

Stima dei costi nei progetti Software:

Function Points e Use Case Points

- Rif. Sommerville, Capitolo 26 (stima dei costi)
- Pressman, Capitolo 20
- Estimating With Use Case Points, by Mike Cohn

Argomenti

- Produttività Software e determinazione del prezzo
- Function Point e Use Case Point

I problemi fondamentali della stima

- Quanto vale lo **sforzo** richiesto per completare una attività? (quanti mesi-uomo?)
- Quanto **tempo** (di calendario) è necessario per completare una attività?
- Qual è il **costo** totale di una attività?
- La stima dei costi e la tempistica di un progetto sono in genere eseguite insieme.

I fattori fondamentali dei costi

- Costi dell' Hardware e del software.
- Costi per viaggi e formazione.
- Costi dello sforzo (la voce più rilevante in molti progetti), pari agli stipendi degli ingegneri/programmatici coinvolti.
- Altri costi da considerare...
 - Costi di forniture, riscaldamento, illuminazione
 - Costi di tutto il personale a supporto: amministrazione, tecnici, personale delle pulizie, etc.
 - Costi di rete e telecomunicazioni
 - Costi di biblioteche e altri servizi
 - Costi di previdenza sociale
- I costi delle risorse “a contorno” di solito valgono almeno quanto i costi legati agli stipendi delle risorse addette direttamente alla produzione

Costi e Prezzi

- Le stime servono a prevedere i costi di sviluppo di un sistema software.
- Il **prezzo** da pagare dovrebbe essere la somma del **costo** più il **profitto**...
- Ma la relazione fra costi di sviluppo e prezzo richiesto al committente non è, in genere, semplice.
- Ci possono essere svariate considerazioni di tipo organizzativo, economico, politico ed aziendali che possono influire sul prezzo che verrà richiesto.

Fattori per la determinazione dei prezzi del Software

Opportunità di Mercato	Una organizzazione può fissare un prezzo di sviluppo più basso per entrare in un nuovo settore di mercato
Incertezza della Stima del Costo	Se si è incerti sui presunti costi di sviluppo, si può aumentare il prezzo
Termini di contratto	Il prezzo può variare a seconda se il contratto richiede la consegna del codice sorgente o no
Volatilità dei requisiti	Se si prevede che i requisiti cambieranno, si abbassa il prezzo per lo sviluppo, prevedendo di guadagnare anche per la manutenzione.
Salute Finanziaria	Se lo sviluppatore è in crisi finanziaria, può anche abbassare il prezzo, per sopravvivere

Produttività di sviluppo Software

- Per stimare i costi di sviluppo, può essere necessario conoscere la **produttività** degli ingegneri.
- La produttività è una misura della velocità con cui gli ingegneri producono software e la relativa documentazione.
- Un problema: in tali valutazioni della produttività non si tiene conto della qualità di quanto prodotto!
 - Confrontare la produttività di due team che sviluppino software con diversi requisiti di qualità non è realmente significativo.

Misure di Produttività

- Richiedono la misura di qualche attributo del software e si dividono tali misure per lo sforzo richiesto per lo sviluppo.
- **Size related measures** basate su qualche output del processo. (Es. LOC /mese programmatore)
- **Function-related measures** basate su una stima della funzionalità del software rilasciato (es. Function-points/ pm)

Le Linee di Codice

- Problema: Che cosa è una linea di codice (LOC)?
 - Il concetto era semplice per linguaggi di vecchia generazione (1 Linea -> 1istruzione->1 Scheda)
 - Non è di immediata traduzione per linguaggi moderni (una istruzione può corrispondere a più linee di codice, o una linea può contenere più istruzioni...)
 - Quali moduli del software vanno conteggiati? E quali no? (il problema dei componenti riusati)

Produttività al variare del linguaggio

- Più il linguaggio è di basso livello, più il programmatore può risultare produttivo.
 - Es. Assembler rispetto a C
 - (es. 700 vs. 300 LOC/mese)
 - Ma il tempo per lo sviluppo in C è complessivamente più breve! Dunque si avrebbe l'assurdo che il programmatore Assembler ha migliore produttività ma consegna dopo!
- Più il programmatore è verboso, più risulta produttivo.

I Function Point e l'analisi dei FP

- Tecnica proposta da **Albrecht** [*Measuring Application Development Productivity*, 1979]
- Approccio indipendente dal linguaggio di programmazione per misurare le funzionalità del sistema.
- Può essere usata per:
 - Stimare il costo richiesto per programmare, testare il software
 - Prevedere il numero di errori che si rileveranno durante il testing
 - Prevedere il numero di componenti o di LOC del sistema

Cosa sono i Function Point (FP)

- I FP sono una misura di funzionalità basata su entità logico-funzionali che l'utente facilmente comprende (es. Input, output, etc.)
 - I FP sono pertanto indipendenti dal linguaggio di programmazione, quindi la produttività può essere confrontata tra diversi linguaggi
- Un FP non è una singola caratteristica ma una combinazione di caratteristiche del sistema
 - Nati per essere applicati a Sistemi Sw di tipo Business
 - Misura che può essere effettuata 'presto' nel CVS

Function Point Analysis

★ I Function Point sono stati standardizzati:

- L'ISO (International Standard Organization) e l'IFPUG (International Function Point User Group) ne hanno promosso l'uso e successivamente la standardizzazione.
- **L'IFPUG 4.1 Unadjusted Function Point Method** è stato approvato dall'ISO/IEC Joint Technical Committee 1 (JTC1) ed è divenuto PAS (Publicly Available Specification) nel 2001.
- Il personale dedito al conteggio dei FP può essere anche certificato.

Function Point

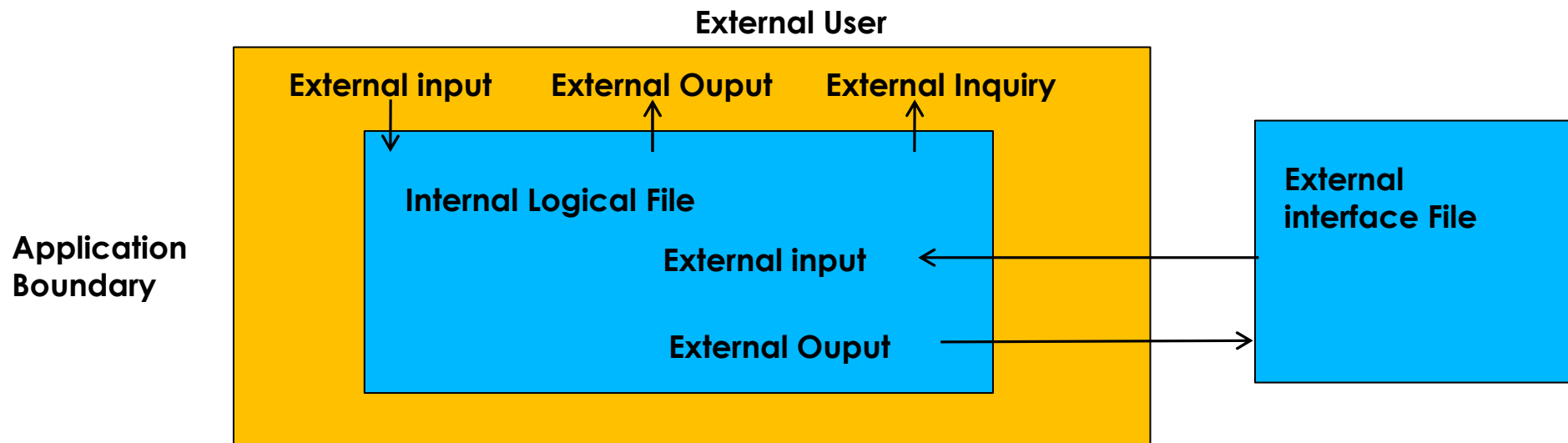
- I FP vengono derivati usando una relazione empirica che si basa su:
 - Misure dirette (esprese in valori interi) del **Dominio delle informazioni** del software
 - Valutazione della **complessità** del software
- Le Misure del Dominio delle Informazioni sono catturate con riferimento al Confine (Boundary) del sistema software.

Boundary

- Secondo il concetto di Boundary, esiste una linea chiusa che definisce il confine del sistema software cui è rivolta la misura.
- Con riferimento al confine, sono misurate o valutate le seguenti caratteristiche del programma:
 - External input (EI)
 - External output (EO)
 - External inquiry (EQ)
 - Internal Logical File (ILF)
 - External Interface File (EIF)

Definizione del Confine Applicativo

- ★ Il **confine applicativo** identifica le “ linee di divisione” fra:
- l'utente e l'applicazione che si vuole misurare
 - l'applicazione che si vuole misurare e eventuali altre applicazioni.



Internal Logical File (ILF)

- Un ILF è un raggruppamento unico di dati logicamente correlati, o di informazioni di controllo, identificabile dall'utente, usato ed aggiornato dall'applicazione.
- (Esempi tipici sono: tabelle di un database, file di configurazione, file di errore, password mantenuti dal sistema)
- I dati relativi non attraversano il confine, ma risiedono internamente al confine dell'applicazione e sono gestiti attraverso gli external input (EI).

External Interface File (EIF)

- Una EIF è un unico gruppo di dati logicamente correlati, o di informazioni di controllo, identificabile dall'utente, che risiede esternamente all'applicazione ma fornisce dati usati dall'applicazione.
- In pratica, i dati relativi attraversano il confine del sistema e risiedono interamente all'esterno del confine dell'applicazione e sono mantenuti da un'altra applicazione.
- Un EIF è un ILF di un'altra applicazione.

External Input (EI)

- External Input: Informazioni distinte fornite dall'utente o da altre parti del sistema o da altre applicazioni, usate come dati di ingresso
- Le informazioni di ingresso possono alterare un ILF (aggiungere, cambiare o cancellare dati in un File Logico Interno), oppure influire sul funzionamento del sistema
- **Esempi**: dati transazionali (una vendita, un appuntamento inserito), messaggi di altre applicazioni, ingressi che portano informazioni di controllo (un sensore).

External Output (EO)

- Un EO è un tipo di dati unico che viene creato nell'applicazione ed è destinato all'utente, o ad altre applicazioni, attraversando il confine dell'applicazione.
 - Un output è unico se il formato o la logica elaborativa che lo produce è unica.
- Un EO consiste in report, schermate, file, messaggi d'errore...
- Un EO può additionally aggiornare un ILF.
- Questi reports o files sono creati a partire da uno o più ILF e EIF.

External Output (EO)

Esempi di EO:

report su terminale o stampante che richiedono l'uso di algoritmi o calcoli, trasferimenti di file o dati da altre applicazioni verso l'utente, documenti di varia natura con calcoli (fatture, bollette, assegni, ...).

Non sono da considerare EO: messaggi di conferma o errore, report generati da un semplice query, più modi di invocare lo stesso processo di output.

External Inquiry (EQ)

- Una interrogazione esterna EQ è definita come una coppia input/output unica, dove un input online produce la generazione di una risposta immediata del software, sotto forma di output online.
 - L'intento primario è di presentare l'informazione all'utente attraverso il recupero da un ILF o EIF senza effettuare elaborazioni nè modifiche ad un ILF o EIF.
- Una EQ ricerca dati o informazioni di controllo (da uno o più ILF e/o EIF) per una risposta immediata.

Calcolo dei Function Point

- Esistono diverse teorie, più o meno complesse, per calcolare i FP in funzione del conteggio degli elementi prima descritti
 - Il livello di complessità di ciascun elemento individuato per ognuna delle 5 precedenti categorie viene classificato in:
 - **Basso (semplice)**
 - **Medio**
 - **Alto (complesso)**
 - Ad ognuno dei livelli è associato un peso che varia da 3 (più semplice) a 15 (più complesso).

Matrice dei pesi: un esempio

	Componenti	Livello di complessità		
		Basso	Medio	Alto
Transactiona l Function Types	Ext. Input (EI)	3	4	6
	Ext. Output (EO)	4	5	7
	Ext. Inquiry (EQ)	3	4	6
Data Function Types	Internal Logical files (ILF)	7	10	15
	Interface files (EIF)	5	7	10

UFC e FP

- Una prima stima dei FP è detta **UFC (Unadjusted Function Point Count)**:

$$\text{UFC} = \sum (\text{number of elements of given type}) \times (\text{weight})$$

- Lo UFC è poi corretto da un fattore moltiplicativo **TFC** compreso tra **0.65 e 1.35**


























- **DFP = UFC * TFC** (Delivered Function Point)

TFC- Fattore di Complessità Totale

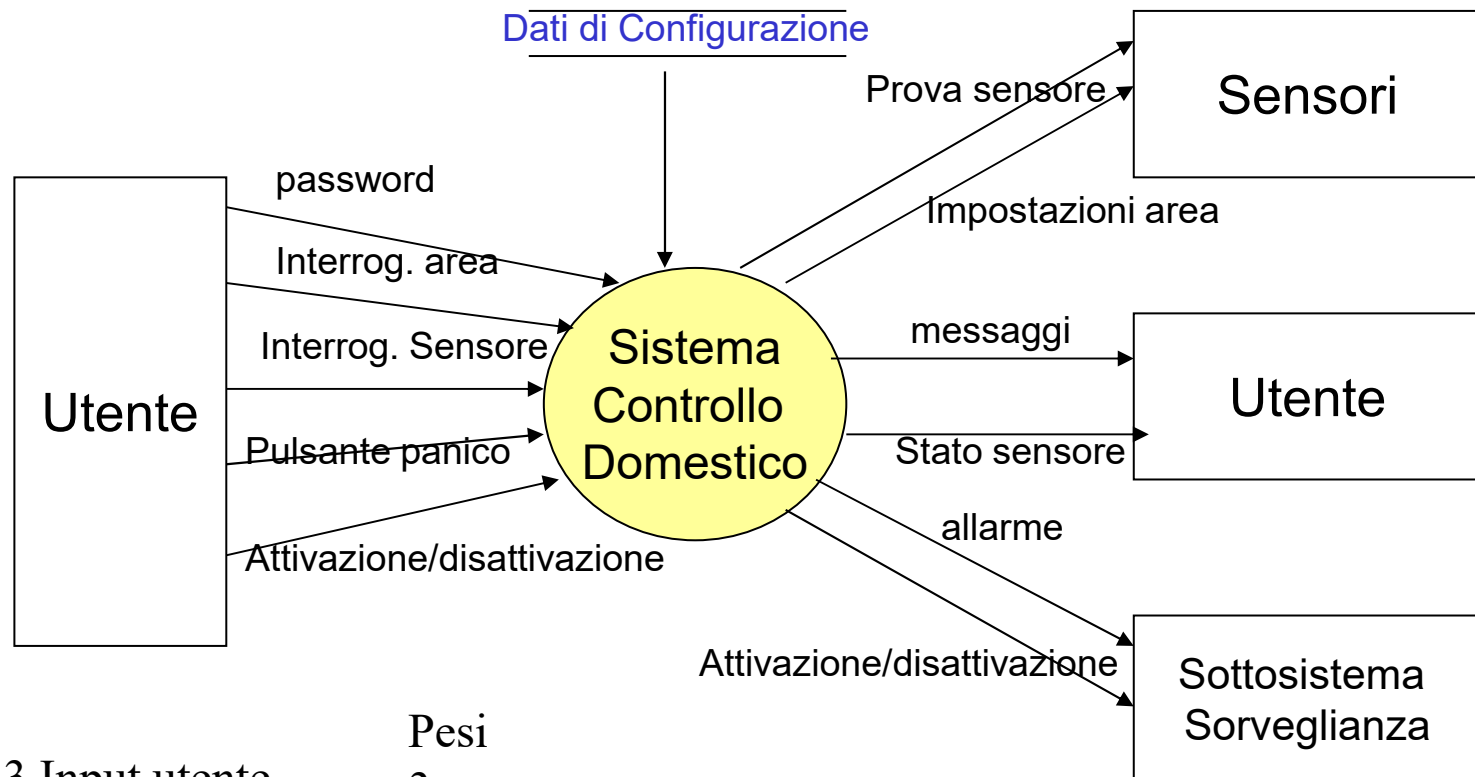
- Il TFC è calcolato sulla base di 14 Fattori di complessità (FC), valutati su una scala da 0 (ininfluente) a 5 (essenziale):
 1. Il sistema richiede backup e recovery affidabili?
 2. Sono richieste comunicazioni specializzate per trasferire dati da/verso l'applicazione?
 3. Vi sono funzionalità richiedenti elaborazioni distribuite?
 4. Le prestazioni sono un elemento critico?
 5. Il software funzionerà in un ambiente operativo esistente già pesantemente utilizzato?
 6. Il sistema richiede inserimenti online di dati?
 7. L'input online di dati richiede che la transazione relativa sia progettata su più schermate o più operazioni?
 8. Il file principali IFL sono aggiornati online?
 9. I dati di I/O, i file, le interrogazioni sono complesse?
 10. L'elaborazione interna è complessa?
 11. Il codice è progettato/scritto per essere riusabile?
 12. Il progetto comprende anche l'installazione e la conversione?
 13. Il software è stato progettato per essere installato presso diversi utenti?
 14. Il software è stato progettato per facilitare le modifiche e per un semplice uso da parte dell'utente finale?

$$\text{TFC} = 0.65 + 0.01 * \sum_{i=1..14} \text{FC}_i$$

Schema di calcolo FP

<u>measurement parameter</u>	<u>count</u>						
	<u>simple</u>	<u>avg.</u>	<u>complex</u>				
number of Ext. inputs	 X 3	+	 X 4	+	 X 6	=	
number of Ext. outputs	 X 4	+	 X 5	+	 X 7	=	
number of Ext. inquiries	 X 3	+	 X 4	+	 X 6	=	
number of Internal files	 X 7	+	 X 10	+	 X 15	=	
number of Ext.interfaces	 X 5	+	 X 7	+	 X 10	=	
count-total							
complexity multiplier							
							X
Function Points							

Un esempio di calcolo FP



	Pesi
3 Input utente	3
2 Interrogazioni	4
1 File Interno	3
2 output utente	7
4 interfacce esterne	5



UFC= 50



DFP= 50*[0.65+(0.01*46)]=56

Vantaggi e svantaggi dei FP

- Vantaggi:
 - Indipendenti dal linguaggio
 - Ottime indicazioni per applicazioni di elaborazione dati, che usano linguaggi convenzionali
 - Basati su quei dati che hanno la maggior probabilità di essere noti all'inizio di un progetto
- Svantaggi
 - Soggettività nell'assegnazione dei pesi
 - Dati sul dominio delle informazioni possono essere difficili da reperire
 - Nessuna valutazione della complessità dell'algoritmo (a parità di pochi input e output potrebbe essere banale o estremamente complicato)
 - I FP non hanno un diretto significato fisico

Utilizzi dei FP

- Per stimare la dimensione finale del codice:
 - Studi hanno rilevato una correlazione tra FP e numero di LOC, variabile a seconda del linguaggio di programmazione.
 - Es. COBOL: 1 FP -> 100 LOC
 - PL1 : 1FP -> 80 LOC
 - OO lang : 1 FP ->60 LOC
- Per stimare lo sforzo (in ore/uomo) di sviluppo:
 - Es. se 1 mese/uomo ->12 FP , allora ...
- Per valutare la completezza del testing
 - studi hanno misurato la correlazione fra FP e numero di difetti scoperti durante il testing

Stime di Produttività

- Per sistemi Real-time embedded, 40-160 LOC/P-month.
- Programmazione di sistema, 150-400 LOC/P-month.
- Applicazioni commerciali, 200-900 LOC/P-month.
- La produttività è influenzata da vari fattori (es. Esperienza nel dominio, qualità del processo, dimensione del progetto, supporto tecnologico, ambiente di lavoro)

Problemi delle misure di produttività

- Tutte le misure di produttività basate sul rapporto volume/unità di tempo sono difettose perchè considerano solo la quantità e non la qualità di ciò che si produce.
- La produttività potrebbe essere ‘aumentata’ a discapito della qualità.
- La metrica di produttività più semplice (LOC/pm), usando le LOC, non è un indicatore valido in assoluto ma dipende dai vari fattori indicati precedentemente.
- Se si usano i FP, il conteggio di FP dipende da valutazioni soggettive sulla complessità del software.

USE CASE POINTS

Usare i Casi d'Uso per stimare la dimensione del software

- USE CASE POINTS: sono basati sull'intuizione che più complessi sono i casi d'uso di un sistema, più tempo sarà necessario per lo sviluppo del codice.
- Concettualmente simili ai Function Point, permettono di misurare la dimensione di una applicazione e dedurre una stima sulla durata del progetto e i progressi del team.
- Use case points furono descritti inizialmente da Gustav Karner, poi da Schneider and Winters (1998) and Ribu (2001).
- *Ci riferiamo all'articolo: "Estimating With Use Case Points, by Mike Cohn"*

Use Case Points: dipendono da...

- il numero e la complessità dei casi d'uso nel sistema
- il numero e la complessità degli attori nel sistema
- vari requisiti non funzionali (quali portabilità, prestazioni, manutenibilità) che non sono stati scritti come casi d'uso
- l'ambiente in cui verrà sviluppato il progetto (come la lingua, la motivazione del team e così via)

1. Conteggio dei Casi d'Uso

- Si contano i Casi d'Uso del sistema
 - Il livello di dettaglio del Caso d'Uso è quello degli UC che permettono di raggiungere un Goal all'attore
- Si pesano i Casi d'Uso in base al numero di Transazioni (sia dello scenario normale che dei flussi alternativi).

Use case complexity	Number of transactions	Weight
Simple	3 or fewer	5
Average	4 to 7	10
Complex	More than 7	15

Table 1. Use case weights based on the number of transactions

Use Case Title:	Pay for a job posting
Primary actor:	Recruiter
Level:	Actor goal
Precondition:	The job information has been entered but is not viewable
Minimal Guarantees:	None
Success Guarantees:	Job is posted; recruiter's credit card is charged.
Main Success Scenario:	
1.	Recruiter submits credit card number, date, and authentication information
2.	System validates credit card.
3.	System charges credit card full amount.
4.	Job posting is made viewable to Job Seekers.
5.	Recruiter is given a unique confirmation number.
Extensions:	
2a:	The card is not of a type accepted by the system:
2a1:	The system notifies the user to use a different card.
2b:	The card is expired:
2b1:	The system notifies the user to use a different card.
2c:	The card number is invalid:
2c1:	The system notifies the user to re-enter the number.
3a:	The card has insufficient available credit to post the ad.
3a1:	The system charges as much as it can to the current credit card.
3a2:	The user is told about the problem and asked to enter a second credit card.
	the remaining charge. The use case continues at Step 2.

Figure 1. A sample use case for pay for a job posting

Si contano 10 transazioni complessive.

- 5 dello scenario normale.
- 5 transazioni alternative (2a1, 2b1, 2c1, 3a1, 3a2)
- Il caso d'uso è considerato complesso.

Unadjusted Use Case Weight, o UUCW

- Si ottiene sommando i prodotti di ciascun caso d'uso per il peso assegnato ad esso.
- Un esempio:

Use case complexity	Weight	Number of use cases	Product
Simple	5	40	200
Average	10	21	210
Complex	15	10	150
Total			560

Table 2. Calculating Unadjusted Use Case Weight (UUCW) for a simple project

2. Conteggio degli Attori

- Anche gli attori sono un fattore di complessità da considerare. Alcuni attori saranno più semplici (es. attore che usa una semplice CUI, o un sistema esterno via API), altri più complessi (uso di GUI complesse molto interattive).
- Possibili Complessità e Pesi per Attori:

Actor Type	Example	Weight
Simple	Another system through an API	1
Average	Another system through a protocol A person through a text-based user interface	2
Complex	A person through through a graphical user interface	3

Table 3. Actor complexity

Unadjusted Actor Weight (UAW)

- L' Unadjusted Actor Weight è la somma dei prodotti degli Attori per il rispettivo Peso.
- Esempio:

Actor Type	Weight	Number of Actors	Product
Simple	1	8	8
Average	2	7	14
Complex	3	6	18
Total			40

Table 4. Calculating Unadjusted Actor Weight (UAW) for a sample project

3. Calcolo dell' Unadjusted Use Case Points (UUCP)

- È la somma di Unadjusted Use Case Weight (UUCW) e Unadjusted Actor Weight (UAW).

$$\text{UUCP} = \text{UUCW} + \text{UAW}$$

- Nell'esempio precedente: $\text{UUCP} = 560 + 40 = 600$

4. Aggiustamenti per la Complessità Tecnica

- La complessità di un sistema dipende anche da altri fattori .
- Un sistema distribuito è più complesso di uno non distribuito.
- Un sistema con stringenti vincoli di prestazioni lo è più di uno senza tali vincoli.
- L'impatto della complessità tecnica è valutato attraverso 13 fattori, a cui viene dato un punteggio da 0 (irrilevante) a 5 (molto importante).

Pesi per i Fattori di Complessità Tecnica

Factor	Description	Weight
T1	Distributed system	2
T2	Performance objectives	2
T3	End-user efficiency	1
T4	Complex processing	1
T5	Reusable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent use	1
T11	Security	1
T12	Access for third parties	1
T13	Training needs	1

Table 5. The weight of each factor impacting technical complexity

Un esempio di valutazione dell'impatto dei Fattori di C. Tecnica per una Web application

An example assessment of a project's technical factors is shown in Table 6. This project is a web-based system for making investment decisions and trading mutual funds. It is somewhat distributed and is given a three for that factor. Users expect good performance but nothing above or beyond a normal web application so it is given a three for performance objectives. End users will expect to be efficient but there are no exceptional demands in this area. Processing is relatively straightforward but some areas deal with money and we'll need to be more carefully developing, leading to a two for complex processing. There is no need to pursue reusable code and the system will not be installed outside the developing company's walls so these areas are given zeroes. It is extremely important that the system be easy to use, so it is given a four in that area. There are no portability concerns beyond a mild desire to keep database vendor options open. The system is expected to grow and change dramatically if the company succeeds and so a five is given for the system being easy to change. The system needs to support concurrent use by tens of thousands of users and is given a five in that area as well. Because the system is handling money and mutual funds, security is a significant concern and is given a five. Some slightly restricted access will be given to third-party partners and that area is given a three. Finally, there are no unique training needs so that is assessed as a zero.

Un esempio di valutazione del Tfactor per una Web application

Factor	Weight	Assessment	Impact
Distributed system	2	3	6
Performance objectives	2	3	6
End-user efficiency	1	3	3
Complex processing	1	2	2
Reusable code	1	0	0
Easy to install	0.5	0	0
Easy to use	0.5	4	2
Portable	2	2	4
Easy to change	1	5	5
Concurrent use	1	5	5
Security	1	5	5
Access for third parties	1	3	3
Training needs	1	0	1
Total (TFactor)			42

Table 6. Example assessment of a project's technical factors

Calcolo del Technical Complexity Factor, TCF

- Ottenuto il Tfactor, si può calcolare il Technical Complexity Factor, TCF :

$$\text{TCF} = 0.6 + (0.01 \times \text{TFactor})$$

- Nell'esempio considerato:
- $\text{TCF} = 0.6 + (0.01 \times 42) = 1.02$.

5. Aggiustamento per la Complessità dell'Ambiente

Factor	Description	Weight
E1	Familiar with the development process	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time staff	-1
E8	Difficult programming language	-1

Table 7. Environmental factors and their weights

Esempio di calcolo dell'EFactor

Factor	Weight	Assessment	Impact
Familiar with the development process	1.5	3	4.5
Application experience	0.5	4	2
Object-oriented experience	1	4	4
Lead analyst capability	0.5	4	2
Motivation	1	5	5
Stable requirements	2	1	2
Part-time staff	-1	0	0
Difficult programming language	-1	2	-2
Total (EFactor)			17.5

Table 8. Example calculation of EFactor

$$EF = 1.4 + (-0.03 \times \text{EFactor})$$

Nel nostro esempio:

$$EF = 1.4 + (-0.03 \times 17.5) = 1.4 + (-0.51) = 0.89$$

Calcolo Finale degli Use Case Points (UCP)

- Si moltiplicano gli UUCP per i due fattori correttivi:

$$\text{UCP} = \text{UUCP} \times \text{TCF} \times \text{EF}$$

$$\text{UCP} = 600 \times 1.02 \times 0.89 = 545$$

Factor	Description	Weight
UUCW	Unadjusted Use Case Weight	560
UAW	Unadjusted Actor Weight	40
TCF	Technical Complexity Factor	1.02
EF	Environmental Factor	0.89

Table 9. Component values for determining total Use Case Points

Stimare la Dimensione e la Durata del Progetto a partire dagli UCP

- Use case points sono una stima della dimensione di un progetto.
- Per stimare la durata, basta conoscere la velocità di sviluppo di UCP da parte del team.
- Inizialmente, Karner propose un rapporto di 20 ore per UCP.
- Successivamente, Kirsten e Ribu raffinarono proponendo un valore tra 15 e 30 ore per UCP.
- Altri approcci tengono conto anche dei Fattori ambientali per proporre rapporto variabile fra 20 e 28.

Stimare la Durata del Progetto

- Una soluzione alternativa si basa su *dati storici aziendali*.
- Si valuta il tempo totale necessario a sviluppare un certo numero di UCP e da questi si ricava il rapporto.
- Ad es. se l'azienda ha sviluppato progetti per:
- 2,000 use case points in 44,000 hours of work, la media aziendale sarà 22 hours per use case point:
- $(44,000 \div 2000 = 22)$

Esempio di stima della durata di un progetto

- Si supponga che:
- Il progetto ha 545 use case points
- Il team impiega fra 20 e 28 ore per use case point
- Le iterazioni saranno lunghe 2 settimane
- Un team di 10 sviluppatori (programmers, testers, DBAs, designers, etc.) lavorerà su questo progetto.
- Saranno necessarie fra 10,900 e 15,260 ore:
- $545 \times 20 = 10,900$ e $545 \times 28 = 15,260$
- Ma quante iterazioni?

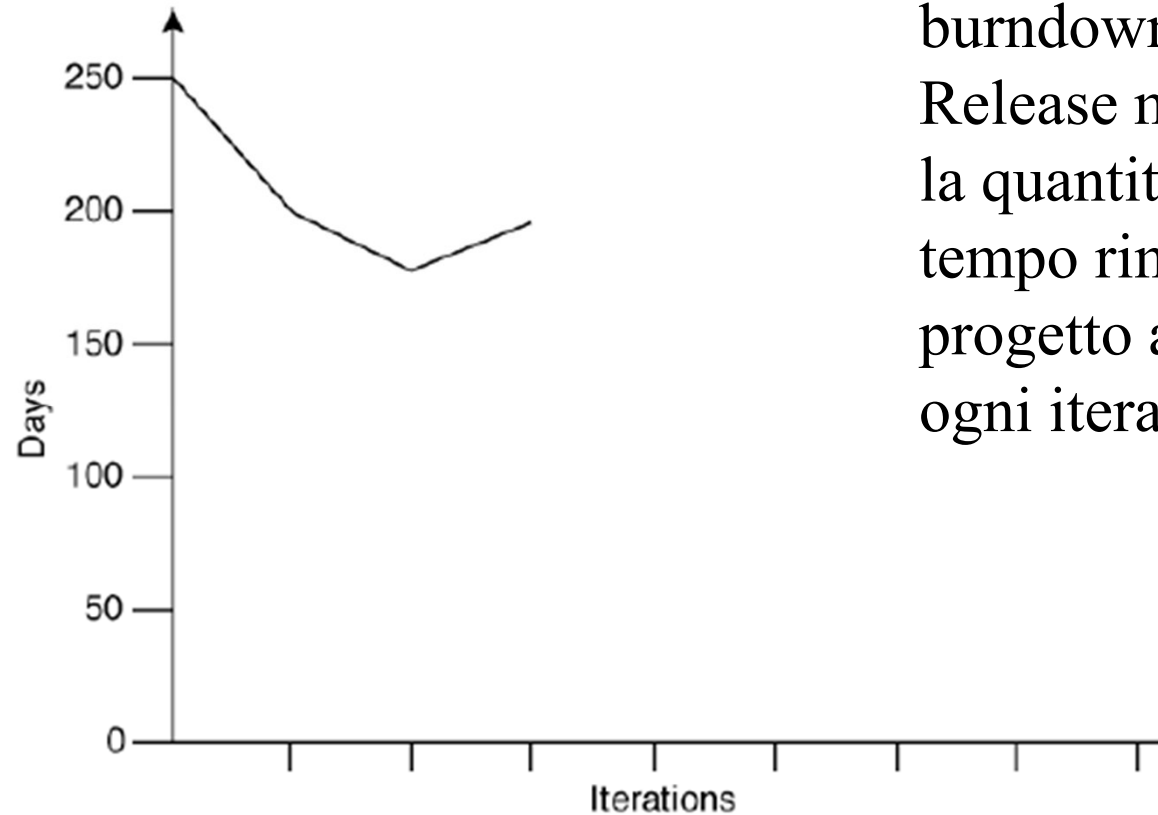
Stimare il numero di Iterazioni

- Se ogni sviluppatore lavora 30 ore a settimana sul progetto (e le restanti ore le dedica a email, meeting, etc.), il team dedicherà $10 \times 30 = 300$ hours per week o 600 hours di progresso per iteration.
- Dividendo 10.900 ore per 600 ore e arrotondando, il progetto complessivo potrebbe richiedere **19** iterazioni di due settimane.
- Dividendo 15.260 per 600 ore e arrotondando, potrebbero essere necessarie **26** iterazioni di due settimane.
- La nostra stima è quindi che questo progetto richiederà tra le 19 e le 26 iterazioni di due settimane (da 38 a 52 settimane).

Adattamenti per i Progetti Agili

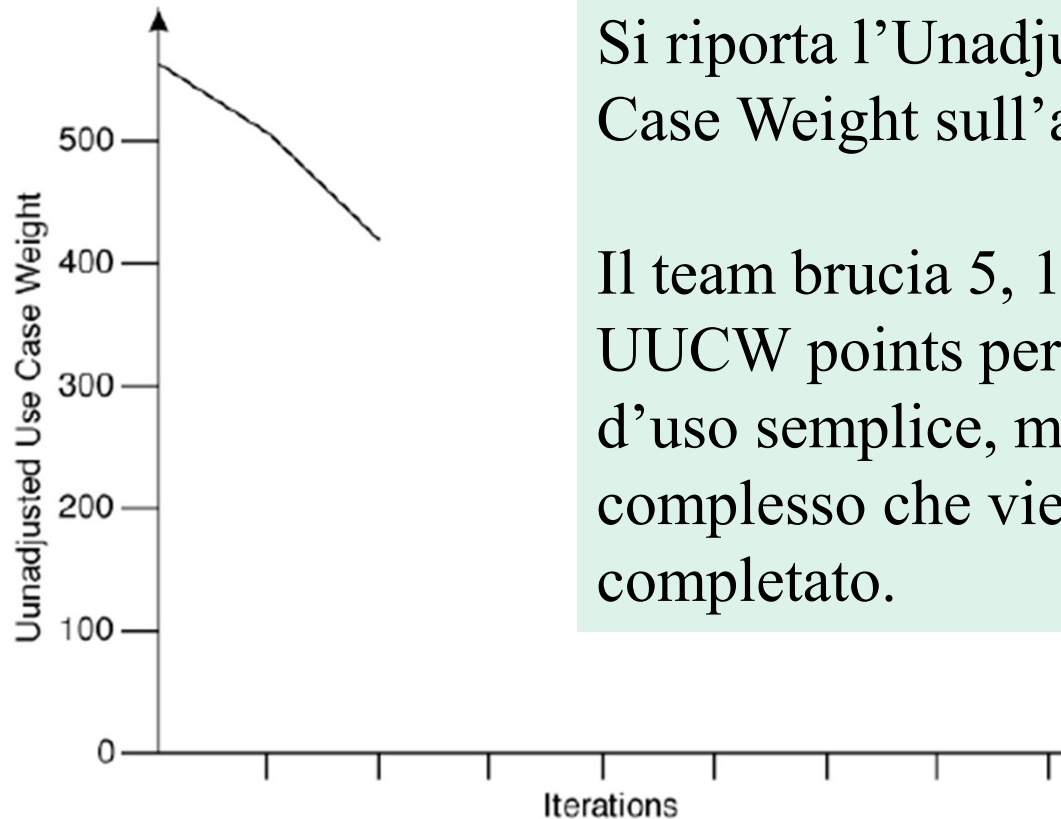
- Il conteggio degli UCP non è perfettamente compatibile con lo sviluppo Agile in cui i casi d'uso di raffinano iterativamente e non esiste un modello dei casi d'uso completo.
- Gli UCP si possono comunque usare per stimare la Velocità delle iterazioni.
- In Scrum, ad esempio, si usa il grafico della Burnrate Chart per controllare il tempo di lavoro che resta da fare
- Riporta sulle ordinate il tempo stimato rimanente di un progetto, all'inizio di ogni iterazione e sulle ascisse le iterazioni da svolgere.

Burnrate Chart



Un tipico grafico di burndown di una Release mostra la quantità stimata di tempo rimanente in un progetto all'inizio di ogni iterazione.

BurnRate Chart con UUCW



Si riporta l'Unadjusted Use Case Weight sull'asse verticale.

Il team brucia 5, 10, o 15 UUCW points per ogni caso d'uso semplice, medio e complesso che viene completato.