Gestione di progetti software (Software Project Management)

- Lo sviluppo di un prodotto software è una operazione complessa che richiede una specifica attività di gestione
- La gestione di un progetto software implica la pianificazione, il monitoraggio ed 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

Le quattro "P"

La gestione efficace di un progetto software si fonda sulle "quattro P":

- **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

Organizzazione dei team

- ***Problema***: sviluppare un prodotto software in 3 mesi con un impegno pianificato di 1 anno/uomo
- ***Soluzione immediata***: 4 sviluppatori che si suddividono il lavoro
- ***Realtà***: i 4 sviluppatori potrebbero impiegare un anno ottenendo un prodotto di qualità inferiore a quello risultante dal lavoro di un singolo sviluppatore
- ***Motivi***:
  - alcuni compiti non possono essere condivisi
  - necessità di frequenti interazioni
- L'aggiunta di uno sviluppatore potrebbe ritardare ulteriormente il progetto, a causa del periodo di formazione e dell'aumento delle interazioni (*legge di Brooks*, 1975)

Approccio democratico

- Introdotto da Weinberg (1971) e basato sul concetto di *egoless programming*, secondo cui ogni sviluppatore valuta la scoperta di fault nel proprio codice come un attacco alla propria persona e non come un evento usuale
- L'approccio democratico 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

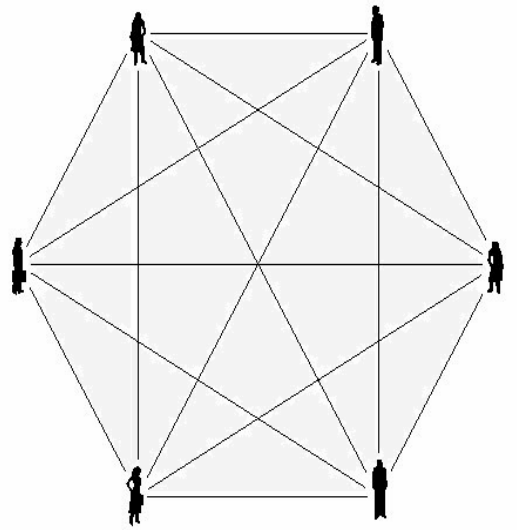
Approccio democratico
vantaggi e svantaggi

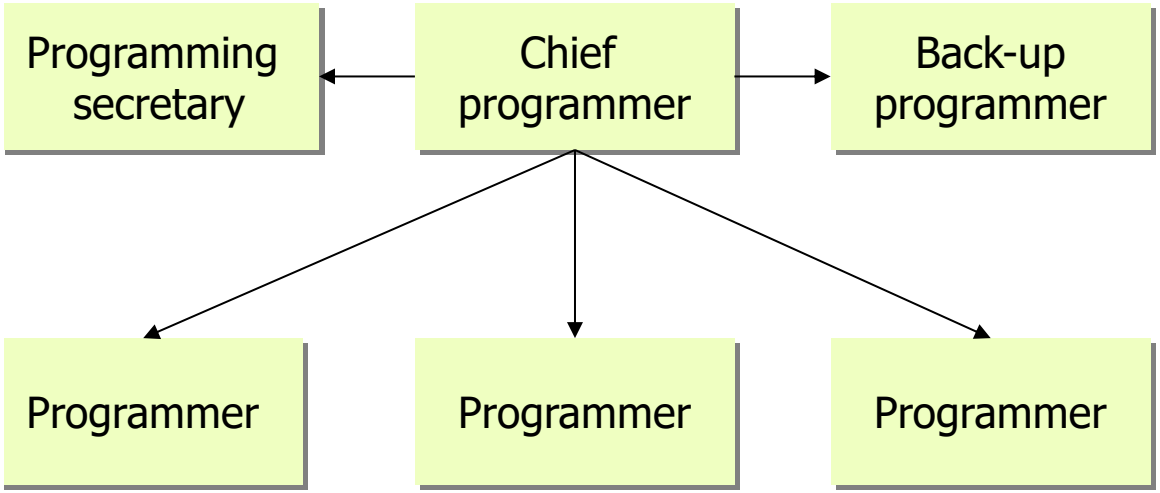
- *Vantaggi*:
  - atteggiamento positivo verso la ricerca di fault
  - molto produttivo in caso di problemi difficili da risolvere (es. ambienti di ricerca)
- *Svantaggi*:
  - l'approccio non può essere imposto dall'esterno ma deve nascere spontaneamente
  - gli sviluppatori più anziani non desiderano essere valutati dai più giovani

Approccio con capo-programmatore

Basato su:

- **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



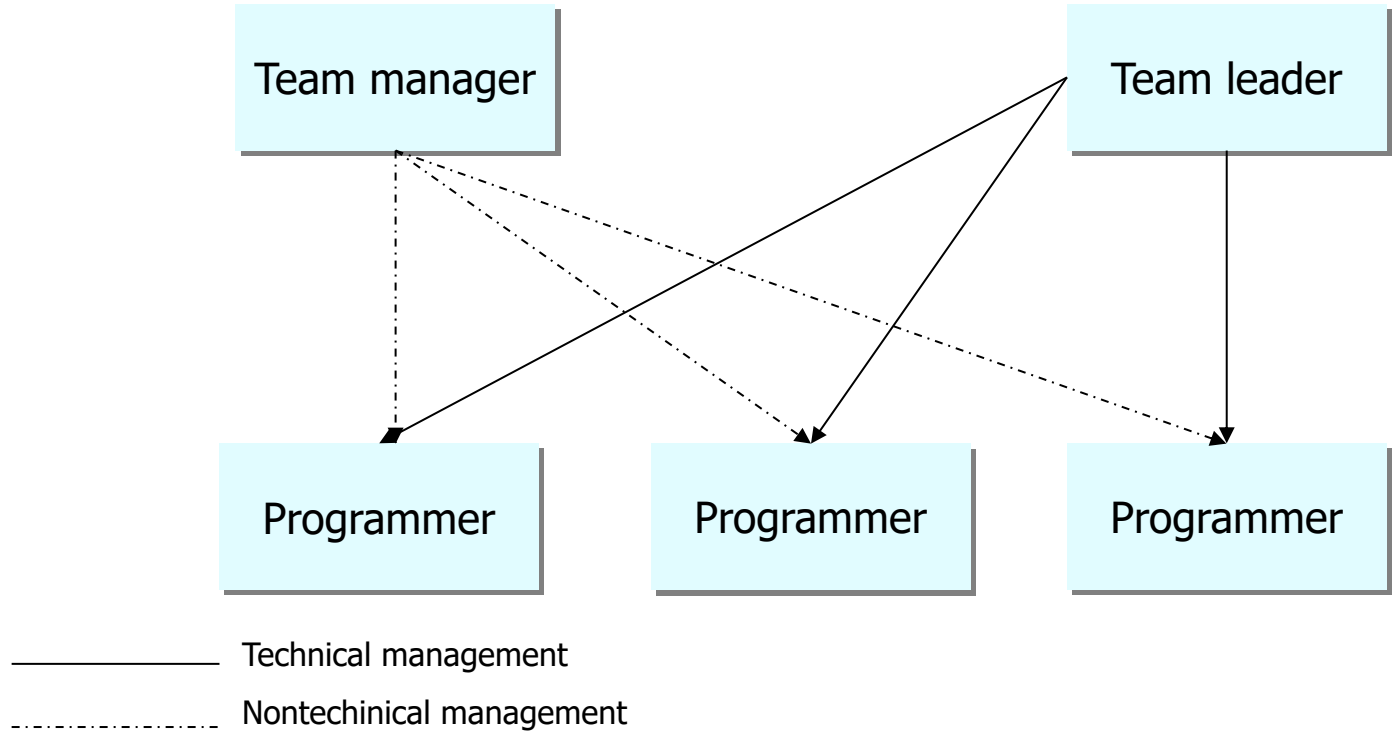


Approccio con capo-programmatore
vantaggi e svantaggi

- Un *vantaggio* immediato è che diminuisce il numero di canali di comunicazione
- Lo *svantaggio* principale 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)

Evoluzione degli approcci

- 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)



Evoluzione degli approcci

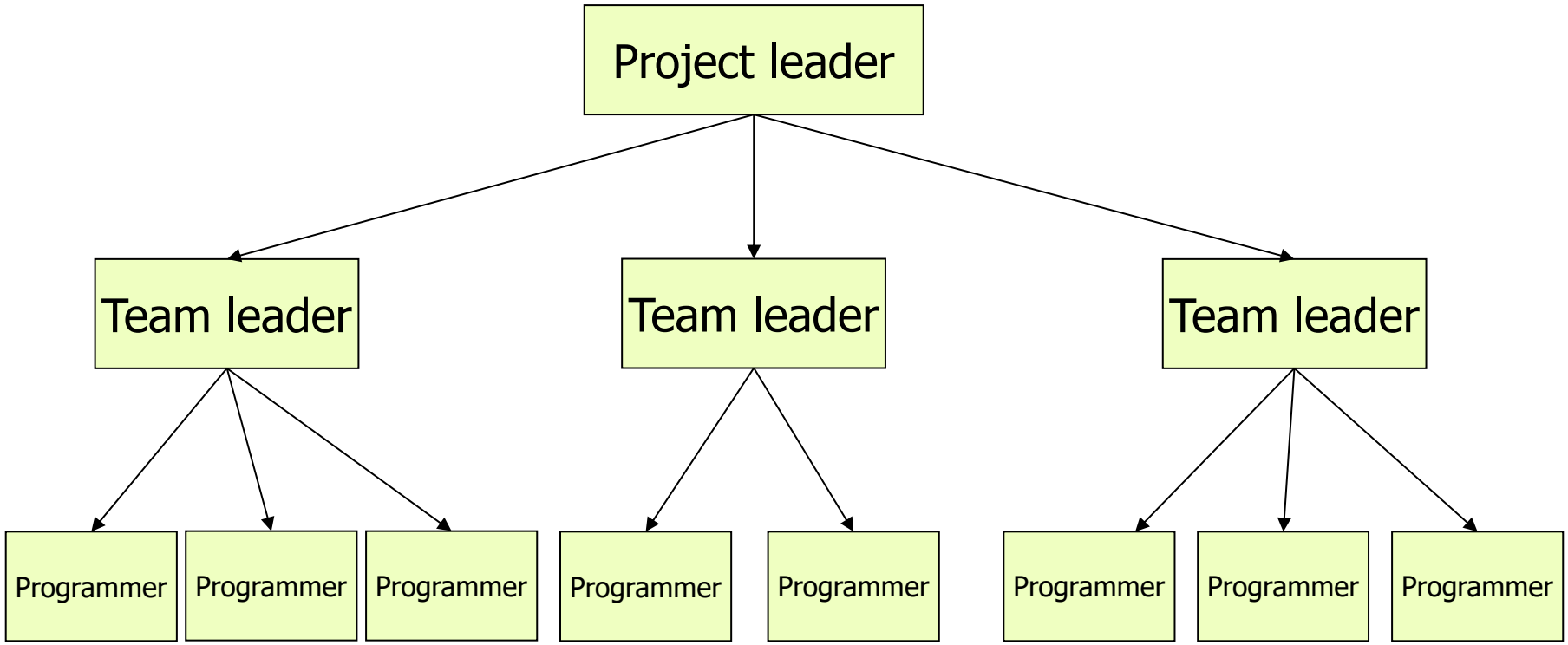
- Problema

  il team manager concede le ferie al programmatore senza contattare il team leader che invece è di parere contrario

- Soluzioni
  - identificare aree di responsabilità condivise
  - introdurre un ulteriore livello di gestione, con un *project leader* che è responsabile dell'intero progetto e che comunica con i team leader

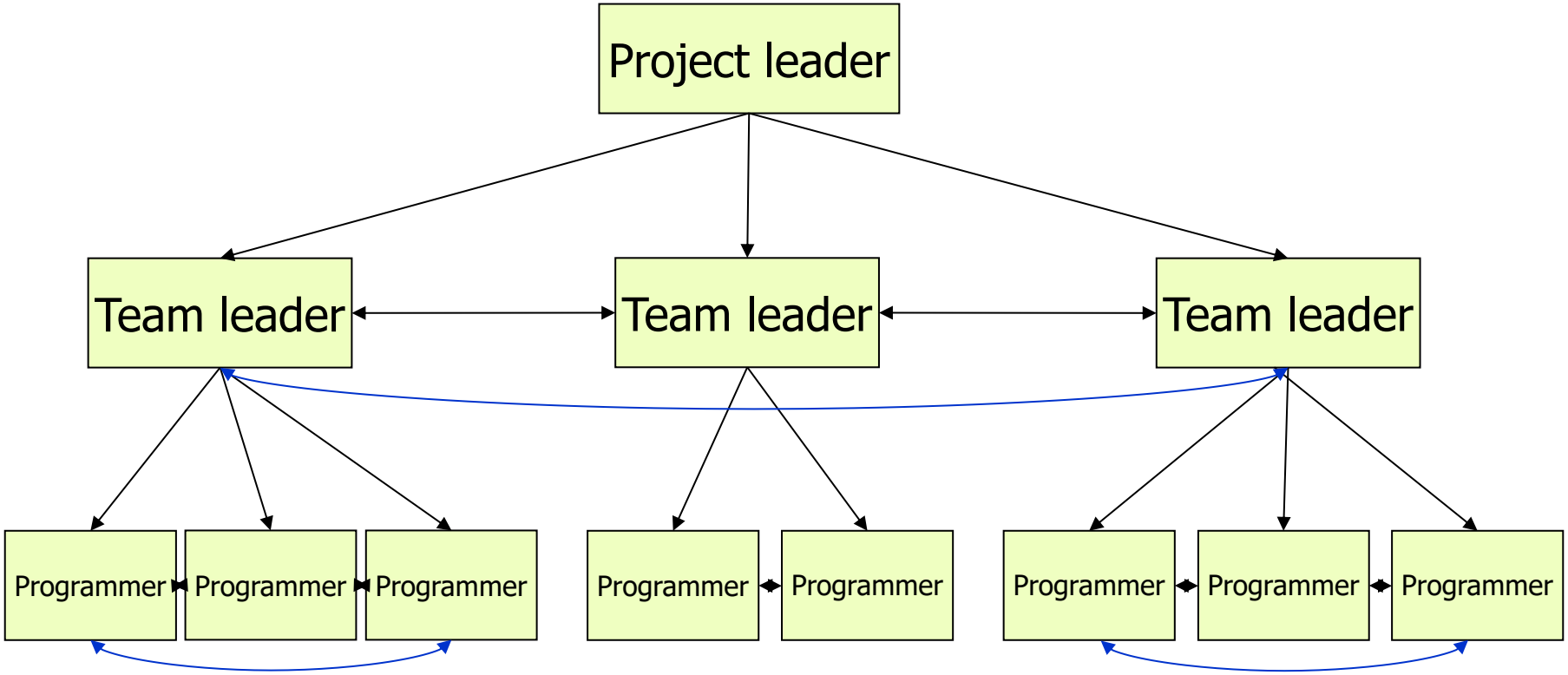
Evoluzione degli approcci

- Organizzazione (scalabile) dei team per *progetti SW di dimensione medio-grande*



Evoluzione degli approcci

- Organizzazione (scalabile) dei team per *progetti SW di dimensione medio-grande*, con decision-making decentralizzato



Importanza del Software Project Mgmt

- Percentuale di progetti on-time, ritardati o cancellati del tutto a causa di costi/ritardi eccessivi o scarsa qualità
- Statistica elaborata su progetti USA dal 1984 al 2016

	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%**

Pianificazione di progetti software

- Obiettivo:

  definire un quadro di riferimento per controllare, determinare l'avanzamento ed osservare lo sviluppo di un progetto software

- Motivazione:

  essere in grado di sviluppare prodotti software nei tempi e costi stabiliti, con le desiderate caratteristiche di qualità

- Componenti fondamentali:
  - Scoping (raggio d'azione): comprendere il problema ed il lavoro che deve essere svolto
  - Stime: prevedere tempi, costi e effort
  - 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

Stime nei progetti software

- 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:
  1. stime su progetti simili già completati (*expert judgement by analogy*)
  2. "tecniche di scomposizione" (approccio *bottom-up*)
  3. modelli algoritmici empirici

Stime nei progetti software (2)

- Le **tecniche di scomposizione** 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
- I **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 i v_i sono le variabili indipendenti (es. LOC o FP stimati)

Esempio di stima di LOC

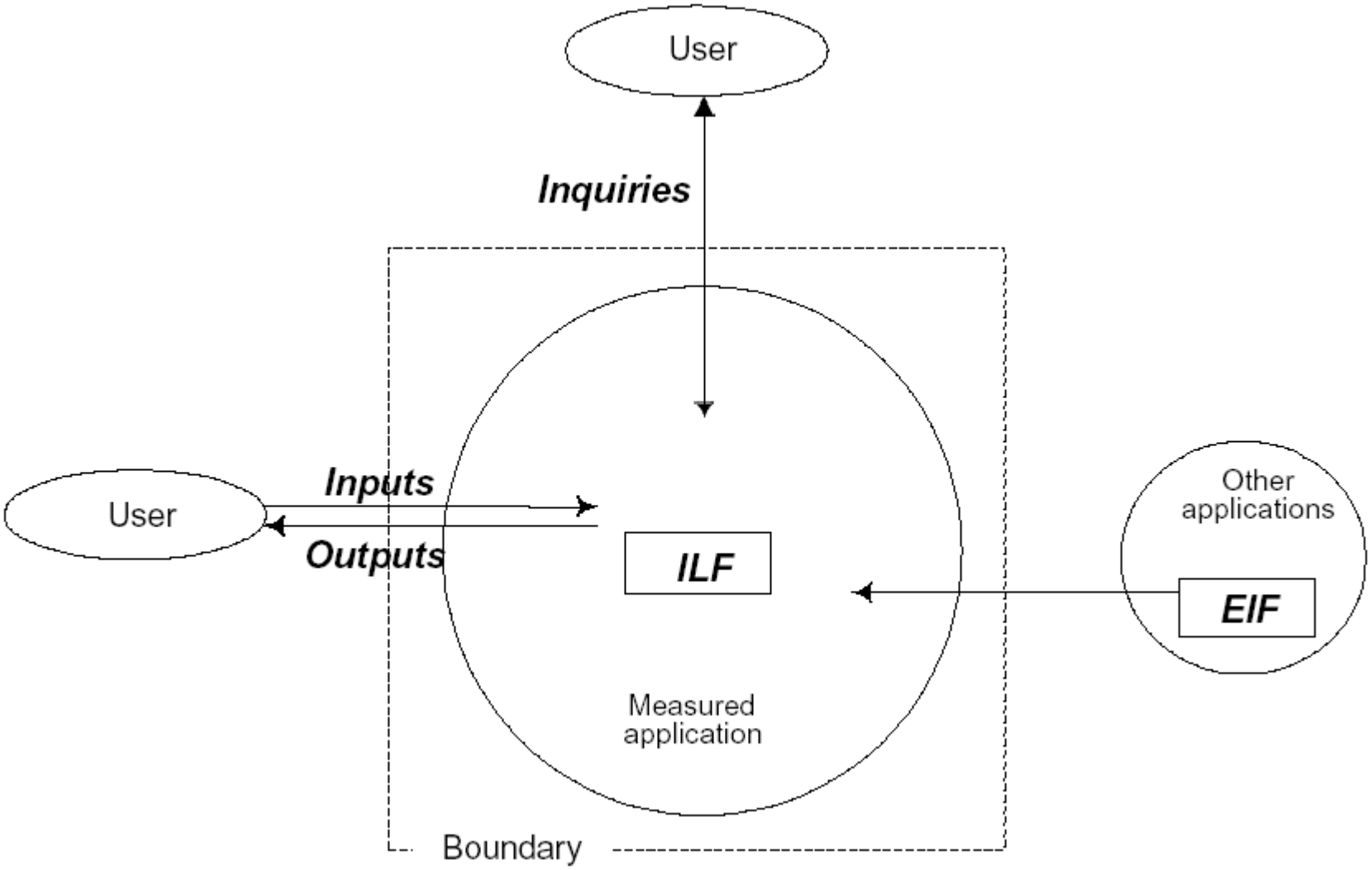
Functions	estimated LOC	LOC/pm	$/LOC	Cost	Effort (MM)
UICF	2340	315	14	32,000	7.4
2DGA	5380	220	20	107,000	24.4
3DGA	6800	220	20	136,000	30.9
DBM	3350	240	18	60,000	13.9
CGDF	4950	200	22	109,000	24.7
PCF	2140	140	28	60,000	15.2
DAM	8400	300	18	151,000	28.0
Totals	33,360			655,000	144.5

Function Point – FP

- Function Point (FP) is a weighted measure of software functionality proposed by Albrecht (1979~1983).
- Function points measure the amount of functionality in a system based upon the system specification (*estimation before implementation*)
- FP is computed in two steps:
  1. Calculating an ***Unadjusted Function point Count*** (***UFC***).
  2. Multiplying the UFC by a ***Technical Complexity Factor*** (***TCF***).
- The final (adjusted) Function Point is:

$$FP = UFC \times TCF$$

FP components



FP count – data categories

- Counts for *data categories*:
  - ***Number of Internal Logical Files* (*ILF*)**

    A group of data or control information that is generated, used or maintained by the software system.
  - ***Number of External Interface Files* (*EIF*)**

    A group of data or control information passed or shared between applications, i.e., machine-readable interfaces to other systems and/or the user.
- The term “file” does not mean file in the traditional data processing sense. If refers to a logically related group of data and not the physical implementation of those groups of data

FP count – transaction categories

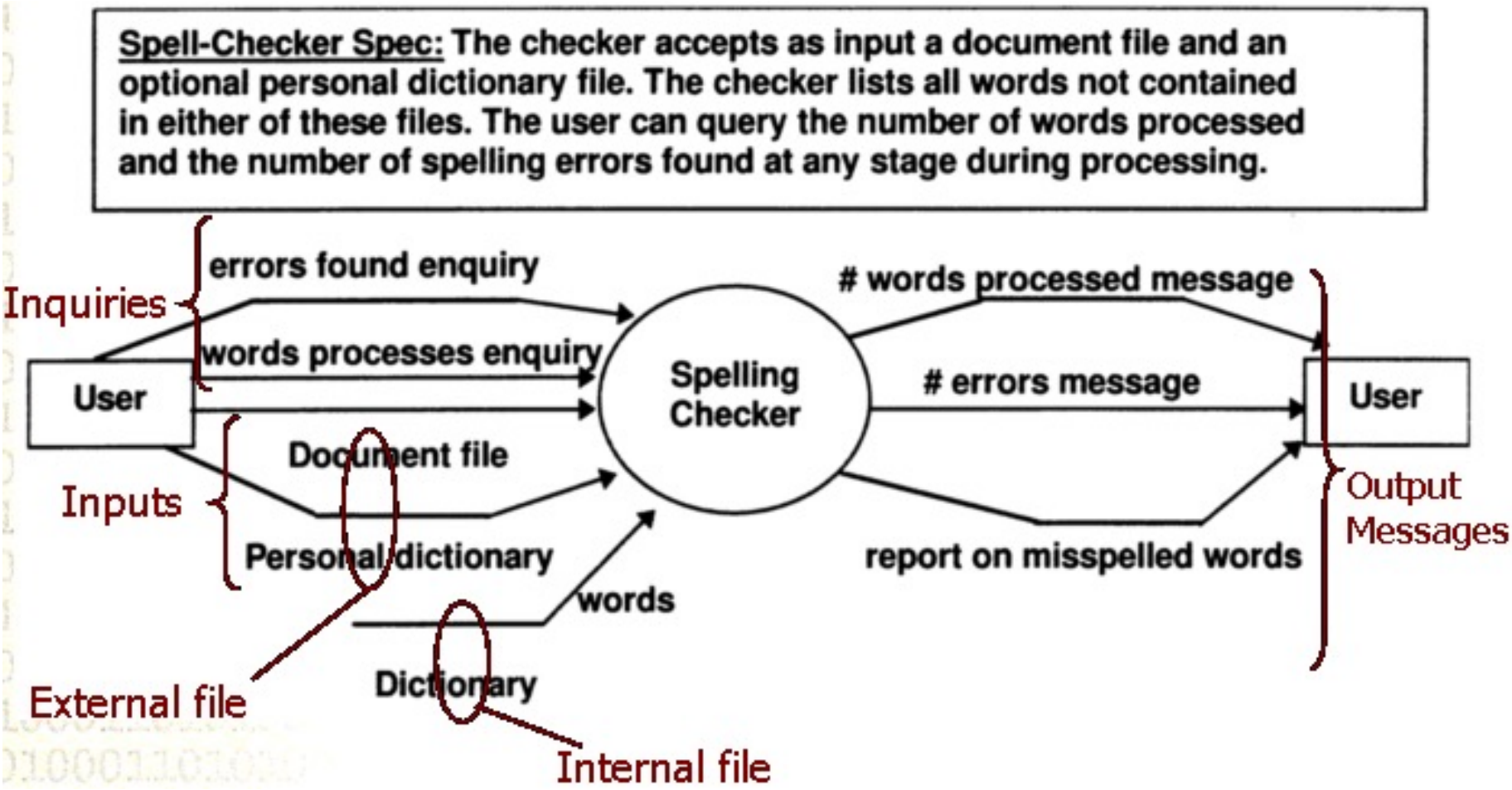
- Counts for *transaction categories*:
  - ***Number of External Inputs*** **(*EI*)**

    Those items provided by the user that describe distinct application-oriented data, control information (such as file names and menu selections), or outputs of other systems that enter an application and change the status of its internal logical file(s).
  - ***Number of External Outputs*** **(*EO*)**

    All unique data or control information produced by the software systems, e.g., reports and messages.
  - ***Number of External Inquiries*** **(*EQ*)**

    All unique input/output combinations, where an input causes and generates an immediate output without changing any status of internal logical files.

Example – Spell Checker



Example – *FP components*

- *EI=2* (external inputs): document filename, personal dictionary-name
- *EO=3* (external outputs): misspelled word report, number-of-words-processed message, number-of-errors-so-far message
- *EQ=2* (external inquiries): words processed, errors so far
- *EIF=2* (external interface files): document file, personal dictionary
- *ILF=1* (internal logical files): dictionary

Example - *UFC*

- Assume average complexity weights for each element
- UFC = 55

Function Type		Functional Complexity		Complexity Totals	Function Type Totals
ILFs		Low	X 7 =		
	1	Average	X 10 =	10	
		High	X 15 =		
					10
EIFs		Low	X 5 =		
	2	Average	X 7 =	14	
		High	X 10 =		
					14
EIs		Low	X 3 =		
	2	Average	X 4 =	8	
		High	X 6 =		
					8
EOs		Low	X 4 =		
	3	Average	X 5 =	15	
		High	X 7 =		
					15
EQs		Low	X 3 =		
	2	Average	X 4 =	8	
		High	X 6 =		
					8
		Total Unadjusted Function Point Count			55

Fattori di degree of influence

1. ***Reliable back-up and recovery***
2. ***Data communications***
3. ***Distributed data processing***
4. ***Performance***
5. ***Heavily used configuration***
6. ***Online data entry***
7. ***Operational ease***
8. ***Online update***
9. ***Complex interface***
10. ***Complex processing***
11. ***Reusability***
12. ***Installation ease***
13. ***Multiple sites***
14. ***Facilitate change***

Ad ogni fattore viene

associato un valore intero

compreso tra

- **0** (influenza *irrilevante*) e
- **5** (influenza *essenziale*)

TCF calculation

- Each component is rated from 0 to 5, where:
  - 0 Not present, or no influence
  - 1 Incidental influence
  - 2 Moderate influence
  - 3 Average influence
  - 4 Significant influence
  - 5 Strong influence throughout
- The TCF can then be calculated as:

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

- The TCF varies from 0.65 (if all F_j are set to 0) to 1.35 (if all F_j are set to 5) ➔ *± 35% adjustment*

Example – *TCF and FP*

- Suppose that:

$F_1 = 3$	Reliable back-up and recovery	$F_7 = 3$	Operational ease
$F_2 = 3$	Data communications	$F_8 = 3$	Online update
$F_3 = 0$	Distributed data processing	$F_9 = 0$	Complex interface
$F_4 = 5$	Performance	$F_{10} = 5$	Complex processing
$F_5 = 0$	Heavily used configuration	$F_{11} = 0$	Reusability
$F_6 = 3$	Online data entry	$F_{12} = 0$	Installation ease
		$F_{13} = 0$	Multiple sites
		$F_{14} = 3$	Facilitate change

- then

$$TCF = 0.65 + 0.01(18+10) = 0.93$$

- and

$$FP = 55 \times 0.93 \approx 51$$

Riferimento per conteggio FP

Function Point Counting Practices Manual

Release 4.3.1

FP vs. LOC

- A number of studies have attempted to relate LOC and FP metrics (Jones' *backfiring*, 1996).
- The average number of source code statements per function point has been derived from case studies for numerous programming languages.
- Languages have been classified into different levels according to the relationship between LOC and FP.

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

Es. di modello algoritmico: **COCOMO**

- **COCOMO** (**CO**nstructive **CO**st **MO**del) è il modello introdotto da Boehm (1981) per determinare il valore dell'effort
- 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 sottosistema)
- La stima dell'effort viene effettuata a partire da:
  - stima delle dimensioni del progetto in **KLOC**
  - 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)
- Nel 1995 è stato introdotto COCOMO II, più flessibile e sofisticato rispetto alla versione precedente

Esempio d'uso di COCOMO

Modello *Intermediate*, modo *organic*

- ***Passo 1***

  Determinare l'**effort nominale** usando la formula:

  $$\textit{effort nominale} = 3.2 \times (\textit{KLOC})^{1.05}\ \textit{MM}$$

  Esempio:

  $$3.2 \times (33)^{1.05} = 126\ \text{MM}$$

- ***Passo 2***

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

  $$\textit{effort} = \textit{effort nominale} \times C$$

  Esempio:

  $$126 \times 1.15 = 145\ \text{MM}$$

- *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)

Tabella di cost driver (Intermediate COCOMO)

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.

Cost drivers ratings

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[a]	35th percentile	55th percentile	75th percentile	90th percentile	
AEXP	≤4 months experience	1 year	3 years	6 years	12 years	
PCAP	15th percentile[a]	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%	

[a] Team rating criteria: analysis (programming) ability, efficiency, ability to communicate and cooperate

Complexity (CPLX) ratings

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

Example cost drivers ratings

Microprocessor-based communications processing software

Cost Driver	Situation	Rating	Effort Multiplier
RELY	Serious financial consequences of software faults	High	1.15
DATA	20,000 bytes	Low	0.94
CPLX	Communications processing	Very High	1.30
TIME	Will use 70% of available time	High	1.11
STOR	45K of 64K store (70%)	High	1.06
VIRT	Based on commercial microprocessor hardware	Nominal	1.00
TURN	Two-hour average turnaround time	Nominal	1.00
ACAP	Good senior analysts	High	0.86
AEXP	Three years	Nominal	1.00
PCAP	Good senior programmers	High	0.86
VEXP	Six months	Low	1.10
LEXP	Twelve months	Nominal	1.00
MODP	Most techniques in use over one year	High	0.91
TOOL	At basic minicomputer tool level	Low	1.10
SCED	Nine months	Nominal	1.00
	Effort adjustment factor (product of effort multipliers)		1.35

COCOMO
Time Schedule

- 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* $T = 2.5\ E^{0.32}$

Development Costs Estimation

- Development costs (C) are estimated by allocating development effort (E) on phases and staff activities, e.g.:
  - 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
- The *cost per person-month* of each staff category (e.g., project manager, analyst, programmer, etc.) is then used to obtain development costs

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

Strumenti di pianificazione

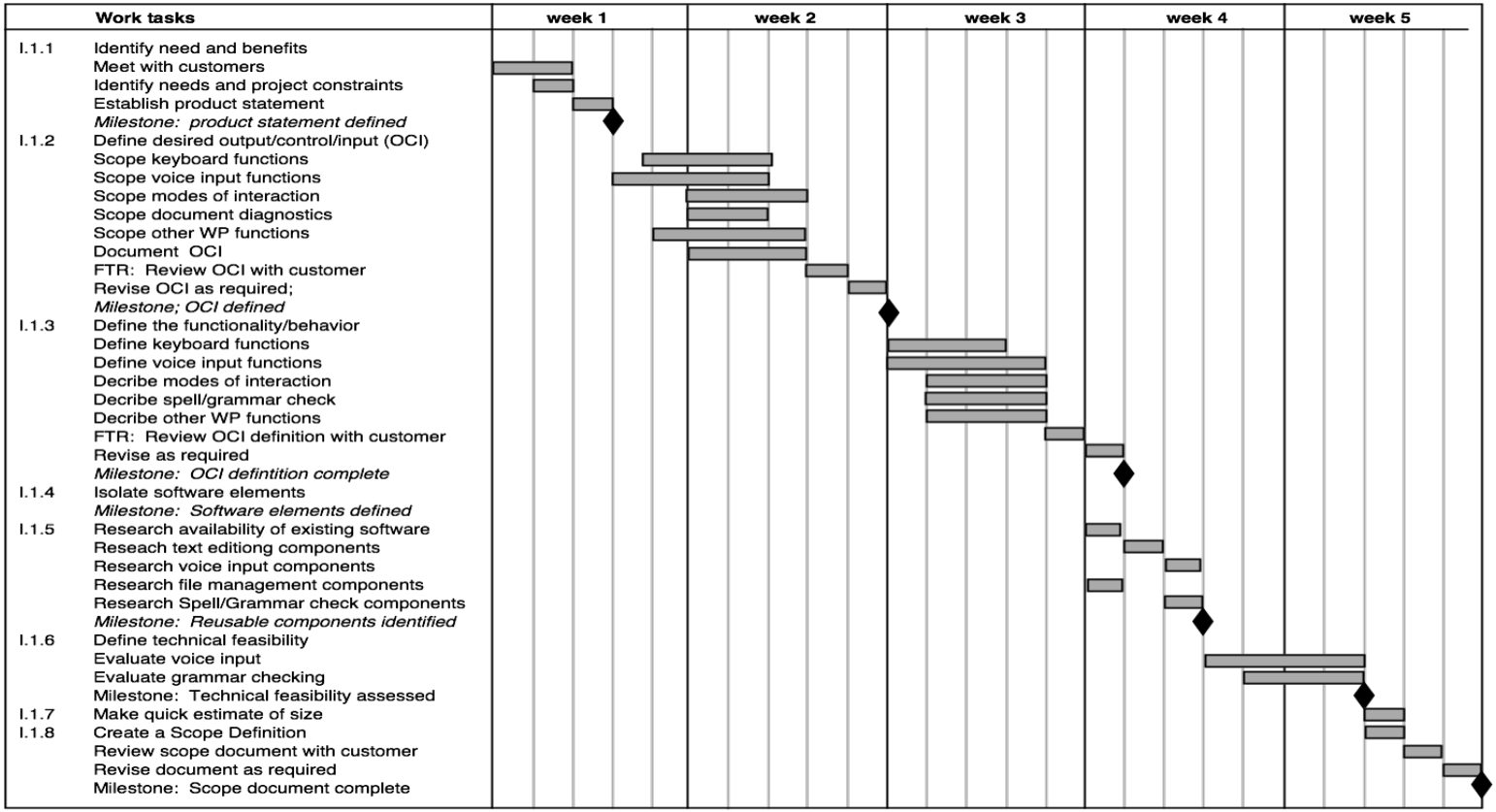
Diagramma PERT (Program Evaluation and Review Technique)

- *grafo* in cui ogni *nodo* rappresenta un ***task*** ed ogni *arco* un ***legame di precedenza***
- 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

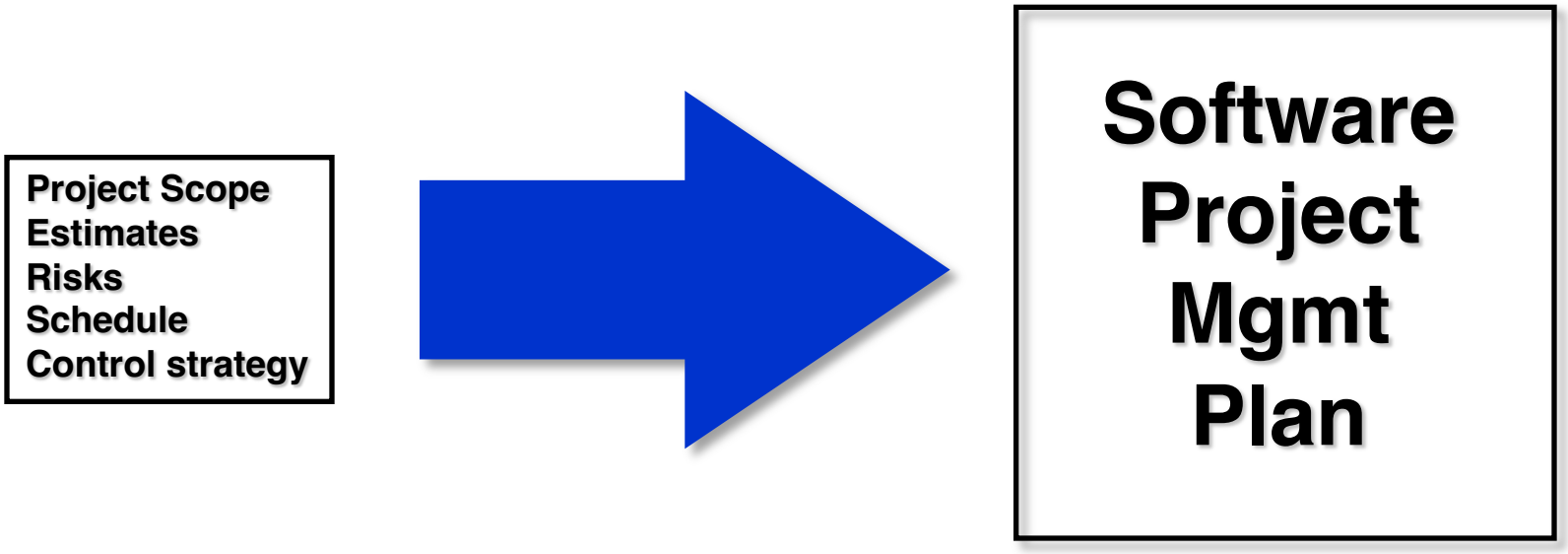
Strumenti di pianificazione (2)

Carta di Gantt

- diagramma a barre che consente di visualizzare l'allocazione temporale dei task
- non appare nessuna indicazione dei legami di precedenza, quindi viene integrata con un diagramma PERT



Software Project Management Plan (SPMP)



Contenuti del SPMP

Manager's Handbook for Software Development
(NASA-SEL-84-101, 1990)
pag 1/2

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.*

Contenuti del SPMP

Manager's Handbook for Software Development (NASA-SEL-84-101, 1990) *pag 2/2*

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.*

Contenuti del SPMP

IEEE Standard for Software Project Management Plans

(IEEE Std. 1058-1998)

pag. 1/3

Title page
Signature page
Change history
Preface
Table of contents
List of figures
List of tables

1. Overview
   - 1.1 Project summary
     - 1.1.1 Purpose, scope and objectives
     - 1.1.2 Assumptions and constraints
     - 1.1.3 Project deliverables
     - 1.1.4 Schedule and budget summary
   - 1.2 Evolution of the plan
2. References
3. Definitions
4. Project organization
   - 4.1 External interfaces
   - 4.2 Internal structure
   - 4.3 Roles and responsibilities

Contenuti del SPMP

IEEE Standard for Software Project Management Plans

(IEEE Std. 1058-1998)

pag. 2/3

- 5. Managerial process plans
  - 5.1 Start-up plan
    - 5.1.1 Estimation plan
    - 5.1.2 Staffing plan
    - 5.1.3 Resource acquisition plan
    - 5.1.4 Project staff training plan
  - 5.2 Work plan
    - 5.2.1 Work activities
    - 5.2.2 Schedule allocation
    - 5.2.3 Resource allocation
    - 5.2.4 Budget allocations
  - 5.3 Control plan
    - 5.3.1 Requirements control plan
    - 5.3.2 Schedule control plan
    - 5.3.3 Budget control plan
    - 5.3.4 Quality control plan
    - 5.3.5 Reporting plan
    - 5.3.6 Metrics collection plan
  - 5.4 Risk management plan
  - 5.5 Closeout plan

Contenuti del SPMP

IEEE Standard for Software Project Management Plans

(IEEE Std. 1058-1998)

pag. 3/3

6. Technical process plans
   - 6.1 Process model
   - 6.2 Methods, tools and techniques
   - 6.3 Infrastructure plan
   - 6.4 Product acceptance plan
7. Supporting process plans
   - 7.1 Configuration management plan
   - 7.2 Verification and validation plan
   - 7.3 Documentation plan
   - 7.4 Quality assurance plan
   - 7.5 Reviews and audits
   - 7.6 Problem resolution plan
   - 7.7 Subcontractor management plan
   - 7.8 Process improvement plan
8. Additional plans

Annexes

Index