POLITECNICO DI MILANO

Facoltà di Ingegneria Informazione

Corso di Laurea in Ingegneria Informatica

Progetto di Ingegneria del Software 2

TravelDream

PROJECT REPORTING

Professore: **Luca Mottola**

Autori:
Emma Balgera
Sara Biancini
Pietro Bressana

Anno Accademico 2013-2014

2 Project Reporting

Questo documento rappresenta la quinta deliverable.
Lo scopo è *stimare alcuni parametri fondamentali come il tempo di consegna e i mesi-uomo necessari per lo sviluppo del progetto TravelDream.*

Indice

1. Function Point

Il **Function Point** è un'unità di misura utilizzata nell'ambito dell'Ingegneria del Software per esprimere la dimensione delle funzionalità fornite da un prodotto software.

Questa unità di misura è stata definita per la prima volta nel 1975 da Allan Albrecht (presso l'IBM), al fine di dimensionare i requisiti funzionali utente (FUR - Functional User Requirements) di un prodotto software già in fase di progettazione, al fine di ottenere una stima più oggettiva possibile dell'impegno richiesto.

1.1. Requisiti Funzionali e non Funzionali

Secondo gli standard i requisiti utente vengono distinti in:

- requisiti funzionali (FUR - Functional User Requirements)
- requisiti non funzionali (NFR - Non Functional Requirements)

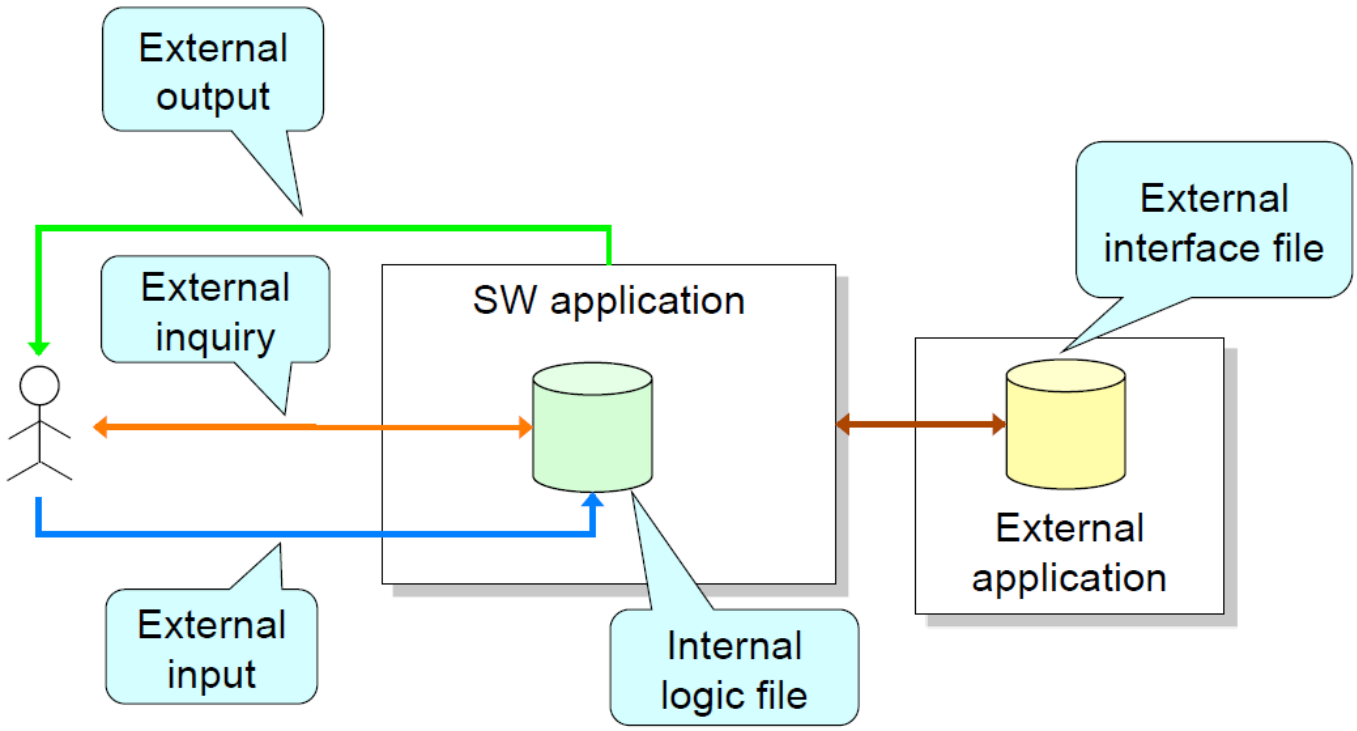
I *Function Point* misurano esclusivamente i FUR di un prodotto software, mentre per i NFR esistono diversi approcci e tecniche.

1.2. Obiettivi d'analisi

Gli obiettivi dell'analisi del metodo dei Function Point sono:

- quantificare l'impegno richiesto per sviluppare un prodotto software considerando le funzionalità che deve offrire
- dare una misura che sia indipendente dalla tecnologia utilizzata
- essere utilizzabile già dalle prime fasi dello sviluppo del software.

1.3. Tipi di funzione



dove:

- **Internal Logical File (ILF):**
  insieme omogeneo di dati utilizzati e gestiti dall'applicazione.

- **External Interface File (EIF):**
  insieme omogeneo di dati utilizzati dall'applicazione, ma generato e mantenuto da altre applicazioni.

- **External Input:**
  operazione elementare di elaborazione dei dati provenienti dall'ambiente esterno.

- **External Output:**
  operazione elementare che genera dati per l'ambiente esterno; di solito comprende l'elaborazione di dati da file logici.

- **External Inquiry:**
  operazione elementare che coinvolge input e output senza una significativa elaborazione dei dati da file logici.

1.4. (Adjusted) function points

Function types	**Weight**		
	Simple	Medium	Complex
Inputs	3	4	6
Outputs	4	5	7
Inquiry	3	4	6
ILF	7	10	15
EIF	5	7	10

Tab. 1.1 Tabella dei pesi metodo di Albrecht

Analizzando le specifiche, stimando il numero di funzioni di ogni tipo e applicando i relativi costi specificati in tabella, possiamo ottenere l'unadjusted function points (UFP).

Inoltre è possibile ottenere una stima del valore dei function points applicando la seguente formula correttiva:

$$FP = UFP * \left(0.65 + 0.01 * \sum_{i=1}^{14} F_i\right)$$

1.5. Applicazione del metodo function points

Nel nostro caso usando il metodo di Albrecht abbiamo:

- **EIF**

  Non sono presenti external interface files (EIF) perché tutti i dati sono gestiti all'interno dell'applicazione web.

- **ILF**

NOME	COMPLESSITA'
USER	BASSA
AMICO	BASSA
INVITO	BASSA
PACCHETTO	BASSA
COMPONENTE	BASSA

- **EI**

AZIONE	TIPO	COMPLESSITA'
MODIFICA COMPONENTE	EI	ALTA
BLOCCA PRENOTAZIONE	EI	ALTA
ACCETTA VIAGGIO	EI	ALTA
CONFERMA VIAGGIO	EI	ALTA
CREARE GIFTLIST	EI	ALTA
ACCETTARE PROPOSTA	EI	ALTA
RIFIUTARE PROPOSTA	EI	ALTA
REGISTRAZIONE	EI	BASSA
LOGIN	EI	BASSA
SALVATAGGIO PACCHETTO	EI	BASSA
RIMUOVERE PACCHETTO	EI	BASSA
RIMOVERE COMPONENTE	EI	BASSA
CREARE PACCHETTO	EI	BASSA
CREARE COMPONENTE	EI	BASSA
MODIFICA PACCHETTO	EI	MEDIA
INVITA AMICO	EI	MEDIA

- **EO**

AZIONE	TIPO	COMPLESSITA'
RICERCA PACCHETTO	EO	BASSA
RICERCA COMPONENTE	EO	BASSA
RICERCA COMPONENTE	EO	BASSA

- **EQ**

AZIONE	TIPO	COMPLESSITA'
DETTAGLI PACCHETTO	EQ	BASSA
DETTAGLI COMPONENTE	EQ	BASSA
DETTAGLI INVITO	EQ	BASSA
PROFILO UTENTE	EQ	BASSA
DETTAGLI GIFTLIST	EQ	BASSA

CALCOLO FUNCTION POINTS

BASSA	7	3	5	5	0	83
MEDIA	2	0	0	0	0	8
ALTA	7	0	0	0	0	42

TOT UFP
133

ORE/UFP	**8**	→	**TOT ORE**	**1064**
ORE/MESE	**100**	→	**TOT MESI**	**10,64**

2. COCOMO

2.1. Introduzione

COCOMO, abbreviazione di COnstructive COst MOdel, è un modello matematico creato da Barry Boehm utilizzato nell'ingegneria del software.
Cocomo è considerato un modello statico e analitico; statico in quanto le variabili di ingresso e di uscita sono ben definite e fisse, analitico perché può essere utilizzato non necessariamente nella sua interezza ma in parti per un progetto.

2.2. Modelli e definizioni

Esistono tre diversi modelli di Cocomo che si differenziano per la raffinatezza e la precisione con cui vengono stimati i diversi valori: Basic, Intermediate e Advanced, chiamato anche Detailed.

- **Basic COCOMO**: è il più facile da calcolare ma anche il meno preciso; la stima viene fatta partendo dalla dimensione del software da sviluppare calcolata in KLOC (totale linee di codice sorgente ad esclusione delle linee bianche/vuote).
- **Intermediate COCOMO**: calcola lo sforzo di sviluppo del software come funzione della grandezza del programma, espressa sempre in KLOC, e su un insieme di "indici di costi", detti Cost-driver, che includono l'assegnazione soggettiva di valutazioni di prodotti, hardware, attributi di progetto e personali.
- **Advanced/Detailed COCOMO**: incorpora tutte le caratteristiche del cocomo intermedio con in aggiunta una valutazione dell'impatto dei vari costi per ogni passo (analisi, progettazione, ecc.) del processo di ingegneria del software.

2.3. Tipologie di progetto per i livelli di COCOMO

Per ciascun livello di Cocomo esisitono tre diverse tipologie di progetto, Organic, Semi-detached e Embedded, che possono essere utilizzate come modello a seconda dei vincoli che si hanno:

- **Organic**: il progetto che si sta sviluppando è di piccole dimensioni, si ha già esperienza su questa tipologia di prodotti e si hanno pochi vincoli esterni.
- **Embedded**: è l'opposto dell'organic, il progetto è di grandi dimensioni, si ha poca esperienza su questa tipologia di prodotti, ci sono forti vincoli esterni sui costi e sui tempi.
- **Semi-detached**: è a metà via tra organic e embedded

Nel nostro caso prendiamo in esame il modello Basic e la tipologia di tipo Organic; la motivazione alla base di questa scelta sta nel fatto che il nostro team di sviluppo è composto da tre membri e le dimensioni del nostro prodotto software si possono considerare "piccole".

Nota Bene:
Una delle osservazioni importanti nel modello è che tutti i parametri sono affiancati da considerazioni del tutto soggettive. Ciò significa che le abilità del team e dell'individuo incaricato di tale valutazione influenzano in modo significativo il modello.

2.4. Basic COCOMO

La stima dell'impegno per lo sviluppo del software è ottenuta come funzione della dimensione del programma espressa in numero di righe di codice stimato (KLOC).

L'equazione di base del Cocomo ha la forma seguente:

$$E = a_b(\text{KLOC})^{b_b}$$

$$D = c_b(E)^{d_b}$$

$$P = \frac{E}{D}$$

dove:

- E è lo sforzo applicato (mesi-persona)
- D è il tempo di sviluppo (mesi)
- KLOC (Kilo Lines of Code) è il numero totale di linee del codice sorgente ad esclusione delle linee bianche/vuote (espresse in migliaia)
- P è il numero di persone richieste.

I coefficienti a_b, b_b, c_b e d_b sono ricavabili dalla tabella che segue:

Progetto Software	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Tab. 2.1 Tabella dei coefficienti per il Basic COCOMO

Concludendo il Basic Cocomo può essere utilizzato per stime veloci ma grezze del costo del software; la sua accuratezza è necessariamente limitata in quanto manca di fattori che tengano conto di differenze hardware, della qualità del personale e della sua esperienza, dell'uso di strumenti moderni e di altri attributi di progetto di cui è risaputa l'influenza sul costo del software.

2.5. Applicazione del metodo Basic COCOMO

Tramite l'uso di un software dedicato (http://cloc.sourceforge.net/) abbiamo calcolato il KLOC relativo al nostro progetto software.

La tabella relativa al nostro codice sorgente è la seguente:

```
http://cloc.sourceforge.net v 1.60  T=1.52 s (75.5 files/s, 7817.9 lines/s)
```

Language	files	blank	comment	code
Java	65	2531	1744	3894
JavaServer Faces	33	907	0	2191
SQL	1	63	96	281
XML	12	9	0	147
Visualforce Component	4	0	0	46
SUM:	115	3510	1840	6559

Tab. 2.2 Tabella del calcolo del LOC riguardante il nostro codice sorgente

Utilizzando il valore calcolato con il software (KLOC = 6,559+1,840 = 8,399), possiamo ora calcolare le equazioni base del COCOMO, considerando i parametri riportati in tabella 2.1.

$$E = a_b(\text{KLOC})^{b_b} = 2.4 * (8{,}399)^{1.05} = 22{,}421$$

$$D = c_b(E)^{d_b} = 2.5 * (22{,}421)^{0.38} = 8{,}151$$

$$P = \frac{E}{D} = \frac{22{,}421}{8{,}151} = 2{,}751$$

3. CONCLUSIONI

Applicando il metodo FUNCTION POINTS si ottiene un tempo di sviluppo dell'applicazione pari a 10 mesi circa.
Considerando che il nostro team è composto da 3 membri, si nota che lo sviluppo dovrebbe richiedere circa 3 mesi.

Applicando il metodo COCOMO BASIC si ottiene un tempo di sviluppo dell'applicazione pari a 8 mesi circa per un team di 3 persone.

La durata complessiva del nostro progetto è stata di circa 4 mesi.
Poiché più di 2 mesi sono stati spesi per la stesura della documentazione, ne è rimasto uno e mezzo per l'implementazione.

Confrontando l'effettiva durata della fase di sviluppo con le stime ottenute si nota che:

- il metodo FUNCTION POINTS fornisce un'approssimazione migliore perché considera le funzionalità dell'applicazione;
- il metodo COCOMO BASIC fornisce una stima eccessiva perché considera solo la dimensione del codice prodotto, senza tener conto della tipologia di codice e del numero di funzionalità implementate.

Indice delle tabelle