- *Sesto Blocco → è un ppt con un esercizio*

- ***Lezione 17***

- *Settimo Blocco*

Lo sviluppo software è un'attività complessa che richiede una specifica Gestione di Progetti Software (Software Project Management). Questa gestione implica:

- la pianificazione,
- il monitoraggio
- il controllo di persone, processi ed eventi durante lo sviluppo del prodotto

Il Software Project Management Plan (SPMP) è il documento che guida la gestione di un progetto software.

La gestione di un prodotto software si fonda su un concetto che viene riassunto con un termine "Le Quattro P". Rappresentano i 4 elementi fondamentali alla base del progetto Software.

- Persone, che rappresentano l'elemento più importante di un progetto software di successo (il SEI ha elaborato il People Management - Capability Maturity Model)
- Prodotto, che identifica le caratteristiche del software che deve essere sviluppato (obiettivi, dati, funzioni, comportamenti principali, alternative, vincoli)
- Processo, che definisce il quadro di riferimento entro cui si stabilisce il piano complessivo di sviluppo del prodotto software
- Progetto, che definisce l'insieme delle attività da svolgere, identificando persone, compiti, tempi e costi

L'ingegneria ci aiuta fornendoci vari approcci che possono essere utilizzati per gestire team o per organizzare le persone che sviluppano il software e così via.

- Approccio democratico

  È stato introdotto da Weinberg, nel 1971. È basato sul concetto di *egoless programming*, secondo cui ogni sviluppatore valuta la scoperta di fault (guasto) nel proprio codice come un attacco alla propria persona e non come un evento usuale. Questo approccio punta ad organizzare team che lavorino con un obiettivo comune, senza un singolo leader. Il codice appartiene al team come entità e non al singolo sviluppatore.

- Vantaggi:

  - atteggiamento positivo verso la ricerca di fault

  - molto produttivo in caso di problemi difficili da risolvere (es. ambienti di ricerca)

- Svantaggi:

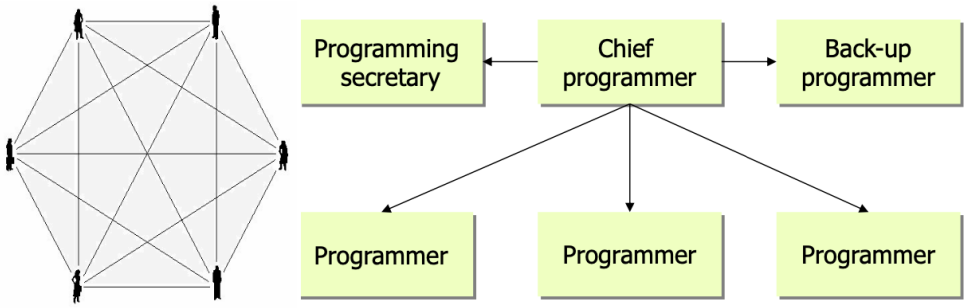
  - l'approccio non può essere imposto dall'esterno ma deve nascere spontaneamente

  - gli sviluppatori più anziani non desiderano essere valutati dai più giovani

- Approccio con Capo – Programmatore, detto anche Gerarchico

  Questo si basa su concetti di:

  - Specializzazione → ogni partecipante svolge i compiti per i quali è stato formato
  - Gerarchia → ogni sviluppatore comunica esclusivamente con il Capo - Programmatore, che dirige le attività ed è responsabile dei risultati



All'interno di ogni team c'è un Chief Programmaer (Capo Programmatore), al quale tutti i programmatori rendono conto.

Esiste la figura da Back-up Programmer, che è una persona (programmatore) che testa i programmi in caso di conflitti tra Programmatori e Capo Programmatore.

- Vantaggio → Un vantaggio immediato è che diminuisce il numero di canali di comunicazione
- Svantaggio → è che richiede personale molto esperto per ricoprire gli incarichi di:

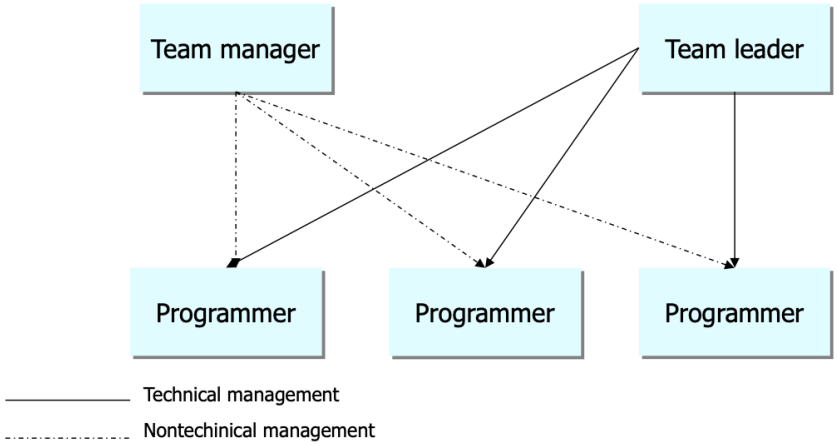
  - capo-programmatore (è sia un manager che un tecnico, sviluppa il progetto architetturale e le parti di codice critiche)

- programmatore di back-up (sostituisce il capo-programmatore ed è responsabile delle attività di testing)

- segretario di programmazione (responsabile della documentazione e dell'archivio di produzione)

C’è stata una sorta di Evoluzione degli Approcci, con il quale si vuole distinguere il ruolo gestionale dal ruolo tecnico. Il capo-programmatore, essendo sia manager che tecnico, risulta essere il valutatore di se stesso. Va dunque sostituito con due individui:

- Team Leader (responsabile aspetti tecnici)
- Team Manager (responsabile aspetti gestionali)



Esiste una statistica elaborata su progetti di sviluppo software negli USA dal 1984 al 2016, per dire quanti di questi progetti sono stati terminati in tempo, ritardati o cancellati.

	Application Types	On-time	Late	Canceled
1	Scientific	68%	20%	12%
2	Smart phones	67%	19%	14%
3	Open source	63%	36%	7%
4	U.S. outsource	60%	30%	10%
5	Cloud	59%	29%	12%
6	Web applications	55%	30%	15%
7	Games and entertainment	54%	36%	10%
8	Offshore outsource	48%	37%	15%
9	Embedded software	47%	33%	20%
10	Systems and middleware	45%	45%	10%
11	Information technology (IT)	45%	40%	15%
12	Commercial	44%	41%	15%
13	Military and defense	40%	45%	15%
14	Legacy renovation	30%	55%	15%
15	Civilian government	27%	63%	10%
	Total Applications	**50.13%**	**37.27%**	**13%**

La riuscita di un Progetto Software dipende dalla pianificazione di Progetti Software

- Obiettivo è definire un quadro di riferimento per controllare, determinare l'avanzamento ed osservare lo sviluppo di un progetto software
- Motivazione è essere in grado di sviluppare prodotti software nei tempi e costi stabiliti, con le desiderate caratteristiche di qualità
- Componenti fondamentali di una corretta pianificazione software sono:
    - Scoping (raggio d'azione): comprendere il problema ed il lavoro che deve essere svolto
    - Stime: prevedere tempi, costi e effort (quantità di lavoro)
    - Rischi: definire le modalità per l'analisi e la gestione dei rischi
    - Schedule: allocare le risorse disponibili e stabilire i punti di controllo nell'arco temporale del progetto
    - Strategia di controllo: stabilire un quadro di riferimento per il controllo di qualità e per il controllo dei cambiamenti

Una volta compreso cosa dovrà fornire il software. Quanto costerà sviluppare un prodotto?

Lo faremo con la “Stima”. Le attività di stima di tempi, costi ed effort nei progetti software sono effettuate con gli obiettivi di:

- ridurre al minimo il grado di incertezza
- limitare i rischi comportati da una stima

Risulta quindi necessario usare tecniche per incrementare l'affidabilità e l’accuratezza di una stima. Le tecniche di stima possono basarsi su:

- stime su progetti simili già completati (expert judgement by analogy)
- "tecniche di scomposizione" (approccio bottom-up), utilizzano una strategia "divide et impera" e sono basate su:
    - stime dimensionali, ad es. LOC (Lines Of Code) o FP (Function Point)
    - suddivisione dei task e/o delle funzioni con relativa stima di allocazione dell'effort

- modelli algoritmici empirici, si basano su dati storici e su relazioni del tipo:

$$d = f\ (\ v_i\ )$$

  dove d è il valore da stimare (es. effort, costo, durata) e v_i sono le variabili indipendenti (es. LOC o FP stimati)

- ***Lezione 18***

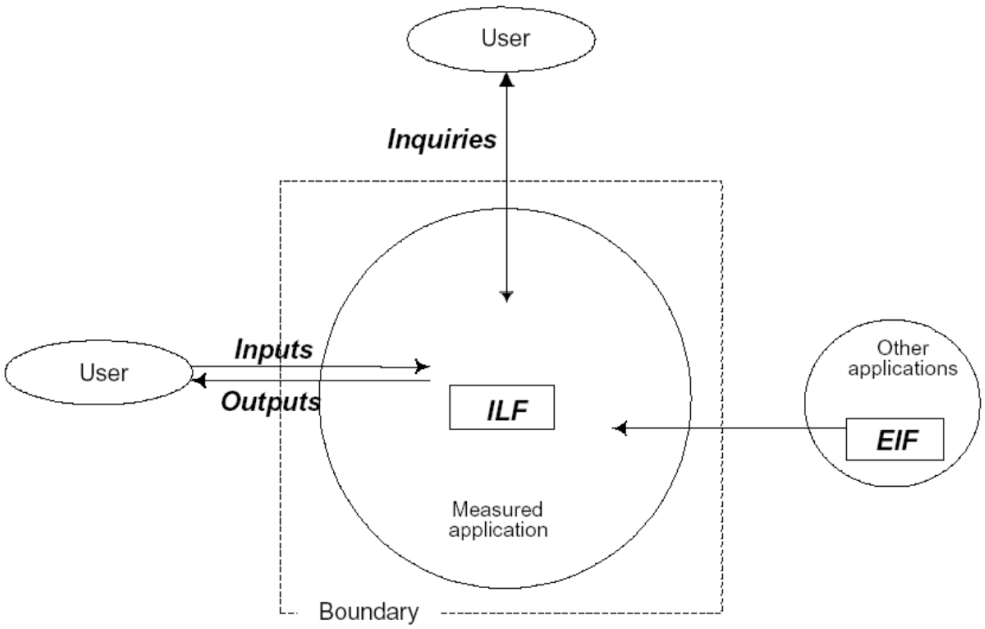
Function Point (Punto Funzione, o FP) è una misura ponderata della funzionalità del software proposta da Albrecht (1979 ~ 1983).

I punti funzione misurano la quantità di funzionalità che il prodotto software contiene. Si calcolano dal documento di specifica. Si è creata una vera e propria organizzazione internazionale, chiamata IFPVG, che ci dice come calcolare la quantità di punti funzione a partire dal documento di specifica. Il Punto Funzione è calcolato in due passaggi:

1. Calcolo di un conteggio dei punti funzione non aggiustato (UFC).
2. Moltiplicare l'UFC per un fattore di complessità tecnica (TCF).

Quindi avremo → $FP = UFC \times TCF$

L'UFC lo si calcola predendo in considerazione il documento di specifica 5 valori citati sotto:



Scrivi img. Prima di 20min

Gli elementi di conteggio che fanno riferimento ai dati:

- ILF – Number of Internal Logical Files → Un gruppo di dati o informazioni di controllo generate, utilizzate o mantenute dal sistema software.

- EIF – Number of External Interface Files → Un gruppo di dati o informazioni di controllo passati o condivisi tra le applicazioni, cioè interfacce leggibili dalla macchina ad altri sistemi e/o l'utente.

Il termine "file" non si inteso come lo intendiamo, ma come un gruppo di dati logicamente correlato e non all'implementazione fisica di tali gruppi di dati.

Per quanto riguarda, gli elementi di conteggio che fanno riferimento alla categoria delle transazioni:

- Numero di input esterni (Number of External Inputs, o EI) → Quegli elementi forniti dall'utente che descrivono dati distinti orientati all'applicazione, informazioni di controllo (come nomi di file e selezioni di menu) o output di altri sistemi che entrano in un'applicazione e cambiano lo stato dei suoi file logici interni.
- Numero di uscite esterne (Number of External Outputs, o EO) → Tutti i dati univoci o le informazioni di controllo prodotti dai sistemi software, ad esempio rapporti e messaggi.
- Numero di richieste esterne (Number of External Inquiries, o EQ) → Tutte le combinazioni di input/output uniche, in cui un input causa e genera un output immediato senza modificare alcuno stato di file logici interni.

La complessità tecnica ci serve per determinare il Punto Funzione. Questa complessità tecnica si calcola su una serie di fattori. Questi fattori sono i seguenti:

1. Backup e ripristino affidabili
2. Comunicazioni di dati
3. Elaborazione dati distribuiti
4. Prestazioni
5. Configurazione pesantemente utilizzata
6. Inserimento dati online
7. Facilitatore operativo
8. Aggiornamento online
9. Interfaccia complessa
10. Elaborazione complessa
11. Riutilizzabilità
12. Facilita l'installazione
13. Siti multipli
14. Facilitare il cambiamento

Ognuno di questi 14 fattori puo avere un’influenza definita da un numero tra:

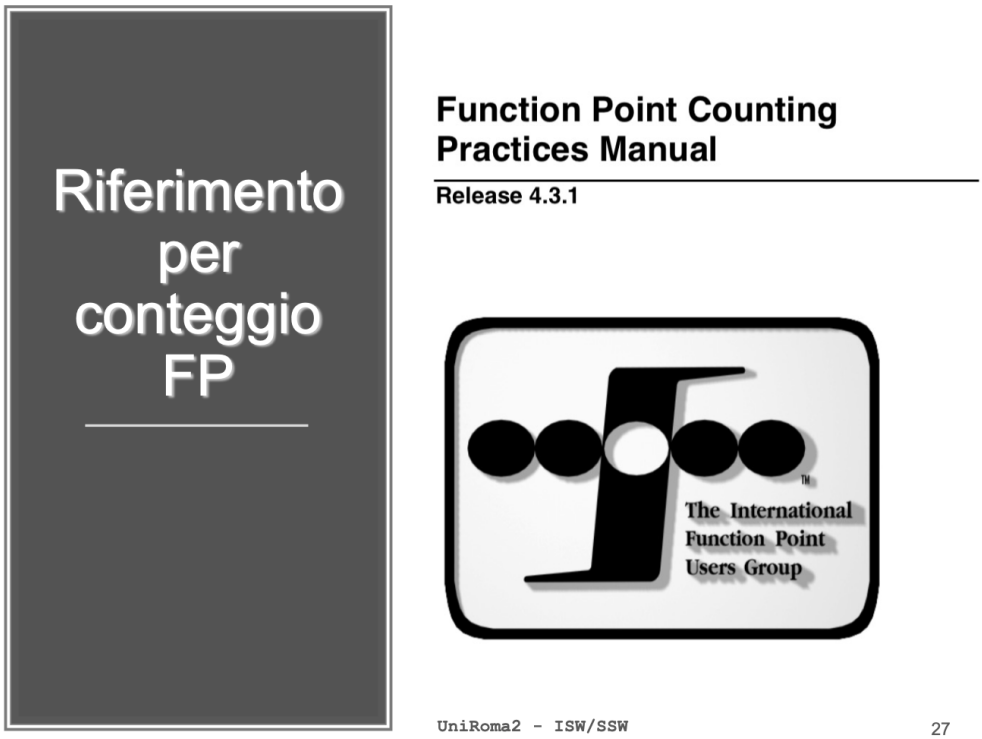
- 0 → influenza irrilevante
- 5 → influenza essenziale

Una volta associato uno tra questi valori possiamo calcolare:

$$TCF = 0.65 + 0.01 \sum_{j=1}^{14} F_j$$

Ovviamente il *TCF* è un numero puro. Il TCF varia da 0,65 (se tutti i F_j sono impostati su 0) a 1,35 (se tutti i F_j sono impostati su 5) → ± aggiustamento del 35%

Esistono dei manuali che ci permette di effettuare il calcolo dei punti funzione, come il seguente:



Un signor di nome Jones, nel 1996, ha calcolato i punti funzione di alcuni documenti di specifica e ha confrontato il numero di righe di codice (LOC) una volta implementato quel prodotto. Abbiamo un estratto che segue:

Language	Nominal level	Source statements per function point		
		Low	Mean	High
First generation	1.00	220	320	500
Basic assembly	1.00	200	320	450
Macro assembly	1.50	130	213	300
C	2.50	60	128	170
Basic (interpreted)	2.50	70	128	165
Second generation	3.00	55	107	165
Fortran	3.00	75	107	160
Algol	3.00	68	107	165
Cobol	3.00	65	107	170
CMS2	3.00	70	107	135
Jovial	3.00	70	107	165
Pascal	3.50	50	91	125
Third generation	4.00	45	80	125
PL/I	4.00	65	80	95
Modula 2	4.00	70	80	90
Ada 83	4.50	60	71	80
Lisp	5.00	25	64	80
Forth	5.00	27	64	85
Quick Basic	5.50	38	58	90
C++	6.00	30	53	125
Ada 9X	6.50	28	49	110
Database	8.00	25	40	75
Visual Basic (Windows)	10.00	20	32	37
APL (default value)	10.00	10	32	45
Smalltalk	15.00	15	21	40
Generators	20.00	10	16	20
Screen painters	20.00	8	16	30
SQL	27.00	7	12	15
Spreadsheets	50.00	3	6	9

- ***Lezione 19***

Per avere una stima di tempo, efford e costo possiamo basarci su modelli algoritmici, uno tra questi COCOMO. Il termine deriva da COnstructive COst MOdel è il modello introdotto da Boehm nel 1981 per determinare il valore dell'effort (colui che ha introdotto il modello a spirale).

Il valore ottenuto per l'effort viene successivamente utilizzato per determinare durata e costi di sviluppo. COCOMO comprende 3 modelli:

- Basic → per stime iniziali
- Intermediate → usato dopo aver suddiviso il sistema in sottosistemi
- Advanced → usato dopo aver suddiviso in moduli ciascun sotto sistema

La stima dell'effort viene effettuata a partire da:

- Stima delle dimensioni del progetto in KLOC (kiloLoc - 1 $KLOC = 10^3\ LOC$)
- Stima del modo di sviluppo del prodotto, che misura il livello intrinseco di difficoltà nello sviluppo, tra:
    - organic (per prodotti di piccole dimensioni)
    - semidetached (per prodotti di dimensioni intermedie)
    - embedded (per prodotti complessi)

Facciamo riferimento ad un particolare caso: Modello Intermidiate, modo Organic

Si tratta di una procedura a 2 step (simile al conteggio dei Punti Funzione):

- Passo_1: determinare l'effort nominale usando la formula:

$$E = a\ \times\ (\ KLOC\ )^{\ b}$$

$$Effort_{Nominale} = 3.2 \times\ (\ KLOC\ )^{\ 1.05}\ MM$$

  MM = Man/Month (Uomo/Mesi) quante persone per mese devono lavorare al progetto

- Passo_2: Ottenere la stima dell'effort applicando un fattore moltiplicativo C basato su 15 cost drivers:

$$Effort = Effort\ nominale \times C$$

  dove C (cost driver multiplier) si ottiene come produttoria dei cost driver c_i. Ogni c_i determina la complessità del fattore i che influenza il progetto e può assumere uno tra più valori assegnati con variazioni intorno al valore unitario (valore nominale)

Abbiamo dei fattori, detti Cost Driver, che hanno un rading che si aggira attorno al valore nominale 1 per tutti. Questo valore scende o sale asseconda dei casi:

Cost Drivers	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.16	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual machine volatility*		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Programmer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience*	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project Attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

*For a given software product, the underlying virtual machine is the complex of hardware and software (operating system, database management system) it calls on to accomplish its task.

Segue una tabella detta Cost Drivers Ratings, che ci dice qual è il criterio che devo usare per scegliere un certo Ratings anziché un altro.

Cost Driver	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
RELY	Effect: slight inconvenience	Low, easily recoverable losses	Moderate, recoverable losses	High financial loss	Risk to human life	
DATA		$\frac{\text{DB bytes}}{\text{Prog. DSI}} < 10$	$10 \leq \frac{D}{P} < 100$	$100 \leq \frac{D}{P} < 1000$	$\frac{D}{P} \geq 1000$	
CPLX	*See next slide*					
Computer attributes						
TIME			≤ 50% use of available execution time	70%	85%	95%
STOR			≤ 50% use of available storage	70%	85%	95%
VIRT		Major change every 12 months Minor: 1 month	Major: 6 months Minor: 2 weeks	Major: 2 months Minor: 1 week	Major: 2 weeks Minor: 2 days	
TURN		Interactive	Average turnaround <4 hours	4–12 hours	>12 hours	
Personnel attributes						
ACAP	15th percentile*	35th percentile	55th percentile	75th percentile	90th percentile	
AEXP	≤4 months experience	1 year	3 years	6 years	12 years	
PCAP	15th percentile*	35th percentile	55th percentile	75th percentile	90th percentile	
VEXP	≤1 month experience	4 months	1 year	3 years		
LEXP	≤1 month experience	4 months	1 year	3 years		
Project attributes						
MODP	No use	Beginning use	Some use	General use	Routine use	
TOOL	Basic microprocessor tools	Basic mini tools	Basic midi/maxi tools	Strong maxi programming, test tools	Add requirements, design, management, documentation tools	
SCED	75% of nominal	85%	100%	130%	160%	

* Team rating criteria: analysis (programming) ability, efficiency, ability to communicate and cooperate

Per quanto riguarda la complessità (CPLX) Ratings, qui ho un ulteriore tabella che mi permette di scegliere un valore in base a certe caratteristiche. Queste sono legate a diversi elementi: le operazioni che hanno un ruolo dal punto di vista computazionale, le operazioni che sono device dependent e le operazioni che operano sui dati di Data Manager. Quindi in base a queste caratteristiche, attraverso la tabella farò una scelta guidata.

Rating	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations
Very low	Straightline code with a few non-nested SP[a] operators: DOs, CASEs, IFTHENELSEs. Simple predicates	Evaluation of simple expressions: e.g., A = B + C * (D − E)	Simple read, write statements with simple formats	Simple arrays in main memory
Low	Straightforward nesting of SP operators. Mostly simple predicates	Evaluation of moderate-level expressions, e.g., D = SQRT (B**2-4.*A*C)	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level. No cognizance of overlap	Single file subsetting with no data structure changes, no edits, no intermediate files
Nominal	Mostly simple nesting. Some intermodule control. Decision tables	Use of standard math and statistical routines. Basic matrix/vector operations	I/O processing includes device selection, status checking and error processing	Multi-file input and single file output. Simple structural changes, simple edits
High	Highly nested SP operators with many compound predicates. Queue and stack control. Considerable intermodule control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns	Operations at physical I/O level (physical storage address translations; seeks, reads, etc). Optimized I/O overlap	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level
Very high	Reentrant and recursive coding. Fixed-priority interrupt handling	Difficult but structured N.A.: near-singular matrix equations, partial differential equations	Routines for interrupt diagnosis, servicing, masking. Communication line handling	A generalized, parameter-driven file structuring routine. File building, command processing, search optimization
Extra high	Multiple resource scheduling with dynamically changing priorities. Microcode-level control	Difficult and unstructured N.A.: highly accurate analysis of noisy, stochastic data	Device timing-dependent coding, micro-programmed operations	Highly coupled, dynamic relational structures. Natural language data management

[a] SP = structured programming

La durata la posso ottenere attraverso un'altra formula "offerta" sempre da COCOMO. Si tratta di una Stima del tempo T alla consegna (product delivery):

- Modo organic → $T = 2.5\ E^{0.38}$ (Months M)
- Modo semi-detached → $T = 2.5\ E^{0.35}$
- Modo embedded → $T = 2.5\ E^{0.32}$

Le persone che fanno parte del team di sviluppo svolgono ruoli differenti all'interno del team. Suddivido l'effort, nelle varie fasi che devo realizzare. I costi di sviluppo (C) sono stimati allocando lo sforzo di sviluppo (E) sulle fasi e sulle attività del personale, ad esempio:

- 16% preliminary design
  - 50% project manager
  - 50% analyst

- 62% detailed design, coding and testing
  - 75% programmer/analyst
  - 25% programmer

- 22% Integration
  - 30% analyst
  - 70% programmer/analyst

La categoria costo per uomo/mese (ad esempio, project manager, analista, programmatore, ecc.) viene quindi utilizzata per ottenere i costi di sviluppo.

Nella fase di pianificazione bisogna occuparsi della Pianificazione Temporale.

Dopo aver scelto il modello di processo, identificato i task da eseguire e stimato durata, costi ed effort, è necessario effettuare la pianificazione temporale ed il controllo dei progetti.

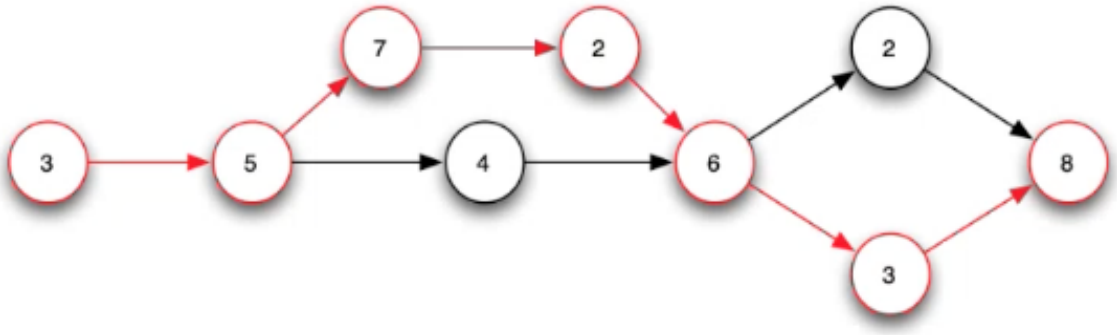
La pianificazione temporale consiste nel definire una "rete di task" in base ai seguenti principi fondamentali:

- Ripartizione → scomposizione di processo e prodotto in parti (task e funzioni) di dimensioni ragionevoli
- Interdipendenza → identificazione delle dipendenze reciproche tra i task individuati
- Allocazione di risorse → determinazione di numero di persone, effort e date di inizio/fine da assegnare ad ogni task
- Responsabilità definite → individuazione delle responsabilità assegnate a ciascun task
- Risultati previsti → definizione dei risultati prodotti al termine di ogni task
- Punti di controllo(milestone) → identificazione dei punti di controllo della qualità da associare al singolo task o a gruppi di task

L'ingegneria del software ci mette a disposizione 2 strumenti:

- Diagramma PERT (Program Evaluation and Review Technique) → grafo in cui ogni nodo rappresenta un task ed ogni arco un legame di precedenza. Questo consente di determinare:

- il cammino critico (sequenza di task che determina la durata minima di un progetto)
- la stima del tempo di completamento di ciascun task, mediante applicazione di modelli statistici
- i limiti temporali di inizio e termine di ciascun task

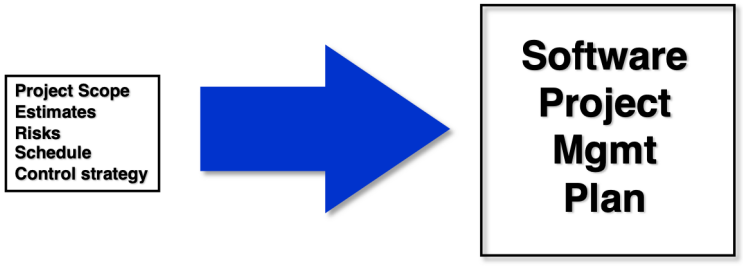


L'informazione di "limiti" di tempo di inizio e fine di ciascun task è determinata dalla:

- Carta di Gantt. Questo infatti è un diagramma a barre che consente di visualizzare l'allocazione temporale dei task. Però non appare nessuna indicazione dei legami di precedenza, quindi viene integrata con un diagramma PERT (sopra).

  [Ha fatto l'esempio: abbiamo un progetto dove un'attività inerente ad esso deve iniziare un giorno, ma un'attività precedente sempre inerente ad esso dura più giorni non possiamo vederlo attraverso la carta di Gantt e ci serve il Diagramma di Pert]

Tutti questi strumenti (COCOMO, Tabelle di Scomposizione, Gantt, ecc..) fa parte del processo ampio di Software Project Management Plan (SPMP o anche Piano di Gestione del Progetto Software)



Dal punto di vista organizzativo questo documento si organizza attraverso dei template. Seguono un paio di template.

- Manager's Handbook for Software Development (fornito dal Laboratorio di Ingegneria del Software (SEL) della NASA, del 1990)

TITLE PAGE — document number, project and task names, report title, and report date.
LEAD SHEET — document identification numbers, project and task names, report title, customer name, preparers, contract and task identifiers, and report date.
TABLE OF CONTENTS — list of subsection titles and page numbers.

1. **INTRODUCTION**
   - 1.1 **Purpose** — brief statement of the project's purpose.
   - 1.2 **Background** — brief description that shows where the software products produced by the project fit in the overall system.
   - 1.3 **Organization and Responsibilities**
     - 1.3.1 **Project Personnel** — explanation and diagram of how the development team will organize activities and personnel to carry out the project: types and numbers of personnel assigned, reporting relationships, and team members' authorities and responsibilities (see Section 3 for guidelines on team composition).
     - 1.3.2 **Interfacing Groups** — list of interfacing groups, points of contact, and group responsibilities.
2. **STATEMENT OF PROBLEM** — brief elaboration of the key requirements, the steps to be done, the steps (numbered) necessary to do it, and the relation (if any) to other projects.
3. **TECHNICAL APPROACH**
   - 3.1 **Reuse Strategy** — description of the current plan for reusing software from existing systems.
   - 3.2 **Assumptions and Constraints** — that govern the manner in which the work will be performed.
   - 3.3 **Anticipated and Unresolved Problems** — that may affect the work and the expected effect on each phase.
   - 3.4 **Development Environment** — target development machine and programming languages.
   - 3.5 **Activities, Tools, and Products** — for each phase, a matrix showing: a) the major activities to be performed, b) the development methodologies and tools to be applied, and c) the products of the phase (see Section 4). Includes discussion of any unique approaches or activities.
   - 3.6 **Build Strategy** — what portions of the system will be implemented in which builds and the rationale. *Updated at the end of detailed design and after each build.*
4. **MANAGEMENT APPROACH**
   - 4.1 **Assumptions and Constraints** — that affect the management approach, including project priorities.
   - 4.2 **Resource Requirements** — tabular lists of estimated levels of resources required, including estimates of system size (new and reused LOC and modules), staff effort (managerial, programmer, and support) by phase, training requirements, and computer resources (see Section 3). Includes estimation methods or rationale used. *Updated estimates are added at the end of each phase.*
   - 4.3 **Milestones and Schedules** — list of work to be done, who will do it, and when it will be completed. Includes development life cycle (phase start and finish dates); build/release dates; delivery dates of required external interfaces; schedule for integration of externally developed software and hardware; list of data, information, documents, software, hardware, and support to be supplied by external sources and delivery dates; list of data, information, documents, software, and support to be delivered to the customer and delivery dates; and schedule for reviews (internal and external). *Updated schedules are added at the end of each phase.*
   - 4.4 **Metrics** — a table showing, by phase, which metrics will be collected to capture project data for historical analysis and which will be used by management to monitor progress and product quality (see Section 6 and Reference 3). If standard metrics will be collected, references to the relevant standards and procedures will suffice. Describes any measures or data collection methods unique to the project.
   - 4.5 **Risk Management** — statements of each technical and managerial risk or concern and how it is to be mitigated. *Updated at the end of each phase to incorporate any new concerns.*
5. **PRODUCT ASSURANCE**
   - 5.1 **Assumptions and Constraints —** that affect the type and degree of quality control and configuration management to be employed.
   - 5.2 **Quality Assurance (QA)** — table of methods and standards used to ensure the quality of the development process and products (by phase). Where these do not deviate from published methods and standards, the table references the appropriate documentation. Means of ensuring or promoting quality that are innovative or unique to the project are described explicitly. Identifies the person(s) responsible for QA on the project, and defines his/her functions and products by phase.
   - 5.3 **Configuration Management (CM)** — table showing products controlled, tools and procedures used to ensure the integrity of the system configuration: when the system is under control, how changes are requested, who makes the changes, etc. Unique procedures are discussed in detail. If standard CM practices are to be applied, references to the appropriate documents are sufficient. Identifies the person responsible for CM and describes this role. *Updated before the beginning of each new phase with detailed CM procedures for the phase, including naming conventions, CM directory designations, reuse libraries, etc.*
6. **REFERENCES**
7. **PLAN UPDATE HISTORY —** *development plan lead sheets from each update indicating which sections were updated.*

Analogamente esiste un template simile pubblicato come IEEE Standard for Software Project Management Plans (IEEE Std. 1058-1998)