ai consumi che si misurano di un componente IT. Questo è spiegato dal fatto che la commutazione dei bit è solamente il livello più basso di un sistema informatico: sopra ad esso esistono tanti livelli infrastrutturali che moltiplicano anche di un fattore pari a 30 il consumo energetico della singola

commutazione.

La Fig.2.5 mostra come viene ripartito il consumo energetico in un *data center* standard (ovviamente la distribuzione può cambiare in base alla tipologia del carico computazionale, dalle macchine installate e dalle caratteristiche fisiche dei locali).

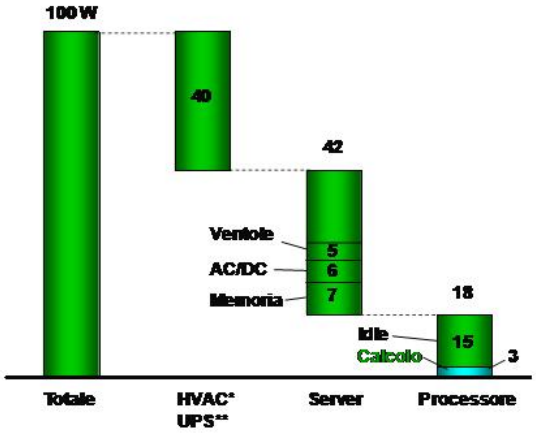


Figura 2.5: Ripartizione percentuale del consumo di energia in un data center; * Heating, Ventilation and Air Conditioning (impianto di ventilazione e climatizzazione); ** Uninterruptible Power Supply (gruppi di continuità).

Il problema del green IT, come si evince anche dal grafico, deve essere affrontato ai vari livelli infrastrutturali del *data center* (utilizzo del processore, alimentazione, raffreddamento, ecc.) in quanto tutti i livelli contribuiscono al consumo totale. Bisogna dunque operare e migliorare l'efficienza energetica di ogni livello del sistema IT, partendo dai livelli più bassi, in quanto i valori di consumo energetico di quest'ultimi vengono poi amplificati da tutti i livelli superiori. Infatti, un software inefficiente richiederà più operazioni del processore, che a sua volta richiederà più memoria e dovrà essere raffreddato di più, e così via. Nella Tabella

2.6 sono presenti valori stimati sull'efficienza energetica dei vari livelli dei sistemi IT comunemente usati.

Livello	Efficienza energetica attuale stimata
Infrastruttura	50%
Sistema	40%
Server	60%
Microprocessore	0,001%
Software	20%
Rete	10%
Database	60%
Pratiche di utilizzo dell'IT	30%

Figura 2.6: Efficienza energetica stimata rispetto all'efficienza massima teorica degli attuali sistemi IT suddivisa per livelli logici e infrastrutturali

Sono quindi numerose le leve su cui è possibile agire per migliorare l'efficienza energetica.

Vediamo come possiamo agire ad ogni livello del sistema IT:

- Analizzando il livello **infrastrutturale** possiamo distingue due possibili cause di inefficienza: i gruppi di continuità e i sistemi di condizionamento. Quest'ultimi, infatti, molto spesso sono a potenza fissa, mentre potrebbero essere regolati in base alle esigenze, a seconda del carico di lavoro effettivo del *data center*. Inoltre la disposizione degli scaffali e delle bocchette di condizionamento sono stabiliti in base a regole empiriche e regolate dall'esperienza, men-

tre si potrebbero fare degli studi specifici di tipo termodinamico degli ambienti che porterebbero sicuramente a notevoli guadagni di efficienza.

- A livello di **sistema** per ottenere risparmi energetici sarebbe opportuno bilanciare in modo opportuno i carichi di lavoro sui vari server e affidarsi alla virtualizzazione, una pratica sempre più diffusa e che è in grado di portare notevoli risparmi di costo.

- Considerando il singolo **server** molta energia è dissipata dalle periferiche e dai componenti ausiliari e piccoli accorgimenti possono portare a notevoli risparmi di energia. Ad esempio si potrebbe regolare la velocità delle ventole in base alle esigenze, ciò potrebbe portare ridurre il consumo di energia delle ventole stesse fino al 45%, inoltre si potrebbero utilizzare meno drive ma più potenti ricudendo fino al 50% il relativo consumo. Altri risparmi possono essere ottenuti dividendo la cache in segmenti alimentati solo se necessario, in alcuni casi si potrebbe sostituire parte degli hard disk con memorie allo stato solido.

- A livello di **processori** si presentano le più grandi inefficienze. Essi non funzionano al massimo delle loro potenzialità e non sono sfruttati in modo efficiente. Per poter portare una riduzione dei consumi relativi ai processori è stato dimostrato che bisogna abbassare la frequenza di clock e passare da processori *single-core* a processori *quad-core*.

- Le operazioni del processore sono dettate dal **software**, anche esso su vari livelli, dal sistema operativo al middleware e alle applicazioni utente. La progettazione del software molto spesso

non tiene affatto conto dell'efficienza energetica e non viene preso in considerazione il trade-off fra costi, prestazioni e qualità. Le ricerche in questo ambito sono immature, non si hanno basi teoriche ed empiriche sulle quali fare affidamento. Si cerca quindi di analizzare l'efficienza energetica al livello software introducendo numerose ipotesi e restrizioni. Ciò ci ha portato a scegliere questo filone di ricerca del quale discuteremo successivamente.

- A livello di **rete** un guadagno energetico può essere ottenuto sia rivedendo in ottica 'green' agli algoritmi di routing, sia ottimizzando i dispositivi hardware per la gestione della rete stessa.

- Al livello di **database** ha un importante impatto sull' efficienza energetica la qualità dei dati memorizzati in esso. La scarsa qualità dei dati (es. dati inesatti e non aggiornati) porta ad eseguire più transazioni ed operazioni del necessario con conseguente consumo di energia.

- Infine analizziamo il modo in cui l' IT è utilizzato (**pratiche di utilizzo**) che può essere causa di enormi sprechi di energia. Recenti ricerche [18] hanno dimostrato che un corretto uso delle funzioni di *power management* ormai presenti su tutti i moderni PC, un giusto uso degli screen saver e lo spegnimento dei sistemi quando non sono in uso possono portare a risparmi dell'ordine del 60%. Impostare di default la stampa fronte e retro porta a notevoli risparmi di carta e anche di energia, infatti il trascinamento di due fogli consuma più energia del trascinamento di un singolo foglio.

In generale per migliorare l' efficienza energetica di un sistema IT è necessario agire sulla cultura aziendale sia degli utenti sia degli amministratori del sistema. La progettazione, la manutenzione e la gestione a tutti i livelli del sistema devono tenere conto dei parametri 'green' e le persone coinvolte dovrebbero essere educate verso il risparmio energetico con la consapevolezza che quasi sempre i risparmi energetici portano anche a risparmi di costo. Ad esempio, il premium price da sostenere per acquistare un PC costruito secondo parametri green può aggirarsi intorno ai 15-20€; poiché il risparmio è di circa 30W considerati i costi attuali dell'energia, l'investimento si ripagherà in meno di due anni, senza contare la riduzione dei costi di condizionamento (Elaborazione su dati Intel 2007).

2.1.5 L' IT per il controllo dei consumi energetici

Una recente ricerca compiuta dall' EILT [20], l'ente per la telecomunicazione britannico, riporta come l'86% dei dipartimenti ICT nel Regno Unito non conosce il peso delle proprie emissioni di CO_2.

Analogamente, una ricerca svolta dal Dipartimento di Elettronica e Informazione del Politecnico di Milano [3] su 140 aziende italiane di tutte le dimensioni e appartenenti a diversi settori mostra che il 95% dei responsabili IT non conosce il consumo energetico dei sistemi affidati alla loro gestione.

Inoltre alla luce di un'altra ricerca nell'81% dei casi il CIO non è responsabile dei costi energetici dei sistemi IT, che sono contabilizzati in altre voci di bugdet.

A questo punto è evidente che è molto difficile eseguire un processo di ottimizzazione su qualcosa che non si conosce, ed è improbabile che un responsabile IT sia interessato e spinto ad investire risorse per diminuire i consumi energetici dei suoi apparati quando i relativi costi non interessano il suo budget.

E' buona norma, quindi, monitorare tramite un opportuno sistema informativo direzionale 'green' l'impatto ambientale dei processi di un'azienda; bisogna tenere in considerazione diversi *Key Performance Indicator*, ad esempio consumo di energia, temperatura, CO_2, ecc., suddividendo il tutto per le opportune dimensioni (es. su base temporale, per fase di lavorazione, per prodotto, ecc.).

L' IT deve essere visto come uno strumento per monitorare e ottimizzare le prestazioni 'green' di un'azienda. Ciò può essere fatto in due diversi modi:

- IT deve supportare la misura dei parametri 'green' tramite opportuni sensori, anche wireless, e tramite reti intelligenti per la raccolta e l'aggregazione dei dati;
- l'analisi e la sintesi dei dati possono essere effettuate tramite cruscotti direzionali e strumenti di *data mining* già molto diffusi in altri campi.

Quanto descritto ovviamente è applicabile non solo all' IT, ma a tutti i processi aziendali che consumano energia. In Italia ci sono aziende con sistemi informativi tecnologicamente molto avanzati, ma che non sono in grado di valutare quanta energia è richiesta per una determinata fase di lavoro.

Per citare un caso di successo, una grande banca italiana ha recentemente realizzato un sistema che memorizza KPI 'green', i quali vengono analizzati periodicamente da professionisti specializzati con l'obiettivo di ottimizzare le performance energetiche. Tutto questo ha portato ad una riduzione dei consumi di corrente elettrica delle proprie filiali del 20% in un anno.

2.2 Focus sul software

La nostra tesi si focalizza sull'efficienza energetica dell' IT e, nello specifico, sull' efficienza energetica del software.

Molte ricerche sul 'green IT' si focalizzano sul consumo energetico dell'hardware, infatti il costo dell'energia è naturalmente associato alla parte hardware, e metà dei TCO sono dovuti ai costi energetici di alimentazione e raffreddamento dei server.

Ci si chiede se l'efficienza energetica può essere ottimizzata da una prospettiva legata alla parte software.

Il software occupa un ruolo fondamentale anche se non consuma direttamente energia. Esso influisce fortemente sui consumi dell'hardware che lo circonda. Le applicazioni software, dai sistemi operativi ed i driver ai sistemi di supporto alle decisioni e gli ERP, indicano come l'informazione vada elaborata e, in un certo qual modo, guidano il funzionamento dell'hardware. Di conseguenza, il software è indirettamente responsabile dei consumi energetici.

Generalmente parlando, attualmente gli ambienti di programmazione sono piuttosto complessi e soltanto programmatori esperti e con elevate

capacità sanno come ottenere performance con un limitato uso delle risorse. I moduli software sono progettati sia per essere preformanti sia efficienti. Tuttavia, se i programmatori non sono adeguatamente istruiti a risolvere i problemi di performance, posso scrivere codice inefficiente. Inoltre, l'industrializzazione, per poter ridurre i costi di sviluppo software, può portare ad ignorare problemi di efficienza energetica e di performance, considerando soltanto la manutenibilità.

2.2.1 Il software impatta sul Green IT

Il software ha un ruolo fondamentale nello scenario del Green IT. Anche se esso non consuma direttamente energia, è coinvolto nel consumo di potenza di tutto l'hardware di un sistema IT. In questo modo il software è indirettamente responsabile del consumo energetico di un sistema IT.

Dalle leggi della fisica sappiamo che la potenza media consumata da un microprocessore mentre esegue un applicazione è data da una formula ben nota:

$$P = VCC \times I[W] = [A][V] \tag{2.1}$$

dove I è la corrente media e VCC è la tensione di alimentazione del sistema. Dal momento che la potenza è il tasso con il quale l'energia è consumata, il consumo energetico di una determinata applicazione è l'integrale del consumo di potenza P sul tempo t:

$$ConsumoEnergia = \int I \times VCCdt \tag{2.2}$$

Misurare il consumo energetico di un'applicazione software tramite l'espressione 2.2 richiede di misurare sia I sia VCC sul sistema hardware.

Di conseguenza, esse si di riferiscono sempre ad una specifica architettura. Inoltre, dall'espressione è possibile analizzare il perché l'energia è consumata.

Per spiegare questo problema, abbiamo bisogno di legare il dominio fisico (ad esempio il consumo di energia) al dominio logico. Il consumo di energia può essere valutato analizzando il perché una certa applicazione richiede una certa quantità di energia per produrre l'output desiderato. Al fine di esaminare il problema del consumo di energia da una prospettiva logica possiamo introdurre alcune definizioni teoriche necessarie: il teorema di Margolus-Levitin, la profondità termodinamica e la profondità logica.

Il teorema di Margolus-Levitin [19] afferma che la massima frequenza per la commutazione di stato di un sistema fisico è direttamente proporzionale al totale dell'energia del sistema stesso. Di conseguenza, l'energia minima di commutazione richiesta da un sistema per operare a una data frequenza può essere calcolata come:

$$E_{min}(f) = \frac{f \cdot t}{4} \tag{2.3}$$

dove f è la frequenza e h è la costante di Planck. Per esempio, se noi rappresentassimo un bit mediante il senso di rotazione di un elettrone, la commutazione di energia necessaria alla frequenza di 1Ghz (e quindi comparabile a quella di un attuale computer desktop, considerando che, per un elettrone, la massima frequenza di commutazione è circa $3 \cdot 1013Hz$) è $E_e = 5 \cdot 10^{-21}J$.

La profondità termodinamica [21] è una proprietà di ogni sistema fisico; in sostanza si tratta di una misura delle informazioni necessarie per descrivere, e conseguentemente costruire, il sistema stesso. Questa metrica è legata al concetto di entropia, che è la misura del livello di 'disordine' in un dato sistema[26].

Partendo dal presupposto che un sistema può sempre essere descritto da una stringa di bit(per esempio, descrivendo la velocità iniziale e la posizione di tutti suoi atomi), l'entropia è il numero di bit del sistema che sono disordinati e non disponibili per produrre lavoro.

D'altra parte, è negentropia la misura che quantifica il numero di bit che sono ordinati e strutturati nel sistema. Ad esempio, un essere umano ha un alto grado di negentropia, mentre un pallone pieno di elio è completamente privo di negentropia. Se volessimo descrivere un tavolo, avremmo bisogno di un certo numero di bit negentropici, ma non abbiamo bisogno di descrivere la posizione di tutti i miliardi di atomi di un tavolo: questi bit possono essere entropici senza pregiudicare la nostra descrizione. Sulla base di queste definizioni, la profondità termodinamica è definita come il numero di bit negentropici che sono stati usati per costruire il sistema.

La profondità logica [21] di una generica stringa di bit, che può essere interpretata come la rappresentazione di un generico sistema (così come l'output di una applicazione), è definita come la complessità computazionale dei più efficienti programmi che sono in grado di produrre tale output. In altre parole, è il più piccolo numero di operazioni logiche elementari necessarie per eseguire il calcolo che produce la desiderata stringa di bit.

Un'applicazione software esegue un certo numero di calcoli in un

numero stabilito di bit al fine di ottenere un risultato. Applicando le definizioni teoriche discusse in precedenza, il consumo di energia di un'applicazione software può essere valutata dal punto di vista logico come:

$$EC_{logical}(f) = E(f) \cdot C_c \cdot T_d \tag{2.4}$$

dove E(f) è l'energia richiesta da un singolo bit in stato di commutazione alla frequenza f, C_c è la complessità computazionale dell'applicazione che è eseguita e T_d è la profondità termodinamica del calcolo che è eseguito sul problema rappresentato. In altre parole, l'espressione 2.4 stima il consumo di energia considerando la quantità di energia richiesta da un singolo bit per commutare E(f) che è applicata su un dato numero di bit (T_d) per un dato numero di operazioni (C_c). Inoltre l'espressione 2.4 consente di analizzare le cause che portano un' applicazione a consumare energia quando si sta elaborando informazione, senza focalizzarsi sul consumo dei meccanismi fisici ed elettrici.

Prima di tutto, notiamo che c'è un inevitabile trad-off tra energia e frequenza: un sistema più veloce richiede più energia. La minimizzazione del consumo di energia può essere ottenuto ottimizzando ciascuno dei tre termini dell'espressione 2.4.

Come discusso prima, l'energia minima richiesta per la commutazione dello stato di un bit ad una certa frequenza è data dal teorema di Margolus-Levitin. Questo è solo un limite inferiore teorico, che è valido se i bit sono rappresentati dalla rotazione degli elettroni. In verità, vari tentativi di costruire un macchina di calcolo che usa la rotazione di elettroni per rappresentare un bit è stata fatta (ad esempio, Isacc Chuang del MIT ha fattorizzato il numero 15 con un 7 qubit del computer [13]).

Si deve considerare che l'energia E_{min} è di gran lunga inferiore a quella effettiva che è consumata per mutare un bit nei computer attuali basati sui transistor (le moderne architetture richiedono circa 10-14 Joule per commutare un bit, e le ricerche sono in corso per ridurre questa energia).

Gli altri due termini, cioè la complessità computazionale C_c e la profondità termodinamica T_d possono invece essere affrontati da una prospettiva di Information System.

La minimizzazione della complessità computazionale richiesta per produrre un certo output può essere ottenuta se la generica applicazione A che è eseguita ha la minima complessità computazionale possibile, che è, esattamente la profondità logica L_d dell'output richiesto.

La profondità termodinamica può essere minimizzata adottando un modo efficiente di rappresentare il problema e i dati necessari per produrre l'output desiderato, che è la minima profondità termodinamica Td_{min}.

Di conseguenza, il limite inferiore dall'espressione 2.4, ad una data frequenza è dato da:

$$EC_{min}(f) = E_{min}(f) \cdot Ld \cdot Td_{min} \tag{2.5}$$

Proprio come la minima energia di commutazione data dal teorema di Margolus-Levitin, anche l'espressione 2.5 è solo un ideale limite inferiore teorico. In particolare, i problemi di scrivere un'applicazione con la minima complessità computazionale necessaria ad ottenere un desiderato output o precisare che è la rappresentazione più efficiente di un dato problema non sono problemi di poco conto. Ad esempio, esistono problemi per i quali non sappiamo se gli algoritmi usati per calcolare la loro soluzione sono i più efficienti (ad esempio l'algoritmo di ordinamento).

Inoltre, esistono classi di problemi per i quali non possiamo sapere se esiste una soluzione efficiente (per esempio, la classe dei problemi NP-completi). D'altra parte, è possibile progettare una metodologia che permette di confrontare le diverse applicazioni da un punto di vista dell'efficienza energetica. Considerando le espressioni 2.2 e 2.4, poniamo che:

$$EC_{physical}(f) \propto EC_{logical}(f) \tag{2.6}$$

Cioè, il consumo di energia descritto dal punto di vista fisico può essere considerato come un indice di misura del consumo di energia definito dal punto di vista logico.

Dal punto di vista teorico, la minimizzazione della complessità computazionale richiederebbe di valutare in che misura è lontano l'attuale complessità computazionale C_c dal limite inferiore di una data applicazione, dalla profondità logica L_d di un output che l'applicazione intende produrre. Tuttavia, come accennato in precedenza, tale soluzione è davvero difficile da raggiungere, se non proprio irrisolvibile (almeno allo stato attuale dell'arte). In primo luogo, sarebbe necessaria una metodologia generale per la definizione della complessità computazionale di un problema generico. In secondo luogo, l'attuale complessità computazionale C_c dell'applicazione deve essere correttamente determinato. In terzo luogo, un modo per identificare il più breve programma che risolve il problema dovrebbe essere determinato (cioè, definire la profondità logica L_d del problema). Come quarto punto, un confronto tra i valori di C_c e L_d dovrebbe essere eseguito in modo da valutare in che misura l'applicazione è lontana dall'ottimo teorico. Dato che, due o tre fasi non possono essere completamente automatizzate, e richiederebbe

di individuare il minimo della profondità logica per ogni possibile problema (che è chiaramente un requisito non soddisfacibile), l'attenzione deve essere portata sulla definizione di metodologie di benchmarking per confrontare applicazioni specifiche.

2.2.2 Green Software nel Mondo

Il Green IT è una realtà sempre più diffusa nel mondo aziendale. L'adesione a questa metodologia, come detto anche in precedenza, comporta miglioramenti non solo per l'azienda che riduce i costi e migliora la propria immagine verso il cliente; anche l'ambiente ne guadagna poichè si ottiene una riduzione del consumo di energia elettrica e, di conseguenza, una diminuzione delle emissioni di carbonio nell'atmosfera.

Fin ad ora la maggior parte degli studi sul Green Computing sono stati focalizzati sul risparmio energetico dell'infrastruttura IT nel complesso, concentrandosi principalmente sui problemi di raffreddamento e gestione dell'energia.

Il software assume un ruolo attivo nel gestire e controllare le infrastrutture, ottenere dati che descrivono il consumo energetico del sistema e rielaborarli.

Ma il termine Green Software può assumere una diversa accezione: lo sviluppo di codice di qualità orientato al risparmio energetico.

Precedenti ricerche sull'argomento si sono focalizzate sul Low Power Software per i sistemi embedded.

Il low power software si occupa di progettazione orientata al risparmio di energia per sistemi che hanno stringenti requisiti di consumo energeti-

co. La diminuzione del consumo in questi dispositivi, permette di ridurre i limiti imposti alla miniaturizzazione, dilatare i tempi tra ricariche dei dispositivi ed essere quindi competitivi sul mercato.

I sistemi embedded sono caratterizzati anche da ristretti limiti di spazio di memoria a disposizione. Una metodologia utilizzata per ovviare alla ristrettezza di memoria è la code compression [22]; effetti positivi si hanno anche sul consumo di energia poichè il codice compresso occupa una quantità inferiore di memoria e, di conseguenza, è necessario un numero minore di accessi alla main memory dalla quale deriva un minore dispendio in termini energetici. Inoltre, la riduzione di accessi in memoria diminuisce la dissipazione di energia nel bus e nelle interconnessioni. Su questo tipo di dispositivi diventa fondamentale anche il *task scheduling* che viene considerato uno dei metodi più comuni per ottenere un consumo energetico inferiore. In particolare, nei light-weight embedded system, lo scheduling risparmia energia spegnendo i dispositivi non operativi in un dato momento. Inoltre, gli elementi di elaborazione in questi sistemi solitamente servono richieste diverse in tempi differenti. L'ordinamento dell'esecuzione dei tasks permette una migliore gestione dei tempi di idle ed un minore spreco di energia. Nei dispositivi mobili, l'unità che consuma più energia è la CPU. Sono state quindi sviluppate diverse tecniche che permettono una riduzione dello spreco energetico della CPU implementando il Dynamic Voltage and Frequency Scaling (DVFS). Questa tecnica permette di modificare dinamicamente il voltaggio della CPU per garantire la minima energia necessaria affinché la frequenza del circuito possa processare il system workload rispettando i vincoli di tempo/throughtput e riducendo il consumo energetico dipendente in modo quadratico dal livello di voltaggio fornito dal sistema.

Studi volti al risparmio energetico hanno riguardato in passato soprattutto l'ottimizzazione del consumo nei portable device per ovvie limitazioni imposte dal funzionamento a batteria dei dispositivi. Tali studi non sono stati svolti in un'ottica *green*, ma il driver fondamentale è stato l'incremento della competitività dei prodotti sul mercato. In questa ottica, ad esempio, Intel ha condotto una ricerca [23] sui portable device e sulle metodologie di risparmio energetico. In questa indagine, Intel afferma che buoni risultati di risparmio energetico derivano da:

- Multi-Threading
- Compilatori, librerie e Istruction Set
- Efficienza dei dati
- Context Awareness

L'utilizzo di applicazioni multithreading apporta miglioramenti dal punto di vista delle performance poichè più job vengono eseguiti in modo parallelo; di conseguenza, poichè le risorse del sistema vengono utilizzate per un tempo minore rispetto a quanto accade con applicazioni single-threaded, si ottengono miglioramenti più o meno significativi anche dal punto di vista energetico.

I risultati dello studio hanno portato a diverse considerazioni:

- Utilizzare applicazioni multi-threading in modo corretto porta ad un miglioramento delle performance e del consumo di potenza. Il

risparmio energetico deriva direttamente dal tempo inferiore impiegato dall'applicazione che utilizza più thread nello svolgere il compito assegnato.

- Thread con carichi di lavoro sbilanciati possono portare ad un degrado delle prestazioni e non apportare miglioramenti nel consumo energetico rispetto a thread bilanciati. Thread non bilanciati possono causare una migrazione di thread tra i core del processore inducendo un abbassamento nella frequenza del processore in sistemi adattivi.

Un altro metodo per ottenere migliori performance ed efficienza computazionale è l'utilizzo di compilatori che ottimizzano il codice, librerie focalizzate sulle performance che contengono algoritmi ottimizzati e codice per implementare funzioni comuni e instruction set avanzati che aiutino gli sviluppatori a trarre vantaggio dalle nuove caratteristiche di processori realizzati per specifiche applicazioni.

Anche una *gestione efficiente dei dati* può ridurre i costi dell'energia tramite la progettazione di algoritmi software che ne minimizzano il movimento dei dati, gerarchie di memoria che mantengono i dati vicino agli elementi da processare e applicazioni che usino la *cache memory* in modo ottimale.

Obiettivo della *Context Awarness* è invece quello di creare applicazioni che possono rispondere o adattarsi ai cambiamenti ambientali.

Per ambienti fisici, ad esempio, i sensori possono generare eventi ai quali le applicazioni possono reagire. Molti computer notebook possono valutare l'energia residua delle batterie e cambiare automaticamente il livello di luminosità del display o alcune applicazioni possono scrivere i dati che si trovano nella cache. In un'ottica energetica, il software deve rispondere ai cambiamenti del sistema intraprendendo azioni atte a conservare energia.

La ricerca sull'efficienza energetica si è finora concentrata sul *software embedded* cercando di migliorare le performance e configurando dinamicamente i componenti a basso livello architetturale. Ciò riduce l'utilizzo complessivo della CPU e, quindi, il consumo di energia. In confronto al software applicativo che solitamente 'eredita' l'hardware e le impostazioni di basso livello, l'embedded software può influenzare più facilmente queste impostazioni oltre ad avere il vantaggio di una minore complessità. Ciò permette lo sviluppo tramite linguaggi di basso livello ed una ottimizzazione del codice basata sul controllo di operazioni elementari. Al contrario, il software applicativo richiede linguaggi di alto livello, librerie e frameworks che hanno lo scopo di ridurre la complessità del compito di sviluppare codice ma che cambiano la natura delle applicazioni trasformandole in complessi ecosistemi nei quali le performance ed il consumo energetico diventano problematiche separate e spesso conflittuali. Non molto è stato fatto quindi, nel campo del green software designing per applicazioni di largo impiego nonostante sia stato mostrato come l'ottimizzazione energetica del software abbia ancora ampio spazio di miglioramento e, soprattutto, come benefici tangibili in termini di diminuzione dei costi dell'energia possano essere apportati. In

tal senso, Microsoft Research in collaborazione con Stanford University sta sviluppando un framework chiamato 'Green' [24] che permette ai programmatori di trarre vantaggi, nell'ambito del consumo energetico, dall'introduzione nel codice di un certo livello di approssimazione.

Secondo gli autori della ricerca, molti domini applicativi offrono la possibilità di valutare un tradeoff tra qualità del servizio (QoS) e risultato, per ottenere miglioramenti nelle performance e nella riduzione del consumo energetico. Green dovrebbe permettere al programmatore di approssimare funzioni particolarmente costose e cicli, operando in due fasi. Nella fase di *calibrazione* viene costruito un modello della perdita di qualità di servizio derivata dall'approssimazione. Questo modello viene poi utilizzato nella fase *operazionale* per permettere decisioni sull'approssimazione da introdurre, basate su vincoli di QoS specificati dal programmatore. La fase operazionale include una 'funzione di adattamento' che monitora il comportamento a runtime ed effettua cambiamenti sulle decisioni prese in precedenza e sul modello costruito per garantire livelli elevati di qualità del servizio.

Le funzioni e i loop vengono considerati l'obiettivo ideale per realizzare l'approssimazione poichè sono modulati e *time-consuming*. E' il programmatore stesso a fornire diverse versioni approssimate della stessa funzione mentre i cicli vengono resi più brevi rispetto ai cicli originali. Per calcolare la perdita di qualità, cioè di precisione, Green calcola la QoS come la differenza tra valori ottenuti dalla funzione approssimata e i valori ottenuti dalla funzione precisa originaria con i medesimi dati in input. Per quanto riguarda i cicli, il programmatore deve sviluppare una funzione che calcoli la perdita di qualità derivata dalla riduzione delle iterazioni del ciclo.

Green realizza poi una versione del programma chiamata 'versione di calibrazione' la cui esecuzione, utilizzando dei dati di prova come input, permette la costruzione di un modello di QoS che quantifichi la perdita di qualità derivata dall'introduzione di approssimazione; nel contempo vengono eseguite misurazioni di performance e consumo energetico. Il modello costruito viene usato in seguito per decidere se approssimare o impiegare il codice originale in modo da garantire il Service Level Agreement specificato inizialmente. Poichè con gli input reali di un programma il degrado di qualità può differire da quello preventivato in fase di calibrazione, Green si occupa di fare controlli a campione durante l'esecuzione del programma, aggiornando di conseguenza le decisioni in merito all'utilizzo dell'approssimazione. Risultati sperimentali indicano che il framework in via di sviluppo può ridurre il consumo energetico di diverse applicazioni al prezzo di un degrado preventivato e controllato in termini di precisione degli output. In particolare, la sperimentazione su *Live Search*, un motore di ricerca web commerciale, ha portato ad un risparmio energetico del 14% a fronte di una perdita qualitativa pari allo 0.27%.

2.3 Metriche per valutare il Software

La scelta di studiare gli aspetti qualitativi del software ha come diretta conseguenza la necessità di avere un set di metriche a disposizione per misurare determinati aspetti. La letteratura è ricca di metriche per valutare diversi aspetti di qualità del software [33], [34], ma nessuno di essi nasce come orientato all'efficienza energetica. La nostra ricerca ci ha portato a pensare ad un set di metriche orientati a questo aspetto, che rispettino le caratteristiche discusse nel successivo paragrafo.

Presenteremo le nostre metriche nel paragrafo 4.3 .

2.3.1 Definizione di metrica

Una metrica di valutazione del software viene generalmente definita come la quantificazione di una generica proprietà associata ad un prodotto software o alle attività del processo di produzione ad esso correlato [31].

Una buona metrica di valutazione del software non deve avere una funzione puramente descrittiva del prodotto o del processo a cui viene applicata, ma deve risultare utile al fine di stimare i parametri ad esso legati.

Una metrica ideale deve avere le seguenti caratteristiche [32]:

- **Semplicità e precisione della definizione**: la chiarezza con cui è definito l'oggetto della metrica;
- **Oggettività**: la metrica deve prescindere il più possibile da criteri di valutazione soggettivi;
- **Facilità di misurazione**: la metrica deve essere ricavabile in modo semplice, con un costo di misurazione ragionevole;
- **Validità**: la metrica deve essere focalizzata su ciò che si intende effettivamente misurare;
- **Robustezza**: la metrica deve risultare relativamente insensibile a piccoli cambiamenti del processo o del prodotto a cui viene applicata.

2.3.2 Qualità del Software

Per qualità del software si intende la misura in cui un prodotto software soddisfa un certo numero di aspettative, rispetto sia al suo funzionamento sia alla sua natura interna.

In questo paragrafo riprenderemo le direttrici di classificazione della qualità relative al software proposte in Ghezzi [34]: qualità interne e qualità esterne da un lato, e qualità di prodotto e qualità di processo dall'altro.

Qualità interne ed esterne

Le qualità esterne sono visibili agli utenti del sistema mentre le qualità interne riguardano gli sviluppatori. In generale, gli utenti del software hanno interesse soltanto nelle qualità esterne, ma sono le qualità interne, che in larga misura hanno a che fare con la struttura del software, ad aiutare gli sviluppatori a raggiungere le qualità esterne. In molti casi tuttavia le qualità sono tra di loro strettamente correlate e la distinzione tra qualità interne ed esterne non è così marcata.

Qualità del processo e qualità del prodotto

Per realizzare un prodotto software si utilizza un processo. E' possibile attribuire alcune qualità ad un processo, anche se molto spesso le qualità del processo sono spesso correlate con quelle del prodotto. Nell'affrontare le qualità del software è bene comunque distinguere tra qualità del processo e qualità del prodotto.

Il termine prodotto di solito si riferisce a quanto viene alla fine consegnato al committente. Anche se quest'ultima è una definizione accettabile dal punto di vista del cliente, non è corretta per lo sviluppatore, in quanto questi, nel corso del processo di sviluppo, produce una serie di prodotti intermedi. Il prodotto effettivamente visibile al committente consiste nel codice eseguibile e nei manuali utente, ma lo sviluppatore produce un numero ampio di artefatti, quali i documenti di specifica dei requisiti e di progetto, i dati di test ecc. Questi prodotti intermedi sono spesso soggetti agli stessi requisiti di qualità del prodotto finale.

Principali qualità del Software

In questa sezione verranno elencate alcune tra le più significative caratteristiche di qualità del software. Per la loro spiegazione si rimanda a Colombo,2006 [35]

- Affidabilità
- Comprensibilità
- Correttezza
- Manutenibilità, riparabilità ed evolvibilità
- Riusabilità
- Robustezza
- Testabilità
- Usabilità

Tutti gli studi fino a questo momento condotti escludono dalle qualità del software un aspetto che per il nostro lavoro sarà fondamentale: l'ef-

ficienza energetica del software, o meglio, una caratteristica legata alle prestazioni energetiche di un software.

Volendo classificare questa caratteristica secondo la classificazione di Ghezzi et al. [34], essa si collocherebbe tra le qualità di prodotto, e sarebbe considerata sia come interna sia come esterna. Infatti, l'efficienza energetica di un software ha, senza dubbio, origine nella natura intrinseca del codice, ma genera effetti misurabili esternamente da qualunque utente.

2.3.3 Metriche Green

La teoria della complessità computazionale di Margolous e Levitin descritta in precedenza, ha guidato la ricerca di definizioni di complessità in altri settori scientifici. Ciò ha permesso, nell'ambito del progetto del Politecnico, di definire metriche *green* che potessero dare un indice del grado di efficienza energetica di una applicazione [27].

Le categorie di metriche individuate sono:

- Metriche innovative sul linguaggio macchina [25]
- Metriche classiche di qualità del design
- Metriche di entropia dell'espressività su codice sorgente
- Metrica Green vettoriale su codice macchina
- Metrica Green vettoriale su bytecode Java

Le **metriche classiche di qualità del design** appartengono alla Suite CK [11] e sono per la maggior parte riconducibili a linguaggi ad

oggetti. Tali metriche possono ulteriormente essere suddivise in due classi:

- Una prima classe è calcolabile su gruppi strutturati di classi che vanno a comporre un programma. Essa comprende: Attribute Hiding Factor, Attribute Inheritace Factor, Coupling Factor, Method Hiding Factor, Method Inheritance Factor, Polymorphism Factor e complessità ciclomatica.

- Una seconda classe che si riferisce ad una serie di caratteristiche del singolo oggetto: Binary Length, Coupling Between Objects, complessità ciclomatica, Data Abtraction Couplings, Depth of Inheritance Tree, funzionalità, Information Flow, numero di figli, numero di costruttori, numero di campi, numero di metodi, Response For a Class, Weighted Method per Class.

L'**entropia dell'espressività del linguaggio** è una metrica che indica quanto un determinato frammento di codice sfrutti la potenzialità espressiva del linguaggio di programmazione che utilizza. In altre parole, maggiore entropia significherà un utilizzo più vario dei diversi simboli presenti nel codice, ciascuno con minore frequenza relativa. La base teorica di tale metrica è rappresentata dall'entropia di Shannon, la quale può essere un indice di comprimibilità di una stringa in un determinato alfabeto. In questo caso, l'alfabeto diviene il linguaggio di programmazione utilizzato mentre la stringa è rappresentata da un certo frammento di codice scritto nel linguaggio stesso. Per simbolo di intende invece una qualunque sequenza di caratteri riconoscibile da un parser del linguaggio. In accordo con questa definizione, i simboli sono

suddivisi in diverse categorie (keywords, variabili, simboli matematici, funzioni, ecc.), a seconda della loro provenienza nella grammatica del linguaggio e del loro significato semantico.

La **metrica Green su codice macchina** è una metrica di tipo vettoriale nella quale ciascun elemento diventa una rappresentazione del peso che il codice ha ad un certo livello di annidamento, come somma dei pesi delle singole microistruzioni a quel livello. Tramite tale metrica è possibile analizzare la struttura del codice considerando i diversi livelli di annidamento dei cicli e le istruzioni di selezione. Il risultato è un vettore di valori, ciascuno dei quali rappresenta il peso totale del codice all' i-esimo livello di annidamento. Il peso di ogni singola istruzione è stato calcolato empiricamente; sono poi state identificate delle regole per individuare inizio e fine di cicli ed istruzioni di selezione.

La **metrica Green vettoriale su bytecode Java** è analoga alla precedente con la differenza che tale metrica opera le sue computazioni su file in bytecode.

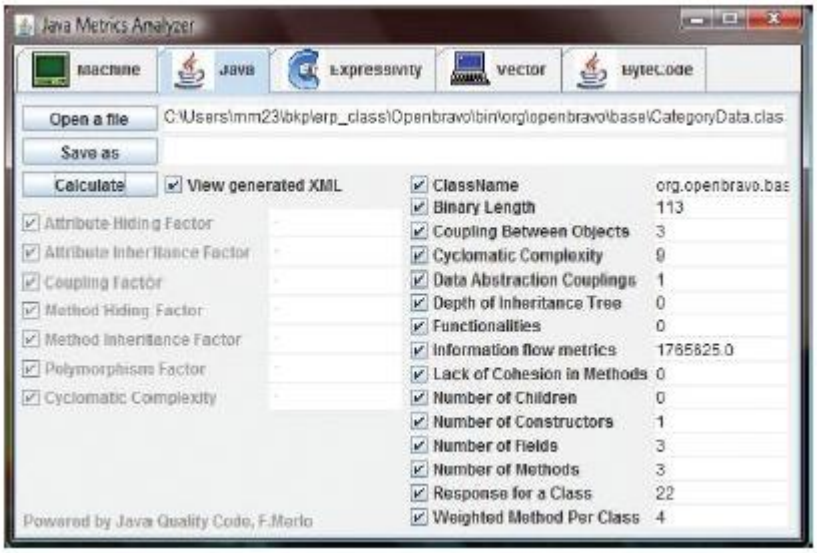


Figura 2.7: Interfaccia del tool sviluppato per la misurazione delle metriche green

Capitolo 3

Ipotesi di ricerca

Nel presente capitolo andremo a spiegare l'ipotesi dalla quale ha avuto inizio il nostro lavoro di tesi.

Supponendo che si abbiano programmatori con la stessa esperienza, in generale, ci aspettiamo che l'intensivo uso di librerie esterne da parte di un programma ha un impatto significativo sul consumo energetico.

Le librerie, in generale, semplificano il lavoro agli sviluppatori e salvaguardano loro dal capire le complesse strutture implementative. Gli aiuti forniti dalle librerie possono portare un programmatore ad essere più efficiente nell'implementazione del software, ma questo può comportare un lavoro extra per il processore durante la fase di esecuzione dell'applicazione.

In secondo luogo le librerie esterne sono progettate per essere generali e abilitate al riutilizzo di porzioni di codice [14],[15]. E' probabile che esse non abbiano caratteristiche specifiche e uniche per svolgere un determinato compito ma che contengono anche funzionalità aggiuntive che sono causa di lavoro supplementare per il processore quando il programma viene eseguito. Ad esempio, un'applicazione può contare su

alcune librerie di riferimento per accedere al database. Queste librerie sono facilmente utilizzabili e gestibili da parte dei programmatori, ma essendo generali, in quanto devono inglobare tutte le operazioni possibili e strutture dati, anche se non sono utilizzate nella totalità, aumentano la complessità di esecuzione sovraccaricando il processore e impattando sul consumo energetico. Questo fenomeno è più evidente per le applicazioni più complesse strutturalmente [1] in quanto, se fanno un uso intensivo delle librerie, sovraccaricano in modo eccessivo il processore, poichè se, ad esempio, si esegue una singola funzionalità si devono attraversare più layer. Si pensa quindi sia consigliato usare un set di istruzioni specifiche per eseguire una singola funzionalità, anche se questo approccio è meno gestibile e può richiedere uno sviluppatore più esperto, ma potrebbe avere il vantaggio di ridurre il consumo energetico.

Perciò, le nostre aspettative sono:

- **(H1)** la struttura del codice di un'applicazione influisce sul consumo energetico

- **(H2)** l'effetto positivo sul consumo energetico, derivante dall'uso intensivo di librerie esterne, è più pronunciato per le applicazioni meno complesse strutturalmente

[1] La complessità strutturale è stata calcolata in base alle metriche classiche di qualità del design

Capitolo 4

Metodologia

In questo capitolo si descriveranno brevemente le linee guida di sviluppo di questo lavoro. Una volta analizzato lo stato dell'arte del Green IT, si è passati a cercare di capire se l'uso di librerie esterne in un software ha un impatto sull'efficienza energetica dello stesso. Da qui l'individuazione delle macrofasi dell'intero progetto. Il lavoro si è dapprima concentrato sullo studio delle metriche esistenti per quanto riguarda l'impatto del software sui consumi, successivamente sono state create delle metriche nuove per valutare l'impatto sui consumi dovuto all'uso delle librerie esterne nell' implementazione del software. Infine, ci si è concentrati su un'articolata fase di applicazioni e validazione empirica dei risultati.

4.1 Macrofasi del progetto

In questo paragrafo è inquadrato tutto il lavoro nell'insieme del suo progetto di appartenenza, mirato a capire che impatto ha sull'efficienza energetica l'uso di librerie esterne nell'implementazione di applicazioni software. Sono quindi elencate le macrofasi del progetto:

1. Comprensione dei problemi e studio dello stato dell'arte
2. Studio e messa in opera di metriche utili a quantificare l'efficienza energetica delle applicazioni software
3. Upgrade di un tool che misuri l'insieme delle metriche
4. Misura dei consumi energetici delle applicazioni
5. Integrazione dei risultati in un tool software che supporti i manager nella gestione del problema dell'efficienza del software
6. Studio dell'impatto dei differenziali di consumo energetico sul Total Cost of Ownership di un data center

Il lavoro copre le prime tre fasi del progetto e prosegue il lavoro fatto da Galli [27]. Le restanti fasi sono coperte dal lavoro di Formenti e Gallazzi [30].

L' approccio generale di questo lavoro è di tipo empirico: dopo aver pensato che il modo di implementare il software ha un impatto sui consumi energetici di un calcolatore, sono state definite delle metriche, poi testate su un campione; dai risultati delle analisi incrociati coi consumi energetici sono state tratte delle conclusioni.

4.2 Impatto della tesi sul progetto

Il lavoro di questa tesi si propone di operare una prima analisi esplorativa sulle cause dell'inefficienza energetica; ma il cuore del lavoro è l'analisi dell'impatto software sui consumi, in particolare l'impatto che ha l'uso di librerie esterne nell'implementazione di applicazioni software. Questo step è necessario per poter poi validare le metriche che saranno introdotte nel prossimo paragrafo. Fatto questo si possono definire delle linee guida per la progettazione e l'implementazione del software in modo green.

4.3 Le metriche individuate

La ricerca di metriche che possano indicare i consumi energetici è un argomento molto complesso, e fino ad oggi sostanzialmente inesplorato. Dopo gli studi eseguiti da Galli [27], durante i quali sono state definite alcune metriche usate come proxy per il consumo energetico, e lo sviluppo di una prima versione di un tool, i nostri studi si sono focalizzati sui consumi energetici legati alla struttura implementativa del software. In particolare si è analizzato quale impatto energetico ha un software implementato con o senza l'aiuto di librerie esterne, quindi se è consigliato o meno sviluppare un'applicazione con l'uso di librerie esterne per poter ottenere riduzioni sul consumo energetico.

A tal fine sono state individuate metriche ad-hoc. La base comune delle metriche individuate è l'analisi del codice statico dell'applicazione. Dall'analisi di quest'ultimo sono state individuate le seguenti metriche:

- EFL

  La prima nuova metrica considerata è chiamata *EFL*. Questa metrica si basa sul conteggio delle linee di codice dei metodi esterni richiamati dalla classe 'main' dell'applicazione normalizzato sul totale delle linee di codice del software.

- EFC

  E' la seconda nuova metrica definita. La seconda metrica si basa semplicemente sul conteggio dei metodi esterni richiamati dalla classe 'main' dell'applicazione normalizzato sul totale dei metodi del software.

- EFLC

  La terza è ultima metrica definita è stata chiamata *EFLC*. Essa è una conseguenza delle metriche *EFL e EFC*, infatti usa informazioni relative alle metriche precedentemente dichiarate.

La tre metriche in un certo senso sono un indice di come è stato progettato e implementato il software. Ad esempio un valore alto delle metriche indica che l'applicazione considerata è stata sviluppata con l'uso intensivo di librerie esterne predefinite.

Nel capitolo 'Implementazione' sono descritte in dettaglio le nuove metriche elencate.

4.4 Validazione dei risultati

La parte empirica di questo lavoro si serve di un set di applicazioni campione aventi determinate caratteristiche che sono usate nella validazione

statistica dei risultati delle metriche rispetto ai consumi energetici.

4.5 Applicazioni Campione

Per poter valutare le metriche a disposizione, era necessario che tutte le applicazioni campione avessero caratteristiche comuni: prima di tutto dovevano essere open source, per poter accedere al codice sorgente, e inoltre dovevano essere scritte nello stesso linguaggio. La scelta è caduta sul linguaggio Java poichè è uno dei linguaggi più comuni usato nel contesto open source.

4.6 Validazione dei risultati

La fase finale del lavoro è quella più tipicamente statistica: avendo a disposizione le nostre metriche, si è verificato se fossero in correlazione con i consumi energetici misurati.

Capitolo 5

Implementazione delle metriche e del tool

La ricerca di un indice di merito dell'efficienza energetica dovuto all'uso delle librerie esterne del software ha portato alla definizione delle tre metriche introdotte nel capitolo precedente:

- EFL
- EFC
- EFLC

In questo capitolo, quindi, andremo a spiegare, anche con l'aiuto di esempi, l'implementazione delle tre metriche individuate, le quali utilizzano, per poter analizzare il codice sorgente, tutte un unico generatore di parser. Inoltre, nell'ultima sezione del capitolo, andremo a illustrare il tool costruito ad-hoc che permette di calcolare tutte le nostre metriche.

5.1 Parser: JavaCC

Per l'implementazione delle nostre metriche abbiamo avuto bisogno di un parser adatto al linguaggio Java. La nostra scelta è caduta su JavaCC[36]. (Java Compiler Compiler), che fa uso del sistema di parsing LL(k) per il linguaggio di programmazione Java. Quindi JavaCC genera un parser per una grammatica fornita nella notazione BNF, notazione per descrivere formalmente la sintassi di linguaggi.

In particolare JavaCC è un plug-in di Eclipse[38], che partendo da una specifica di una grammatica (con azioni semantiche), contenuta in un file testuale con estensione .jj, è capace di generare automaticamente una serie di classi Java che realizzano un analizzatore sintattico top-down per la grammatica. Il file .jj è suddiviso in tre sezioni:

1. lista di opzioni per JavaCC
2. unità di compilazione Java che inizia con emphPARSER_BEGIN(nome_parser) e termina con emphPARSER_END(nome_parser)
3. lista di regole lessicali e sintattiche
    - si possono aggiungere azioni semantiche, ovvero, porzioni di codice Java che vengono eseguite al momento dell'espansione delle produzioni.

5.2 Regole SLOC

Le nostre metriche principalmente si basano sul conteggio delle linee di codice e dei metodi. Per far questo ci siamo serviti di regole di SLOC. Esistono due tipi di misure di SLOC:

- Physical SLOC: si contano tutte le righe di testo del codice sorgente includendo anche i commenti e le linee bianche
- Logical SLOC: si contano gli "statements", ovvero le effettive istruzioni

Consideriamo per esempio il seguente segmento di codice:

```
for(i=0; i ≤ 100; i++)
{
    printf("Hello World!");
}
```

Le SLOC saranno:

- 4 Physical SLOC
- 2 Locigal SLOC

La nostra scelta è caduta sulle Logical SLOC e le regole di Java standard di conteggio che abbiamo seguito sono quelle redatte dal *Center for Systems and Software Engineering*[28] che sono riportate in Fig.5.1.

La Tabella 5.1 e la Tabella 5.2 propongono alcuni esempi di conteggio di Logical SLOC counting.

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	"*for*", "*while*", "for each" or "*if*" statement	1	Count once.	"*while*" is an independent statement.
R02	*do* {…} *while* (…); statement	2	Count once.	Braces {…} and semicolon ; used with this statement are not counted.
R03	Statements ending by a semicolon	3	Count once per statement, including empty statement.	Semicolons within "*for*" statement are not counted. Semicolons used with R01 and R02 are not counted.
R04	Block delimiters, braces {…}	4	Count once per pair of braces {..}, except where closing brace is followed by a semicolon, i.e. }; or an opening brace comes after a keyword "*else*".	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by {…}.
R05	Compiler directive	5	Count once per directive.	

Figura 5.1: Regole di conteggio di Logical SLOC

STATEMENT DESCRIPTION	*GENERAL FORM*	*SLOC COUNT*
if	if(boolean expression)	1
	statements;	1
else	if(boolean expression)	1
	statements;	1
	else	0
	statements;	1
else if	if(boolean expression)	1
	statements;	1
	else if(boolean expression)	1
	statements;	1
	else	0
	statements;	1
switch	switch(expression)	1
	{	0
	case(constant1):	0
	statements;	1
	break;	1
	default:	0
	statements;	1
	}	0
try{}catch{}	try{	1
	statements;	1
	}	0
	catch{	1
	statements;	1
	}	0

Tabella 5.1: Logical SLOC rules

STATEMENT DESCRIPTION	*GENERAL FORM*	*SLOC COUNT*
for	for(initial;condition;increment)	1
	statements;	1
empty statements	;	1
while	while(boolean expr.)	1
	statements;	1
do-while	do	1
	{	0
	statements;	1
	}while(boolean expr.);	1
for-each	for(String name: Names)	1
	statements;	1
while	while(boolean expr.)	1
	statements;	1
return	return expression;	1
break	break;	1
exit function	void exit(int return_code);	1
continue	continue;	1
function call	function_name (params);	1
assignament statements	name=value;	1
directive types	import package_name;	1
	package package_name;	1
variable declaration	type name;	1

Tabella 5.2: Logical SLOC rules

Per poter applicare le regole di *Logical SLOC* descritte, e quindi poter assegnare un valore alle metriche, le quali usano il conteggio delle linee e dei metodi del codice dell'applicazione analizzata, ci siamo serviti di azioni semantiche definite da noi nel file con estensione .jj del generatore di parser JavaCC.

5.3 *Metrica EFL*

La prima metrica che andremo a considerare per poter studiare e valutare l'impatto energetico dovuto all'uso di librerie esterne durante la fase di implementazione di un progetto software, si basa sulla seguente formula:

$$\frac{EFL}{EFL+IFL}$$

External Function Lines

EFL: Numero di linee dei metodi usati nelle librerie esterne del file sorgente

Internal Function Lines

IFL: Numero di linee del file sorgente

Il valore del numeratore rappresenta il totale delle righe di codice dei soli metodi, delle librerie esterne, usati nel file sorgente. Al denominatore è presente il valore dato dalla somma di quello al numeratore e del totale delle righe di codice del file sorgente, così da poter ottenere un valore assoluto.

5.3.1 Modulo calcolo Metrica EFL: Implementazione

Nella seguente sezione andremo a spiegare come è stato realizzato il modulo per il calcolo della metrica EFL.

Inizialmente abbiamo parsato il file sorgente per poter individuare:

- Librerie Usate (import)
- Metodi interni
- Metodi esterni

Per poter calcolare il numero di linee di codice totali per quanto riguarda il file sorgente abbiamo:

1. Conteggiato le righe di codice di ogni singolo metodo interno
2. Conteggiato il numero di librerie usate dal file sorgente(import)
3. Sommato i valori ottenuti nei precedenti punti

Diversamente per poter calcolare la somma di tutte le righe di codice dei metodi esterni abbiamo:

1. Individuato le librerie esterne del file sorgente
2. Individuato i metodi interni di ogni singola libreria usata
3. Confrontato i metodi interni della libreria con quelli esterni del file sorgente

Se viene individuato nella libreria il metodo esterno contenuto nel file sorgente, ne vengono contate le righe di codice. Tutti i singoli risultati ottenuti sono sommati, ottenendo così il numeratore della nostra metrica.

A questo punto avendo a disposizione tutti i parametri relativi alla EFL, di quest'ultima viene calcolato il valore numerico.

5.3.2 Esempio per calcolo Metrica EFL

Nella Fig.5.2 è riportato un esempio di calcolo della metrica EFL.

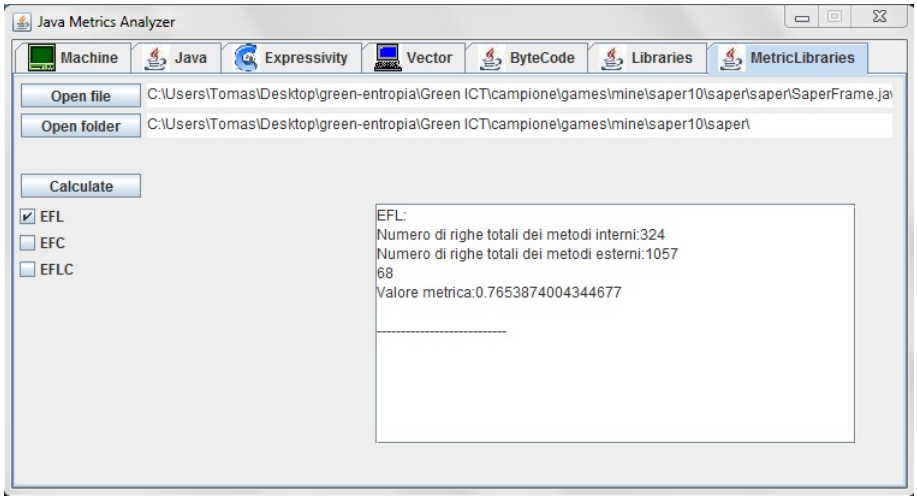


Figura 5.2: Esempio di calcolo metrica EFL

5.4 *Metrica EFC*

La metrica EFC che abbiamo considerato va semplicemente a considerare il numero dei metodi esterni e interni, mettendoli in relazione tra loro tramite la seguente formula:

$$\frac{EFC}{EFC+IFC}$$

External Function Called

EFC: Numero dei metodi usati nelle librerie esterne del file sorgente

Internal Function Called

IFC: Numero di metodi del file sorgente

Al numeratore è presente il valore numerico che equivale alla somma del numero dei metodi esterni chiamati nel file sorgente, mentre il valore assegnato al denominatore deriva dalla sommatoria del valore numerico precedentemente ricavato e dal totale dei metodi interni del file sorgente.

Per poter valutare il numero dei metodi interni ed esterni, e per poter assegnare il valore numerico al numeratore e al denominatore ci viene in aiuto, come nel caso della metrica EFL, il generatore di parser JavaCC; andremo a dettagliare l'implementazione della metrica EFC nella successiva sezione.

5.4.1 Modulo calcolo Metrica EFC: Implementazione

L'implementazione della metrica EFC è molto semplice. Principalmente ci siamo serviti di ciò che è stato fatto per la precedente metrica, infatti nella metrica EFL per prima cosa andiamo a individuare i metodi interni ed esterni, per poi contare le linee di codice che compongono ogni metodo.

Avendo quindi a disposizione i metodi interni ed esterni, possiamo individuare il loro numero relativamente all'applicazione considerata. Successivamente, per le finalità della metrica in questione, andiamo ad effettuare l'operazione di divisione come mostrato nella formula della metrica EFC, calcolando il valore numerico di interesse.

5.4.2 Esempio per calcolo Metrica EFC

Nella Fig.5.3 è riportato un esempio di calcolo della metrica EFC.

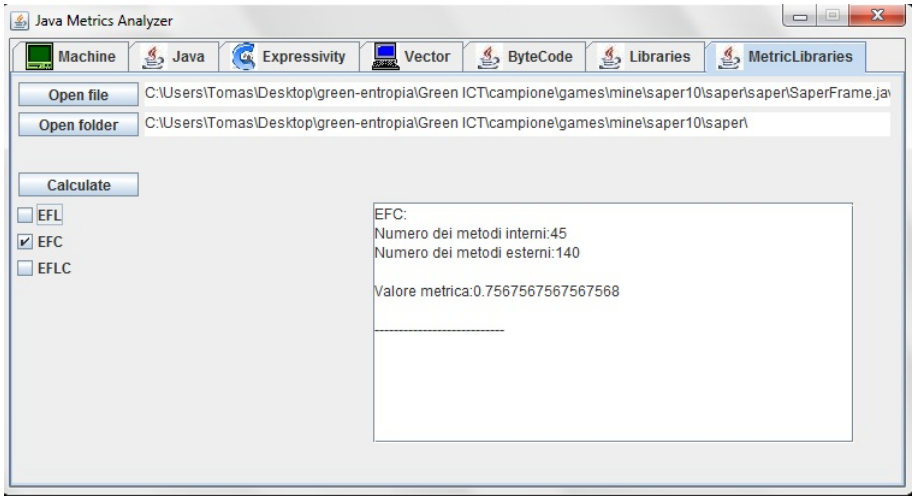


Figura 5.3: Esempio di calcolo metrica EFC

5.5 *Metrica EFLC*

La metrica EFLC mette in relazione alcune informazioni ricavate dalle due metriche descritte in precedenza e usa altri dati ricavati sempre mediante l'aiuto del generatore di parser JavaCC.

La formula che andiamo ad utilizzare per ricavare il valore numerico relativo alla metrica che ci interessa descrivere in questa sezione è la seguente:

$$\frac{EFLC}{EFLC+IFLC}$$

External Function Lines Called

EFLC:*numero di linee dei metodi usati nelle librerie esterne* $*$ *numero di chiamate al metodo*

Internal Funtion Lines Called

IFLC:*numero di linee dei metodi del file sorgente* $*$ *numero di chiamate al metodo*

Il valore numerico al numeratore rappresenta il totale delle linee di codice di ogni metodo esterno, le quali prima però vengono moltiplicate per il numero di volte che il metodo esterno è presente nel file sorgente, e successivamente vengono sommate.

Ciò che compare al denominatore deriva dal valore che abbiamo al numeratore al quale si somma il totale delle linee di codice di ogni metodo, questa volta, interno del file sorgente, le quali, come descritto in precedenza, in un primo momento sono moltiplicate per quante volte è presente il metodo interno considerato, e successivamente sono sommate.

Riteniamo che questa metrica sia la più significativa in quanto tiene conto sia del numero delle chiamate che delle linee di codice.

5.5.1 Modulo calcolo Metrica EFLC: Implementazione

Come accennato in precedenza, la metrica EFLC utilizza le informazioni messe a disposizioni dai due moduli precedentemente descritti, in particolare parte dell'implementazione della metrica EFL viene riutilizzata per conteggiare le linee di codice di ogni metodo, sia interno sia esterno, mentre parte della realizzazione della metrica EFC è usata per poter conteggiare il numero dei metodi usati sia interni sia esterni.

Quindi avendo a disposizione tutte le chiamate ai metodi con le proprie linee di codice, si effettua la sommatoria dei prodotti del numero di chiamate di ogni metodo per le proprie linee di codice ottenendo così il numeratore della metrica EFLC. Successivamente per ottenere il valore della metrica EFLC andiamo ad applicare la formula della metrica in questione.

5.5.2 Esempio per calcolo Metrica EFLC

Nella Fig.5.4 è riportato un esempio di calcolo della metrica EFLC.

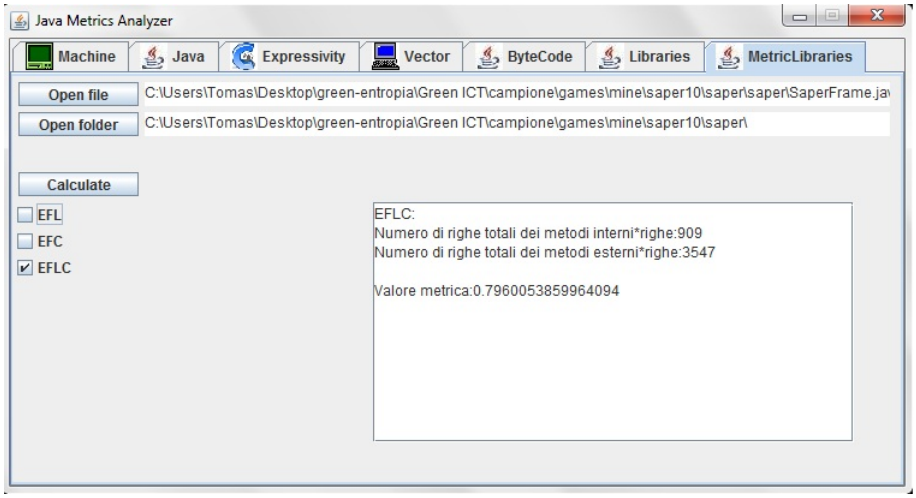


Figura 5.4: Esempio di calcolo metrica EFLC

5.6 Il tool

Le metriche fin qui descritte con le altre discusse nel progetto di tesi di Galli [27] sono tutte calcolabili tramite un tool creato ad-hoc che si chiama JMA (Java Metrics Analyzer). Sviluppato in Java, esso si suddivide in sette sezioni, ciascuna delle quali fornisce la possibilità di eseguire computazioni diverse. In seguito si descriveranno brevemente le sette sezioni.

5.6.1 Sezione sulle metriche innovative

La prima sezione si occupa delle metriche innovative. Oltre a mettere a disposizione le funzionalità di calcolo delle metriche di entropia di Shannon, entropia di Renyi, informazione di Chernoff, dimensione frattale, correlazione e entropia in eccesso, fornisce altre informazioni sul codice macchina analizzato: il numero totale di istruzioni lette, il numero di istruzioni diverse trovate, il numero di sub-routine ed una lista delle istruzioni macchina trovate, ciascuna con la propria frequenza di utilizzo.

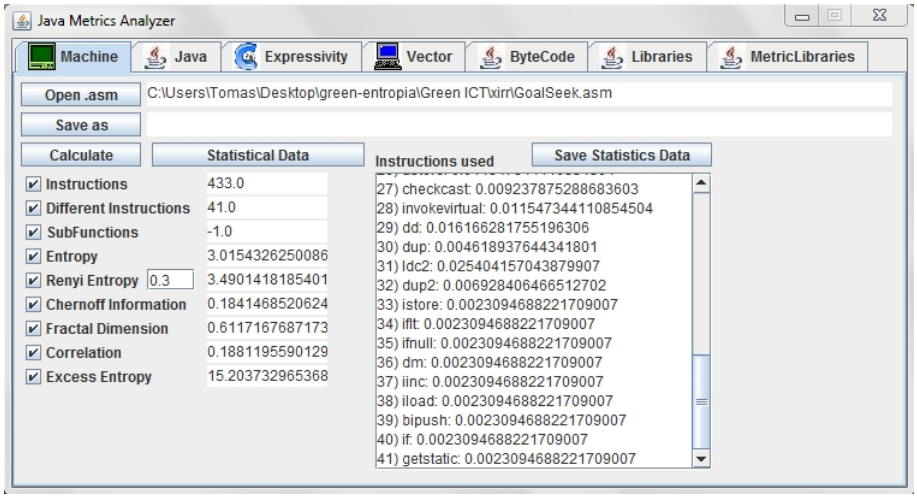


Figura 5.5: Snapshot del tab sulle metriche innovative nel tool JMA

5.6.2 Sezione sulle metriche classiche

La seconda sezione del tool riguarda le metriche classiche di qualità del design. Per far ciò, il tool si appoggia ad un altro software, JQC (*Java Quality Code*), fornendogli un'interfaccia user-friendly.

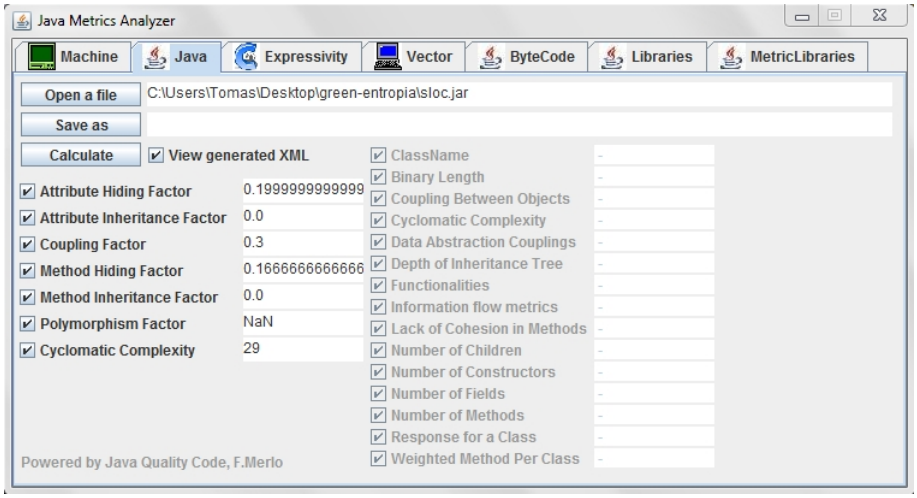


Figura 5.6: Snapshot del tab sulle metriche classiche di qualità del design nel tool JMA

5.6.3 Sezione sull' entropia dell'espressività

La terza sezione del tool tratta il calcolo dell'*entropia dell'espressività*, che è basata sull'*entropia di Shannon* e sottolinea quanto un'applicazione sfrutta le funzionalità di un linguaggio. Per calcolare questa metrica è richiesto il codice sorgente e al momento il tool è in grado di analizzare correttamente soltanto il linguaggio Java. Se il valore in output della metrica è alto, significa che il programma usa un alto numero di differenti simboli, se è basso, significa che gli stessi tipi di simboli sono usati più volte.

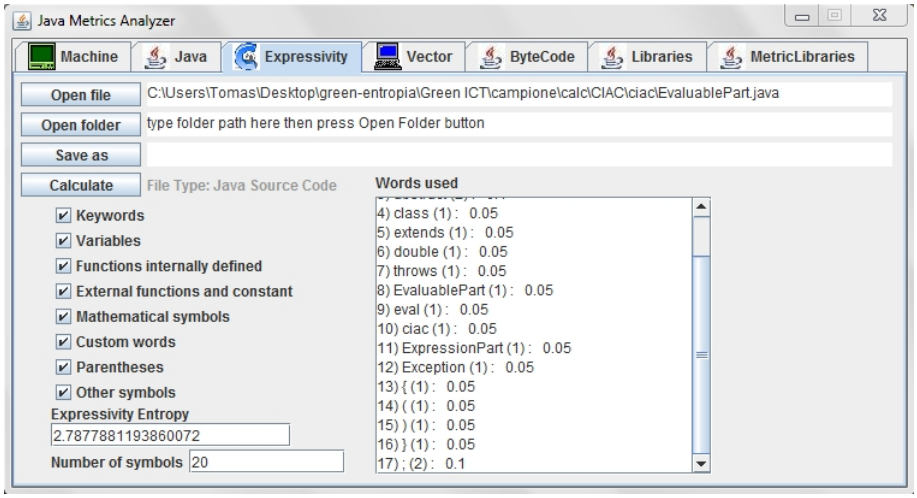


Figura 5.7: Snapshot del tab sull'entropia dell'espressività nel tool JMA

5.6.4 Sezione sulla metrica Green applicata al codice macchina

La quarta sezione del tool fornisce la funzionalità di calcolo della metrica Green. Essa da la possibilità di inserire in maniera parametrica il peso delle istruzioni di selezione, e fornisce in output, oltre che il vettore della metrica, la lista delle istruzioni trovate e la loro struttura di annidamento.

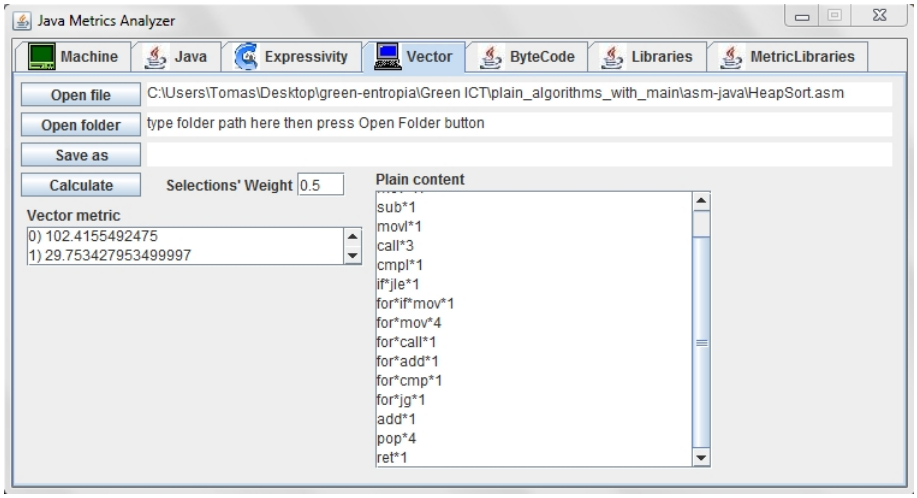


Figura 5.8: Snapshot del tab sulla metrica Green applicata al codice macchina nel tool JMA

5.6.5 Sezione sulla metrica Green applicata al bytecode Java

La quinta sezione del tool è simile alla quarta: essa fornisce la funzionalità di calcolo della metrica Green sul bytecode Java, che mira ad essere un buon proxy per il reale consumo di un'applicazione. Fornisce inoltre la possibilità di modificare i parametri di peso per le istruzioni di selezione e per le eccezioni, oltre a quella di selezionare un file di configurazione apposito per la gestione dei pesi delle singole istruzioni.

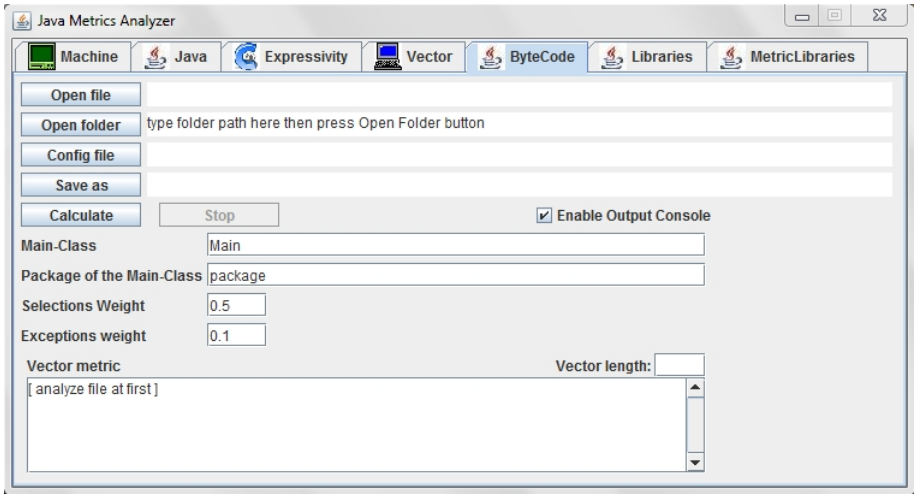


Figura 5.9: Snapshot del tab sulla metrica Green applicata al bytecode Java nel tool JMA

5.6.6 Sezione sulle librerie utilizzate

La sesta sezione, non direttamente legata ad un gruppo di metriche, fornisce informazioni riguardanti le librerie usate da un dato software. Avendo a disposizione il percorso della cartella in cui sono contenuti tutti i file sorgenti del programma da analizzare, esso li legge e riassume in maniera più comprensibile le librerie utilizzate, siano esse esterne e non.

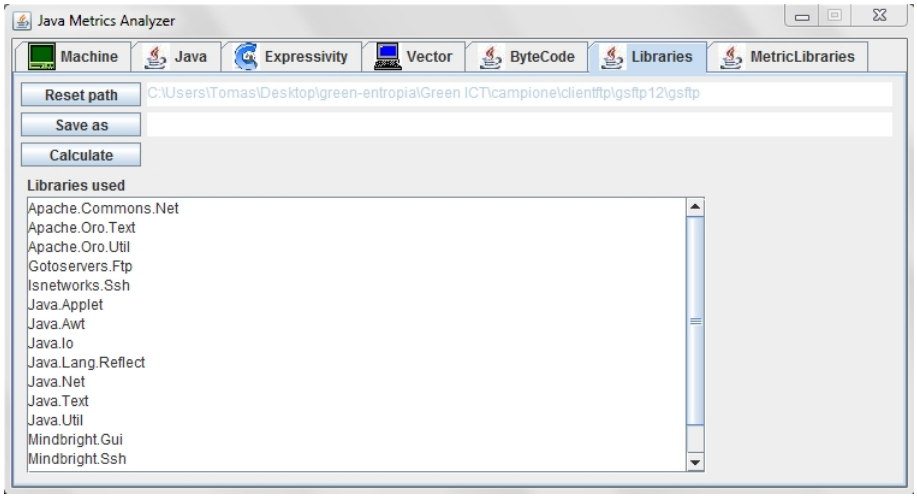


Figura 5.10: Snapshot del tab sulla raccolta di informazioni riguardanti le librerie usate da un dato software nel tool JMA

5.6.7 Sezione sulle metriche EFL, EFC, EFLC

L'ultima sezione del tool fornisce il calcolo delle metriche *EFL, EFC, EFLC*. Dopo aver inserito il percorso del file "main" e impostato il percorso dove sono contenuti tutti i file sorgenti dell'applicazione da analizzare, è possibile scegliere, attraverso l'interfaccia utente, il tipo di metrica di proprio interesse; si può anche impostare il calcolo delle tre metriche contemporaneamente. Oltre al valore numerico delle metriche vengono fornite informazioni aggiuntive come il numero di linee di codice, il numero dei metodi sia delle librerie esterne sia del file sorgente principale. Al momento sono analizzabili in modo corretto soltanto programmi scritti in linguaggio Java.

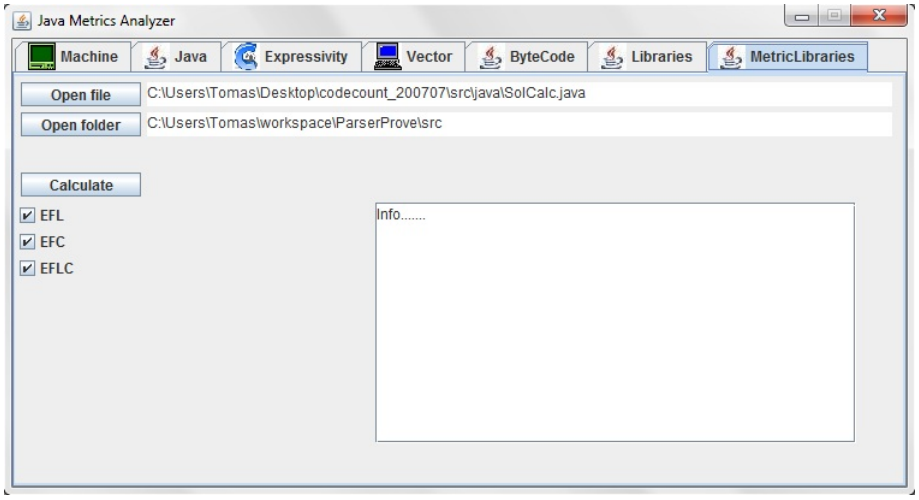


Figura 5.11: Snapshot del tab sul calcolo delle metriche EFC, EFL, EFLC di un dato software nel tool JMA

5.7 Validazione del tool

In questa sezione andremo ad illustrare il modo in cui il corretto funzionamento della finestra del tool sviluppata da noi, riguardante il calcolo delle metriche EFC, EFL, EFLC, è stato validato.

In particolare per verificare se il conteggio delle righe e dei metodi di un file scritto in Java, effettuato dal nostro tool, risultasse corretto, abbiamo realizzato dei file Java ad-hoc. Tali file, di cui noi conosciamo il numero di linee di codice e dei metodi, sono stati dati in pasto al nostro tool, il quale ha dato in output i valori di SLOC e del numero di metodi. Successivamente i risultati dati dal nostro tool sono stati confrontati con i valori che inizialmente ci erano noti.

La realizzazione dei file Java di test è stata effettuata per piccoli passi, nel senso che abbiamo aggiunto gradualmente tutti i costrutti del linguaggio Java.

Per validare ulteriormente il nostro tool abbiamo preso un campione numeroso di file Java di piccole e medie dimensioni, scritti da terzi, e ne abbiamo contato manualmente le righe e i metodi presenti, rispettando sempre le regole di SLOC, delle quali abbiamo parlato nel paragrafo 5.2, per poi confrontare il nostri valori con i risultati forniti dal tool.

Capitolo 6

Analisi Empirica

Il capitolo presenta la validazione statistica delle metriche illustrate nel precedente capitolo. Saranno illustrati la scelta delle applicazioni target che sono state considerate, come sono stati misurati i consumi energetici delle applicazioni, il calcolo delle varie categorie di metriche e l'analisi empirica dei risultati.

6.1 Campione di applicazioni Target

La natura delle metriche da calcolare ha implicato una serie di scelte sul set di applicazioni considerate: esse sono scritte in Java e sono open source per poter accedere al codice sorgente. Questi requisiti hanno portato ad appoggiarci a repository come Sourceforge [17], nei quali purtroppo esistono ben pochi criteri di standardizzazione degli upload: molti di essi non funzionano, sono versioni parziali, danno problemi con determinate versioni di Java o addirittura non forniscono il codice sorgente, in contraddizione con la natura di tali repository. Tali problemi sorgono a causa del basso livello di controllo dei progetti, a fronte di un numero di upload molto elevato.

Il campione delle applicazioni è stato ereditato dal lavoro di Galli [27] e ad esso sono state aggiunte altre applicazioni in linea con le precedenti. Esso ha la caratteristica di avere anche applicazioni di natura diversa dal punto di vista funzionale e di dimensioni non sempre confrontabili. Segue l'elenco delle categorie di programmi presi in considerazione:

- Giochi
- Mail Server
- FTP Server
- FTP Client
- Calcolatrici
- Browser
- Browser
- Editor di Testo
- Editor di Immagini
- Calendari
- eCommerce
- SpreadSheet
- ERP
- Pdf Merge

Il campione di per sè è sufficientemente esteso e vario, anche se per alcuni problemi tecnici e di diversa natura ci hanno costretto ad escludere alcuni programmi nell'analisi.

Ci siamo dovuti limitare a programmi open source escludendo molti dei software usati nel mondo aziendale 'reale', considerando un mondo in cui le tecniche di sviluppo e le caratteristiche dei prodotti sono ben diverse da quelle del mondo del codice proprietario.

6.2 Misurazione dei consumi

Il lavoro di Formenti e Gallazzi [30] è incentrato sulla misura diretta dei consumi dei calcolatori impegnati nell'esecuzione di un programma. Il consumo energetico non è fornito calcolandolo in maniera indiretta a partire dalla temperatura dei componenti, ma è misurato fisicamente dalla quantità di corrente utilizzata per il funzionamento del calcolatore e dei suoi componenti. In seguito saranno presentati due aspetti del loro lavoro svolto, l'apparato fisico utilizzato e la generazione automatica dei carichi di lavoro per i software testati.

6.2.1 Apparato di misura

La misura della corrente assorbita da una macchina può essere portata a termine, nel caso più semplice e generale possibile, con delle semplici pinze amperometriche (come quella in Fig. 6.1).

Figura 6.1: Pinza amperometrica standard

Se la macchina da controllare è un calcolatore, sorge un grosso problema nell'utilizzo di questi apparecchi: la precisione della misura.

Attrezzature di questo tipo nascono per misurare i consumi di macchine molto più grosse e dispendiose di un semplice PC desktop, e quindi sono resistenti ad alte correnti, ma poco precise nel rilevare variazioni di consumo relativamente piccole. Sorge quindi la necessità di avere uno strumento di misura di maggior precisione, non presente sul mercato. Questa lacuna è colmata da uno strumento costruito da Formenti e Gallazzi nel loro precedente lavoro di tesi [30]: la *System Board*, una scheda di misurazione della corrente che scorre in un generico cavo (Fig. 6.2).

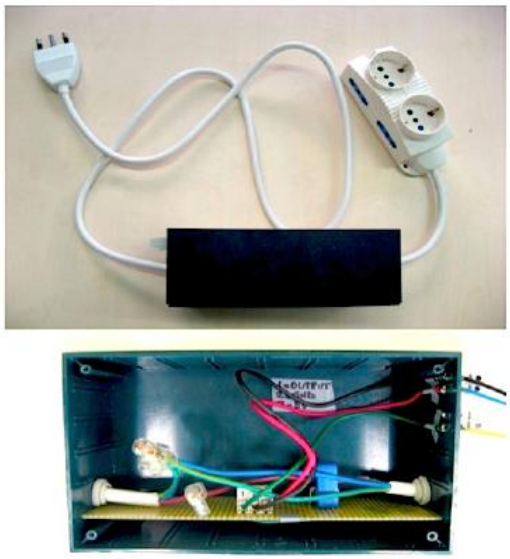


Figura 6.2: System Board per la misurazione dei consumi

La System Board permette di misurare il consumo di potenza di tutto il sistema poichè si pone a monte dell'alimentatore del computer. Essa misura la corrente mediante sensori ad effetto Hall con picchi massimi di 3,7 KW (per tensioni di corrente secondo gli standard italiani di 230 Volt), ed è uno strumento estremamente portabile e non invasivo.

La System Board viene collegata ad una DAQ Board 6.4 (Data Acquisition Board) che ha il compito di acquisire i segnali inviati dalla System Board, di elaborarli e inviarli al computer. Per acquisire i dati è stato utilizzato una DAQ della *National Instruments*, in particolare il modello NI USB-6210 DAQ [12].

Infine per la visualizzazione e il salvataggio dei dati è stato utilizzato un software sviluppato da Formenti e Gallazzi [30]. Tale software si basa su *LabVIEW* (*Laboratory Virtual Instrumentation Engineer-*

Canali Analogici d'ingresso	16
Sample Rate	250 kS/s
Risoluzione	16 bit
Range di Massimo Voltaggio	-10V /+10 V
Range di precisione	2,69 mV
Range di Minimo Voltaggio	-200 /-200 mV
Range di precisione	0,088 mV
Memoria Scheda	2095 samples
Canali Digitali I/O	4 DI / 4 DO
Massimo Input Range	0 / 5,25V
Massimo Output Range	0 / 3,8V
Connettori	Morsetti a vite

Figura 6.3: Specifiche tecniche della scheda NI USB-6210 DAQ

ing Workbench), un ambiente di programmazione visuale della *National Instruments*.

L'ultimo passo per ottenere dei dati utilizzabili e confrontabili è la loro elaborazione per il calcolo di medie ed integrali. Per questo scopo sono stati utilizzati software matematici appositi, come *Matlab* e altri.

Figura 6.4: DAQ Board NI USB-6210 della National Instruments

6.3 I carichi di lavoro

Nel misurare i consumi del software, si presenta il problema relativo ai carichi di lavoro da eseguire. Per confrontare in maniera diretta due programmi è necessario, oltre che siano *funzionalmente comparabili* (per la specifica definizione di *comparabilità* si veda [30]), essere in grado di sottoporli a carichi di lavoro più simili possibili. Questa necessità è molto intuitiva, dal momento che sarebbe assurdo confrontare i consumi energetici di due programmi, ancorché *comparabili*, mentre eseguono task molto diversi tra loro.

Scorrendo il set di applicazioni considerate, si noti che molte fondano l'interazione con l'utente tramite l'interfaccia di un browser web. Questa caratteristica ricorrente ha portato ad esplorare una serie di utility (si veda ad esempio IMacros, http://www.iopus.com/imacros e Selenium, http://seleniumhq.org) che permettono di 'registrare' le interazioni dell'utente con il browser e generare un frammento di codice (è possibili scegliere tra i vari linguaggi di programmazione) che ne ripeta il flusso. Essi sono scritti in forma di plugin e sono disponibili per tutti i browser più diffusi (Internet Explorer, Firefox, Opera,...). Modificando il codice generato, è possibile far ripetere un numero arbitrario di volte lo stesso task, eliminando il problema della variabilità del *think time* dell'utente e creando carichi di lavoro adatti alla misura dei consumi. In questo, test molto rigorosi e precisi sui consumi di determinati software sono stati portati a termine; inoltre, queste categorie di software sono, tra quelle analizzate, le più vicine al mondo del software *enterprise*: ERP, CRM, DBMS, CMS e programmi di eCommerce ne sono un esempio.

Purtroppo, data la natura molto diversa dei programmi inclusi nel set, non è stato possibile trovare il modo di uniformare i carichi di lavoro per tutti i software. Per molti di essi (si prenda ad esempio la categoria dei giochi), l'interazione con l'utente varia molto per ogni singolo programma, essendo legata alla specifica interfaccia grafica sviluppata dal software. E' risultato invece piuttosto facile pensare a carichi di lavoro plausibili, anche se non rigorosissimi dal punto di vista tecnico.

6.4 Prime Evidenze

L'obiettivo preliminare di questo lavoro è quello di dimostrare che impatto sui consumi energetici della macchina ha l'uso di librerie esterne in un applicazione. Sul consumo energetico della macchina si concentra il lavoro di Formenti e Gallazzi [30]. Installando software piuttosto diffusi e comparabili con le medesime condizioni al contorno (hardware, sistema operativo, ...), sono stati creati dei flussi equivalenti per i programmi e ne sono stati misurati i consumi.

Nel seguito di questo paragrafo si riporteranno soltanto alcuni dei risultati trovati, a dimostrazione del fatto che il software di per sè ha un forte impatto sul consumo. Si consideri il grafico in Fig. 6.5, esso rappresenta il consumo energetico di due software ERP open source, Open Bravo (www.openbravo.com) ed Adempire (www.adempire.com), sottoposti allo stesso carico di lavoro creato ad hoc.

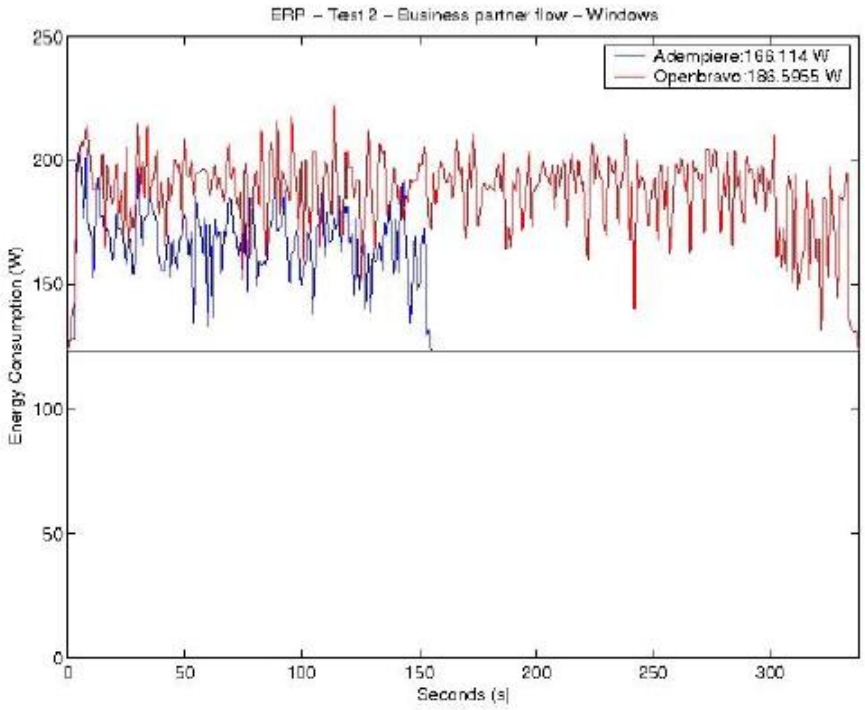


Figura 6.5: Consumi energetici nel tempo di due software ERP

Si nota subito come a parità di task i due programmi abbiamo risposte molto diverse in termini di consumo: Adempire (linea blu), oltre ad avere tempi di risposta molto più brevi (circa 150 secondi contro i circa 350 di OpenBravo), ha nel complesso prestazioni energetiche migliori. Il riquadro in alto a destra riassume i valori integrali di potenza dei due software: la differenza percentuale di circa 11% giustifica ampiamente l'affermazione fatta in precedenza: la qualità del software (intesa in una concezione relativamente nuova, in termini di consumo energetico), ha un impatto sensibile sui consumi dell'intera macchina.

6.5 Raccolta dei dati

Per ogni applicazione appartenente al set di programmi target sono stati calcolati i valori delle metriche Green generati dal tool JMA. Tali valori sono stati riportati in un foglio di calcolo elettronico che ci ha permesso di eseguire in seguito le nostre analisi. Tra le principali voci, riguardanti le nostre applicazioni, riportate sul foglio di calcolo sono presenti:

- Categoria di appartenenza dell'applicazione
- Nome dell'applicazione
- ID e Group ID dell'applicazione: Identificativo del software nel database di Sourceforge
- Valori delle metriche EFL, EFC, EFLC
- Energia media differenziale e Energia consumata dall'applicazione
- Energy Metric[1] data dalla differenza tra l'energia media differenziale e il consumo medio della categoria, il tutto diviso il consumo medio della categoria
- Entropia dell'Espressività
- Valori delle metriche classiche di design
- SLOC file "main" e SLOC librerie

In corso di analisi, sul foglio di lavoro a queste voci sono state aggiunte, in maniere metodologica, delle altre che ci hanno permesso di arrivare ad importanti risultati.

[1]Energy Metric= 1 - Efficienza Energetica

6.6 Risultati empirici di validazione

In questo paragrafo saranno presentati i grafici e i risultati derivanti dall'incrocio tra i consumi energetici delle applicazioni e i valori delle metriche formulate da noi e calcolate con il tool JMA (Capitolo 5).

6.6.1 Metrica EFL

Si inizierà con il mostrare in Fig. 6.6 il grafico che presenta su un asse il valore dell'Energy Metric incrociato con il valore della nostra metrica EFL.

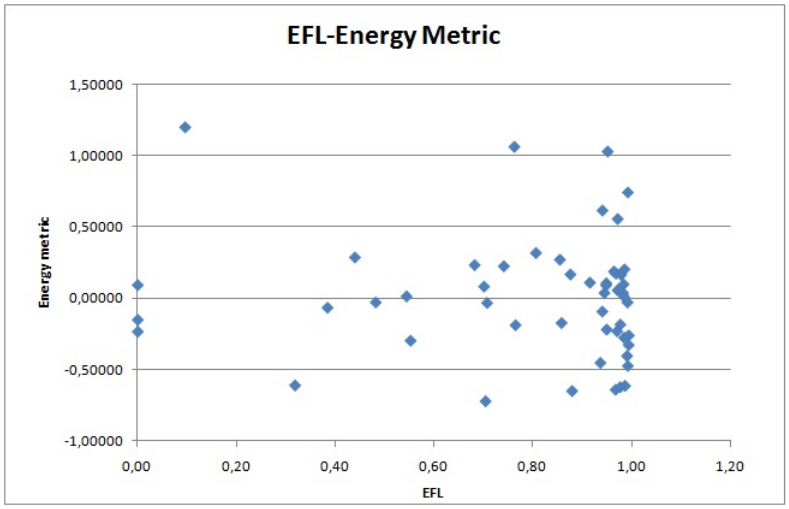


Figura 6.6: Energy Metric rispetto alla metrica EFL

Il grafico in Fig. 6.6, apparentemente, non mostra comportamenti interessanti. Poichè la nostra metrica EFL è indirettamente influenzata dalla dimensione del programma stesso, abbiamo pensato di normalizzare i valori della nostra metrica con un indice della dimensione del programma analizzato. In questo caso, si è deciso di dividere i valori per il *Binary Length*, che rappresenta la lunghezza del bytecode del

programma. Operando questa normalizzazione, si ottiene il grafico di Fig.6.13.

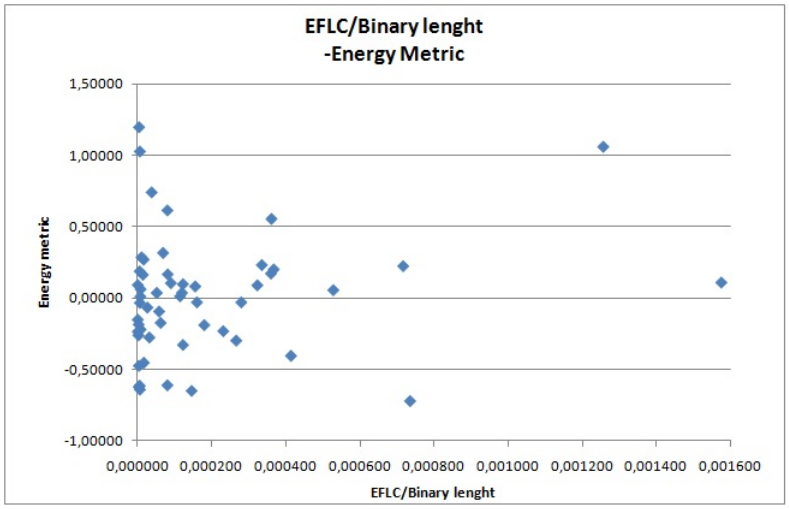


Figura 6.7: Energy Metric rispetto alla metrica EFL (normalizzata su Binary Lenght)

Anche in questo caso sembrerebbe che non ci siano comportamenti interessanti. Abbiamo quindi provato a clusterizzare il campione di applicazioni, secondo i criteri illustrati nel paragrafo 6.8, ottenendo il grafico di Fig. 6.8.

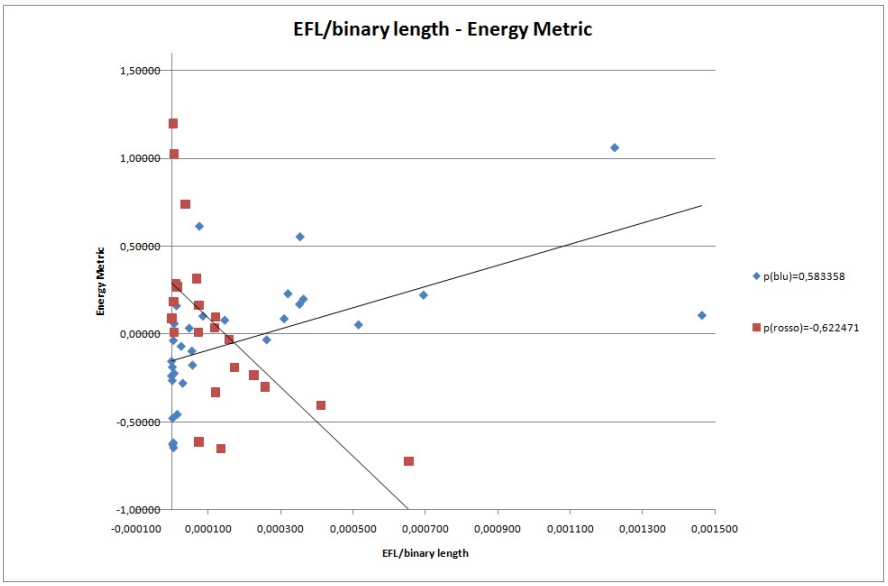


Figura 6.8: Energy Metric rispetto alla metrica EFL (normalizzata sul Binary Lenght), clusterizzato

6.6.2 Metrica EFC e EFLC

Per le metriche EFC e EFLC abbiamo seguito ugualmente lo stesso procedimento svolto per la metrica EFL, cioè:

1. Mostrato il grafico che presenta su un asse il valore dell'Energy Metric incrociato con il valore della nostra metrica
2. Normalizzato il valore della nostra metrica con il Binary Length
3. Diviso le applicazioni in cluster

- Per EFC Fig.6.9, 6.10 e 6.11:

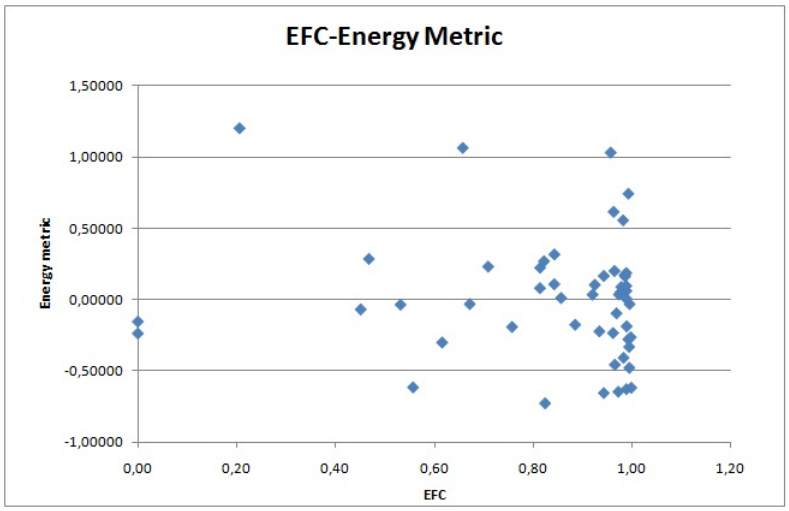


Figura 6.9: Energy Metric rispetto alla metrica EFC

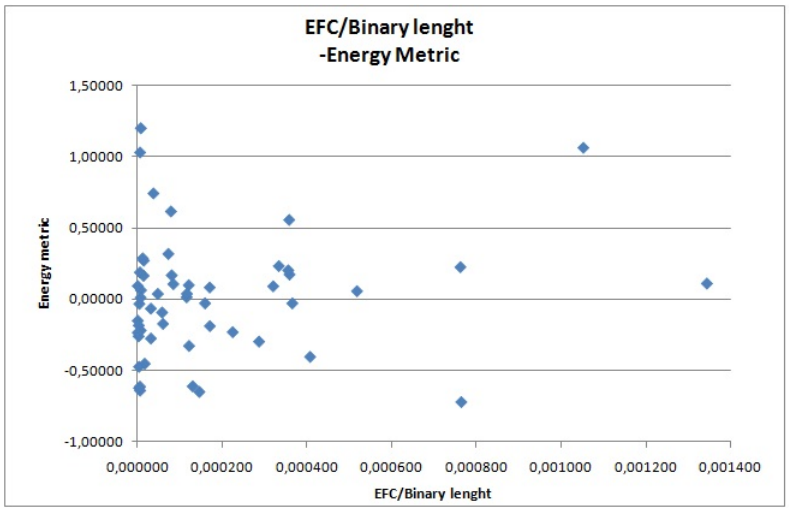


Figura 6.10: Energy Metric rispetto alla metrica EFC (normalizzata su Binary Lenght)

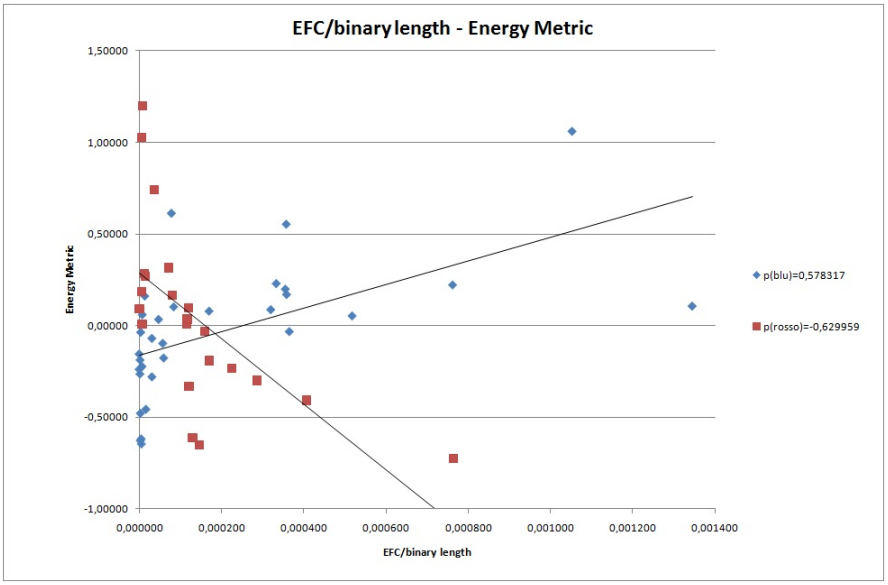


Figura 6.11: Energy Metric rispetto alla metrica EFC (normalizzata su Binary Lenght), clusterizzato

- Per EFLC Fig.6.12, 6.13 e 6.14:

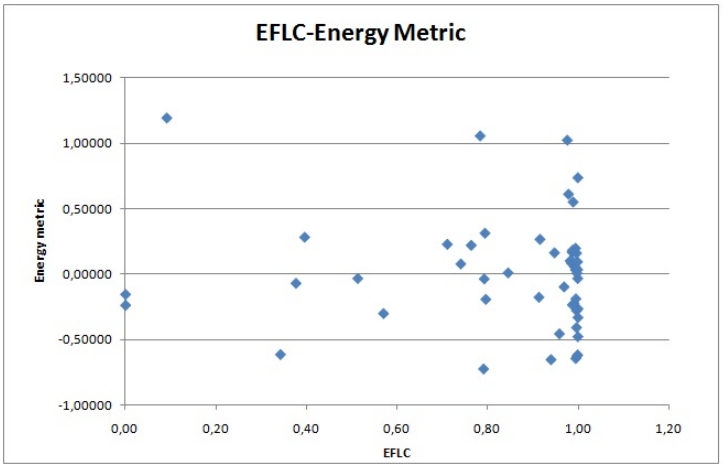


Figura 6.12: Energy Metric rispetto alla metrica EFLC

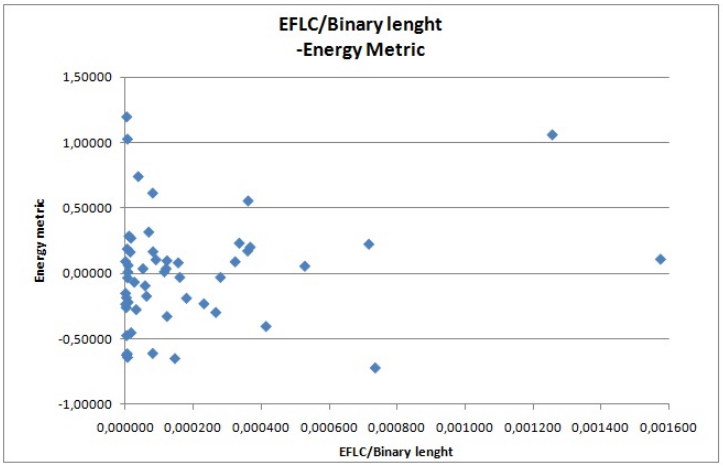


Figura 6.13: Energy Metric rispetto alla metrica EFLC (normalizzata su Binary Lenght)

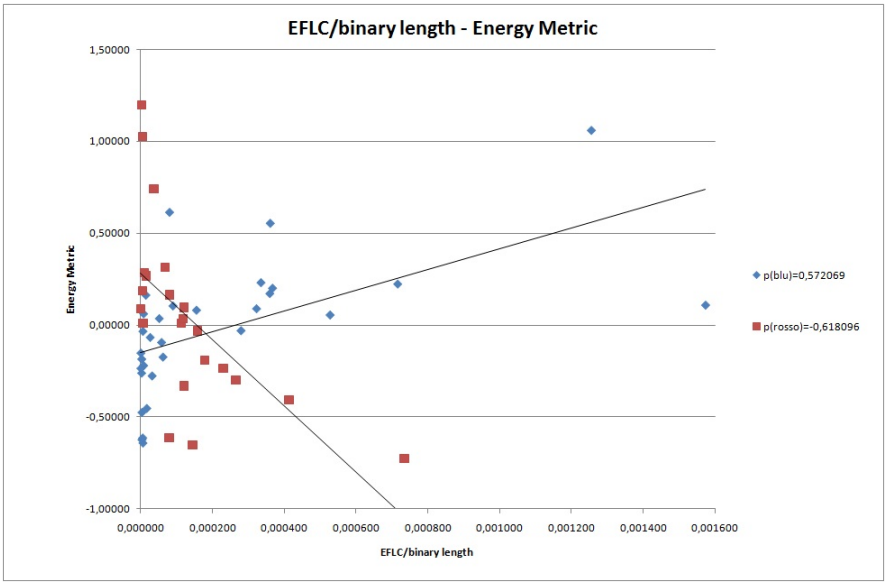


Figura 6.14: Energy Metric rispetto alla metrica EFLC (normalizzata su Binary Lenght), clusterizzato

Dai grafici clusterizzati delle tre metriche si può individuare il medesimo comportamento. Si può notare che il primo cluster (di colore blu), cioè quello con le applicazioni più complesse, mostra consumi crescenti al crescere del valore delle metriche con una correlazione di circa 0.58 per la metrica EFL e EFC e di circa 0.57 per la metrica EFLC; il secondo cluster (di colore rosso), cioè quello con le applicazioni meno complesse, presenta un trend inverso, cioè mostra consumi decrescenti al crescere del valore delle metriche con una correlazione di circa -0.62 per la metrica EFL e EFLC, mentre per la metrica EFC di circa -0.63.

6.7 Correlazione tra Entropia e le nuove metriche

In questa sezione andremo ad osservare che esiste un legame tra le nuove metriche e la metrica dell'Entropia dell'espressività.

Ricordiamo che la metrica dell'Entropia è una metrica che indica quanto un determinato frammento di codice sfrutti la potenzialità espressiva del linguaggio di programmazione che utilizza.

In accordo con il lavoro di tesi di Galli [27], la Fig.6.15 mostra il consumo di energia in relazione con l'Entropia dell'espressività, in particolare il grafico è una rappresentazione logaritmica del valore della $\frac{Entropia\ dell'\ espressivita}{Binary\ Length}$ e del valore energetico normalizzato secondo la categoria dell'applicazione. Come possiamo vedere in Fig.6.15 i dati sono divisi in due cluster: nel primo cluster l'Entropia dell'espressività è proporzionale al consumo energetico dell'applicazione, mentre nel secondo cluster esso è inversamente proporzionale. I cluster sono stati formati in base alla descrizione fatta nel paragrafo 6.8.

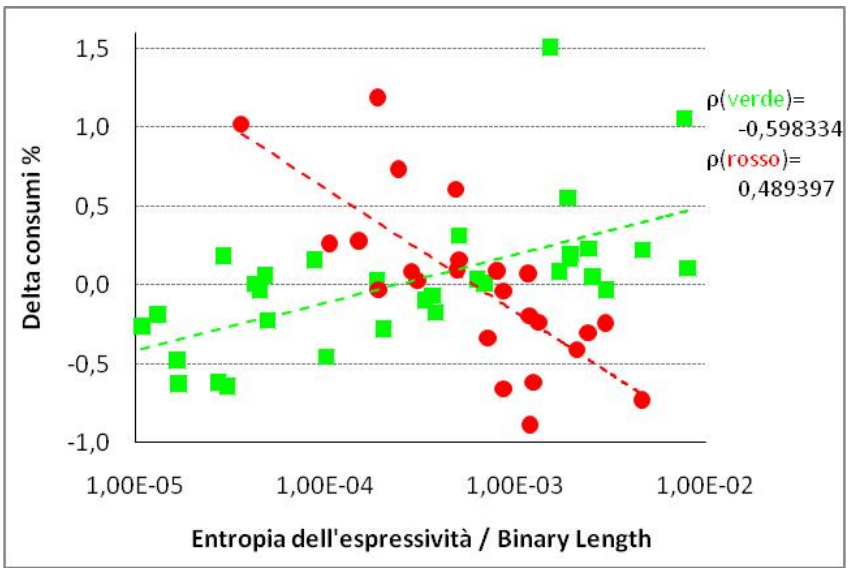


Figura 6.15: Delta dei consumi medi rispetto all'Entropia dell'espressività (normalizzata sul Binary Length)

Si può interpretare la crescita di Entropia come l'utilizzo crescente di librerie esterne, le cui funzionalità vanno ad affiancare i costrutti base del linguaggio, aumentando la cardinalità dell'alfabeto a disposizione dello sviluppatore.

In accordo con i risultati della nostra ricerca, per il primo cluster (applicazioni più complesse), un basso consumo di energia è associato ad un basso uso di librerie, mentre per il secondo cluster (applicazioni meno complesse) un consumo energetico basso è associato ad un alto uso delle librerie.

Anche dall'analisi dei dati, effettuata sul foglio di calcolo elettronico, si nota che c'è una elevata correlazione tra le nostre metriche e la

metrica dell'Entropia dell'espressività presentata nel lavoro di Galli [27], infatti come si può notare anche dalla Fig.6.16 nel caso del primo cluster avremo una correlazione tra la metrica dell'Entropia/BL e:

- la metrica EFL/BL di circa 0.986
- la metrica EFC/BL di circa 0.988
- la metrica EFLC/BL di circa 0.984

nel caso del secondo cluster avremo una correlazione tra la metrica dell'Entropia/BL e:

- la metrica EFL/BL di circa 0.963
- la metrica EFC/BL di circa 0.981
- la metrica EFLC/BL di circa 0.969

	Entropia/BL	
	Cluster1	Cluster2
EFL/BL	0,985565601	0,963148462
EFC/BL	0,988560768	0,981871465
EFLC/BL	0,98431787	0,969254488

Figura 6.16: Correlazione tra Entropia/BL e Metriche EFL, EFC, EFLC

In conclusione le analisi illustrano che esistono due modi differenti di programmazione: un modo che fa uso di librerie esterne contrariamente ad un altro che non ne richiede l'utilizzo.

Le nostre metriche, quindi, supportate dalla metrica dell'Entropia dell'espressività, possono essere usate come uno strumento a supporto dello sviluppo del software.

6.8 Analisi dei cluster

In questo paragrafo si illustrerà il mondo in cui sono stati divisi i due cluster. Principalmente i programmi sono stati assegnati ad uno specifico cluster in base ad alcune caratteristiche come:

- la loro grandezza
- la loro complessità strutturale

Per far cioè è possibile far leva sulle metriche classiche di qualità del design.

6.9 Approccio statistico

Le analisi statistiche sono state effettuate utilizzando il programma statistico Gretl [37] sul campione delle nostre applicazioni.

Al fine di verificare le nostre ipotesi, H1 e H2, abbiamo usato il metodo dei minimi quadrati ordinati[2] per stimare il nostro modello di

[2]Il metodo dei minimi quadrati è una tecnica di ottimizzazione che permette di trovare una funzione che si avvicini il più possibile ad un'interpolazione di un insieme

regressione lineare [29]. La principale differenza tra la correlazione, usata in precedenza, e un modello di regressione è la seguente: la prima analizza se esiste una relazione tra due variabili, cioè come e quanto le due variabili vanno insieme, mentre la regressione analizza la forma della relazione tra le variabili, cioè determina la relazione causa-effetto.

La regressione è importante perché ci permette di costruire un modello funzionale della risposta di una variabile (effetto) ad un'altra (causa). In sostanza conoscendo la forma della relazione funzionale tra la variabile indipendente e dipendente è possibile stimare il valore di quella dipendente conoscendo il valore dell'altra indipendente.

Abbiamo testato le nostre ipotesi utilizzando il seguente modello di regressione multipla:

$$EM = \beta_1 * Cluster + \beta_2 * Metr + \beta_3 * Metr * Cluster + \beta_4 + \epsilon \quad (6.1)$$

dove:

- EM sta per Energy Metric
- Cluster vale 0 se l'applicazione appartiene al primo cluster o 1 se appartiene al secondo cluster
- Metr sta per la metrica EFC/BL o EFL/BL o EFLC/BL

In particolare le ipotesi di regressione sono supportate dai risultati se:

di dati. In particolare la funzione trovata deve essere quella che minimizza la somma dei quadrati delle distanze dei punti dati.

- i segni dei coefficienti della regressione sono coerenti con la direzione delle relazioni ipotizzate
- i coefficienti della regressione sono statisticamente significativi se il p-value dei valori è minore del 5%

I pesi dei termini della regressione che sono moltiplicati per il cluster indicano come i rapporti cambiano quando è considerato il cluster delle applicazioni meno complesse.

6.9.1 Risultati dell'analisi statistica

La Fig.6.17 mostra i coefficienti del modello di regressione rappresentato dall'equazione 6.1 nel caso in cui il parametro Metr è uguale a EFL/BL e riporta le variabili del modello di regressione:

Variabile dipendente: EnergyMetric

	coefficiente	errore std.	rapporto t	p-value	
const	-0,152425	0,0748242	-2,037	0,0473	**
Cluster	0,457419	0,125032	3,658	0,0006	***
EFL_Bl_cluster	-2626,63	529,247	-4,963	9,53e-06	***
EFL_binarylengt	603,851	181,767	3,322	0,0017	***

Media var. dipendente	0,003496	SQM var. dipendente	0,427777
Somma quadr. residui	5,733959	E.S. della regressione	0,349284
R-quadro	0,373315	R-quadro corretto	0,333314
F(3, 47)	9,332615	P-value(F)	0,000060
Log-verosimiglianza	-16,63767	Criterio di Akaike	41,27534
Criterio di Schwarz	49,00264	Hannan-Quinn	44,22817

Note: SQM = scarto quadratico medio; E.S. = errore standard

Figura 6.17: Coefficienti del modello di regressione con variabile indipendente EFL/BL

Il valore R-quadro (0.38) è buono, e indica che l'adattamento del modello ai dati è abbastanza corretto. Il coefficiente β_3, che esprime

l'effetto dell'interazione tra la metrica EFL/BL e la complessità dell'applicazione sull'Energy Metric, è altamente significativo, il che suggerisce che, come ipotizzato da noi, vi è un'interazione tra la complessità delle applicazioni e la struttura del codice.

Lo stesso discorso vale anche nel caso in cui il parametro Metr è uguale a EFC/BL e EFLC/BL. Di seguito, nelle Fig.6.17 e Fig.6.19, sono riportati le variabili del modello di regressione:

Variabile dipendente: EnergyMetric

	coefficiente	errore std.	rapporto t	p-value	
const	-0,159712	0,0760425	-2,100	0,0411	**
Cluster	0,461488	0,124834	3,697	0,0006	***
EFC_Bl_cluster	-2475,83	485,173	-5,103	5,93e-06	***
EFC_binarylengt	644,149	195,218	3,300	0,0019	***

Media var. dipendente	0,003496	SQM var. dipendente	0,427777
Somma quadr. residui	5,711270	E.S. della regressione	0,348592
R-quadro	0,375795	R-quadro corretto	0,335952
F(3, 47)	9,431927	P-value(F)	0,000055
Log-verosimiglianza	-16,53657	Criterio di Akaike	41,07314
Criterio di Schwarz	48,80044	Hannan-Quinn	44,02597

Note: SQM = scarto quadratico medio; E.S. = errore standard

Figura 6.18: Coefficienti del modello di regressione con variabile indipendente EFC/BL

Variabile dipendente: EnergyMetric

	coefficiente	errore std.	rapporto t	p-value	
const	-0,148504	0,0751476	-1,976	0,0540	*
Cluster	0,445010	0,124985	3,561	0,0009	***
EFLC_binaryleng	563,214	174,180	3,234	0,0022	***
EFLC_Bl_cluster	-2409,53	492,444	-4,893	1,21e-05	***

Media var. dipendente	0,003496	SQM var. dipendente	0,427777
Somma quadr. residui	5,817770	E.S. della regressione	0,351827
R-quadro	0,364155	R-quadro corretto	0,323570
F(3, 47)	8,972475	P-value(F)	0,000083
Log-verosimiglianza	-17,00770	Criterio di Akaike	42,01539
Criterio di Schwarz	49,74269	Hannan-Quinn	44,96822

Note: SQM = scarto quadratico medio; E.S. = errore standard

Figura 6.19: Coefficienti del modello di regressione con variabile indipendente EFLC/BL

Il valore R-quadro per il modello di regressione con parametro Metr uguale a EFC/BL è pari a 0.38 e con parametro Metr uguale a EFLC/BL è pari a 0.37 e come detto in precedenza è buono, e indica che l'adattamento del modello ai dati è abbastanza corretto.

Mediante il programma statistico Gretl sono stati tracciati graficamente i valori effettivi e stimati dell'Energy Metric rispetto alle nuove metriche. Il grafico di Fig.6.20 mostra graficamente i valori effettivi e stimati dell'Energy Metric rispetto alla metrica EFL/BL, il grafico di Fig.6.21 mostra graficamente i valori effettivi e stimati dell'Energy Metric rispetto alla metrica EFC/BL e il grafico di Fig.6.22 mostra graficamente i valori effetti e stimati dell'Energy Metric rispetto alla metrica EFLC/BL.

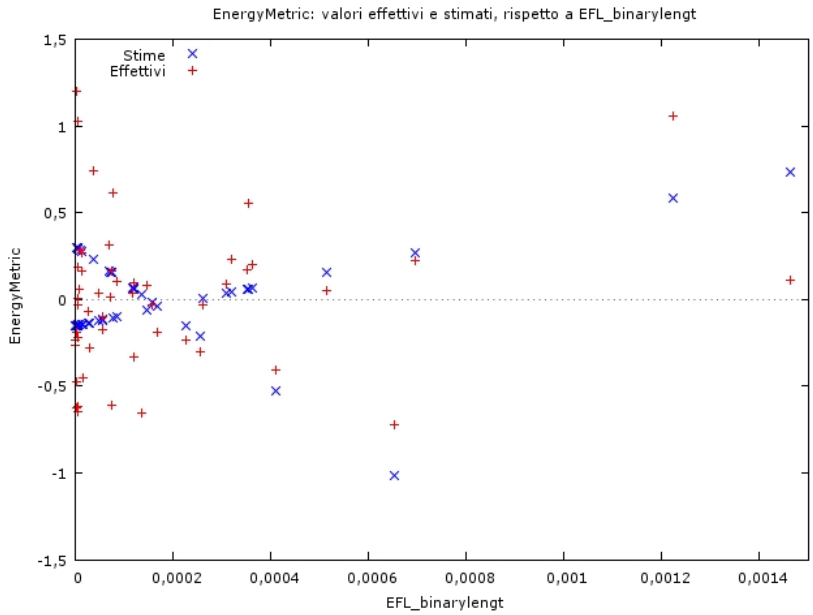


Figura 6.20: Energy Metric: valori effettivi e stimati, rispetto a EFL/BL

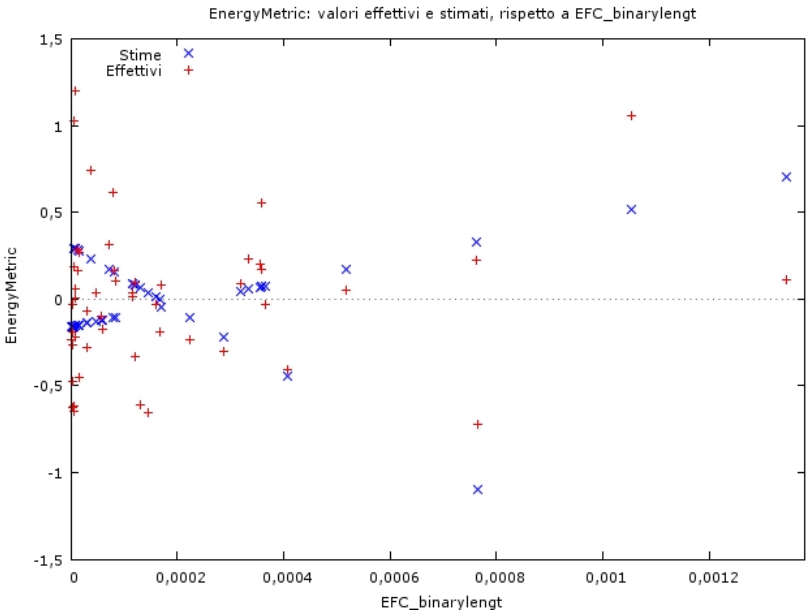


Figura 6.21: Energy Metric: valori effettivi e stimati, rispetto a EFC/BL

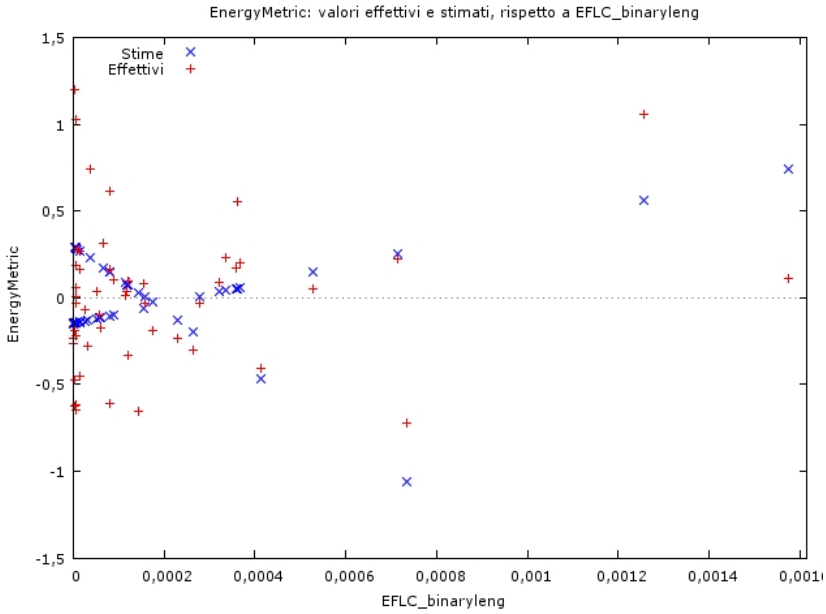


Figura 6.22: Energy Metric: valori effettivi e stimati, rispetto a EFLC/BL

Questi grafici sono in linea con i grafici di Fig.6.8,Fig.6.11 e Fig.6.14.

6.10 Discussione dei risultati

Si andranno ora a discutere i risultati ottenuti in precedenza.

L'obiettivo della nostra ricerca è stato quello di esaminare l'impatto della struttura del codice di un'applicazione sul consumo energetico, partendo dall'idea che il software gioca un ruolo critico sul consumo energetico complessivo di un'infrastruttura IT.

Oggigiorno non esistono metriche che possano dire in che modo l'organizzazione strutturale del codice possa impattare sull'efficienza energetica. Quindi, il nostro lavoro propone una metodologia per misurare empiricamente se il consumo energetico è legato al modo in cui il software è stato implementato. Abbiamo introdotto nuove metriche, adoperabili con l'aiuto di un apposito tool, che potrebbero essere un importante indice di valutazione per lo sviluppatore per poter seguire una certa linea guida nell'implemetazione software.

Come precedentemente discusso, l'uso di librerie, in generale, può semplificare il lavoro agli sviluppatori così da salvaguardarli dal comprendere le complesse strutture implementative rendendo più efficiente il codice, ma questo non necessariamente porta ad una maggior efficienza energetica. Abbiamo implementato delle metriche, chiamate EFL, EFC e EFLC, che misurano quanto pesa l'uso delle librerie esterne, in termini di linee di codice e di chiamate ai metodi, sull'intera struttura dell'applicazione.

I risultati mostrano che le applicazioni appartenenti al cluster contenete le applicazioni meno complesse, ovvero quelle che mostrano consumi decrescenti al crescere dei valori delle nuove metriche, traggono benefici dall'utilizzo crescente di librerie esterne; in questo caso è con-

sigliabile l'utilizzo di costrutti esterni (librerie) in fase di implementazione del software.

Invece, le applicazioni appartenenti al cluster contenente le applicazioni più complesse, cioè quelle che mostrano consumi crescenti al crescere dei valori delle nostre metriche, presentano un comportamento inverso rispetto all'altro cluster. Per questo tipo di applicazioni un uso intenso di costrutti esterni (librerie) non porta benefici in termini energetici: si deduce che in simili situazioni è tendenzialmente consigliabile riscrivere le funzionalità dell'applicazione piuttosto che appoggiarsi a quelle fornite dalle librerie esterne.

Quindi in accordo con i risultati ottenuti, l'ipotesi H1 è verificata in quanto la struttura del codice di un'applicazione influisce sul consumo energetico, e anche l'ipotesi H2 è verificata in quanto un'applicazione strutturalmente meno complessa risulta energeticamente più efficiente quando si usano le librerie esterne. Si ha quindi che la struttura del codice di un'applicazione ha un diverso effetto sull'efficienza energetica.

Una possibile spiegazione di questi risultati, supponendo che si abbiano programmatori con la stessa esperienza, è che in applicazioni più complesse strutturalmente porzioni di codice e molte funzionalità delle librerie esterne sono causa di lavoro supplementare per il processore quando l'applicazione viene eseguita, poichè l'uso di librerie aumenta il numero di strati che devono essere attraversati per eseguire una funzionalità. Mentre per le piccole applicazioni, una possibile spiegazione dei risutlati è che un programmatore, che sviluppa nuove funzionalità

da zero, difficilmente è in grado di implementare codice ottimizzato.

Per esempio, per confermare i nostri risultati per quanto riguarda le applicazioni complesse, se consideriamo i due ERP, OpenBravo, che usa HIBERNATE (un Object Relational Mapping), e Adempire, che usa direttamente istruzioni SQL per accedere al DataBase, notiamo che è più semplice sviluppare, modificare e personalizzare una classe che utilizza HIBERNATE piuttosto che SQL, in quanto non richiede capacità avanzate di programmazione per interagire con il DataBase. Tuttavia l'esecuzione di HIBERNATE rappresenta un overhead per il processore, quindi richiede molta più potenza rispetto all'esecuzione di istruzioni SQL. In altre parole si può dire che HIBERNATE ha un impatto molto limitato sul tempo di esecuzione rispetto alle istruzioni SQL, ma ha un enorme impatto sul consumo energetico.

I nostri risultati vengono supportati anche dai risultati trovati da Galli[27]. Infatti le nuove metriche hanno una correlazione molto alta, come si può vedere nel paragrafo 6.7, con la metrica dell'entropia dell'espressività, della quale ricordiamo che crescente espressività corrisponde ad un maggior uso di librerie.

In conclusione, è necessario individuare delle caratteristiche per poter collocare un'applicazione in un cluster o nell'altro prima che esso venga implementato, o per lo meno fornire delle linee guida per avvicinarsi ad uno dei due cluster.

Capitolo 7

Conclusioni e Sviluppi Futuri

In questo capitolo sono presentate le conclusioni del lavoro svolto e i possibili sviluppi futuri.

Nella prima sezione del capitolo illustreremo i risultati raggiunti, mentre nella sezione finale verranno proposti alcuni possibili sviluppi futuri per proseguire il nostro lavoro.

7.1 Risultati raggiunti

Il lavoro di tesi si pone all'interno di un più ampio progetto che mira ad esplorare il concetto di green software, studiare le tematiche relative allo sviluppo di linee guida per una programmazione green ed identificare i fattori che hanno un impatto sul consumo energetico.

In particolare è stato studiato l'influenza delle diverse modalità di implementazione del software sui consumi energetici, effettuando un'analisi sul codice statico delle applicazioni.

Il tutto ci ha permesso di capire che relazione c'è tra l'uso di librerie esterne e il consumo energetico nelle applicazioni.

Il nostro studio ci ha portato ad individuare due differenti risultati:

- le applicazioni meno complesse mostrano consumi decrescenti al crescere dell'utilizzo di librerie esterne
- le applicazioni più complesse presentano un comportamento inverso rispetto a quelle meno articolate

Nonostante i risultati siano soddisfacenti, e l'esito della nostra ricerca ha un forte legame con il lavoro svolto da Galli[27], non è ancora possibile dire di aver trovato una relazione certa, per due motivi:

- il campione di applicazioni non è ancora sufficientemente numeroso
- il campione di applicazioni utilizzato considera solo software open-source, scritti nello stesso linguaggio di programmazione

Se definitivamente provati, i risultati ottenuti, potrebbero avere un notevole impatto sul modo di produrre software dalle aziende. Si potrebbero creare nuovi indirizzi di business, per esempio: classificare in ottica green le applicazioni, confrontare le prestazioni in termini energetici di diversi team di sviluppatori, creazione di una nuova leva commerciale per formulare offerte green.

7.2 Sviluppi futuri

Il lavoro non può ancora essere considerato completo e potrebbe essere integrato da una serie di task:

- Estensione del campione di applicazioni da analizzare in modo da validare i risultati da noi ottenuti
- Applicazione delle nostre metriche EFC, EFL e EFLC, modificando opportunamente il parsificatore, a software sviluppati con linguaggi di programmazione differenti
- Analisi di applicazioni non open-source, così da includere molti dei software usati nel mondo aziendale, e considerare un mondo in cui le tecniche di sviluppo e le caratteristiche dei prodotti sono ben diverse
- Determinare se esiste un legame, ad esempio, tra l'età, i bugs del software e il consumo energetico
- Valutazione dell'esperienza del programmatore ai fini di comprendere se anche questa ha un impatto sul consumo energetico del software. In pratica se uno sviluppatore "ninja", quindi più esperto ed in grado di usare al meglio le sue abilità, possa implementare applicazioni più efficienti energeticamente rispetto ad uno sviluppatore "operaio", magari considerando anche la complessità del software implementato

Inoltre, sarebbe anche interessante poter analizzare l'impatto energetico del software direttamente dall'ambiente di sviluppo.

Appendice A

Applicazioni analizzate e valori delle metriche

In questa sezione è mostrata una lista con le applicazioni analizzate. La lista si compone del nome dell'applicazione, dell'indirizzo web mediante il quale è possibile fare il download dell'applicazione, dell' ID e del Group_id dell'applicazione (se presente sul sistema per la gestione dello sviluppo di software SourceForge).

In seguito è mostrata una serie di tabelle che presentano i valori delle metriche (Energy Metric, Entropy/BL, nuove metriche, ecc.) per ogni applicazione che è stata analizzata.

Applicazioni	Indirizzo Web	ID su SouceForge	Group_id su SourceForge
Anomic	http://www.anomic.de/AnomicFTPServer/Download.html		
Billy	http://sourceforge.net/projects/billy/	717955	130479
bsheet - Bean Sheet	http://sourceforge.net/projects/bsheet/	701558	125101
Cleansheet	http://sourceforge.net/projects/csheets/	740910	138542
ColoradoFTP	http://cftp.coldcore.com/		
Danoftp	http://sourceforge.net/projects/danoftp/	442939	45441
Dracman	http://sourceforge.net/projects/dracman/		103542
Hermes	http://sourceforge.net/projects/hermesftp/	833334	164846
Jac	http://sourceforge.net/projects/jac-calc/	874762	175819
James	http://james.apache.org/		
Jarky	http://sourceforge.net/projects/jarky/	994269	205594
JavaSolitaire	http://sourceforge.net/projects/javasol/	471261	53693
JavaTextEditor	http://sourceforge.net/projects/javatexteditor/	791679	154412
Jbackgammon	http://sourceforge.net/projects/jbackgammon/	563086	81479
jBrowser	http://sourceforge.net/projects/jbrowser/	624815	99639
jCalcAlfa - Standard & Scientific Calculator	http://sourceforge.net/projects/jcalculator/	435186	43112
Jeppers	http://sourceforge.net/projects/jeppers/	445386	46203
Jes	http://sourceforge.net/projects/jes/		263288
Jexplosion	http://sourceforge.net/projects/jexplosion/	892968	180384
Jgammon	http://sourceforge.net/projects/jgam/	658048	111028
Jmjrst -JavaMassJPEG ResizerTool	http://sourceforge.net/projects/jmjrst/	942609	192740
Jmorpheus	http://sourceforge.net/projects/jmorpheus/	613790	96144
jPDF tweak	http://sourceforge.net/projects/jpdftweak/	944877	193304
Jreversy	http://sourceforge.net/projects/jreversi/	726766	133288
Jrezz	http://sourceforge.net/projects/jrezz/		246822
Jscale	http://sourceforge.net/projects/j-scale/	1108782	239177
JSFTP	http://www.softtechdesign.com/products/components/jsftp.htm		
Jsnake	http://sourceforge.net/projects/jsnake/	716421	129980
Jupiter	http://sourceforge.net/projects/jupiter-ftp/	499619	62159
JvFTP	http://sourceforge.net/projects/jvftp/	590386	89516
JXWB - Java Extensible Web Browser	http://sourceforge.net/projects/jxwb/	602927	93048

Figura A.1: Elenco delle applicazioni analizzate

Applicazioni	Indirizzo Web	ID su SouceForge	Group_id su SourceForge
Kalender	http://sourceforge.net/projects/jkalender/	964472	198187
Kurvetest (kurveserver)	http://sourceforge.net/projects/kurve-online/	718264	130572
MyTaskScheduler	http://sourceforge.net/projects/mytaskscheduler/	991051	204795
Packman(pacmancontroller)	http://packman.links2linux.org/		
Panda	http://sourceforge.net/projects/hgpanda/	879178	176922
Pcalendar - Periodic Calendar	http://sourceforge.net/projects/linuxorg/		47415
Pdfsam	http://sourceforge.net/projects/pdfsam/	814265	160044
Pelletquest	http://sourceforge.net/projects/pelletquest/	1046741	219489
Phoenix	http://sourceforge.net/projects/phoenixpacman/	393833	28605
Pom	http://sourceforge.net/projects/nrmpom/		104483
Reminder	http://sourceforge.net/projects/jreminder/		
RoxMine	http://sourceforge.net/projects/roxminesweeper/	997852	206502
Rtext	http://sourceforge.net/projects/rtext/	610804	95266
Saper	http://sourceforge.net/projects/isaper/	990526	204666
ScientificCalculator	http://sourceforge.net/projects/jscicalc/	797960	155983
SharpTools	http://sourceforge.net/projects/sharptools/	392975	27033
SimpleJ	http://sourceforge.net/projects/simplej/	1126579	251727
Snaax - Classic Snake Game	http://sourceforge.net/projects/snaax/	516681	67044
Snake VS Snake	http://sourceforge.net/projects/snakevssnake/	904388	183226
SolCalc	http://sourceforge.net/projects/solcalc/	950676	194749
TextTrix	http://sourceforge.net/projects/texttrix/		48910
Virgoftp	http://sourceforge.net/projects/qftp/	750315	141731
xiftp	http://sourceforge.net/projects/xiftp/	712651	128752
Xsheet	http://sourceforge.net/projects/xsheet/	1089491	233197
YaFTP	http://sourceforge.net/projects/yaftp/	503345	63252

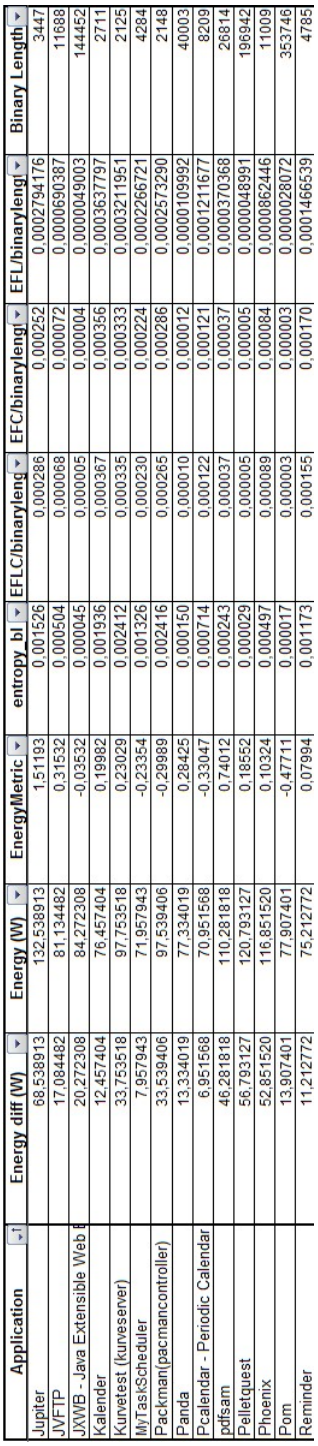
Figura A.2: Elenco delle applicazioni analizzate

Application	Energy diff (W)	Energy (W)	EnergyMetric	entropy_bl	EFLC/binarylengt	EFC/binarylengt	EFL/binarylengt	Binary Length
Anomic	9,470702	73,470702	-0,65290	0,000863	0,000145	0,000145	0,0001358234	6479
Billy	1,679842	65,679842	-0,72464	0,004623	0,000735	0,000765	0,0006543504	1077
bsheet	9,113768	73,113768	-0,45523	0,000101	0,000016	0,000016	0,0000157801	59402
ciac - calcolatrice	7,875163	71,875163	0,08805	0,001699	0,000322	0,000320	0,0003106432	3055
Cleansheet	16,878713	80,878713	0,00891	0,000043	0,000007	0,000007	0,0000067001	147078
ColoradoFTP	10,466369	74,466369	-0,61641	0,000028	0,000005	0,000005	0,0000049274	200386
danoftp	9,711398	73,711398	-0,64408	0,000030	0,000006	0,000005	0,0000054277	178344
Dracman	48,438332	112,438332	0,01112	0,000685	0,000114	0,000115	0,0000734484	7417
Hermes	10,208137	74,208137	-0,62587	0,000017	0,000003	0,000003	0,0000028039	348653
jac	7,673759	71,673759	0,06023	0,000048	0,000007	0,000007	0,0000073201	133264
James	43,054822	107,054822	0,09572	0,000802	0,000122	0,000120	0,0001199496	8207
Jarky	46,786474	110,786474	-0,03136	0,000869	0,000160	0,000159	0,0001584497	6263
JavaTextEditor	46,874742	110,874742	-0,15372	0,023751	0,000000	0,000000	0,0000000000	209

Figura A.3: Valori Metriche

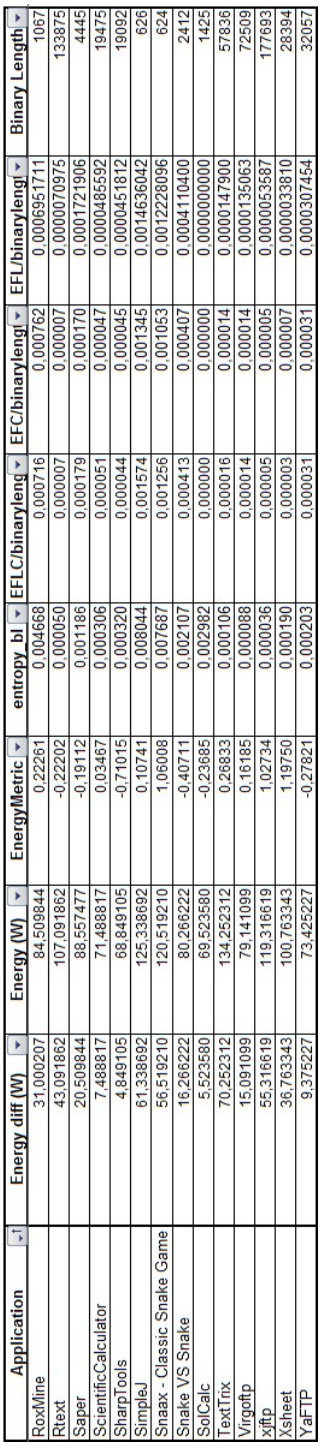
Application	Energy diff (W)	Energy (W)	EnergyMetric	entropy_bl	EFLC/binarylengt	EFC/binarylengt	EFL/binarylengt	Binary Length
Jbackgammon	2,720185	66,720185	-0,61344	0,001246	0,000080	0,000130	0,0000742570	4292
jBrowser	21,756563	85,756563	0,03532	0,000629	0,000119	0,000116	0,0001173159	8382
jCalcAlfa - Standard & Scientif	7,627911	71,627911	0,05389	0,002533	0,000527	0,000518	0,0005157933	1885
Jes	35,532123	99,532123	-0,09572	0,000335	0,000057	0,000057	0,0000558334	16860
Jexplosion	24,557477	95,000207	-0,03149	0,002999	0,000279	0,000365	0,0002623193	1838
Jgammon	11,353593	75,353593	0,61344	0,000487	0,000080	0,000078	0,0000766663	12281
Jmjrst - JavaMassJPEG ResizerTool	57,581822	121,581822	0,16512	0,000509	0,000081	0,000080	0,0000746399	11746
Jmorpheus	7,141971	71,141971	0,17072	0,001927	0,000359	0,000359	0,0003529363	2746
jpdf tweak	19,601632	83,601632	-0,26301	0,000011	0,000002	0,000002	0,0000017874	556697
Jreversy	9,479709	73,479709	0,55392	0,001878	0,000361	0,000358	0,0003545157	2743
Jrezz	46,042519	110,042519	-0,06837	0,000364	0,000026	0,000031	0,0000263062	14616
Jscale	40,190525	104,190525	-0,18678	0,000013	0,000002	0,000002	0,0000022042	443628
JSFTP	10,709330	74,759330	-0,17550	0,000380	0,000061	0,000060	0,0000577808	14861
Jsnake	3,202721	67,202721	-0,88326	0,001193	0,000199	0,000172	0,0001814308	4571

Figura A.4: Valori Metriche



Application	Energy diff (W)	Energy (W)	EnergyMetric	entropy_bl	EFLC/binaryleng	EFC/binaryleng	EFL/binaryleng	Binary Length
Jupiter	68,538913	132,538913	1,51193	0,001526	0,000286	0,000252	0,002794176	3447
JVFTP	17,084482	81,134482	0,31532	0,000504	0,000068	0,000072	0,0000690387	11688
JXWB - Java Extensible Web B	20,272308	84,272308	-0,03532	0,000045	0,000005	0,000004	0,0000049003	144452
Kalender	12,457404	76,457404	0,19982	0,001936	0,000367	0,000356	0,0003637797	2711
Kurvetest (kurveserver)	33,753518	97,753518	0,23029	0,002412	0,000335	0,000333	0,0003211951	2125
MyTaskScheduler	7,957943	71,957943	-0,23354	0,001326	0,000230	0,000224	0,0002266721	4284
Packman(pacmancontroller)	33,539406	97,539406	-0,29989	0,002416	0,000265	0,000286	0,0002573290	2148
Panda	13,334019	77,334019	0,28425	0,000150	0,000010	0,000012	0,0000109992	40003
Pcalendar - Periodic Calendar	6,951568	70,951568	-0,33047	0,000714	0,000122	0,000121	0,0001211677	8209
pdfsam	46,281818	110,281818	0,74012	0,000243	0,000037	0,000037	0,0000370368	26814
Pelletguest	56,793127	120,793127	0,18552	0,000029	0,000005	0,000005	0,0000048991	196942
Phoenix	52,851520	116,851520	0,10324	0,000497	0,000089	0,000084	0,0000862446	11009
Pom	13,907401	77,907401	-0,47711	0,000017	0,000003	0,000003	0,0000028072	353746
Reminder	11,212772	75,212772	0,07994	0,001173	0,000155	0,000170	0,0001466539	4785

Figura A.5: Valori Metriche



Application	Energy diff (W)	Energy (W)	EnergyMetric	entropy_bl	EFLC/binaryleng	EFC/binaryleng	EFL/binaryleng	Binary Length
RoxMine	31,000207	84,509844	0,22261	0,004668	0,000716	0,000762	0,0006951711	1067
Rtext	43,091862	107,091862	-0,22202	0,000050	0,000007	0,000007	0,0000070975	133875
Saper	20,509844	88,557477	-0,19112	0,001186	0,000179	0,000170	0,0001721906	4445
ScientificCalculator	7,488817	71,488817	0,03467	0,000306	0,000051	0,000047	0,0000485592	19475
SharpTools	4,849105	68,849105	-0,71015	0,000320	0,000044	0,000045	0,0000451812	19092
SimpleJ	61,338692	125,338692	0,10741	0,008044	0,001574	0,001345	0,0014636042	626
Snaax - Classic Snake Game	56,519210	120,519210	1,06008	0,007687	0,001256	0,001053	0,0012228096	624
Snake VS Snake	16,266222	80,266222	-0,40711	0,002107	0,000413	0,000407	0,0004110400	2412
SolCalc	5,523580	69,523580	-0,23685	0,002982	0,000000	0,000000	0,0000000000	1425
TextTrix	70,252312	134,252312	0,26833	0,000106	0,000016	0,000014	0,0000147900	57836
Virgoftp	15,091099	79,141099	0,16185	0,000088	0,000014	0,000014	0,0000135063	72509
xjftp	55,316619	119,316619	1,02734	0,000036	0,000005	0,000005	0,0000053587	177693
Xsheet	36,763343	100,763343	1,19750	0,000190	0,000003	0,000007	0,0000033810	28394
YaFTP	9,375227	73,425227	-0,27821	0,000203	0,000031	0,000031	0,0000307454	32057

Figura A.6: Valori Metriche

Bibliografia

[1] Murugesan S.: Harnessing Green IT: Principles and Pratices. IT Professional, Vol.10, n.1, 2008

[2] Bryan Gardiner, How important will new energy star be for pc makers?, PC Magazine, 2007

[3] C. Francalanci e E. Capra, "Green IT, sfide e opportunità", Mondo Digitale, `http://www.mondodigitale.net/Rivista/08_numero_4/Francalancip.36-42_.pdf`, 2007.

[4] E. Capra, "Green IT: Gestione dei dati e impatto ambientale sono correlati in modo molto stretto, basta volerne prendere atto".

[5] Directive 2002/95/ec of the european parliament and of the council. Official Journal of the European Union, January 2003.

[6] Directive 2002/96/ec of the european parliament and of the council on waste electrical and electronic equipment (weee). Official Journal of the European Union, Janaury 2003.

[7] Brown E.G. Lee C.: Topic Overview: GreenIT. Forrester Research report, 2007

[8] Kurmar R.: Important Power, Cooling and Green IT Concerns. Gartner report, 2007

[9] C. Francalanci, E. Capra, and G. Agosta. Developing Energy-Efficient Software: Enersoft., 2009.

[10] Restorick T.: An Inefficient Truth. Global Action Plan Report, 2007 `www.globalactionplan.org.uk/research.aspx`

[11] S.R. Chidamber e C.F. Kemerer.A metrics suite for object oriented design, 20:pp. 476-493, 1994

[12] National Instruments. Ni usb-6210 16-bit, 250ksa/s daq multifunzione serie m, bus-powered. Technical report, National Instruments corporation, 2008.

[13] G.Breyta C.S. Yannoni M.H. Sherwood I.L. Chuang L.M.K. Vandersypen, M. Steffen. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 2001.

[14] C. Szyperski, J. Bosch, and W. Weck, Component oriented programming, Springer, 1999

[15] W. Kozaczynski, G. Booch, "Component-based software engineering", IEEE Software, Sept-Opt. 1998, pp. 34-36, 1998

[16] `www.tpc.org`

[17] `http://sourceforge.net/`

[18] US Environmental Protection Agency (EPA): Energy Conservation: Past and Present Projects: Green Computing Guide. University of Colorado at Boulder, USA, 2005

[19] N.Margolus and L.B. Levitin. The maximum speed of dynamical evolution. *Phisica*, D120:pp 188-195, 1998

[20] F. Renzi, Scenari evolutivi nei sistemi e nella tecnologia e loro impatti sui CED e sui loro consumi energetici, IBM, Presentazione per la conferenza 'Green ICT', 22 Novembre 2007, Milano.

[21] S. Lloyd. Programming the Universe. Knopf, 2006.

[22] Bogliolo A., Benini L., and Lattanzi E. and DeMicheli G., Specification and analysis of power-managed system. *Proceedings of the IEEE*

[23] Bob Steigerwald, Rajshree Chabukswar, Karthik Krishnan, and Jun De Vega. Creating energy efficient software. Technical report, Intel, 2007

[24] Woongki Baek and Trishul Chilimbi. Green: A system for supporting energy-conscious programming using principled approximation. Technical report, Microsoft Research, 2009

[25] S.Lloyd. Measures of complexity, a non-exhaustive list. Technical report, Department of Mechanical Engineering, Massachusetts Istitute of Technology, 2007

[26] Enel: il centro storico di Roma sarà adeguato alla eurotensione. più efficienza per la rete. Technical report, 29 Marzo 1999.

[27] Gabriele Galli, Un approccio alla valutazione dei consumi energetici del software basato sull'analisi statica del codice, Politenico di Milano, 2009

[28] University of Southern California, Java CodeCount^{TM}Counting Standard

[29] Eugenio Capra, Chiara Francalanci e Sandra A. Slaughter, Is software "green"? Application development environments and energy efficiency in opensource applications, 2010

[30] G. Formenti e S. Gallazzi, 'A methodology to evaluate empirically software energy consumption and its impact on Total Cost of Ownership.', Politecnico di Milano, 2009

[31] E.E. Millis, Software Metrics, Technical report, Carnegie Mellon University, Software Engineering Institute, 1988

[32] S.D. Conte, H.E. Dunsmore e V.Y. Shen, Software Engineering Metrics and Models, Benjamin Publishing Company, 1986

[33] ISO/IEC, 'Software engineering - Product quality - Part 3: Internal metrics', ISO/IEC TR9126-3, 2003.

[34] C. Ghezzi, M. Jazayeri e D. Mandrioli, 'Fundamentals of Software Engineering', International Edition, 2003.

[35] G. Colombo, 'Valutazione dell'impatto della complessità strutturale sui costi di manutenzione nelle applicazioni Open Source', Politecnico di Milano, 2006.

[36] Wikipedia, `http://it.wikipedia.org/wiki/JavaCC`
JavaCC, `https://javacc.dev.java.net/`

[37] `http://gretl.sourceforge.net/gretl_italiano.html`

[38] JavaCC Eclipse Plug-In, `http://eclipse-javacc.sourceforge.net/`