Sistemi Informativi Aziendali

Appunti per il corso - Capitolo 10

Luca Ardito Marco Torchiano

Politecnico di Torino – Dipartimento di Automatica e Informatica

Versione 0.3.3

3 febbraio 2021

INDICE

CAPITOLO 10

STIMA DEI COSTI

chi più spende, meno spende
Detto popolare

Dopo aver definito le caratteristiche di un sistema informativo tramite diversi modelli, è importante fare una stima dei costi di realizzazione.

Fino a questo momento abbiamo definito per esempio quali informazioni devono essere gestite dal nostro sistema informativo: lo abbiamo fatto rappresentando un diagramma UML delle classi e, successivamente, abbiamo definito il processo che il nostro sistema andrà a supportare. Per fare ciò abbiamo identificato gli attori e come questi ultimi devono interagire per raggiungere l'obiettivo prefissato. Poi abbiamo creato i casi d'uso e, per ognuno di essi, è stata fatta la narrativa per descrivere le interazioni che l'utente deve avere con il sistema per arrivare al suo obiettivo ed infine abbiamo creato i mockup che servono a dare una rappresentazione grafica del sistema che noi vogliamo realizzare.

A questo punto dobbiamo essere in grado di dare una stima dei costi del sistema che sarà realizzato. I costi di sviluppo del software sono essenzialmente costi di personale, quindi quello che si stima è il monte ore (*effort*) necessario.

Possiamo farlo utilizzando la nostra esperienza quindi, in base alla funzionalità che dobbiamo relizzare sappiamo più o meno i problemi che si tipicamente si incontrano e sapremo quante persone dovremo mettere in campo per realizzarlo. Sapremo dare più o meno una stima oppure possiamo avere delle metriche che ci diano una informazione a riguardo

In generale, le tre famiglie di tecniche adottate per la stima sono:

- Per analogia (*Analogy based*)
- Giudizio di esperti (*Expert judgment*)
- Tramite misure (*Metrics based*)

L'effort richiesto può essere calcolato attraverso:

- la dimensione del software che dobbiamo realizzare,
- la produttività del nostro team

Chi calcola la stima deve però essere in grado di dare dei valori a queste a queste metriche e ci sono diversi metodi per poter arrivare a queste informazioni. Possiamo avere ad esempio i function points, che sono abbastanza comuni nel contesto dello sviluppo software, oppure possiamo usare gli use case points che rappresentano la tecnica che studieremo in questo capitolo.

10.1 Stima tramite misure

$$Effort = Size_{sw} \times ProductivityNorm_{team} \tag{10.1}$$

L'$Effort$ necessario per sviluppare il software, di solito espresso in ore-persona o giorni-persona, è il prodotto della dimensione stimata del software $Size_{sw}$ per la norma di produttività del team di sviluppo $ProductivityNorm_{team}$.

La dimensione del software può essere calcolata in diversi modi, ad esempio tramite il numero di linee di codice (LoC). Tuttavia la stima spesso viene fatta *prima* dello sviluppo del software, quando del sistema sono note solo le funzionalità definite tramite i requisiti. Per poter stimare l'effort sulla base dei requisiti è necessario utlizzare un approccio noto come *Functional Size Measurement* (FSM)[1] che misura la dimensione dei requisiti usandola come una stima della dimensione del software.

L'approccio FSM ha dato origine a diverse tecniche per misurare i Function Points ed una tecnica per misurare gli Use Case Points [2].

10.1.1 Function points

L'analisi dei function points è stata sviluppata da Allan J. Albrecht alla fine degli anni '70. Ci sono diverse varianti, codificate negli standard ufficiali:

- ISO/IEC 19761 (COSMIC),
- ISO/IEC 20926 (IFPUG)
- ISO/IEC 20968 (Mk II),
- ISO/IEC 24570 (NESMA), and
- ISO/IEC 29881 (FiSMA)

Per avere un'idea di come funzionano questi metodi, riassumiamo qui di seguito, in maniera semplificate il il metodo COMSIC FP[3].

Si considera un sistema software come un'entità che interagisce con i suoi utenti attraverso un confine (interfaccia), e con i moduli di memorizzazione.

I requisiti degli utenti possono essere mappati in processi funzionali unici.

Ogni processo funzionale è costituito da sottoprocessi:

- movimentazione dati
- elaborazione dati

Un movimento di dati sposta un singolo gruppo di dati

- Entry: dati dall'utente al sistema.
- Exit: dati da sistema a utente.
- Write: dati dal sistema alla memorizzazione persistente.
- Read: dati dalla memorizzazione persistente al sistema.

I movimenti di dati trasferiscono dei *Data group*, ovvvero un insieme di attributi che descrivono un singolo oggetto di interesse.

Ogni processo viene avviato con l'attivazione del movimento dei dati di ingresso.

La dimensione del sistema in function points è calcolato come:

$$FP = Ne + Nx + Nr + Nw$$

dove:

- Ne = numero of Entries

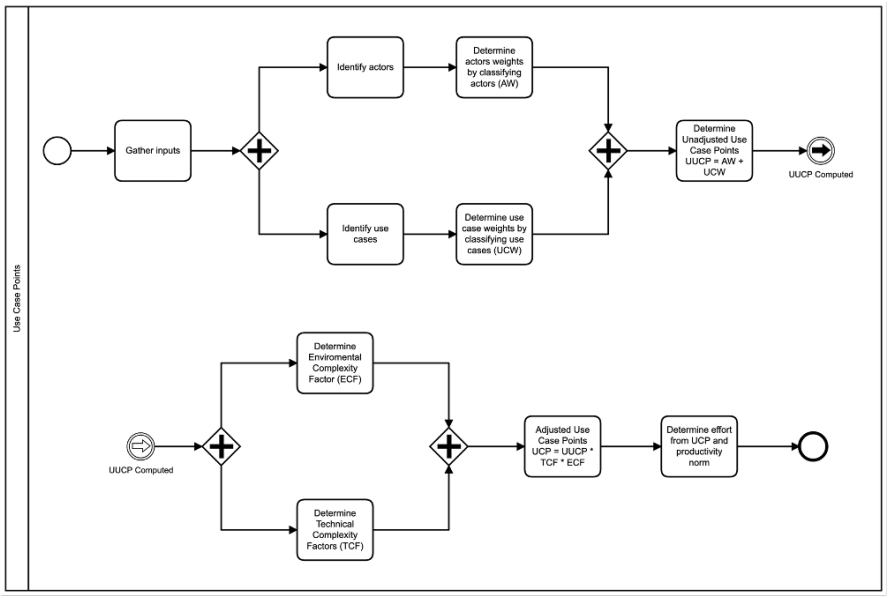

Figura 10.1: Processo che caratterizza il calcolo dell'effort attraverso gli Use Case Points

- Nx = numero of Exits
- Nr = numero of Reads
- Nw = numero of Writes

10.2 Use case points

Rispetto ai Function Points, la tecnica degli Use Case Points [2] risulta essere più astratta e tende a sovrastimare i costi in quanto produce delle stime per eccesso. Con l'esperienza, l'analista che andrà a fare questa questa stima riuscirà a calibrare i coefficienti in modo da avere valori affidabili. Inserendo poche informazioni relative ad il caso d'uso da implementare, saremo in grado di fornire una stima affidabile. Questa tecnica è diffusa a livello industrial dove diversi strumenti di supporto sono stati sviluppati [4].

I dati utili a stimare la dimensione di un'applicazione sono:

- il numero di attori che la utilizzeranno e la complessità delle relative interfacce;
- il numero di casi d'uso che questa applicazione deve essere in grado di supportare e la complessità delle transazioni che descrivono.

Successivamente entrano in gioco dei fattori di contesto:

- technical complexity factors che riguardano come le funzionalità dell'applicativo saranno progettate ed implementate;
- environmental complexity factor che riguardano il team che lavorerà al progetto.
- fattori di produttività del team che determinano l'effort a partire dalla dimensione del progetto.

Il processo che noi dobbiamo seguire per poter definire il costo è definito in Figura 10.1.

Per prima cosa è necessario dare un peso al agli attori quindi comprendere che tipo di attori utilizzeranno il sistema, successivamente bisogna definire il peso dei casi d'uso. Una volta calcolati, questi due valori vanno sommati, e si ottengono gli Unadjusted Use

Case Points (UUCP). Gli UUCP tengono in considerazione i requisiti funzionali del sistema che sono descritti tramite i casi d'uso. Poi bisogna calcolare i Technical Complexity Factors (TCF) e gli Environmental Complexity Factors (ECF). Ottenuti questi due valori si moltiplicano tra loro e per gli UUCP ottenendo gli Adjusted Use Case Points:

$$UCP = UUCP \cdot TCF \cdot ECF \tag{10.2}$$

Attraverso le productivity norms scaliamo gli UCP ottenendo l'effort in person hours:

$$PersonHours = UCP \cdot PN \tag{10.3}$$

Questo risultato va poi diviso per 8 in modo da ottenere i person days:

$$PersonDays = PersonHours/8 \tag{10.4}$$

Moltiplicando per il costo medio giornaliero del team otterremo il costo totale per implementare il nostro caso d'uso:

$$CostoUseCase = PersonDays \cdot CostoMedioGiornalieroTeam \tag{10.5}$$

10.2.1 Calcolo Actors Weight

Dobbiamo dare dei pesi diversi in base alla tipologia di interfaccia che il sistema deve avere verso gli attori. Si ha:

- un attore di complessità *semplice* che interagisce con il sistema attraverso una API,
- un attore di complessità *intermedia* che interagisce con il sistema attraverso un protocollo definito su una connessione di rete oppure attraverso un'interfaccia testuale (come tramite un terminale),
- un attore di complessità *elevata* che interagisce con il sistema attraverso un'interfaccia grafica.

Un attore semplice avrà peso 1 in quanto l'uso di una API non ha bisogno di progettazioni particolari ed è semplice da realizzare, un attore di complessità media avrà peso 2 poiché è necessario implementare dei protocolli di rete oppure deve gestire un'interazione tramite un'interfaccia testuale, mentre un attore di complessità elevata avrà peso 3 perché l'interfaccia grafica va sia progettata sia implementata e questo caratterizza difficoltà e costi superiori rispetto alle precedenti. Se più attori partecipano ad un caso d'uso i loro pesi vanno sommati.

Un esempio di attore semplice è quello di un POS per i pagamenti con il quale è possibile interagire tramite una libraria che espone una API direttamente utilizzabile dal sistema.

10.2.2 Calcolo Use Case Weight

Possiamo determinare il peso dei casi d'uso a partire dalla loro narrativa. Per farlo è necessario contare le transazioni. Il concetto di transazione nella versione originale è complesso e si riferisce allo stimolo di un attore che avvia elaborazioni che producono una risposta del sistema. Tuttavia una versione semplificata del calcolo delle transazioni è stato proposto sulla base del numero di passi nello scenario principale di successo [5], tale semplificazione non altera significativamente l'accuratezza delle predizioni [6].

Per transazioni intendiamo il numero di interazioni presenti nel main success scenario. Le transazioni si contano in questo modo:

- la prima e l'ultima interazione descritta nel main success scenario svolte dall'attore principale valgono entrambe come una transazione
- in tutti gli altri casi due interazioni del main success scenario formano una transazione.

Esempio:

1. Il cliente chiede di inserire un nuovo ordine
2. Il sistema chiede di inserire e confermare l'indirizzo di consegna
3. Il cliente inserisce l'indirizzo e conferma
4. Il sistema registra i dati inseriti e chiede di inserire i dati dell'ordine
5. Il cliente inserisce i piatti, le relative quantità e conferma
6. Il sistema mostra il riepilogo con il totale dell'ordine
7. Il cliente conferma
8. Il sistema salva la selezione

Il totale delle transazioni per questo caso d'uso sarà 5.
Il peso di un caso d'uso si assegna in questo modo:

- Con meno di 4 transazioni avremo un caso d'uso semplice con un peso pari a 5
- Con un numero di transazioni compreso tra 4 e 7 avremo un peso uguale a 10
- Con un numero di transazioni superiore a 7 avremo un peso uguale a 15

10.2.3 Calcolo Unadjusted Use Case Points

A partire dal peso degli attori e dei casi d'uso è possibile procedere al calcolo degli Use Case Point grezzi (Unadjusted Use Case Points - UUCP) sommando i pesi degli attori e dei casi d'uso.

Ad esempio nel caso d'uso precedente:

- Il peso degli attori (AW) è pari a 3 in quanto possiamo supporre che l'interazione dell'attore avviene tramite un'interfaccia web, quindi un attore complesso..
- Il peso del caso d'uso (UCW) è pari a 10 (essendo il numero di transazioni tra 4 e 7)

$$UUCP = AW + UCW = 3 + 10 = 13 \tag{10.6}$$

10.2.4 Technical Complexity Factors

Sono 13 fattori che è necessario contestualizzare all'interno del progetto. Questi fattori avranno un peso diverso e, per ogni fattore bisogna assegnare un voto da 0 a 5 in base all'importanza che ha questo fattore all'interno del sistema da progettare. Ogni voto assegnato va moltiplicato con il peso che ogni fattore possiede, ottenendo così un Technical Factor (TF) per ogni fattore. Il Technical Factor globale si ottiene sommando tutti e 13 i TF ottenuti.

T1 - Sistema Distribuito

Immaginiamo di avere una semplice applicazione installata sul nostro computer e che interagisce con l'utente solo attraverso un'interfaccia grafica. Chiaramente sarà semplice da realizzare perché noi non ci dobbiamo preoccupare di interagire con altri sistemi. Se invece il nostro sistema deve essere distribuito dovrà essere gestito il fatto che debba stare su macchine diverse, che debba essere raggiungibile dall'esterno cioè deve avere tutta una gestione aggiuntiva. Più il voto che assegniamo è alto più il nostro sistema dovrà essere distribuito. Se si sta progettando una applicazione standalone che è installata solo in un computer metteremo 0 se invece avremo un sito web distribuito con il back end che gira su dei server il front end che gira su altri server allora avremo un valore sicuramente più alto che avvicinerà a 5.

Factor	Description	Weight	Rating (0-5)	TF (W*R)
T1	Distributed System	2		
T2	Response time	2		
T3	End User Efficiency	1		
T4	Complex Internal Processing	1		
T5	Reusable Code	1		
T6	Easy to install	0.5		
T7	Easy to use	0.5		
T8	Cross-platform support	2		
T9	Easy to change	1		
T10	Concurrent	1		
T11	Includes Security Features	1		
T12	Provides Access for 3rd parties	1		
T13	User Training Required	1		

T2 - Response time

Il response time rappresenta il tempo minimo che il sistema deve impiegare per rispondere. Ad esempio, immaginiamo di dover inserire dei dati e di ottenere velocemente il risultato entro un tempo X. È però molto importante e importante che il response time sia costante. Se il sistema deve gestire la prenotazione di un biglietto di un concerto è ovvio che se la risposta arriva tra tra due secondi o un minuto non è così importante in quanto il concerto sarà tra due o tre mesi, però d'altra parte, potrebbe essere importante la reattività delle pagine. Si potrebbe quindi pensare che le pagine debbano essere caricate entro un certo tempo X. Tuttavia, questo vincolo è sicuramente meno importante rispetto al risultato di una elaborazione molto complessa. È necessario dare dei valori da 0 a 5 in base alla difficoltà (0 semplice, 5 molto complesso) che sistema incontrerà a fornire un risultato entro un certo limite.

T3 - End User Efficiency

È necessario capire se l'applicazione è stata sviluppata per ottimizzare l'efficienza che il l'utente a all'interno del suo contesto (lavorativo oppure no). Se il sistema da progettare ha un impatto molto elevato sull'efficienza dell'utente, allora dovremo inserire un valore molto alto altrimenti no. L'efficienza riguarda la capacità del sistema di permettere all'utente di svolgere i propri compiti in maniera veloce, ad esempio fornendo alcune informazioni già precompilate invece di richiederle direttamente all'utente.

T4 - Complex Internal Processing

Bisogna definire quanto sia complesso il lavoro che il sistema deve fare; per elaborazioni molto complicate daremo dei voti alti altrimenti, se semplicemente si tratta di salvare e leggere dei valori, metteremo valori molto bassi.

T5 - Reusable Code

Si tratta di valutare se il codice che si sviluppa dovrà essere facilmente riutilizzabile. Se buona parte del codice sviluppato nel progetto dovrà essere riutilizzabile si assegnerà un valore elevato, se le porzioni riutilizzabili sono molto limitate si potranno inserire valori bassi, o addirittura zero. Se il codice verrà riutilizzato in altri progetti è necessario dedicare del tempo aggiuntivo per scrivere la documentazione, rifattorizzarlo in modo che sia facilmente comprensibile e prevedere consizioni di uso all'esterno di quelli strettamente necessari per il progetto in corso.

T6 - Easy to Install

Se un'applicazione deve essere molto semplice da installare, allora chi sviluppa deve implementare tutta una serie di operazioni che facilitano il lavoro all'utente. Se invece l'applicazione è molto complicata da installare, allora richiederà l'intervento di una persona specializzata e ciò non richiederà di sostenere ulteriori costi all'interno del mio progetto per facilitare il lavoro all'utente. Per un'applicazione molto semplice da installare dovrò mettere un voto molto alto in questo campo. Se stiamo sviluppando applicazioni da distribuire al pubblico, questo fattore avrà rating relativamente alti, se l'installazione viene svolta, ad esempio, da personale addetto sui dispositivi degli utenti, possiamo avere rating bassi.

T7 - Easy to Use

L'usabilità è un fattore molto importante poiché bisognerà investire tempo nel creare una user experience di un certo livello e di conseguenza il nostro progetto costerà di più rispetto ad un caso in cui la facilità di utilizzo non è una priorità (ad esempio nel caso di un proof of concept). Se è necessaria una user experience di livello elevato bisognerà mettere un valore molto alto, mentre per una user experience minimale si metterà un valore basso. È importante sottolineare che sebbene la facilità di uso spesso giochi a favore dell'efficienza dell'end-user, rappresenta di solito aspetti molti più specifici di design dell'interfaccia e di accessibilità su diversi dispositivi.

T8 - Cross Platform Support

Esistono diverse categorie dei sistemi operativi e molte volte è necessario riprogettare la nostra applicazione per renderla compatibile con un diverso sistema operativo. Si assegneranno valori molto alti per assicurare compatibilità con più sistemi operativi e valori molto bassi in caso di compatibilità con uno solo. Se stiamo sviluppando applicazioni da distribuire ad un pubblico ampio questo fattore avrà rating alti: vogliamo che chiunque, indipendemente dal dispositivo mobile che ha o del browser che usa (sul proprio PC), possa utilizzare l'applicazione. Se invece stiamo sviluppando un sistema che dovrà funzionare all'interno di un'organizzazione che ha, ad esempio, un solo tipo di browser il rating sarà basso.

T9 - Easy to change

Se il cliente deve poter personalizzare in autonomia l'applicazione, bisogna fornire delle funzionalità che consentano personalizzazioni elevate. Analogamente se si prevede che il sistema evolverà nel futuro bisogna impostare la sua struttura di conseguenza. Questo costerà molto in termini di sviluppo rispetto ad una soluzione confezionata per un caso molto specifico. Quindi una personalizzazione elevata richiederà di assegnare voti molto alti mentre nessuna personalizzazione richiederà un voto pari a 0.

T10 - Concurrent

Ci sono operazioni possono avvenire in parallelo bisogna gestire eventuali conflitti che possono portare all'invalidazione dei dati presenti. Se questo avviene allora bisogna assegnare valori tendenzialmente alti se le operazioni avvengono in isolamento sicuramente assegneremo valori molto bassi. La domanda a cui bisogna rispondere è se più utenti devono poter accedere contemporaneamente agli stessi dati; più questo è frequente e più sono estesi i dati condivisi più è alto il sistema.

T11 - Security features

Se è possibile utilizzare delle soluzioni di sicurezza standard, conosciute letteratura e presenti sul mercato allora si possono assegnare valori molto bassi mentre se bisogna imple-

mentare delle funzionalità di sicurezza adattate al contesto specifico, allora costerà di più e quindi cui assegnerò un rating molto alto.

T12 - Provides Access for 3rd parties

Occorre valutare se il sistema sviluppato dovrà fornire delle API aggiuntive per consentire l'accesso alle funzionalità di base da parte di terze parti. Tale accesso sarà disponibile a SI terzi oppure ad App sviluppate all'esterno del progetto in considerazione. Se la porzione di funzionalità che devono essere rese accessibili è significativa si utilizzano valori elevati, se invece non è necessario fornire alcun accesso a terze parti è possibile inserire 0. Nel caso di accesso da terze parti occorre progettare o adottare (se definite da qualche standard) delle API per fornire l'accesso. È necessario del lavoro aggiuntivo per realizzare le API e verificare che siano integrate adeguatamente con il sistema. Un esempio significativo di fornitura di accesso verso terze parti è data dalla normativa PSD2[1] che prevede che le bance debbano fornire accesso alle proprie funzioni di base tramite API.

T13 - User Training Required

Se il sistema progettato richiede tanto training per poter essere utilizzato al meglio il vuol dire che sarà un software molto complicato e di conseguenza sarà difficile da implementare. Inoltre sarà richiesto un lavoro aggiuntivo per preparara esempi e documentazione per il training. Quindi se non è richiesto (o è richiesto poco) user training si assegneranno voti bassi, altrimenti si assegneranno voti molto alti. La complessità del training è un altro fattore da considerare: se basta qualche video-tutorial il rating sarà relativamente basso, se prevedo di dover svolgere seminari di formazione allora il rating sarà alto.

Una volta assegnati questi 13 voti, moltiplicheremo ognuno di essi per il suo peso e sommeremo tutti i valori ottenuti ottenendo così il Technical Factor Globale per il caso d'uso analizzato.

$$TFactor = \sum Tf \tag{10.7}$$

Si otterrà poi il Technical Complexity Factor attraverso questa formula:

$$TCF = 0.6 + 0.01 \cdot TFactor \tag{10.8}$$

10.2.5 Environmental Complexity Factors

Sono 8 fattori che è necessario contestualizzare all'interno del team che implementerà progetto. Questi fattori avranno un peso diverso e, per ogni fattore bisogna assegnare un voto da 0 a 5 in base all'importanza che ha questo fattore del team. Ogni voto assegnato va moltiplicato con il peso che ogni fattore possiede ottenendo così un Environmental Factor (EF) per ogni fattore. L'Environmental Factor globale si ottiene sommando tutti e 8 gli EF ottenuti.

Rispetto ai Technical Complexity Factor, danno un contributo negativo in termini di effort quando assegnamo valori alti (tranne gli ultimi due casi che hanno un peso negativo, dando quindi un contributo positivo all'effort). Per cui più alto sarà il voto assegnato più io risparmierò nel mio sviluppo.

F1 - Familiarity With The Project Domain

Questo fattore rappresenta l'esperienza nel team a gestire progetti simili. Non si tratta quindi del progetto specifico ma al contesto più generale. Ad esempio, se il mio team è esperto nel dominio del sistema da sviluppare, sicuramente conoscerà tutti i problemi e risolverà una serie di criticità più velocemente. Più elevata è l'esperienza più alto sarà il rating che assegneremo.

[1] Si veda ad es.: `https://www.agendadigitale.eu/cittadinanza-digitale/pagamenti-digitali/psd2-la-banca-piattaforma-le-api-aperte-nuovi-scenari/`

Factor	Description	Weight	Rating (0-5)	EF (W*R)
F1	Familiarity With The Project Domain	1.5		
F2	Application Experience	0.5		
F3	Object Oriented Experience	1		
F4	Lead Analyst Capability	0.5		
F5	Motivation	1		
F6	Stable requirements	2		
F7	Part Time Workers	-1		
F8	Difficulty of programming language	-1		

F2- Application experience

Questo fattore rappresenta quanto il team conosca l'applicazione da sviluppare. Se l'applicazione nuova assegneremo un voto 0 in quanto non la conosce nessuno. Se invece si tratta di sviluppi aggiuntivi come ad esempio inserire delle funzionalità ad un'applicazione esistente creata dallo stesso team, si velocizzeranno i tempi e quindi avremo dei costi inferiori assegnando un voto pari a 5.

F3 - Object Oriented Experience

Questo fattore indica il livello di esperienza nello sviluppo di codice utilizzando la programmazione ad oggetti. Con un team molto esperto assegnerò un valore molto alto, mentre con esperienza limitata assegnerò un rating molto basso.

F4 - Lead Analyst Capability

Quanto il Lead Analyst sia esperto influisce sulla qualità dei requisiti ottenuti dal cliente. Più è chiaro il lavoro da svolgere e meno volte il team sarà costretto a fare degli sviluppi ulteriori per rimediare in caso di errori. Quindi più alta è l'esperienza (e la capacità) del Lead Analyst, più alto sarà il voto assegnato.

F5 - Motivation

Se abbiamo un team motivato e molto probabile che sarà più veloce ed efficace nello sviluppo. I voti assegnati per questo fattore sono direttamente proporzionali alla motivazione del team.

F6 - Stable Requirements

Più i requisiti sono stabili, meno cambiamenti avremo nella nostra applicazione e di conseguenza meno volte il team dovrà ritornare sui suoi passi. I voti assegnati per questo fattore sono direttamente proporzionali alla stabilità dei requisiti. La stabilità dei requisiti dipende da fattori contestuali (non dipendenti dal team di sviluppo) e dalle capacità del lead analyst che potrebbe avere problemi a definire dei requisiti stabili. Caratteristiche che potrebbero portare ridurre la stabilità dei requisiti sono la scarsa conoscenza del dominio del progetto da parte del lead analyst, per problemi di comunicazione con il cliente (ad esempio scarsa conoscenza della lingua) e all'interno del team (ad esempio team sparpagliato geograficamente).

F7 - Part Time Workers

Se il personale si suddivide in più progetti allora esisterà un tempo di switch tra un progetto e l'altro. Questo tempo in cui il personale fa mente locale sull'ennesimo progetto a cui sta lavorando non è produttivo e rappresenta un costo aggiuntivo per il progetto. I voti assegnati saranno molto alti se il personale sarà impegnato in molti progetti e sarà 0 se

sarà impegnato in un unico progetto. Occorre considerare il part-time a livello di team e individuale:

- Lo stesso team lavora a più progetti contemporaneamente, questo porta ad un livello intermedio di impatto (rating 2-3)
- Ogni sviluppatore individualmente lavora in diversi team (ciascuno su diversi progetti) questo porta ad un impatto elevato (rating 4-5)

F8 - Difficulty of programming the language

Utilizzando un linguaggio di programmazione molto complesso avrò un team meno efficiente e sicuramente più costoso per la skill particolare. Si assegneranno voti molto alti se si utilizzano linguaggi di programmazione, o più in generale tecnologie, molto complessi, mentre si assegneranno voti molto bassi se si utilizzeranno linguaggi di programmazione più comuni. Un caso tipico in cui si assegnano rating elevati è quando il team è obbligato ad utilizzare tecnologie complesse o con cui ha scarsa familiarità (quindi difficili da utilizzare).

Una volta assegnati questi 8 voti, moltiplicheremo ognuno di essi per il suo peso e sommeremo tutti i valori ottenuti ottenendo così il Technical Factor Globale per il caso d'uso analizzato.

$$EFactor = \sum Ef \tag{10.9}$$

Si otterrà poi i Environmental Complexity Factor attraverso questa formula:

$$ECF = 1.4 - (0.03 \cdot EFactor) \tag{10.10}$$

Dopo aver calcolato UUCP, TCF e ECF si può procedere a calcolare gli Adjusted Use Case Points moltiplicando tra loro i tre valori.

$$UCP = UUCP \cdot TCF \cdot ECF \tag{10.11}$$

10.2.6 Calcolo della produttività

Dopodiché si procede al calcolo della Productivity Norm. Si ragiona su tre livelli di produttività

1. Alta: da 10 a 20 person hours per UCP (pari a 1,25 – 2,5 person days per UCP)
2. Media: da 14 a 28 person hours per UCP (pari a 1,75 – 3,5 person days per UCP)
3. Bassa: da 18 a 36 person hours per UCP (pari a 2,25 – 4,5 person days per UCP)

Questi valori devono essere calibrati in base al team con cui si lavora. Per i nostri esercizi useremo i gli estremi inferiori da tali intervalli: 10, 14, e 18.

Per decidere quali dei tre valori utilizzare bisogna calcolare n1 e n2.

Per calcolare n1 è necessario contare quanti fattori tra F1 e F6 hanno dei voti inferiori a 3. Per calcolare n2 è necessario contare quanti fattori tra F7 e F8 hanno voti superiori a 3.

Se n1 + n2:

- È <= 2 allora assumo una produttività alta (10 Ph per UCP)
- È compreso tra 3 e 4 allora assumo una produttività media (14 ph per UCP)
- È superiore a 4 allora assumo una produttività bassa (18 ph per UCP)

In pratica la somma n1+n2 indica quanti dei fattori ambientali possono risultare problematici, più sono minore sarà la produttività.

A questo punto avrò l'effort in person hours che equivale a:

$$EffortPH = UCP \cdot PN \tag{10.12}$$

Se voglio avere l'effort in person days devo dividere EffortPH per 8 (che rappresentano le 8 ore lavorative giornaliere)

$$EffortPD = \frac{EffortPH}{8} \tag{10.13}$$

L'effort indica quanto dovranno lavorare collettivamente i membri del team per sviluppare il software richiesto. L'ultimo passaggio è quello di calcolare quanto costerà lo sviluppo. Per ottenere il costo totale del caso d'uso devo moltiplicare EffortPD per il corso giornaliero medio del team

$$CostoUC = EffortPD \cdot CostoMedioTeam \tag{10.14}$$

10.3 Esempio

Un noto ristorante di un centro commerciale utilizza un sistema per la gestione della lista d'attesa. Quando arriva un cliente, il cameriere, una volta conosciuto il numero di persone del gruppo, controlla il tempo di attesa stimato per un tavolo sul sistema.

Contesto L'azienda decide di affidare parte dello sviluppo ad un team specializzato basato negli Stati Uniti e parte ad un team locale basato in Europa. Si decide di utilizzare l'inglese come lingua di riferimento ed il Lead Analyst è un esperto del settore di madrelingua svedese con un livello di inglese professionale. Entrambi i team sono impegnati su molti altri progetti, ma hanno un livello di esperienza molto elevato sia nel contesto di questo tipo di applicativi sia nella programmazione ad oggetti ed hanno un costo giornaliero medio di 300 euro. Sono richieste misure di sicurezza molto elevate per la gestione dei dati bancari degli utenti.

Estratto Narrativa Caso d'uso Use Case: Lista d'attesa Level: User-goal Intention in context: stimare il tempo di attesa Primary Actor: Cameriere Stakeholder interest: Il cliente vuole sapere l'esatto tempo di attesa per accomodarsi al tavolo Main Success Scenario

1. Il cameriere richiede una stima del tempo di attesa
2. Il Sistema richiede il numero di persone
3. Il cameriere inserisce il numero di persone
4. Il sistema fornisce una stima

Il caso d'uso termina con successo
Extensions: 4a non ci sono tavoli, il caso d'uso fallisce

Calcolo Actor Weight AW = 3 (si interagisce con il sistema attraverso una GUI)

Calcolo UseCase Weight UCW = 5 (3 transazioni 1, la coppia 2-3, 4) caso d'uso semplice

Calcolo Use Case Points Grezzi UUCP = AW + UCW = 8

Calcolo Technical Complexity Factors

$$TCF = 0.6 + (0.01 \cdot 14) = 0.74 \tag{10.15}$$

Calcolo Environmental Complexity Factors

$$ECF = 1.4 + (-0.03 \cdot 25) = 0.65 \tag{10.16}$$

Factor	Description	Weight	Rating (0-5)	TF (W*R)	
T1	Distributed System	2	1	2.0	
T2	Response time	2	2	4.0	
T3	End User Efficiency	1	3	3.0	
T4	Complex Internal Processing	1	0	0.0	
T5	Reusable Code	1	1	1.0	
T6	Easy to install	0.5	1	0.5	
T7	Easy to use	0.5	1	0.5	
T8	Cross-platform support	2	0	0.0	
T9	Easy to change	1	1	1.0	
T10	Concurrent	1	0	0.0	
T11	Includes Security Features	1	2	2.0	
T12	Provides Access for 3rd parties	1	0	0.0	
T13	User Training Required	1	0	0.0	
				TOTAL	**14.0**

Factor	Description	Weight	Rating (0-5)	EF (W*R)
F1	Familiarity With The Project	1.5	3	4.5
F2	Application Experience	0.5	4	2.0
F3	Object Oriented Experience	1	4	4.0
F4	Lead Analyst Capability	0.5	5	2.5
F5	Motivation	1	5	5.0
F6	Stable requirements	2	5	10.0
F7	Part Time Workers	-1	3	-3.0
F8	Difficulty of programming language	-1	0	0.0
			TOTAL	**25.0**

Calcolo Use Case Points

$$UCP = 8 \cdot 0.74 \cdot 0.65 = 3.848 \tag{10.17}$$

Calcolo Productivity Norm Productivity norm:

- n1 = 0
- n2 = 0

Seleziono la prima fascia:
10 Phrs / UCP = 1.25 Pdays / UCP

Calcolo Effort

$$EffortPH = 3.848 \cdot 10 = 38.48 Phrs \tag{10.18}$$

$$EffortPD = 38.48 Phrs = 4.81 Pdays \tag{10.19}$$

Assumendo un costo giornaliero medio di 300 euro al giorno
Costo = 1 443 €

BIBLIOGRAFIA

[1] A. Albrecht, "Measuring application development productivity," in *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, pp. 83—92, 1979.

[2] G. Karner, "Metrics for objectory'," Master's thesis, University of Linkoping, Linkoping, Sweden, December 1993.

[3] ISO/IEC, *Software Engineering — COSMIC-FFP — A functional size measurement method.*, vol. ISO/IEC 19761:2002. ISO/IEC, 2002.

[4] S. Kusumoto, F. Matukawa, K. Inoue, S. Hanabusa, and Y. Maegawa, "Estimating effort by use case points: method, tool and case study," in *10th International Symposium on Software Metrics, 2004. Proceedings.*, pp. 292–299, 2004.

[5] R. K. Clemmons, "Project estimation with use case points," *The Journal of Defense Software Engineering*, vol. 19, no. 2, pp. 18–22, 2006.

[6] M. Ochodek, J. Nawrocki, and K. Kwarciak, "Simplifying effort estimation based on use case points," *Information and Software Technology*, vol. 53, no. 3, pp. 200 – 213, 2011.

Licenza e Colophon

Questo volume è stato redatto con il sistema di composizione LaTeX[2] utilizzando il modello di stile `memoir`[3].

Il contenuto del testo è rilasciato con la licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 2.5 Italia (CC BY-NC-SA 2.5)[4].

[2] `http://www.latex-project.org/`
[3] `http://www.ctan.org/tex-archive/macros/latex/contrib/memoir/`
[4] `http://creativecommons.org/licenses/by-nc-sa/2.5/it/`