

# Project Management dei Progetti Software



*Corso di Ingegneria del Software*  
*CdL Informatica*  
*Università di Bologna*

# Obiettivi della lezione

- Cos'è il Project Management
- Tecniche per la stima dei costi sw
- Misurare le dimensioni di un progetto sw
  - Linee di codice
  - Function Point
- Stima Algoritmica
  - COCOMO

# La legge di Brooks

*Aggiungere personale ad un progetto sw  
in ritardo lo farà ritardare ancora di più*

F. Brooks, *The mythical Man-Month*, 1975

[en.wikipedia.org/wiki/The\\_Mythical\\_Man-Month](https://en.wikipedia.org/wiki/The_Mythical_Man-Month)

# Discussione

- Cos'è un progetto? come si gestisce?



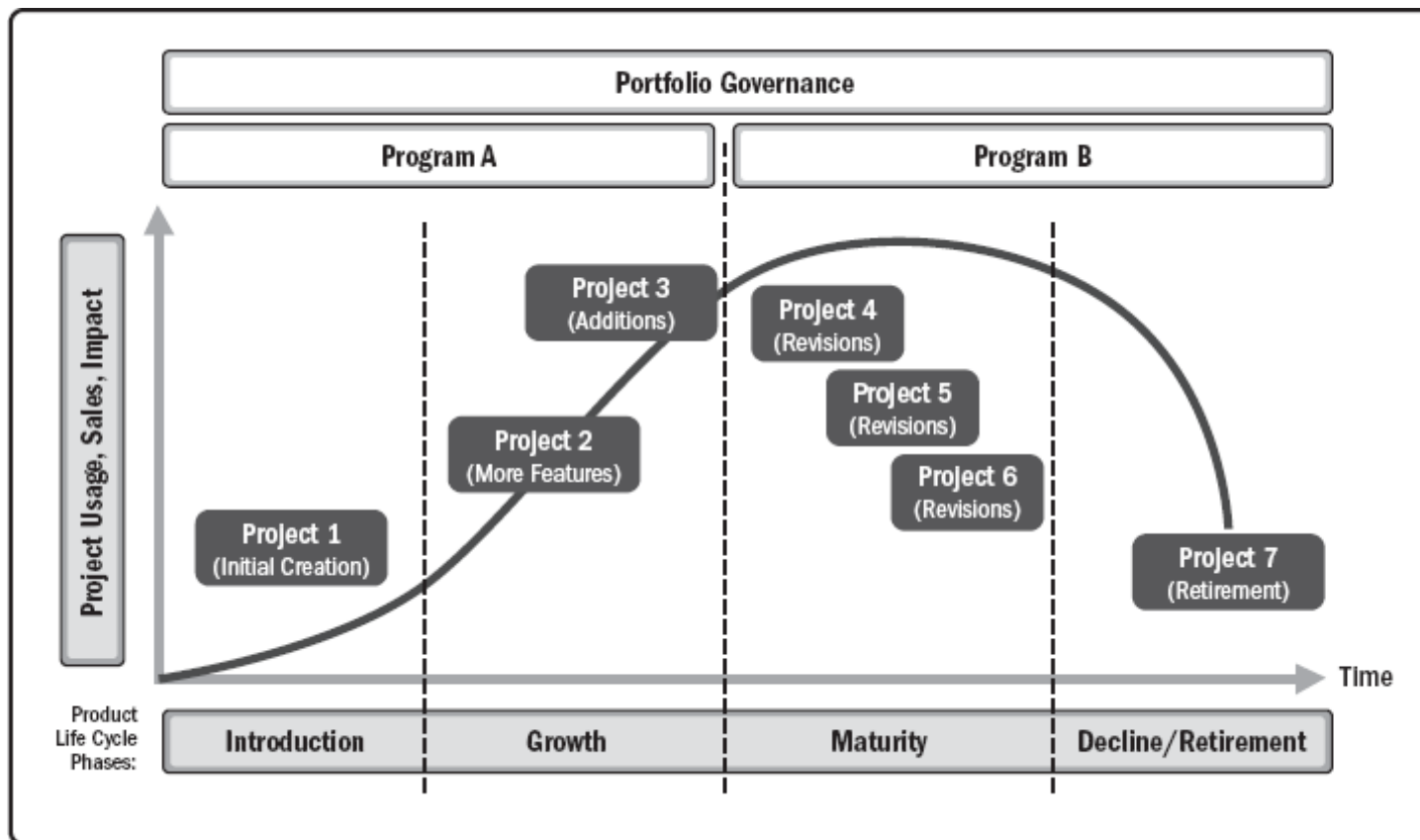


# Alcune definizioni (fonte: PMI)

- **Progetto**: impresa temporanea intrapresa per raggiungere uno scopo
- **Project Management**: applicazione di conoscenze, tecniche e strumenti alle attività di uno specifico progetto allo scopo di portarlo a termine con successo
- **Product Management** : funzione aziendale di pianificazione, produzione e marketing di uno specifico prodotto che segue un ciclo di vita di prodotto

# Esempio: ciclo di vita di prodotto articolato su più progetti

(fonte: Guida PMI 2021)

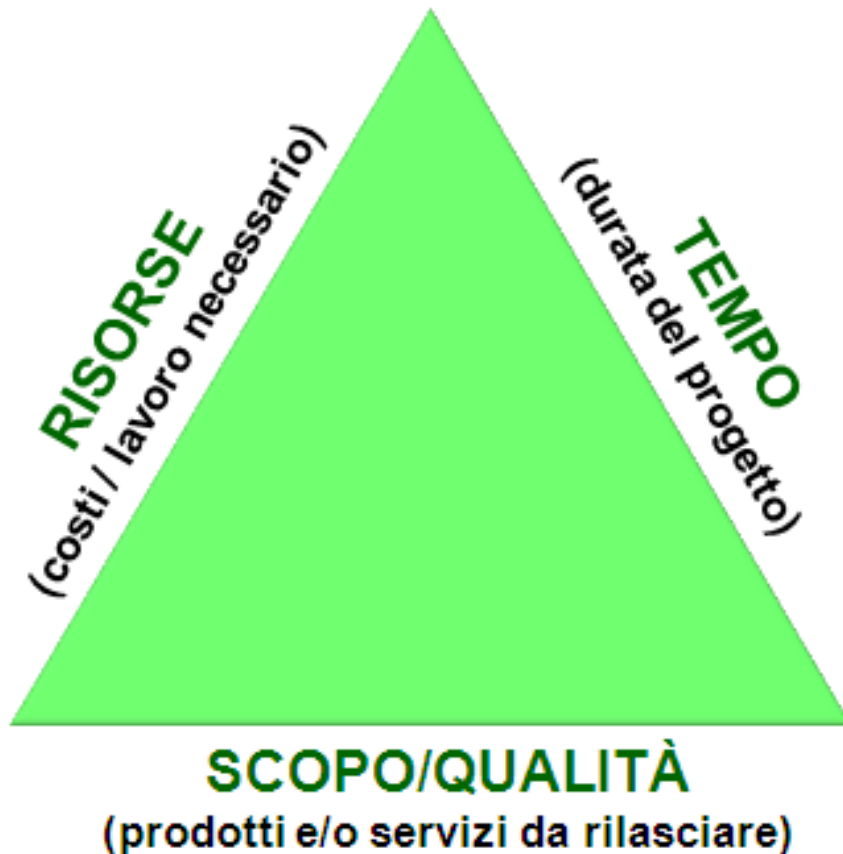


# Variabili che governano un progetto

- **Scopo**: requisiti di utente, specifiche di qualità, elementi da consegnare
- **Costo**: budget e risorse necessarie
- **Tempo**: inizio, fine e calendario delle attività

Un progetto si considera concluso con successo se termina in tempo, rispettando il budget, e consegna ciò che l'utente si aspetta

# Il triangolo del project management



Per ogni progetto i tre vincoli: *scopo*, *risorse*, e *tempo* sono correlati

- Incrementare lo scopo e fissarlo (funzioni+qualità) significa aumentare i tempi e i costi del progetto;
- Ridurre i tempi richiede costi più alti (più risorse) o uno scopo più ristretto;
- Un budget risicato (meno risorse) può implicare tempi più lunghi o una riduzione dello scopo.

Lo scopo di solito è più importante degli altri due vincoli: nei modelli pianificati è la variabile indipendente

Nei **modelli agili** invece le variabili indipendenti sono la durata e i costi, lo scopo e la qualità si contrattano

# Il Project Manager

- Il Project Manager (PM, o responsabile di progetto) controlla la *pianificazione* ed il *budget* di progetto
- Durante tutto il ciclo di vita il PM deve controllare **costi** e **risorse** di un progetto:
  - **Inizialmente** per verificare la fattibilità del progetto
  - **Durante il progetto** per controllare che le risorse non vengano sprecate ed i budget vengano rispettati
  - **Alla fine** per confrontare preventivo e consuntivo

# Compiti del PM

1. Comprendere l'obiettivo del progetto
2. Identificare il processo di sviluppo
3. Determinare la struttura del team di progetto
4. Identificare il processo manageriale
5. Sviluppare un calendario di attività del team
6. Pianificare l'acquisizione di persone nel team
7. Iniziare la gestione dei rischi
8. Identificare i documenti da produrre
9. Iniziare il processo...

# I team agili non hanno un PM

- Perché i team agili NON hanno un Project Manager?
- Nella filosofia Agile, ruolo e responsabilità tradizionalmente attribuiti al “**project manager**” sono distribuiti su più ruoli
- Esempio: nel modello agile Scrum i ruoli che hanno le responsabilità di PM sono **Product Owner** e **Team di sviluppo** (lo Scrum Master è un facilitatore, owner di processo ma non responsabile della gestione)

# Project Management Body of Knowledge

- Il manuale PMBOK è una pubblicazione del PMI che descrive le nozioni e i metodi necessari ai PM “nella maggior parte dei progetti”
- Viene aggiornato periodicamente (edizione più recente: 2021)
- Struttura il Project Management definendo nove aree di competenza
- IEEE 1490-2011 è “Guide adoption of PMI standard to the project management BoK”



# La gestione del tempo di progetto

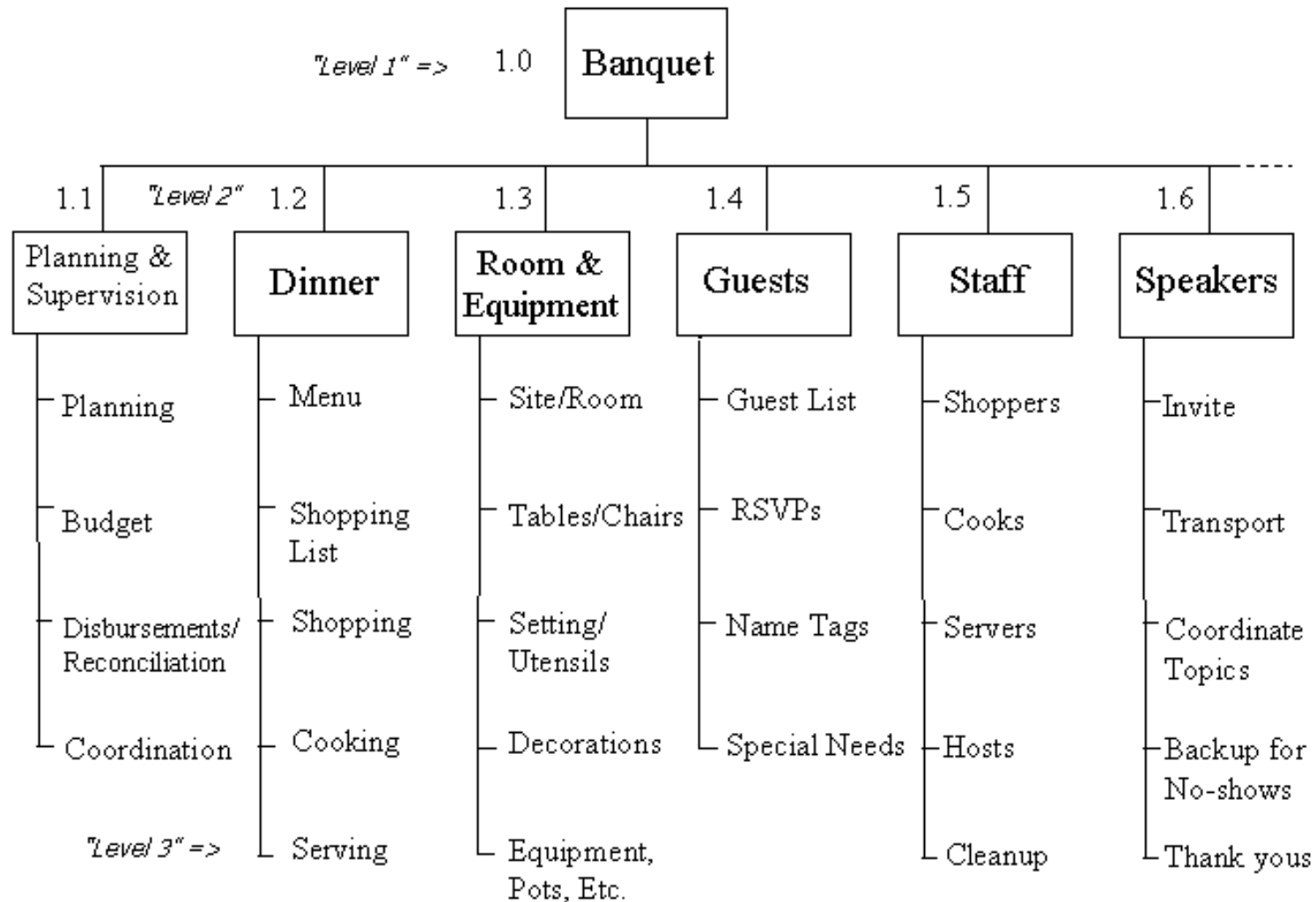
- Definizione delle attività
- Sequenziamento delle attività
- Stima delle risorse per le attività
- Stima della durata delle attività
- Schedulazione delle attività
- Controllo delle attività

# Terminologia

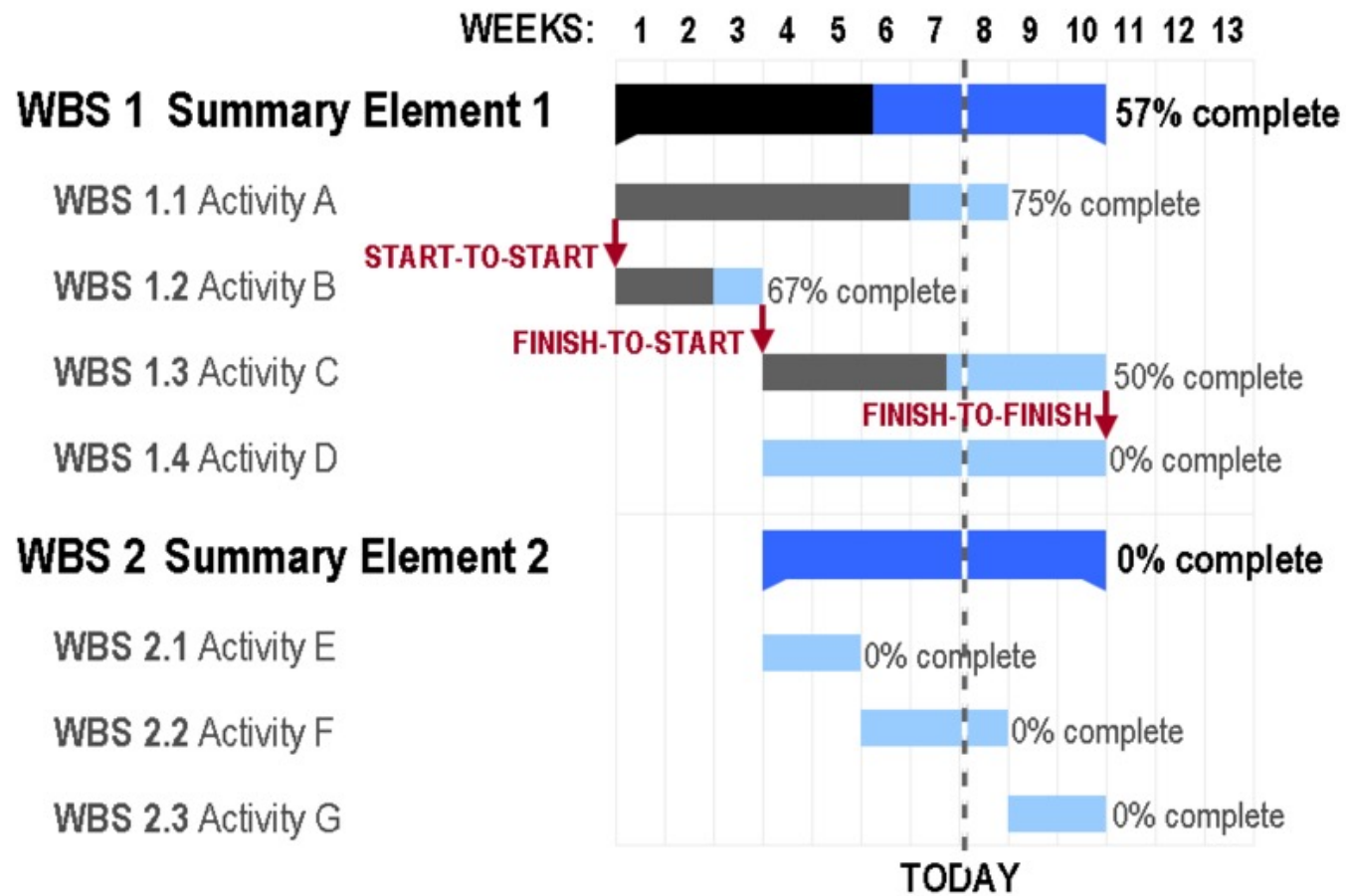
- Work breakdown: articolazione (cioè elenco e relazioni) delle attività di progetto
- Milestone: evento significativo nella vita di un progetto
- Percorso critico: sequenza delle attività la cui durata è incompressibile

# Work Breakdown Structure (WBS)

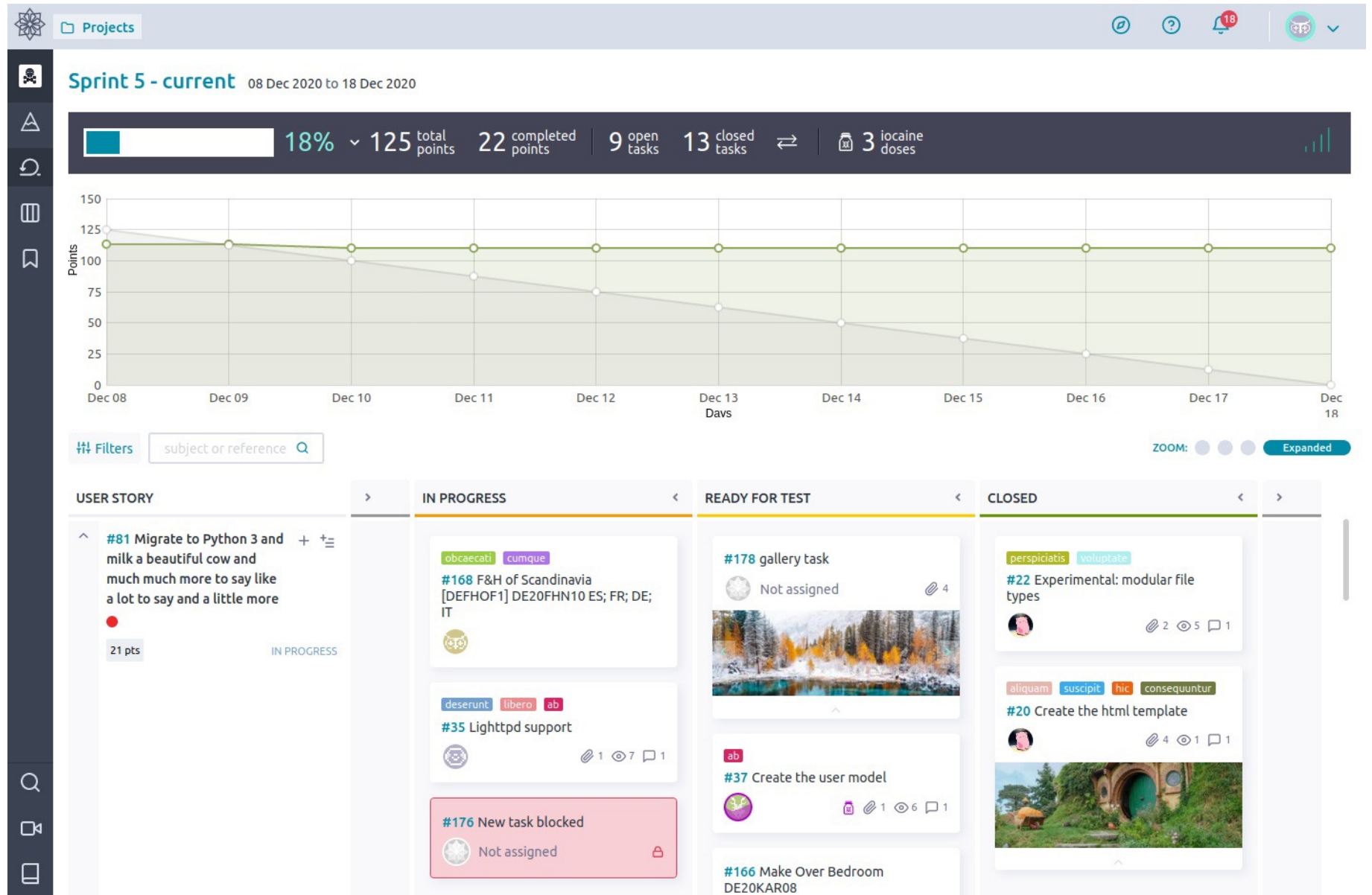
## WBS Example - Banquet



# Grafico Gantt



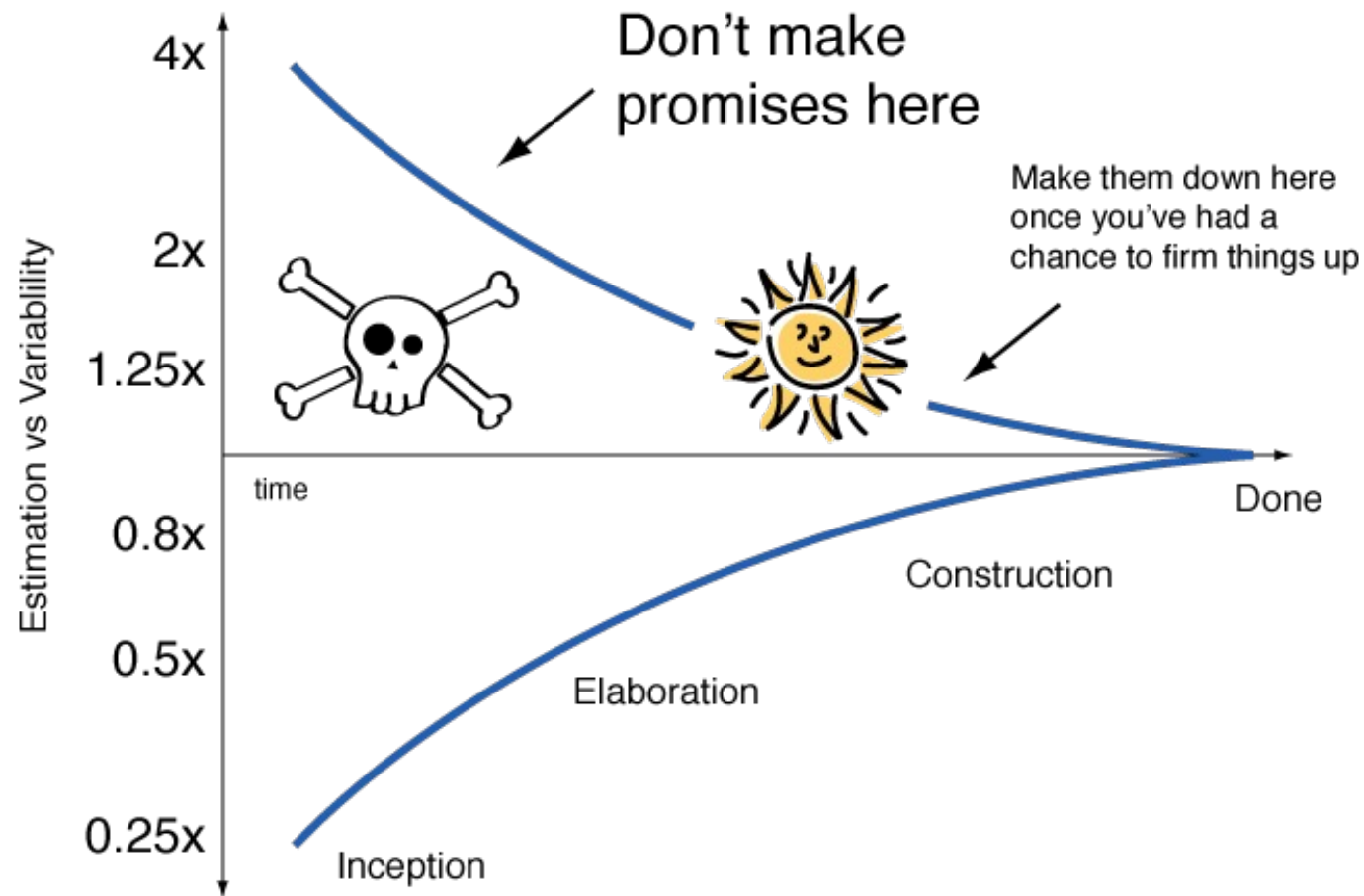
# Taiga



# Stimare e misurare un progetto sw

- La progettazione del software si può analizzare studiando cinque variabili: costo, durata, sforzo, produttività e numero di errori nel prodotto sw [Putnam 1992]
- Il project manager rispetto a tali variabili ha due compiti importanti: stimarle a preventivo, misurarle durante il progetto, e rendicontarle a consuntivo
- Come possiamo stimare costo, durata, sforzo, produttività e numero di errori prima che il progetto inizi?

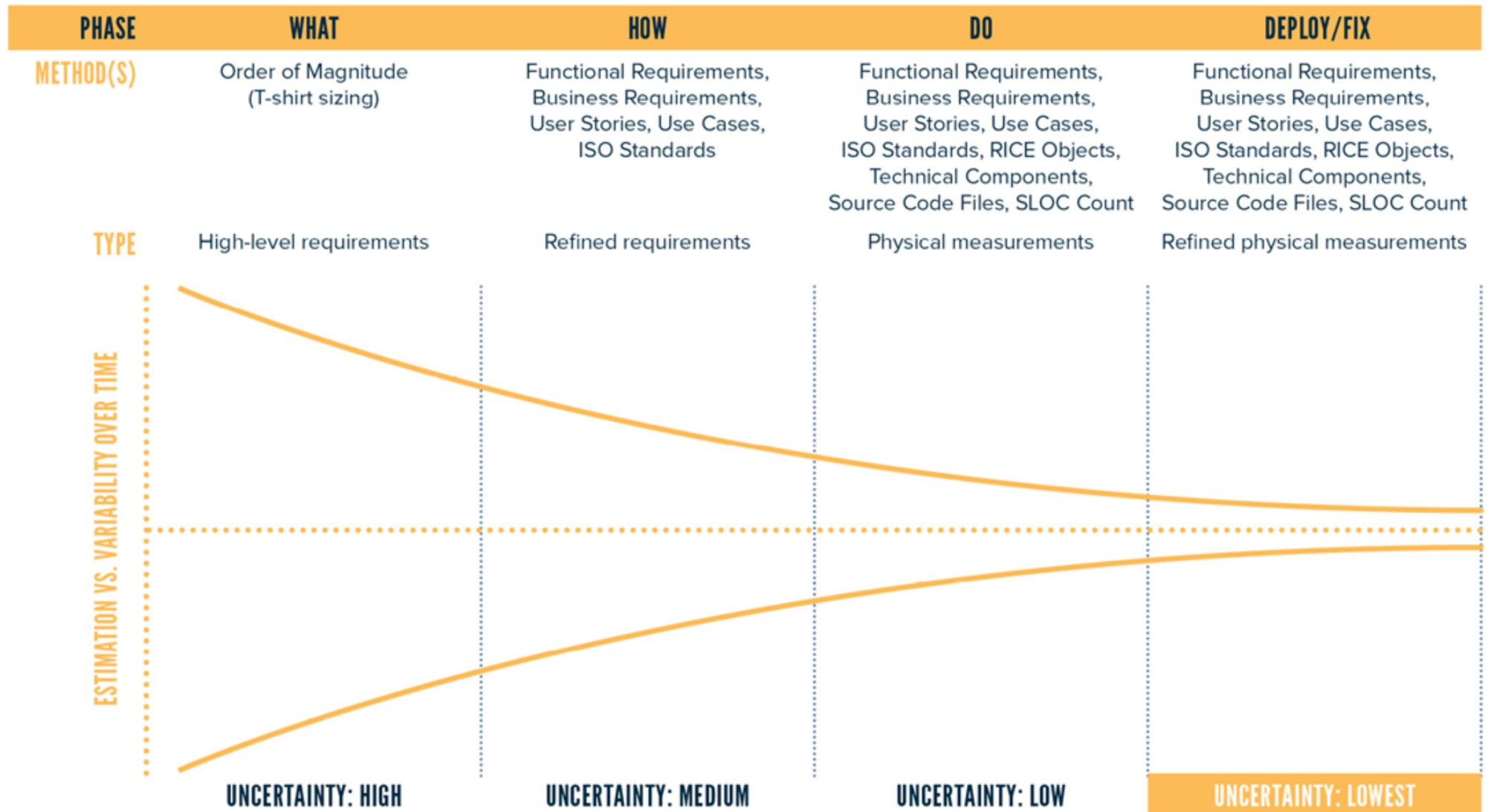
# Il cono dell'incertezza nello sviluppo sw



# L'incertezza nel sizing

fonte:QSM

<https://www.qsm.com/articles/sizing-matters?utm=gcaccess>





# Valore e costo del sw

Il *valore* di un sw è proporzionale al numero di utenti

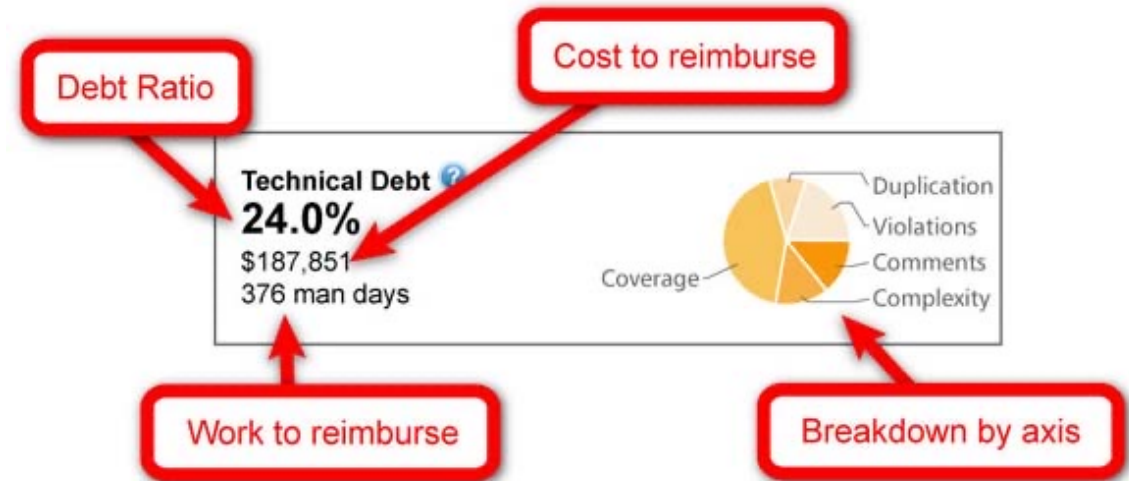
Valore (sw) = #utenti\*ricavo medio per utente

Il *costo* di un software è proporzionale allo sforzo di sviluppo (in mesi/persona), che a sua volta è proporzionale alla dimensione (*size*) del sw

# Produttività

- La **produttività** è un'astrazione economica che si calcola col rapporto output/input
- La **produttività di un team** che ha prodotto un sw di dimensione **size** con un dato effort si calcola  **$size(sw)/effort$**
- Esempio: «*il team ha avuto una produttività di 5K LOC/mese-persona*»

# Il debito tecnico



Il debito tecnico è una stima del costo di futuro sforzo addizionale causato da una soluzione prematura adottata oggi pur di consegnare un prodotto con qualche valore (lo sforzo futuro andrà ripagato con gli interessi)

In un software di buona qualità dovrebbe valere per ogni *versione* la disuguaglianza:

Valore (*versione*) >> Costo (*versione*) + Debito tecnico (*versione*)

# Tecniche di stima dei costi di sviluppo

Le tecniche principali per stimare i costi di un progetto:

- **top-down** (o analogica): uso del costo di un progetto analogo o con componenti di costo noto come base della stima del nuovo progetto
- **bottom-up**: stima dei compiti individuali che compongono il progetto e loro somma per ottenere la stima totale
- **Design-to-cost**: uso di esperti per determinare quanta funzionalità può essere prodotta col budget disponibile
- **parametrica**: stima basata sull'uso di un modello matematico che usa una griglia di parametri

# Cosa va stimato

- **Durata:** estensione temporale del progetto, dall'inizio alla fine; dipende dalle **dipendenze** tra le attività di progetto; si misura di solito in **mesi**
- **Sforzo:** somma dei tempi di tutte le attività di progetto; si misura di solito in **mesi/persona** (mp)
- La **dimensione del team** si ricava dalla relazione sforzo/durata

# Esempio

- Un progetto deve durare un anno, coinvolgere quattro persone di cui due a metà tempo
- La durata è 12 mesi
- Lo sforzo è 36 mesi/persona

# Non confondere durata e sforzo!

## Esempio:

- La *durata* legale di un corso di laurea è tre anni
- Lo *sforzo* è 180 cfu (stabilito per legge)
- Un cfu è pari a 25 h/studente

**Nota 1:** Il regolamento di CdL potrebbe definire durate diverse (es. sei anni), mantenendo però lo sforzo di 180 cfu richiesto dalla legge

**Nota 2:** il cfu (=25h/studente) è una unità di misura dello sforzo (diversa dal mese/persona, che non è definito univocamente, ma che per esempio in COCOMO è pari a 152 h/persona)

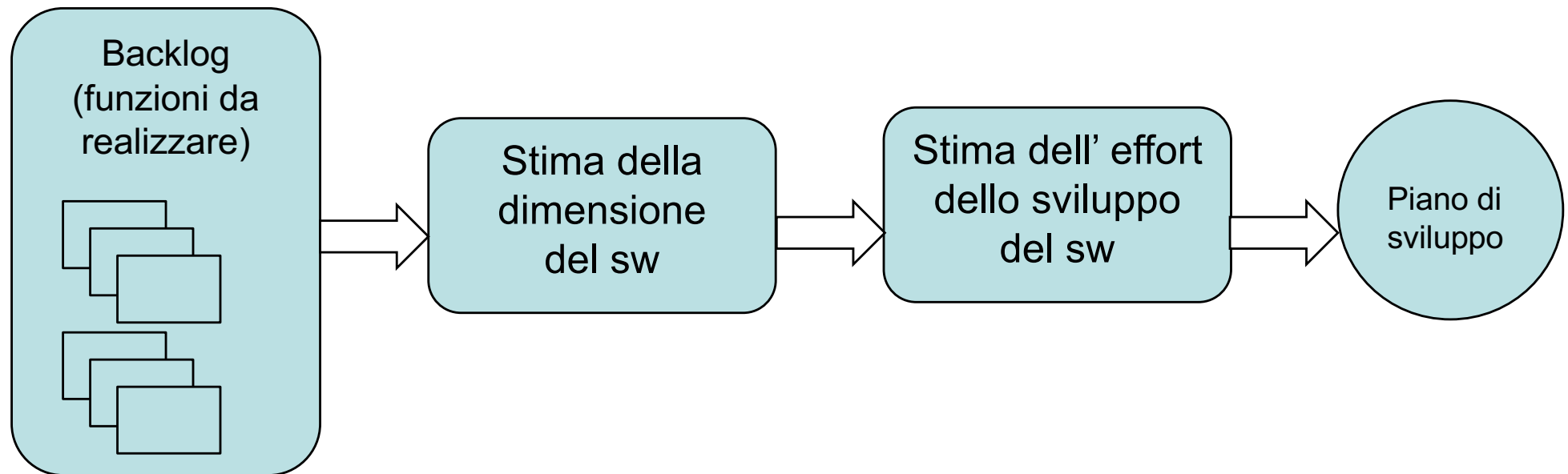
# I componenti del costo del Sw

I principali componenti di costo sw sono:

- Costo dell'hardware di sviluppo
- Costo del software di sviluppo
- Costo delle risorse umane (**sforzo**)
- **Durata** complessiva



# La pianificazione inizia con la stima della dimensione del software



# Misurare le dimensioni del progetto

- La **dimensione** (size) di un progetto sw è il primo coefficiente di costo in molti modelli che stimano durata e sforzo
- Esistono tre misure che stimano la dimensione di un software da produrre:
  - Le linee di codice
  - I punti funzione (Function Points)
  - I punti-storia (history points) nei processi agili
- Queste misure hanno bisogno di dati storici per poter essere “**calibrate**” rispetto all’organizzazione che le usa

# Linee di codice

- La misura più comune della dimensione di un progetto software è il numero di linee di codice *sorgente* (*Lines of Code*, LoC; 1 KLoC = mille **linee di codice** sorgente)
- LoC può tener conto delle linee vuote o dei commenti (CLoC); in generale si ha:  $\text{LoC} = \text{NCLoC} + \text{CLoC}$ , cioè i commenti si contano
- La distinzione più comune è tra LoC **fisiche** e LoC **logiche**

# Esercizio: contare le LOC

/\* Strncat() appends up to count characters from string src to string dest, and then appends a terminating null character. If copying takes place between objects that overlap, the behavior is undefined. \*/

```
char *strncat (char *dest, const char *src, size_t count)
{
    char *temp.dest;
    if (count ) {
        while (*dest )
            dest++;
        while ((*dest.. .*src.. )) {
            if (count .. 0) {
                *dest. '\0';
                break;
            }
        }
        return temp;
    }
}
```

Quante LOC?

# LoC fisiche e logiche

```
for (i=0; i<100; ++i) printf("hello");
```

Una LoC fisica, due LoC logiche

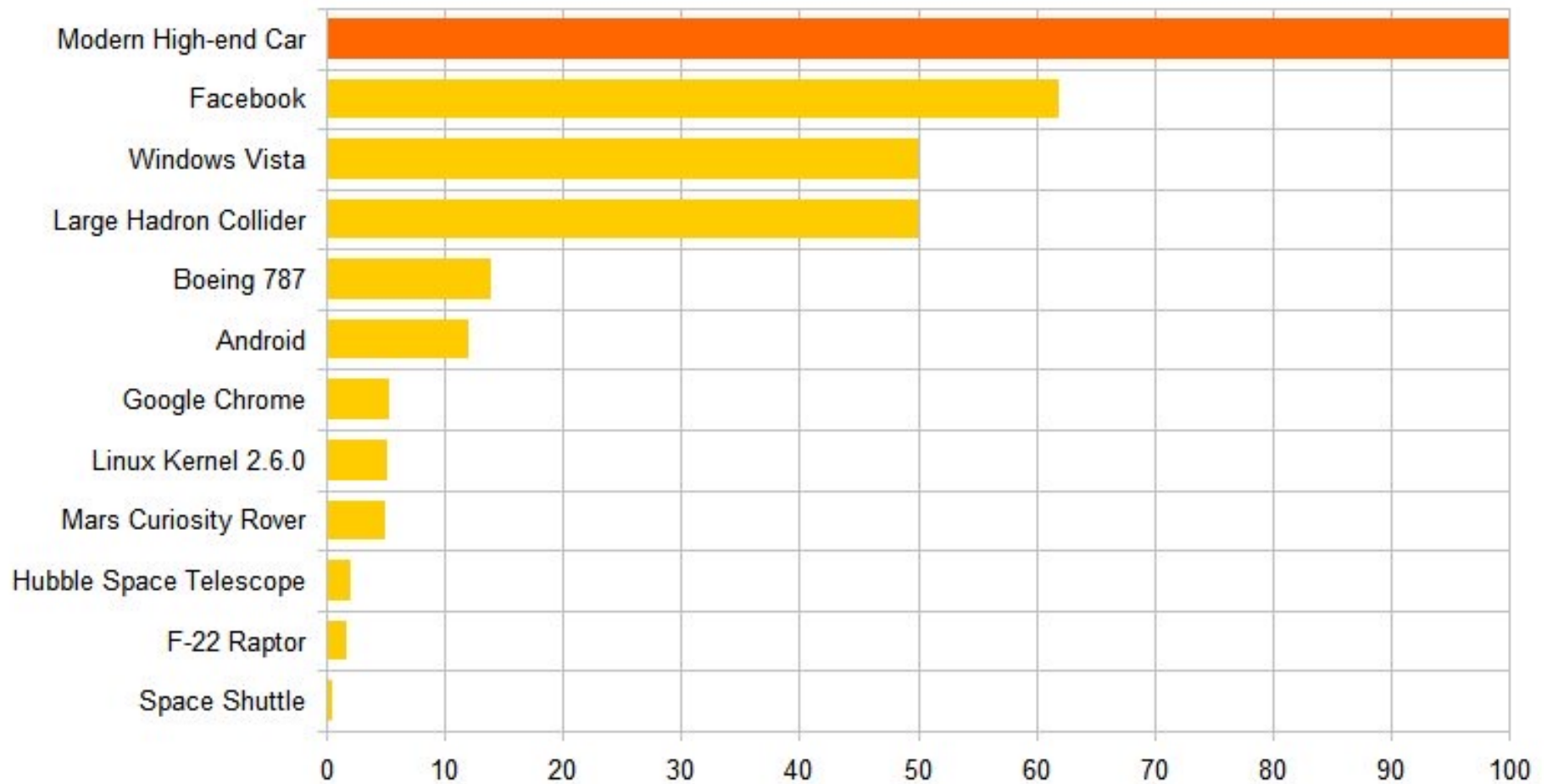
```
for (i=0; i<100; ++i)
{
    printf("hello");
}
/* Now how many lines of code is this? */
```

Cinque LoC fisiche, due LoC logiche

# Esempio: Linux LOC

- March 1994, Linux 1.0.0 - 176,250 lines of code.
- March 1995, Linux 1.2.0 - 310,950 lines of code
- January 1999 - Linux - 1,800,847 lines of code
- January 2001 - Linux 2.4.0 - 3,377,902 lines of code
- December 2003 - Linux 2.6.0 - 5,929,913 lines of code
- 2012, Linux 3.2 - 14,998,651 lines of code.
- Fonte: Wikipedia- Linux kernel

### Software Size (million Lines of Code)



<https://www.linkedin.com/pulse/20140626152045-3625632-car-software-100m-lines-of-code-and-counting/>

# Esempio: stima LOC (da Pressman)

• Interfaccia utente	2.300
• Gestione dati bidimensionali	5.300
• Gestione dati tridimensionali	6.800
• Gestione del db	3.350
• Visualizzazione grafica	4.950
• Controllo dispositivi	2.100
• Moduli di analisi del progetto	8.400
Totale LOC stimate	33.200

Produttività “storica” per sistemi di questo tipo = 620 LOC/mp.

Costo del lavoro = €8000 /mese, quindi ogni LOC costa €13.

La stima di costo totale è €431,000 mentre la stima di sforzo è 54 mp



# LOC: pro e contro

- Metriche derivate:
  - \$ per KLOC
  - errori o difetti per KLOC
  - LOC per mese/persona
  - pagine di documentazione per KLOC
  - Errori per mese-persona
  - \$/pagina di documentazione
- Il codice sorgente è il prodotto tangibile del processo di sviluppo, ed esiste già parecchia letteratura sulla sua misurazione LOC
- Però, la misura delle LOC dipende dal linguaggio di programmazione
- Inoltre, penalizza (sottovaluta la produttività) programmi scritti bene e concisi

# Aspetti critici delle stime dimensionali

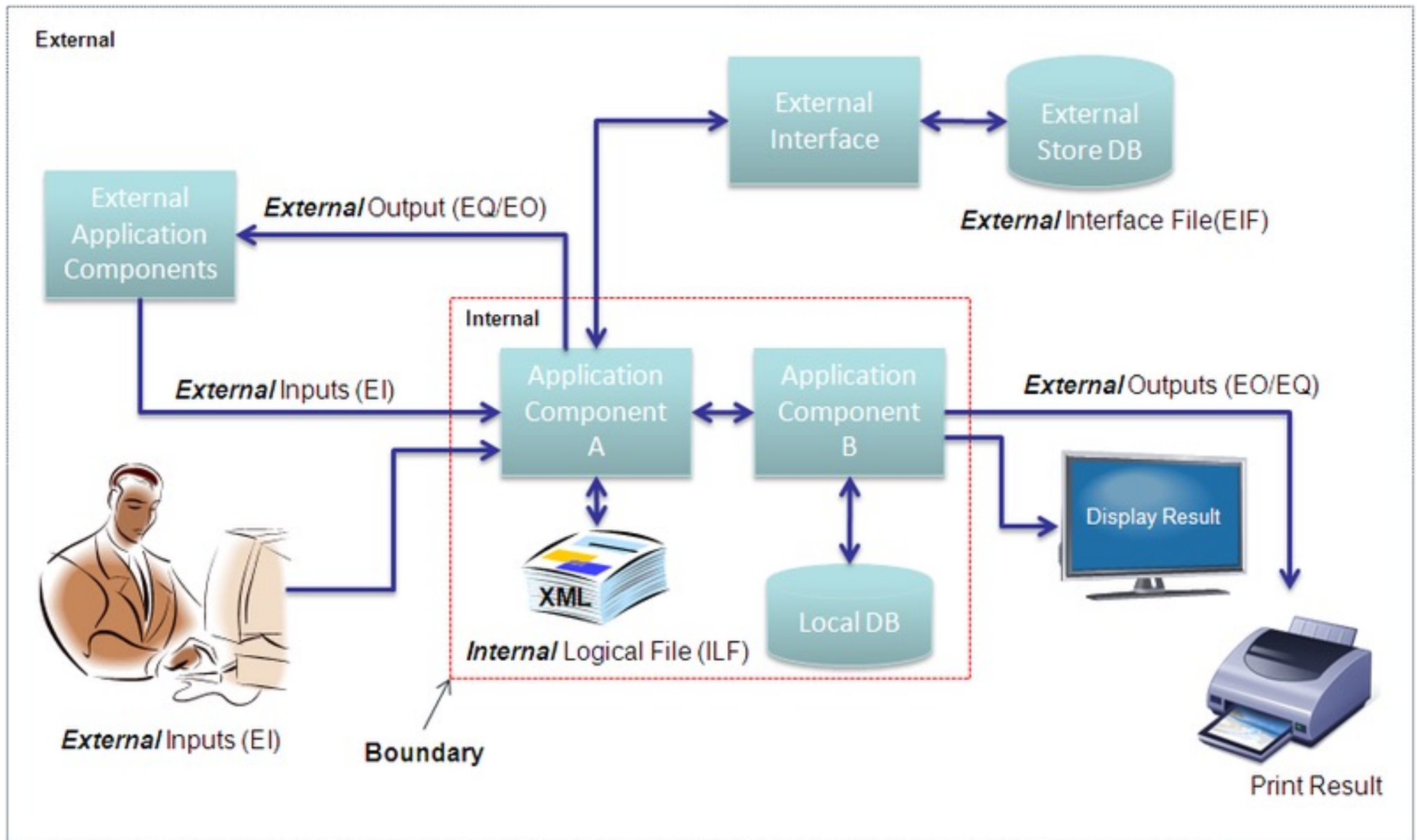
1. È difficile stimare la dimensione in LOC di una nuova applicazione
2. La stima LOC non tiene conto della diversa complessità e potenza delle istruzioni (di uno stesso linguaggio o di linguaggi diversi)
3. È difficile definire in modo preciso il criterio di conteggio (istruzioni spezzate su più righe, più istruzioni su una riga...)
4. Una maggior produttività in LOC potrebbe comportare più codice di cattiva qualità?
5. Il codice non consegnato (es. test) va contato?

# Function Point [Albrecht]

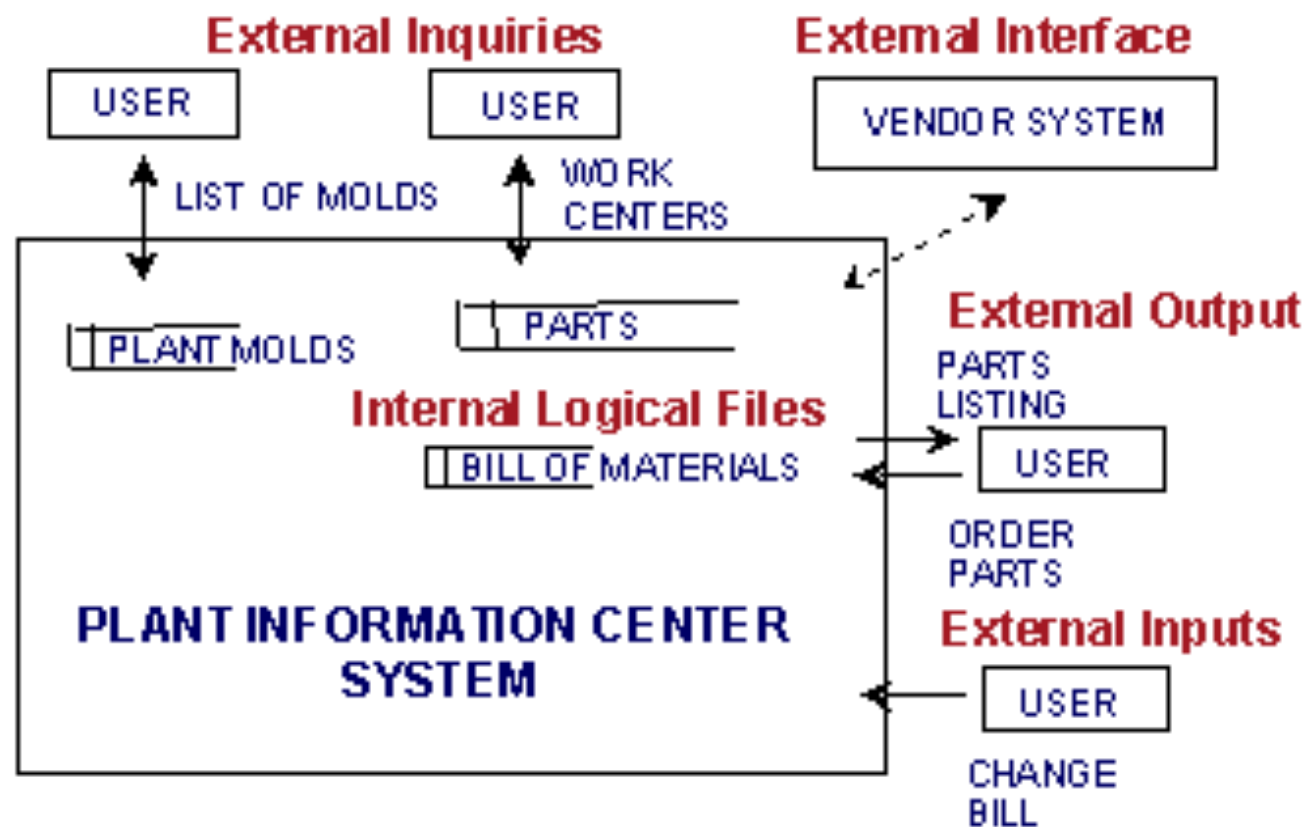
L'analisi Function Point enumera le funzionalità di un sistema dal punto di vista **utente**

*“The original objective of the Function Points work was to define a measure [that would] help managers analyze application development and maintenance work and highlight productivity improvement opportunities.”*

*“The Function Points method measures the equivalent functions of end-user applications regardless of the language, technology, or development environment used to create the application.”*



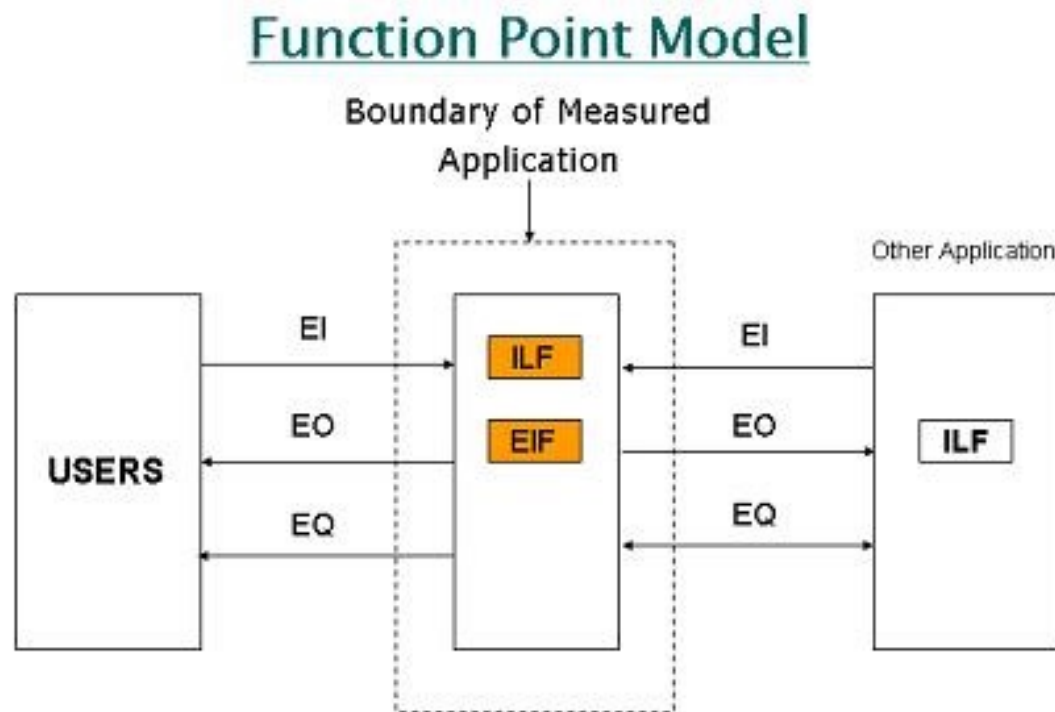
# Esempio: sistema informativo d'impianto



# Calcolo di FP: passo 1

Si descrive l'applicazione da realizzare

Si parte da una **descrizione** del sistema  
(**specific**) di natura **funzionale**



# Calcolo di FP: passo 2

Individuare nella **descrizione dei requisiti utente** i seguenti elementi:

- **External Input**: informazioni, dati forniti dall'utente (es. nome di file, scelte di menù, campi di input)
- **External Output**: informazioni, dati forniti all'utente dell'applicazione (es. report, messaggi d'errore)
- **External Inquiry**: sequenze interattive di richieste – risposte
- **External Interface File**: interfacce con altri sistemi informativi esterni
- **Internal Logical File**: file principali logici gestiti nel sistema

# Calcolo di FP: passo 3

Occorre classificare i singoli elementi funzionali per complessità di progetto, usando la seguente tabella di pesi

<b>Tipo Elementi</b>	Semplice	Medio	Complesso
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
External files	7	10	15
Internal files	5	7	10



# Calcolo di FP: passo 4

Censire gli elementi di ciascun tipo,  
moltiplicare per il lor “peso” e sommare:

$$UFC = \sum_{[i=1\_5]} (\sum_{[j=1\_3]} (\text{elemento } i,j * \text{peso } i,j))$$

Dove i = tipo elemento

j = complessità (semplice o media o complessa)

UFC = Unadjusted Function Count

# Calcolo di FP: passo 5

Definire il fattore di complessità tecnica dell' applicazione (TFC)

Calcolo di TFC:

$$\text{TFC} = 0.65 + 0.01 * \sum_{[i=1-14]} F_i$$

Ciascun fattore  $F_i$  viene valutato tra 0 (irrilevante) e 5 (massimo)

Il valore complessivo di TFC varia nell'intervallo da 0.65 (sviluppo facile) a 1.35 (sviluppo difficile)

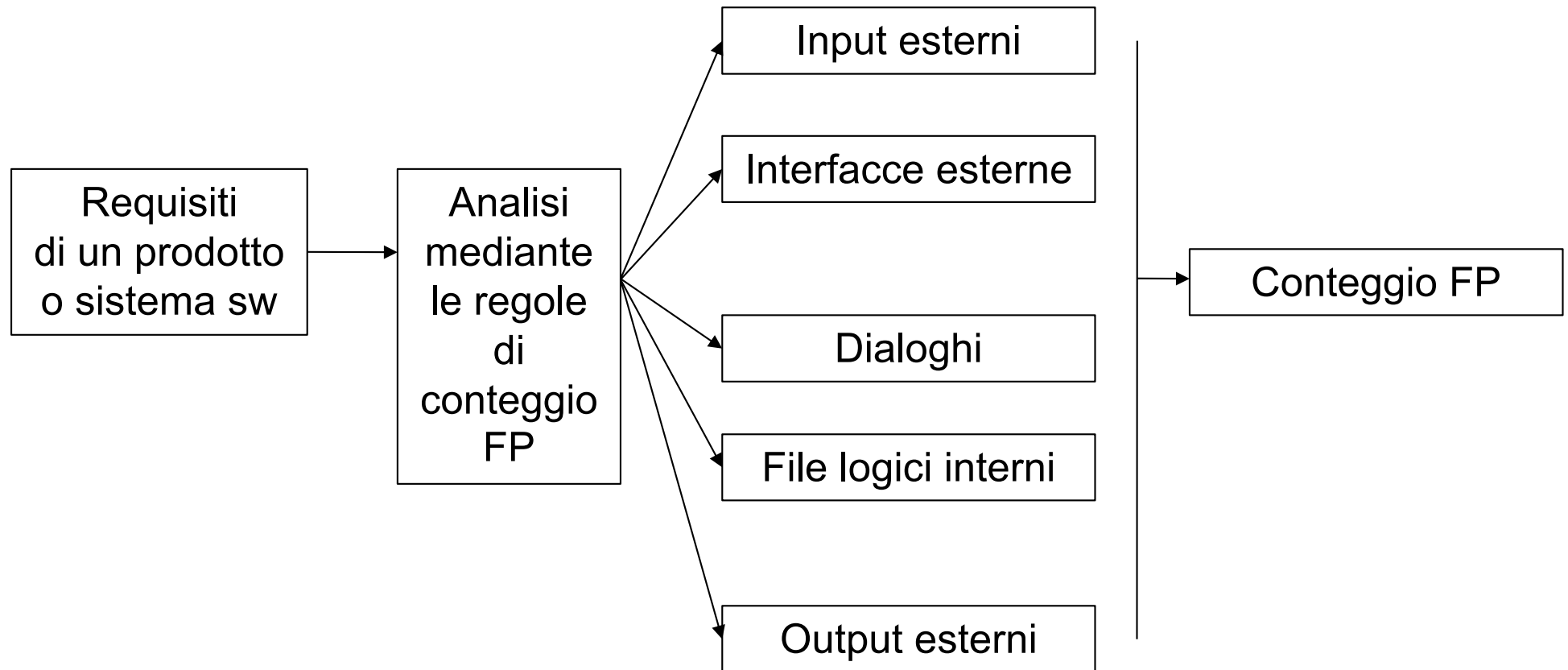
F1	Reliable backup and recovery	F2	Data Communications
F3	Distributed functions	F4	Performance
F5	Heavily used configuration	F6	Online data entry
F7	Operational ease	F8	Online update
F9	Complex interface	F10	Complex processing
F11	Reusability	F12	Installation ease
F12	Multiple sites	F14	Facilitate change

# Calcolo di FP: passo finale

Alla fine la formula risulta:

$$FP = UFC * TFC$$

# Riassunto del metodo



# Esempio

- File interni logici
  - DB Clienti
  - DB conti correnti
  - DB Pagamenti
  - DB Banche
- File esterni di interfaccia
  - GUI responsabile gestione clienti
  - GUI responsabile commerciale
  - Input esterni
  - Creazione account
  - Eliminazione account
  - Ricarica tessera
  - Saldo tessera
  - Richiesta pagamento importo
- Interrogazioni esterne
  - Visualizzazione stato cliente
  - Visualizzazione stato pagamenti
  - Visualizzazione saldi
- Input esterni
  - Notifica pagamenti
  - Gestione invio tessera

# Esempio

La descrizione funzionale di un sistema contiene

6	Input “medi”	x	4	=	24
6	Output “complessi”	x	7	=	42
2	File “medi”	x	10	=	20
3	Inquiries “semplici”	x	2	=	6
2	Interfacce “complesse”	x	10	=	20
	Unadjusted FC			=	112

# Esempio (continua)

Data communications	3
Distributed processing	2
Online processing	4
Complex internal processing	5
Multiple sites	3
TFC	17

Calcolo finale:  $UFC * TFC = 112 * (.65 + .17) = 92 \text{ FP}$

# Esempio (continua)

Se si sa, per esempio a causa di esperienze passate, che il numero medio di FP per mese-persona è pari a 18, allora si può fare la stima che segue:

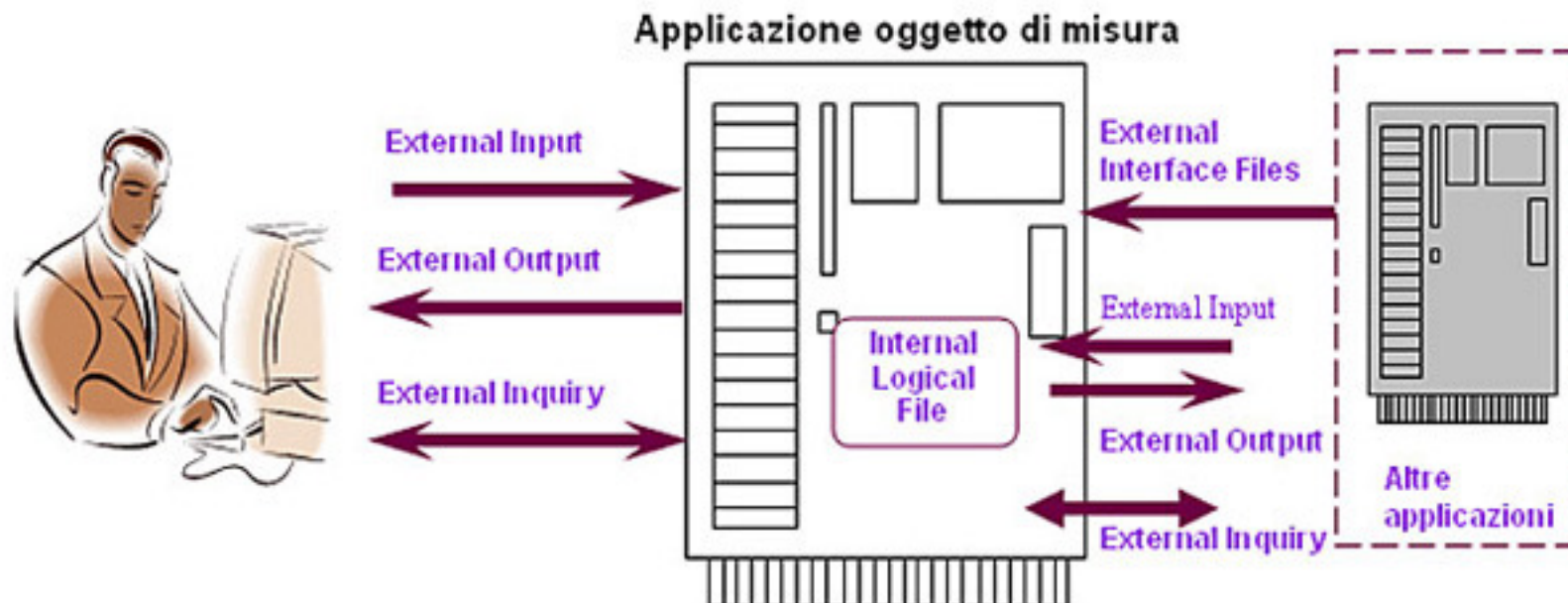
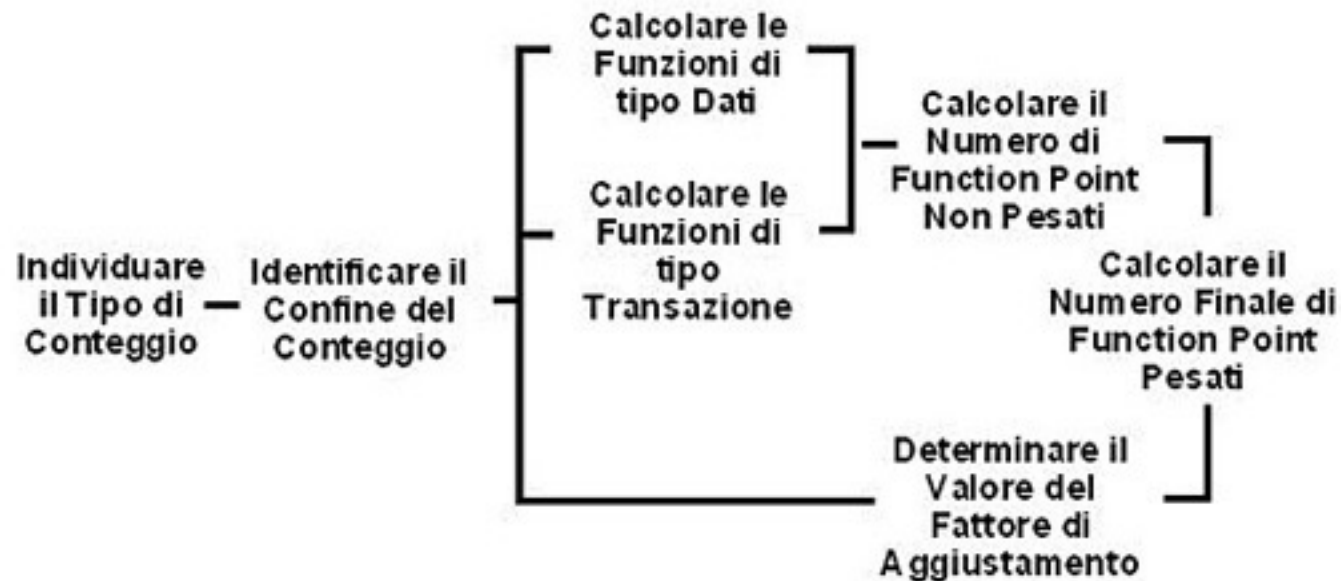
$$92 \text{ F.P.} \div 18 \text{ F.P./mp} = 5.1 \text{ mp}$$

Se lo stipendio medio mensile per lo sviluppatore è di €6.500, allora il costo [dello sforzo] del progetto è

$$5.1 \text{ mp} \times €6.500 = €33.150$$



# Riassunto



# Calibrazione

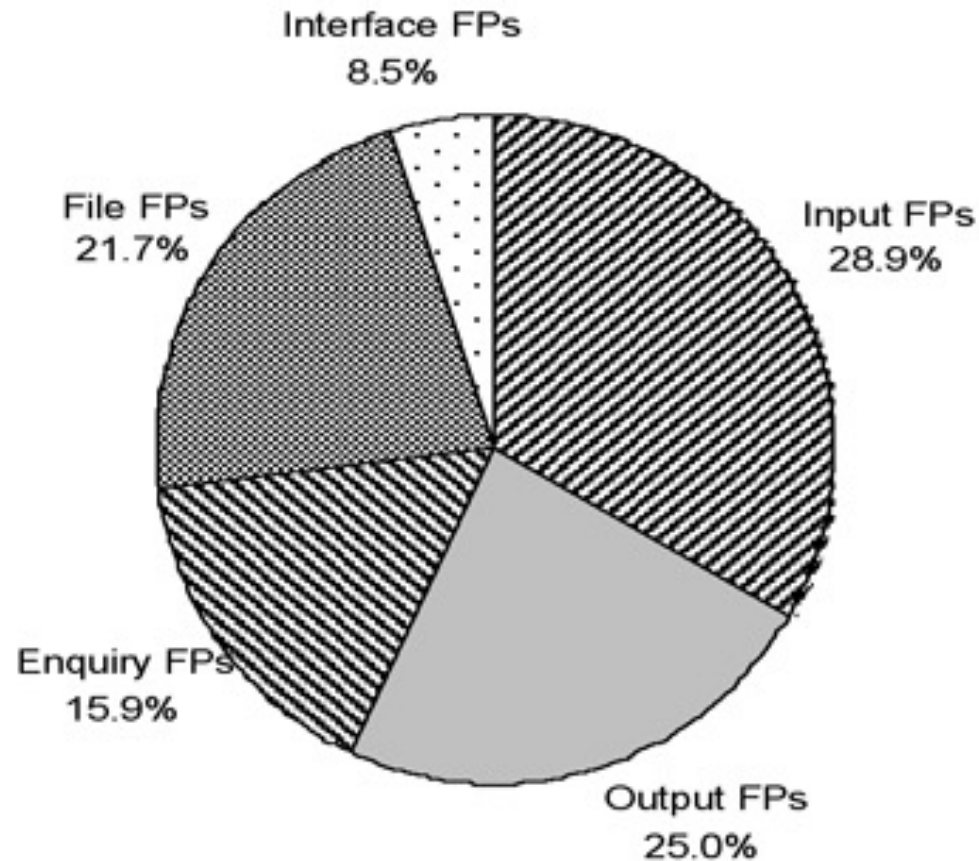
- Il conteggio dei FP si basa su giudizi **soggettivi**, quindi persone diverse possono raggiungere risultati diversi
- Quando si introduce l'Analisi FP (in sigla: FPA) in una organizzazione, è necessaria una fase di **calibrazione**, usando i progetti sviluppati in passato come base del sistema di conteggio

# Misurazioni di alcuni sistemi

(Capers Jones 2010)

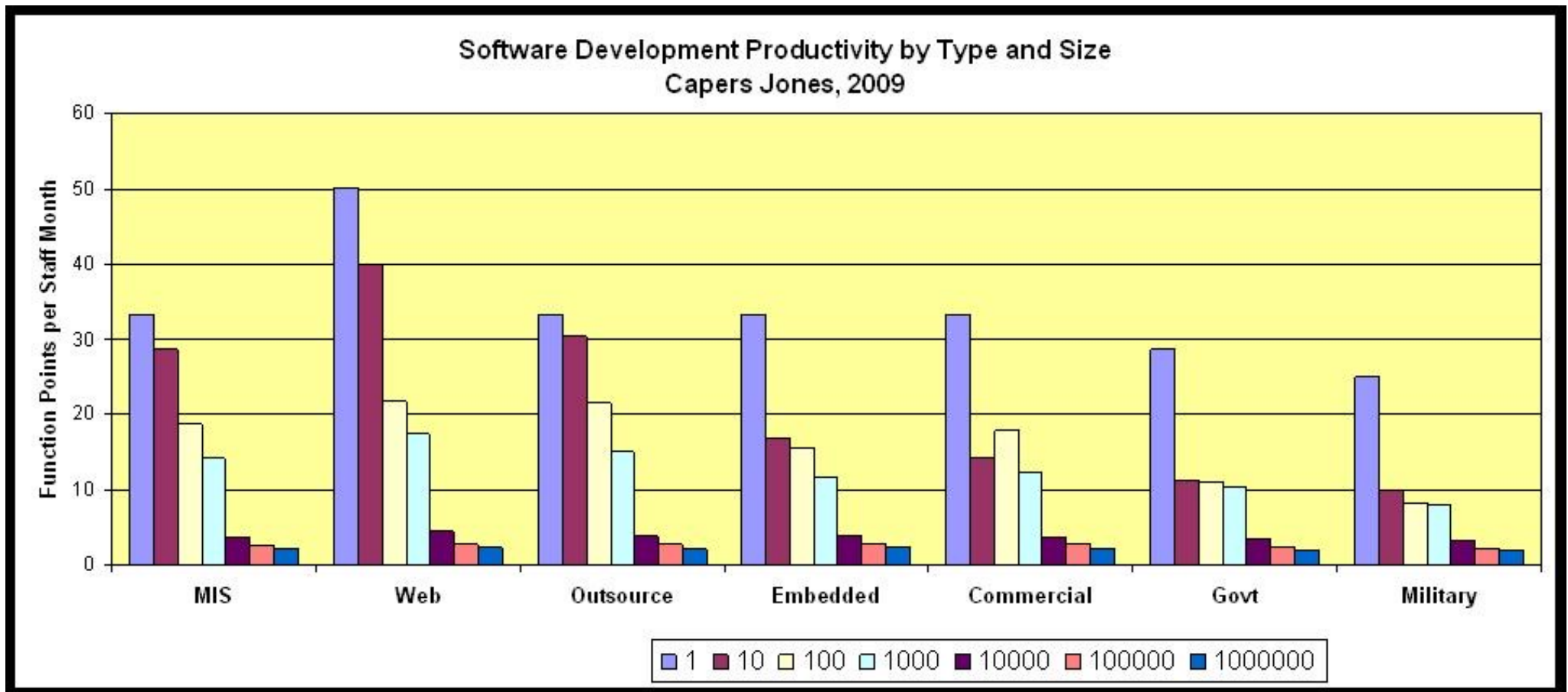
Sistema	Size in FP	KLOC	LOC/FP
US Air traffic control	306.324	65.000	213
Israeli Air Defense	300.655	24.000	80
SAP	296.764	24.000	80
Iran air defense	260.100	24.000	91
MS Vista	157.658	10.090	64
MS Office Pro	93.500	6.000	64
Iphone IOS	19.300	516	27
Google search	18.640	1.193	64
Amazon Website	18.080	482	27
Apple Leopard	17.884	477	27
Linux	17.505	700	40

# Function Point Mix New Developments



Source: Estimating, Benchmarking & Research Suite Release 9  
[209 projects - FPA METHOD: IFPUG 4]

# Produttività FP/mese/persona per diversi mercati di sw



# Costo per FP

Alcuni sviluppatori usano scale di costo basate su FP per i contratti

- Requisiti iniziali = \$500 per FP;
- Modifiche nei primi tre mesi = \$600 per FP;
- Modifiche successive = \$1000 per FP

# Stima basata su Modelli di Costo

- I modelli di costo permettono una stima «rapida» dello sforzo, utile durante le primissime fasi di un progetto
  - Questa prima stima viene poi raffinata, più avanti nel ciclo di vita, mediante dei fattori detti **cost driver**
    - Il calcolo si basa sull'**equazione dello sforzo**:
      - $E = A + B \cdot S^C$
- dove E è lo sforzo (in mesi-persona),  
A, B, C sono parametri dipendenti dal progetto e dall'organizzazione che lo esegue,  
S è la dimensione del prodotto stimata in KLOC o FP.

# Problemi dei costi del software

I costi del software sono dominanti, in percentuale dei costi totali dei sistemi informativi.

I problemi di stima dei costi sw sono causati da:

- incapacità di dimensionare accuratamente un progetto;
- incapacità di definire nel suo complesso il processo e l'ambiente operativo di un progetto;
- valutazione inadeguata del personale, in quantità e qualità;
- mancanza di requisiti di qualità per la stima di attività specifiche nell'ambito del progetto

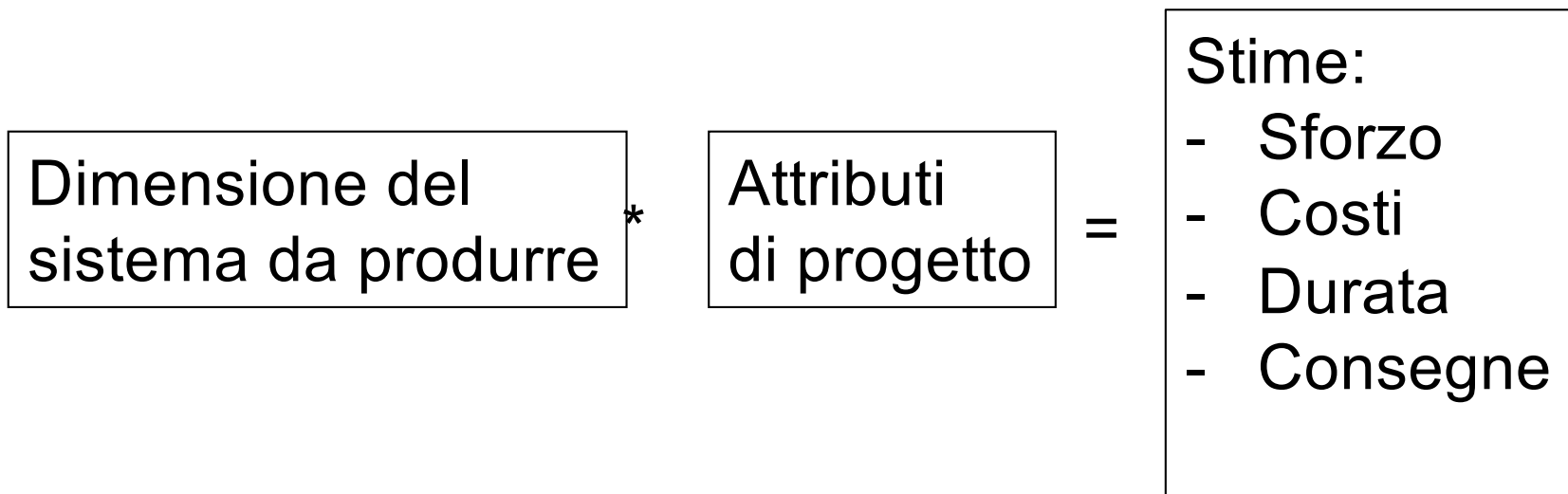


# Modelli dei costi del software

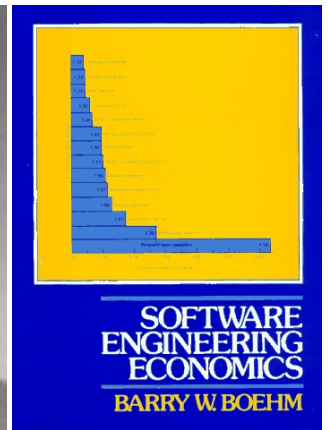
## Esempi di modelli commerciali

- **COCOMO**
- COSYSMO
- COSTXPERT
- SLIM
- SEER
- Costar, REVIC, etc.

# Uso di parametri per le stime



# Modelli di costi sw: COCOMO



- Il **Constructive Cost Model (COCOMO)** di Barry Boehm è uno dei modelli parametrici più diffusi per fare stime nei progetti software
- COCOMO 1 è descritto nel libro *Software Engineering Economics*, 1981
- COCOMO 2 è descritto nel libro *Software Cost Estimation*, 2000
- COCOMO è un modello basato su regressione (cioè su un archivio storico) che considera vari parametri del prodotto e dell'organizzazione che lo produce, pesati mediante una griglia di valutazione

# COCOMO: forma equazionale

- Il principale calcolo di COCOMO si basa sull' Equazione dello Sforzo per stimare il numero di mesi-persona mp necessari per un progetto

$$\text{Costo\_Stimato} = \# \text{ mp} * \text{costo\_lavoro}$$

- La maggior parte delle altre grandezze stimate (la durata, la dimensione del personale, ecc.) vengono poi derivate da questa equazione

# COCOMO: forma equazionale

$$\text{Performance} = (\text{Complexity})^{(\text{Process})} * (\text{Team}) * (\text{Tools})$$

- dove:

- **Performance** = Sforzo
- **Complexity** = Dimensione del codice generato
- **Process** = Maturità del processo e metodo
- **Team** = Abilità, esperienza, motivazione
- **Tools** = Automazione del processo

# COCOMO 1

- Boehm costruì la prima versione di un modello di costo chiamato CoCoMo 1 nel 1981
- CoCoMo 1 è una collezione di tre modelli:
  - **Basic** (applicato all'inizio del ciclo di vita del progetto)
  - **Intermediate** (applicato dopo la specifica dei requisiti)
  - **Advance** (applicato al termine della fase di design)

# COCOMO1

- I tre modelli hanno forma equazionale:

$$\text{Effort} = a * S^b * \text{EAF}$$

- Effort è lo sforzo in mesi-persona
- EAF è il *coefficiente di assestamento*
- S è la dimensione stimata del codice sorgente da consegnare, misurata in migliaia di linee di codice (KLOC)
- a e b sono dei coefficienti che dipendono dal tipo di progetto

# Tipi di progetti in COCOMO1

- **Organic mode** (progetto semplice, sviluppato in un piccolo team)
- **Semidetached mode** (progetto di difficoltà intermedia)
- **Embedded** (progetto con requisiti molto vincolanti e in campo non ben conosciuto)



# Formule per il Modello Base

Tipo fase basic	A	B	EAF	FORMULA RISULTANTE
Organic	2.4	1.05	1	$E = 2.4 * S^{1.05}$
Semi detached	3.0	1.12	1	$E = 3.0 * S^{1.12}$
Embedded	3.6	1.20	1	$E = 3.6 * S^{1.20}$

# Un esempio

Dimensione = 200 KLOC

Sforzo(in mp) =  $a * Dimensione^b$

**Organic:** Sforzo =  $2.4 * (200^{1.05}) = 626$  mp

**Semidetached:** Sforzo =  $3.0 * (200^{1.12}) = 1133$  mp

**Embedded:** Sforzo =  $3.6 * (200^{1.20}) = 2077$  mp

# Modello Intermedio

- Prende il modello basic come riferimento
- Identifica un insieme di attributi che influenzano il costo (detti *cost driver*)
- Moltiplica il costo di base per un fattore che lo può accrescere o decrescere
- La stima di questo modello azzecca i valori reali con approssimazione  $\pm 20\%$  circa il 68% delle volte
- (Modello più usato con COCOMO 1)

# Fattori di costo (COCOMO1, Intermediate)

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.

# Calcolo del fattore moltiplicativo

- I fattori moltiplicativi dovuti agli attributi hanno ciascuno un valore che indica lo spostamento dal valore normale di quel determinato attributo
- Il valore dei diversi fattori si determina tenendo conto dei progetti passati (calibrazione)
- Il prodotto della valutazione degli attributi rilevanti forma EAF

# Esempio

Dimensione = 200 KLOC

Sforzo =  $a * Dimensione^b * EAF$

Cost drivers:

Low reliability	0.88
High product complexity	1.15
Low application experience	1.13
High programming language experience	0.95

$EAF = 0.88 * 1.15 * 1.13 * 0.95 = 1.086$

**Organic:** Sforzo =  $2.4 * (200^{1.05}) * 1.086 = 906$  mp

**Semidetached:** Sforzo =  $3.0 * (200^{1.12}) * 1.086 = 1231$  mp

**Embedded:** Sforzo =  $3.6 * (200^{1.20}) * 1.086 = 2256$  mp

# Esempio 2

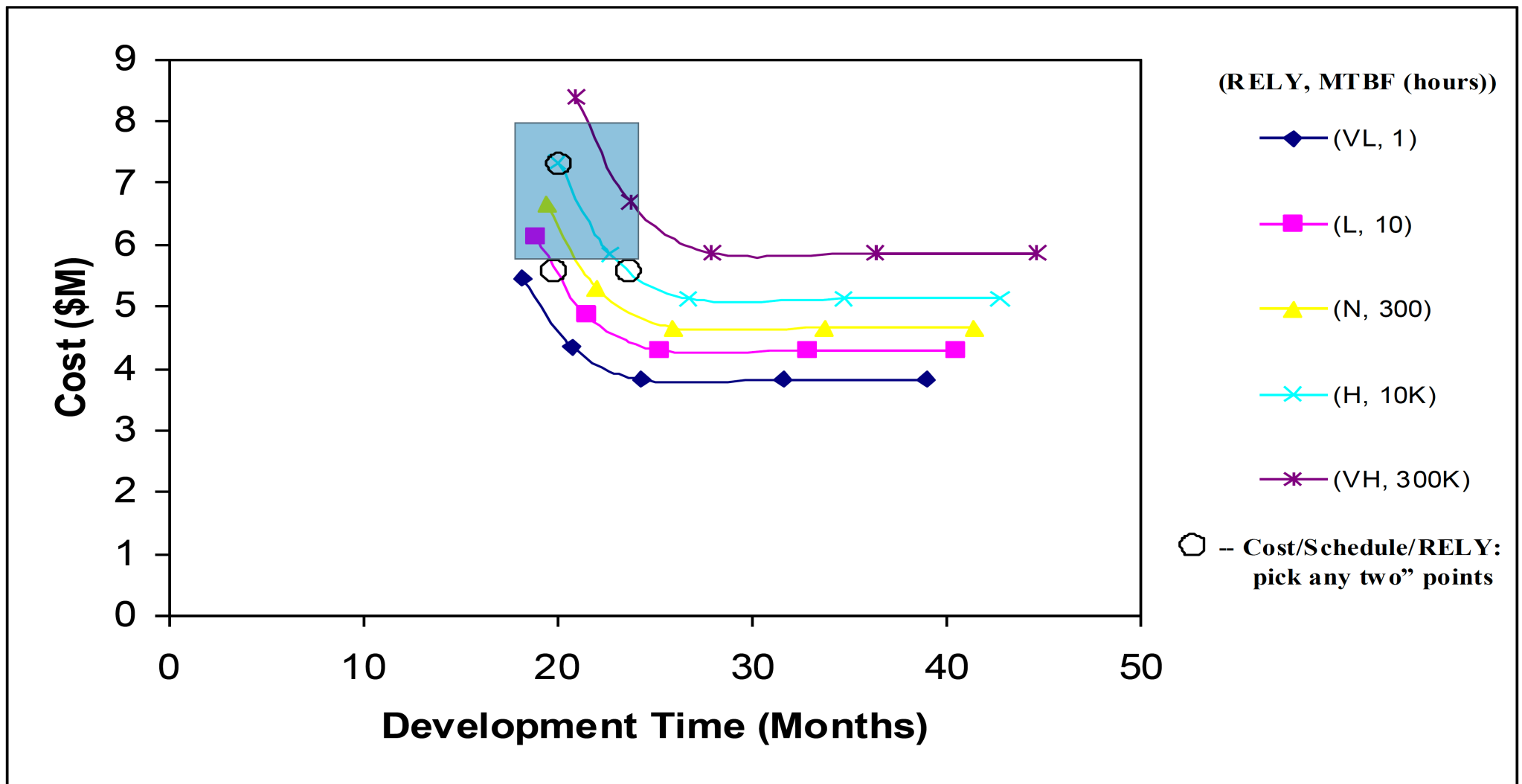
- Sw di comunicazione per trasferimento fondi (per es. Bancomat)
- Embedded mode
- 10 KLOC

Cost Drivers	Situation	Rating	Effort Multiplier
Required software reliability	Serious financial consequences of software fault	High	1.15
Database size	20,000 bytes	Low	0.94
Product complexity	Communications processing	Very high	1.30
Execution time constraint	Will use 70% of available time	High	1.11
Main storage constraint	45K of 64K store (70%)	High	1.06
Virtual machine volatility	Based on commercial microprocessor hardware	Nominal	1.00
Computer turnaround time	Two hour average turnaround time	Nominal	1.00
Analyst capabilities	Good senior analysts	High	0.86
Applications experience	Three years	Nominal	1.00
Programmer capability	Good senior programmers	High	0.86
Virtual machines experience	Six months	Low	1.10
Programming language experience	Twelve months	Nominal	1.00
Use of modern programming practices	Most techniques in use over one year	High	0.91
Use of software tools	At basic minicomputer tool level	Low	1.10
Required development schedule	Nine months	Nominal	1.00

# Esempio 2

- $\text{Effort} = 2.8 \times (10)^{1.20} = 44 \text{ mp}$
- Moltiplicatori di effort = 1.35
- Sforzo stimato reale  $44 \times 1.35 = 59 \text{ mp}$
- Questo indice (59 pm) si può usare in altre formule per calcolare o stimare
  - Costo in Euro
  - Piani di sviluppo
  - Distribuzioni di attività
  - Costi dell'hw
  - Costi di manutenzione annui
  - Ecc.





La figura mostra il bilanciamento di costo, durata, e affidabilità (RELY cost driver) per 100 KSLOC

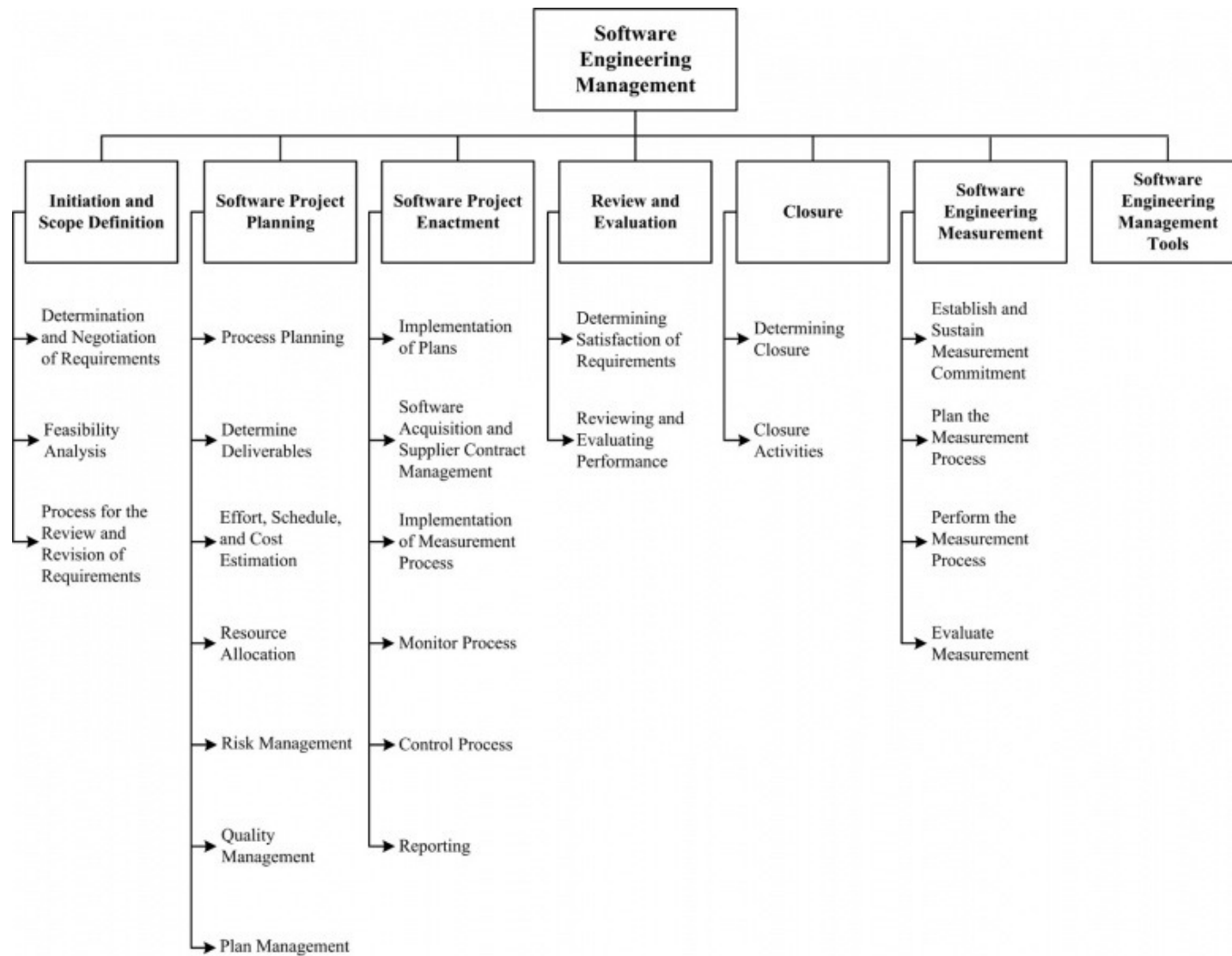
Si supponga che il progetto richieda alto livello RELY (10K-ore MTBF), una durata 20 mesi e un budget di \$5.5M.

Non si possono avere le tre cose insieme: Il budget deve salire a \$7.33M.

Per un costo di \$5.5M, si può avere un livello alto RELY e una durata di 23.5 mesi, oppure una durata di 20 mesi e un basso livello RELY level.

Per avere le tre cose insieme occorre ridurre la dimensione a 77 KSLOC.

# Project Management nel SWEBOK



# Domande di autotest

- Quali sono i compiti di un project manager?
- Cos'è il cono dell'incertezza?
- Cos'è un mese-persona? Quante ore-persona vale?
- Il processo influenza il costo di sviluppo del sw?
- Cos'è una linea di codice? Cos'è un function point?
- Un programmatore preferisce essere pagato a LOC oppure a FP realizzati?
- Cosa calcola il modello COCOMO?
- Cos'è uno history point?

# Riferimenti

- Wysocki, *Effective Project Management* (Traditional, Agile, Extreme), Wiley 2014
- Fairley, *Managing and Leading Software Projects*, Wiley, 2009
- McConnell, *Professional Software Development*, AW 2004
- IEEE Standard 1058-1998 for Software Project Management Plans
- PMI, *A Guide to the Project Management Body of Knowledge*, 6ed, 2017
- Longstreet, *Function Points Analysis Training Course*, 2004
- Bohem, *Software Cost Estimation with COCOMO 2*, PrenticeHall, 2000
- Putnam & Myers, *Measures for Excellence*, YourdonPress, 1992

# Siti

- [www.pmi.org](http://www.pmi.org)
- [www.mindtools.com/pages/article/newPPM\\_01.htm](http://www.mindtools.com/pages/article/newPPM_01.htm)
- [sunset.usc.edu/csse/research/COCOMOII/cocomo\\_main.html](http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html)
- [www.ifpug.org](http://www.ifpug.org) (associazione FP)
- [www.gufpi-isma.org](http://www.gufpi-isma.org)
- [www.dwheeler.com/sloc/](http://www.dwheeler.com/sloc/)
- [www.devdaily.com/FunctionPoints/FunctionPoints.shtml](http://www.devdaily.com/FunctionPoints/FunctionPoints.shtml)
- [www.cosmic.com](http://www.cosmic.com)
- [brodzinski.com](http://brodzinski.com) (blog su sw project management)
- [gispict.wikispaces.com/Analisi+dei+costi+software](http://gispict.wikispaces.com/Analisi+dei+costi+software)
- [alvinalexander.com/FunctionPoints/](http://alvinalexander.com/FunctionPoints/) (esempio FP)
- [conferences.embarcadero.com/article/32094#Eqs](http://conferences.embarcadero.com/article/32094#Eqs) (stesso esempio, FP)

# Strumenti

- <http://softwarecost.org/tools/COCOMO/>
- [www.dotproject.net](http://www.dotproject.net)
- [sourceforge.net/projects/projectlibre](http://sourceforge.net/projects/projectlibre)
- [www.planningpoker.com](http://www.planningpoker.com)

# Domande?



© 2003 United Feature Syndicate, Inc.

# La qualità del software



*Corso di Ingegneria del Software*  
*CdL Informatica*  
*Università di Bologna*



# Obiettivi della lezione

- I rischi della cattiva qualità del software
- Gli standard di qualità
- Il testing

# Software Engineering Disaster Hall of Fame

<http://catless.ncl.ac.uk/Risks/>  
<http://bugsniffer.blogspot.com/2007/11/infamous-software-failures.html>

#	Name	When	Cost	Impact	Cause(s)
1	<a href="#">Soviet Nuclear Missile False Alarm</a>	Sep 26, 1983	500,000,000 lives at risk	nearly-averted nuclear holocaust	» deadline pressure » faulty signal analysis » false positive
2	<a href="#">NORAD Nuclear Missile False Alarm</a>	Jun 3-6, 1980	500,000,000 lives at risk	nearly-averted nuclear holocaust	» scope creep » corrupted data » test message was a blank attack warning message
3	<a href="#">HMS Sheffield Exocet Missile Mis-classification</a>	May 4, 1982	30 deaths 257 lives at risk £23,200,000	» failure to intercept incoming missile » ship loss	» inadequate requirements » unnecessary simplicity » insufficient maintenance
4	<a href="#">Chinook Helicopter Crash</a>	Jun 2, 1994	29 deaths		
5	<a href="#">Patriot Missile Clock Drift</a>	Feb 25, 1991	28 deaths 98 injured	failure to intercept incoming missile	» short-sighted temporal requirements » rounding error » clock drift
6	<a href="#">Panama Radiation Therapy Overdose</a>	2000	18 deaths 10 injured	radiation overdose (10-110%)	» overreliance on automation » inconsistent output » double counting » inadequate testing
7	<a href="#">V-22 Osprey Crash</a>	Dec 11, 2000	4 deaths	aircraft crash	confusing output
8	<a href="#">Therac-25</a>	1985-1987	3 deaths 3 injured	radiation overdose (10000%)	» unrealistic risk assessment » lack of defensive design » inadequate testing » arithmetic overflow » race condition » complacency
9	<a href="#">USS Yorktown</a>	Sep, 1996	400 lives at risk 2 day delay	» complete systems crash » propulsion system failure » ship disabled	» lack of defensive design » inadequate training » insufficient input validation » edge case » divide by zero
10	<a href="#">F-22 Raptor International Dateline Crossing</a>	Feb 11, 2007	6 lives at risk	» complete system crash » mission aborted » retreated by visually following tankers	» lack of defensive design » date/time mishandling

# Fiat Chrysler recalls more than 1-million vehicles due to software problem

12 MAY 2017 - 14:34 by DAVID SHEPARDSON



Fiat Chrysler Automobiles CEO Sergio Marchionne. Picture: REUTERS

Washington — On Friday, Fiat Chrysler Automobiles said it was recalling more than 1.25-million trucks worldwide to address a software error linked to reports of one death resulting from a crash, and two injuries.

The Italian-American vehicle maker said it would reprogram computer modules because an error code could temporarily disable side airbag and seatbelt pretensioner deployment when a car was rolled, if the vehicle "were subjected to a significant underbody impact".

The recall covers about 1-million Ram 1500 and 2500 pick-up vehicles made between 2013 and 2016, and Ram 3500 vehicles made between 2014 and 2016 in the US, 216,007 vehicles in Canada; 21,668 in Mexico and 21,530 outside North America, Fiat Chrysler said.

*Reuters*

E-GOV

83 commenti

3 minuti



430  
condivisioni

# Liquidazione IVA e fatture online, sito chiuso per falla

Buco nella piattaforma online dell'Agenzia delle Entrate, gestita da Sogei, per trasmettere liquidazioni IVA e fatture. Quanti e quali dati sensibili dei contribuenti sono stati diffusi per errore? Non si sa ancora.

di **Pino Bruno** @pinobruno · 25 Settembre 2017, 11:15 · (Fonte **Il Sole 24 Ore** - **Il Corriere della Sera**)

## Fatture e Corrispettivi



**Il servizio web è temporaneamente sospeso per manutenzione.  
Restano attivi tutti gli altri canali di trasmissione.**

Ci scusiamo per l'inconveniente.

# Adesso è l'AI che sbaglia

**technology** > innovation > inventions

## How a 'confused' AI may have fought pilots attempting to save Boeing 737 MAX8s

Two Boeing 737 MAX airliners crashed, killing everyone aboard. Now investigations are zeroing in on one single faulty component.

---

Jamie Seidel

News Corp Australia Network 🕒 MARCH 19, 2019 3:24PM

---

<https://www.news.com.au/technology/innovation/inventions/how-a-confused-ai-may-have-fought-pilots-attempting-to-save-boeing-737-max8s/news-story/bf0d102f699905e5aa8d1f6d65f4c27e>



# Michigan, errori del software per il conteggio dei voti?

Di Jack Phillips

9 NOVEMBRE 2020



Gli scrutatori del Dipartimento delle Elezioni di Detroit scrutinano i voti postali presso il Comitato Centrale di Scrutinio di Detroit il 4 novembre 2020. (Elaine Cromie/Getty Images)

I repubblicani del Michigan hanno annunciato nuove indagini sul software della Dominion Voting System, dopo che la scorsa settimana migliaia di voti sono stati trasferiti per un errore dai repubblicani ai democratici.

Il 7 novembre Tony Zammit, addetto stampa del Partito Repubblicano del Michigan, ha dichiarato al Washington Examiner: «Il nostro team sta attualmente contattando i responsabili delle contee di tutto il Michigan e sta analizzando i risultati delle elezioni in ciascuna delle contee che utilizzano questo software per vedere quanto sia diffuso questo errore».

Il presidente del Partito Repubblicano del Michigan, Laura Cox, ha riferito che 47 contee del Michigan hanno utilizzato il software della Dominion proprio come ha fatto la contea di Antrim, dove si è scoperto che 6 mila voti sono stati erroneamente assegnati al candidato democratico Joe Biden invece che al presidente Donald Trump.

## Scelti dalla Redazione

**1** Storie d'umanità in un villaggio sul confine conteso tra India e Pakistan



**2** Le elezioni americane svelano la battaglia in corso tra libertà e comunismo



**3** Kamala Harris è comunista, altro che moderata



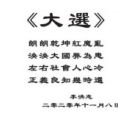
**4** Non c'è niente di normale nelle elezioni presidenziali del 2020



**5** Sui pericoli dell'Egualitarismo



**6** Sulle elezioni presidenziali



**7** L'«industria della pace» si è sbagliata sul Medio Oriente



**8** Nuovi Gandhi, gli



# Tesla richiama quasi 12.000 auto per un errore software

---

*L'errore potrebbe essere presente sulle auto vendute dal 2017 a oggi negli Stati Uniti, per una cifra totale di 11.704 veicoli*

Pubblicato il 02 Novembre 2021 ore 19:25

News > Economy

November 21 2022 07:00:28



---

## Tesla recalls 300K vehicles over taillight software glitch

LOS ANGELES

Online il video della Tesla impazzita che in Cina ha ucciso 2 persone e ferito altre 3 dopo una folle corsa: ancora non chiare le cause del grave incidente

[https://www.youtube.com/watch?v=6Kf3I\\_OyDI](https://www.youtube.com/watch?v=6Kf3I_OyDI)

---

14 novembre 2022 - 12:00

# **Il nuovo treno della metro di Napoli è finito in officina per problemi software dopo un mese di lavoro**

*Primo stop per il nuovissimo convoglio della metropolitana di Napoli: problemi al sistema informatico che segnala anomalie inesistenti. Necessario stop.*



16.8.2023

www.corriere.it/esteri/23\_agosto\_16/bank-of-ireland-errore-prelievo-bancomat-1e7

/singlelogin.re Essence – Dr. Stefan Malich Scacchi Mieì siti News Reti sociali Search Interessanti Pisa Viaggi Acquisti Arch Sw Quiz Corsi Sw Eng

Sezioni Edizioni Locali Servizi **CORRIERE DELLA SERA** Scopri tutti i podcast Lettore\_13749858

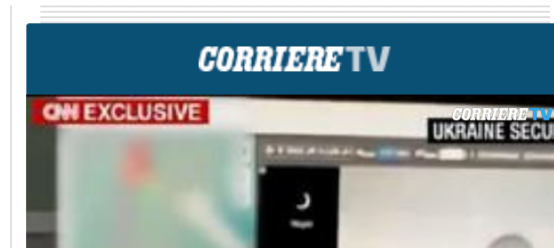
IN EVIDENZA

Le ultime notizie sulla guerra in Ucraina, la diretta

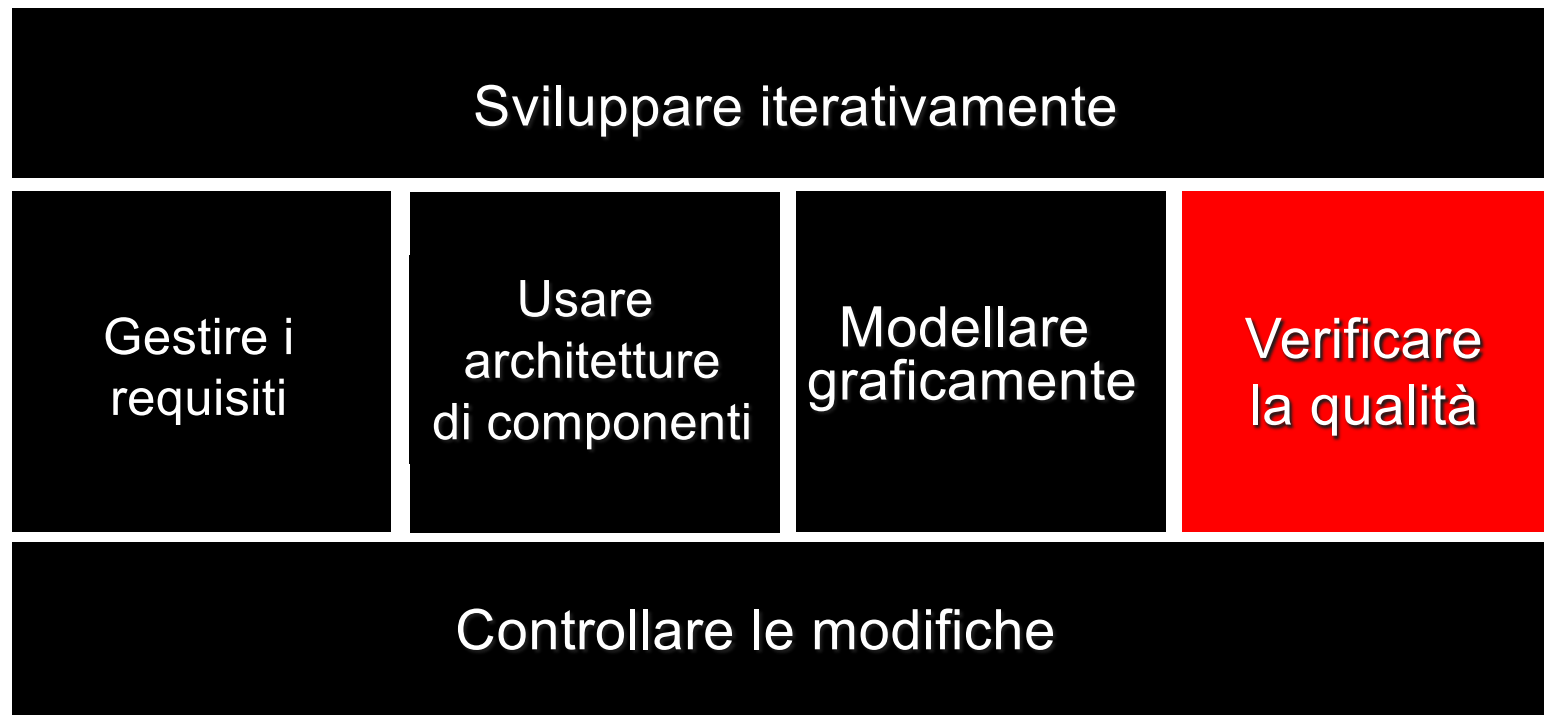
# Bank of Ireland, un errore nel sistema informatico consente il prelievo al bancomat a chi è in rosso: file chilometriche agli sportelli

di Monica Ricci Sargentini

L'Istituto ha sottolineato che «il denaro dovrà essere restituito e sarà considerato un debito». Sull'accaduto sarà condotta un'indagine con il coinvolgimento della Banca centrale.

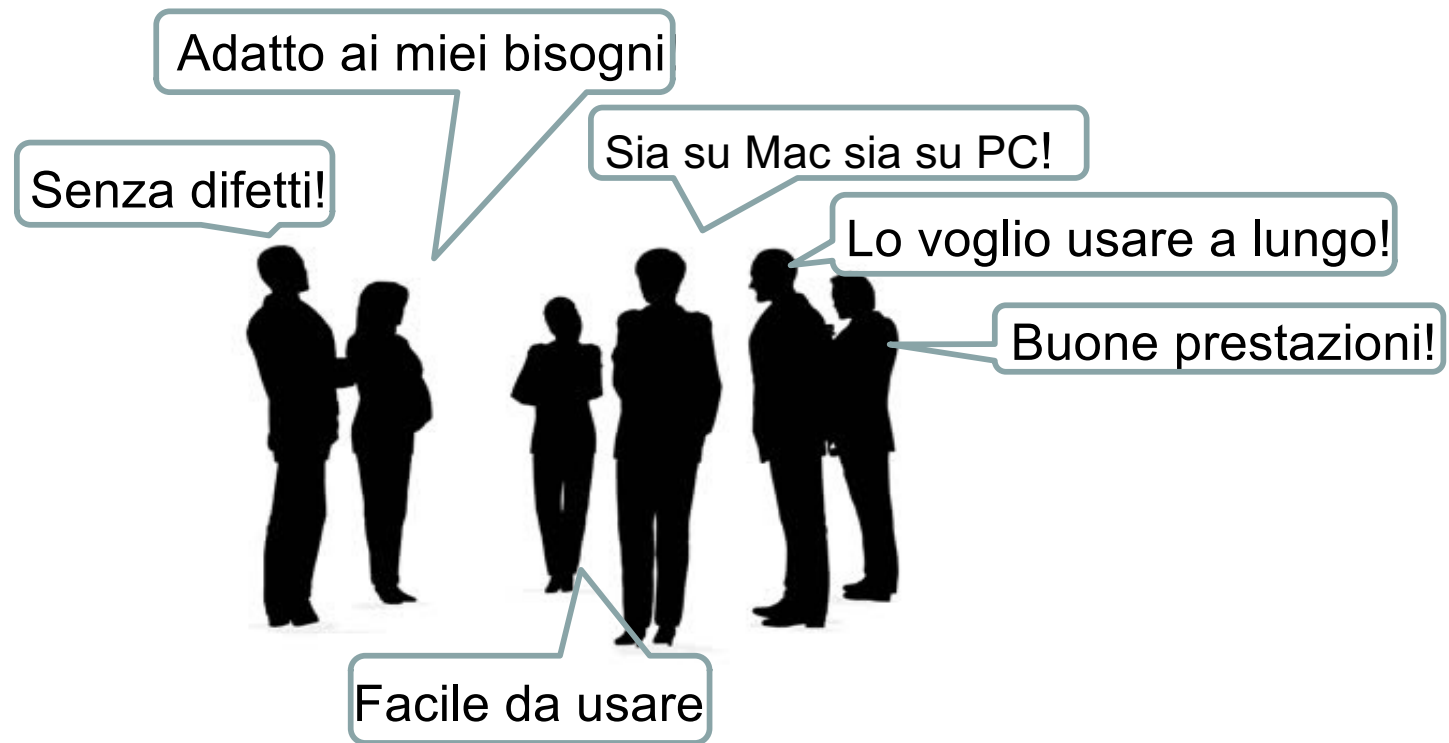


# Principi guida dello sviluppo software



La qualità del software

# Cos'è la qualità del software?



Tutti: deve costare poco!

# Errori, difetti e guasti

La “vita” di un bug:

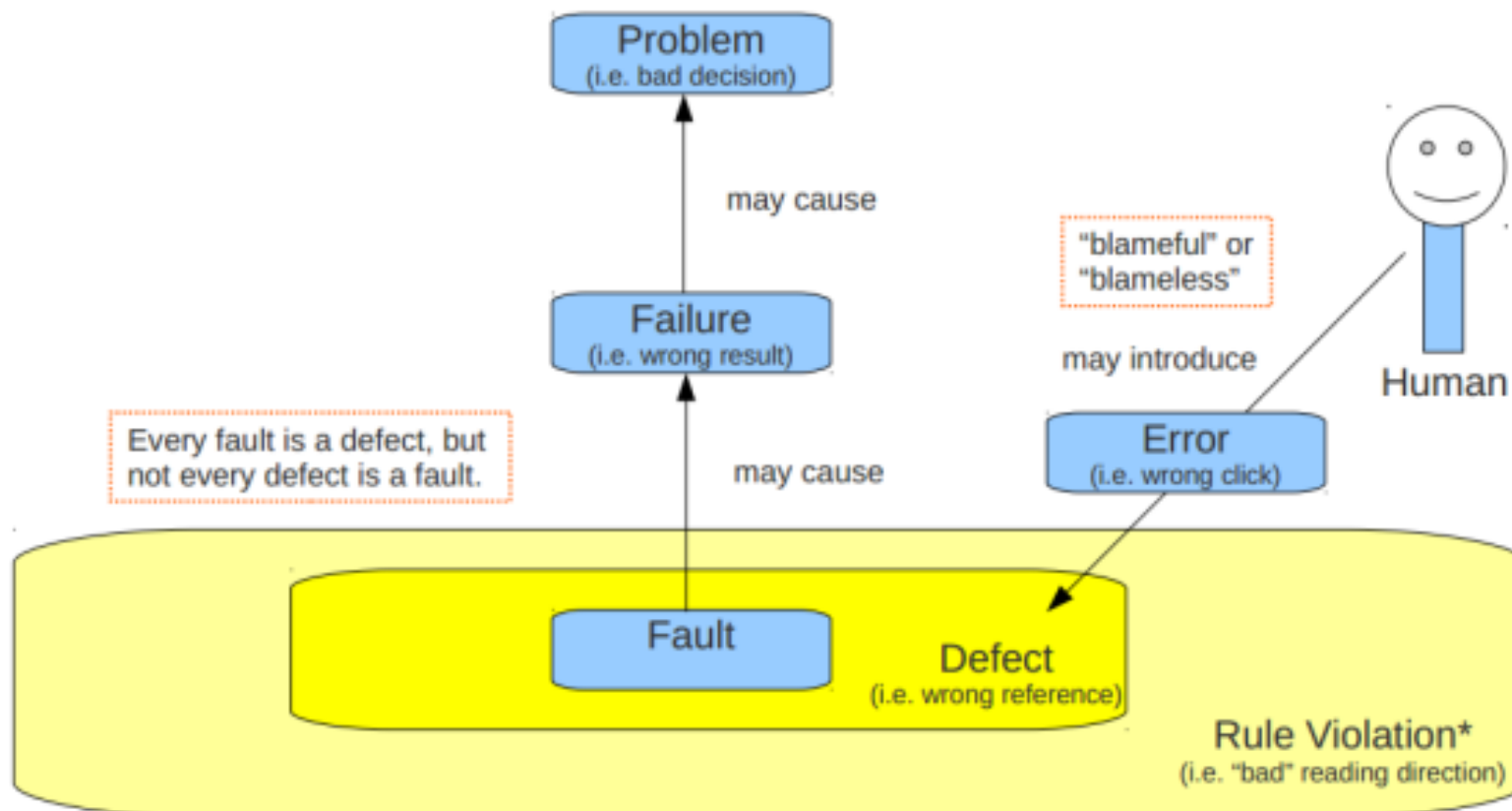
- Il programmatore commette un **errore** mentre scrive una linea di codice;
- L'errore può avere diversi effetti negativi: diventa così un **difetto** (*fault*)
- Il difetto causa direttamente una eccezione/ un output indesiderato: diventa così un **guasto** (*failure*)

## Da IEEE Standard Glossary of SE Terminology

- **Error**: causa di un difetto;  
esempio: un errore umano d'interpretazione della specifica o nell'uso di un metodo
- **Fault**: difetto del sorgente (*bug*), causa di un guasto
- **Failure**: comportamento del sw non previsto dalla sua specifica

# Classificazione standard dei problemi software

(according to IEEE Std 1044-2009)

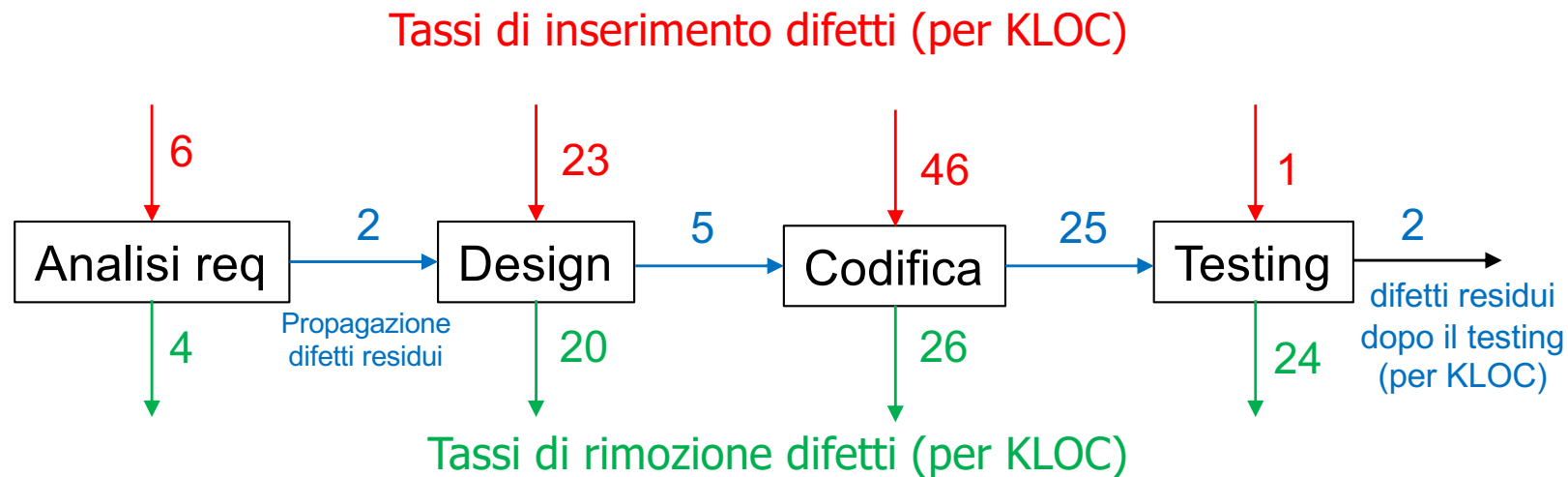


# Perché il sw è difettoso?

- Gli esseri umani commettono errori, specie quando eseguono compiti complessi; ciò è inevitabile
- I programmatori esperti commettono in media un errore ogni 10 righe
- Circa il 50% degli errori di codifica vengono catturati a tempo di compilazione
- Altri errori vengono catturati col testing
- Circa il 15% degli errori sono ancora nel sistema quando viene consegnato al cliente

(W. Humphrey: „What if your life depended on software?” EuroSPI conference, Copenhagen, April 2000)

# Da dove arrivano i difetti?

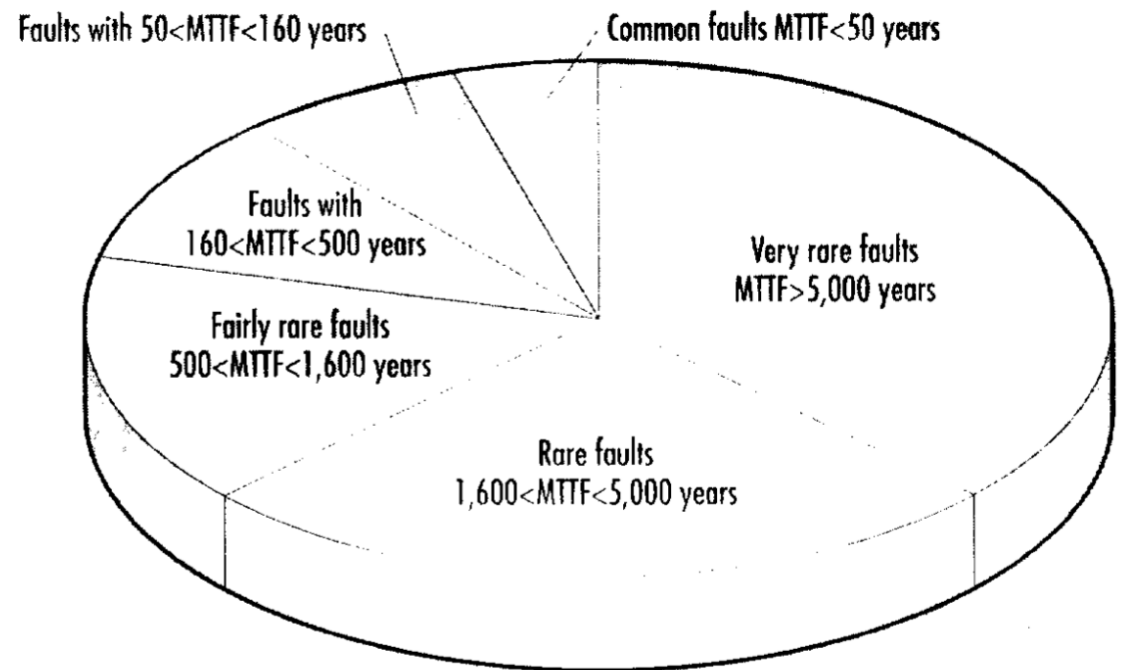


Fonte: [Eick & Loader, ICSE 1992](#)

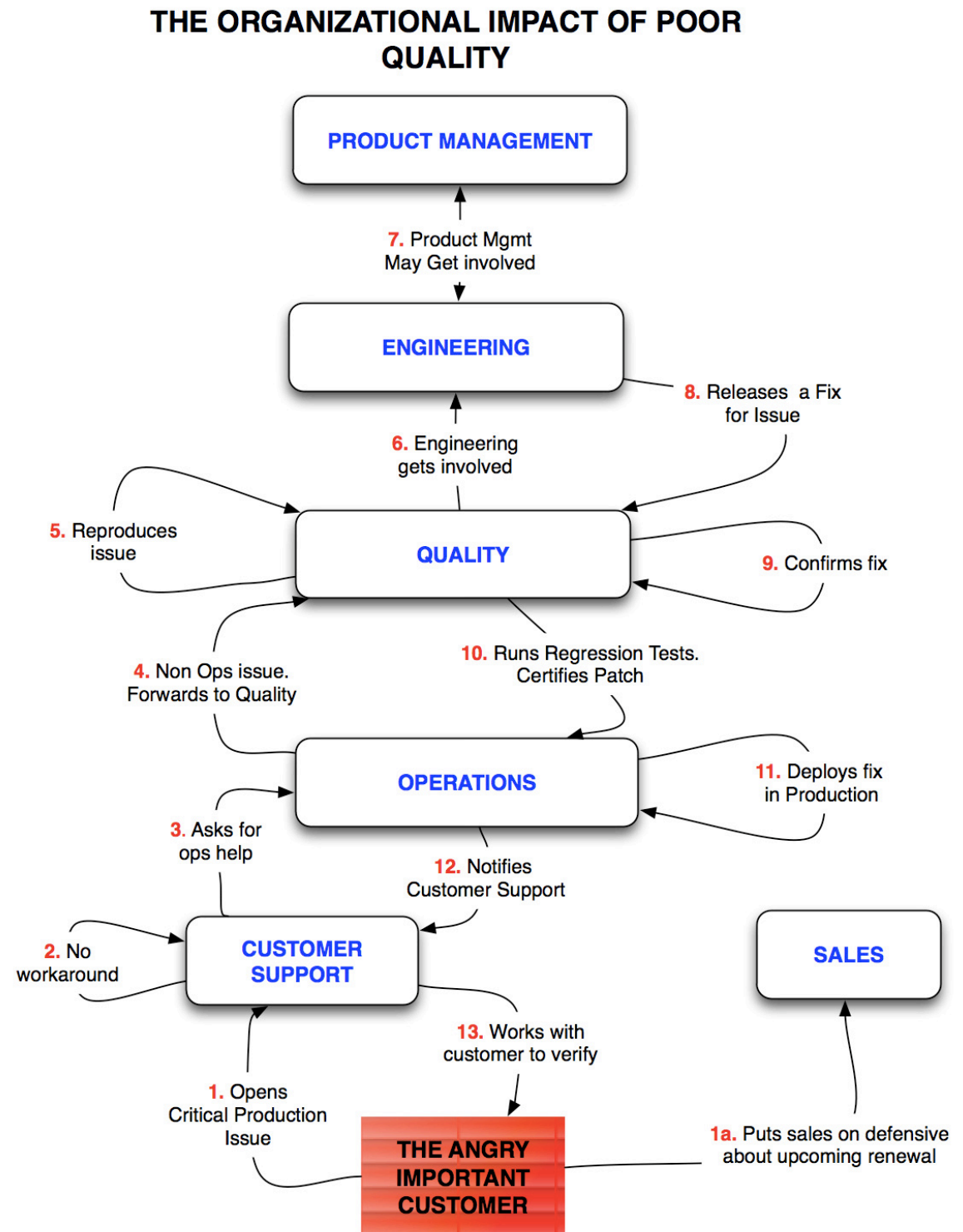


# Guasti vs difetti

- MTTF: mean time to failure
- i difetti sono tanto più pericolosi quanto più frequenti sono i guasti che ne derivano
- La figura mostra che bisogna cercare i difetti più comuni che causano i guasti più frequenti,
- Tali difetti “comuni” sono pochi rispetto al totale dei guasti: moltissimi sono rari o molto rari
- Quindi eliminare difetti non necessariamente migliora la qualità di un sistema software!



# Gestione della qualità nelle grandi aziende



# Obiettivi di questa lezione

- Definire la qualità
- Definire le qualità di processi e prodotti sw
  - Qualità interne - visibili al progettista
  - Qualità esterne - visibili all'utente
- Come valutare la qualità
  - La misurazione di indicatori di qualità
  - Il metodo Goal-Question-Metric (GQM)
- Verificare la qualità: tecniche di testing

# Discussione

- Come possiamo dire che un prodotto software è di buona qualità (rating)?
- Come possiamo dire che un prodotto software è migliore di un altro (ranking)?



# Qualità dei prodotti software

## **1. Il prodotto deve soddisfare la sua specifica**

- I test che verificano il codice rispetto ai requisiti sono il fondamento su cui misurare la qualità

## **2. Il prodotto non deve contenere difetti**

- Ma è molto difficile produrre software privo di errori

## **3. Il prodotto deve rispettare gli standard industriali**

- Quali standard internazionali definiscono i criteri di qualità per valutare il prodotto e il suo processo di sviluppo?

## **4. Il prodotto deve presentare le caratteristiche che ci si aspetta da una realizzazione professionale**

- Per esempio, il costo di manutenzione di un prodotto sw deve essere contenuto e corrispondente alle aspettative del cliente

# I test garantiscono la qualità

*Qualsiasi codice «consegnato» senza i suoi test è fatto male.*

*Non importa se è ben scritto; non importa se è orientato agli oggetti o ben formattato.*

*Se abbiamo i test, possiamo modificare il comportamento del codice rapidamente e in modo controllabile.*

*Senza test non sappiamo davvero se il nostro codice peggiora o migliora.*

# Qualità del software

- Qualità funzionale: grado di soddisfazione dei **requisiti funzionali**, valutato mediante verifica (**testing**)
- Qualità strutturale: grado di soddisfazione dei **requisiti non funzionali**, valutato mediante **validazione**

# Attributi di qualità di prodotto

- Legati alle caratteristiche operative
  - Correttezza (soddisfa la sua specifica funzionale?)
  - Affidabilità (correttezza nel tempo)
  - Efficienza (spreca risorse?)
  - Integrità (danneggia dati o risorse?)
  - Resilienza (capacità di recupero da situazioni anomale)
- Legati alla capacità di subire modifiche
  - Leggibilità e testabilità
  - Manutenibilità
- Legati all'adattabilità a nuovi ambienti
  - Portabilità
  - Riutilizzabilità
  - Interoperabilità
  - Usabilità (quanto è fruibile con utenti e in contesti diversi?)

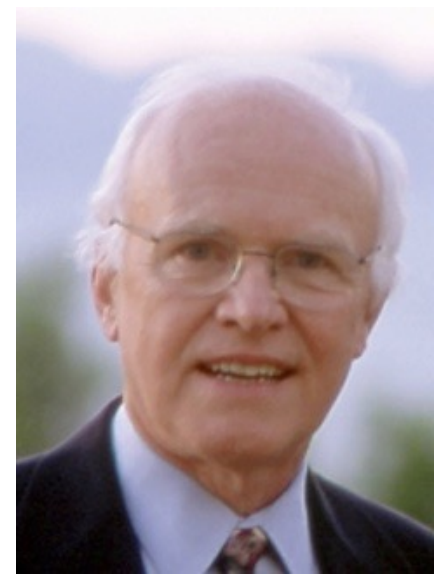


# ISO/IEC 25010:2011 per la software quality

(NB: Questo standard ha sostituito ISO/IEC 9126 )



# Principio di Tom DeMarco

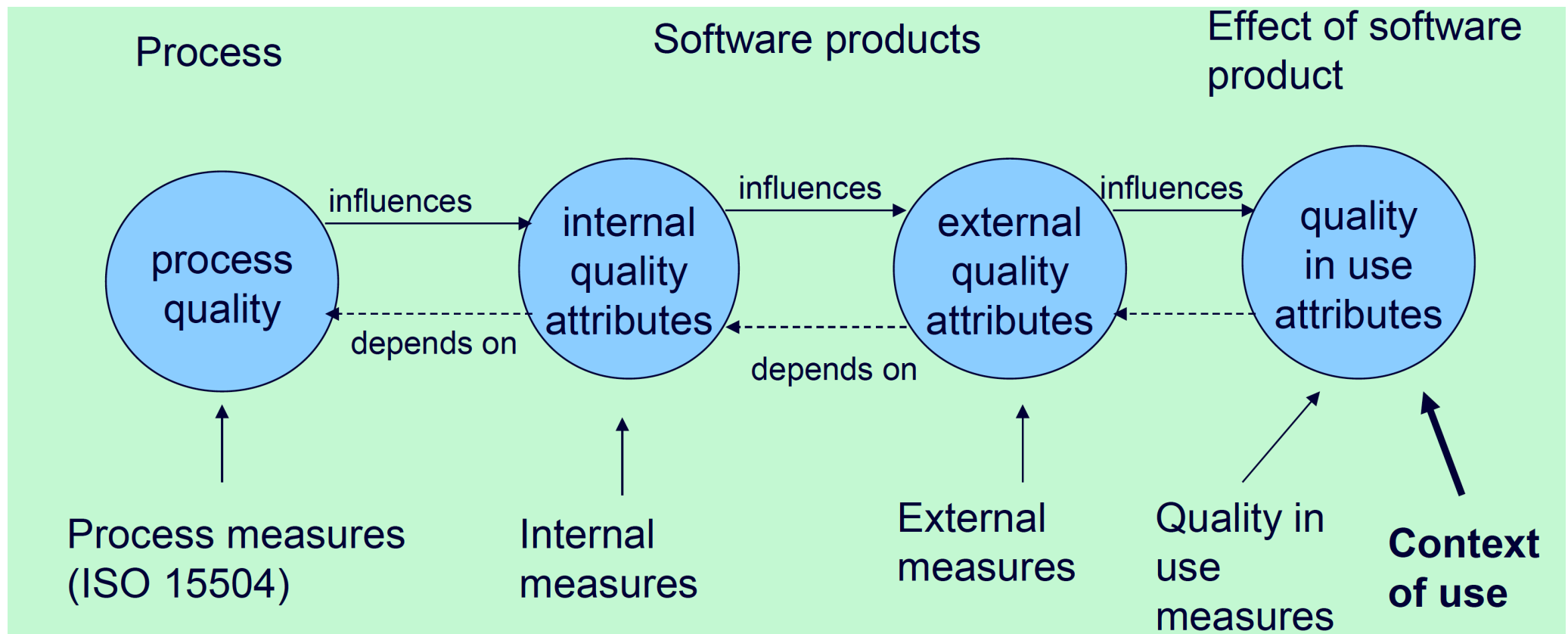


Non puoi controllare quel che  
non puoi misurare!

*Controlling Software Projects, Management Measurement & Estimation, 1982*

# Misurare le qualità del sw

La qualità del software si misura in molti modi



# Misurare la qualità

In un qualsiasi ciclo di vita del software le entità

**misurabili** sono

le **risorse** impiegate (tempo, effort)

alcuni **attributi del processo** (es. produttività)

alcuni **attributi del prodotto** (es. difetti)

# Quanto è importante misurare?

Progetti	Con misurazioni	Senza misurazioni
puntuali	75%	45%
In ritardo	20%	40%
cancellati	5%	15%
Rimozione difetti	95%	<85%
Stima risorse necessarie	Accurata	Ottimistica
Soddisfazione del cliente	Alta	Bassa
Morale del team	Alto	Basso

Fonte: Capers Jones, Measurement, Metrics and Industry Leadership, 2009 e  
Software Engineering Best Practices, McGraw Hill, 2010

# Ragioni per cause legali sul software

<b>Requisiti instabili, modificati continuamente</b>	<b>95%</b>
Controllo di qualità inadeguato, senza misurazioni	90%
Cattivo monitoraggio del processo di sviluppo	85%
Stime sbagliate dei costi o dei tempi	80%
False promesse dei venditori	80%
Stime ottimistiche o arbitrarie	75%
Sviluppo informale, non strutturato	70%
Clienti inesperti incapaci di definire i propri requisiti	60%
Project manager inesperti	50%
Mancato uso di tecniche di analisi statica o ispezioni	45%
Riuso di software con errori	30%
Team di progettisti inesperto o non qualificato	20%

- Sviluppatori individuali: **A chi interessano le misure del sw**
  - Distribuzione dello sforzo
  - Durata e sforzo a preventivo (stimati) e a consuntivo
  - Codice coperto da testing di unità
  - Numero di difetti trovati dal test di unità
  - Complessità del progetto e del codice
- Team di progetto
  - Dimensione del prodotto
  - Distribuzione dello sforzo
  - Stato dei requisiti (#approvati, #implementati, #verificati)
  - % dei casi di test superati
  - Durata stimata e reale tra due milestone principali
  - Livelli di staff stimati e reali
  - #difetti trovati dai test di integrazione e di sistema
  - #difetti trovati dalle ispezioni
  - Stato dei difetti
  - Stabilità dei requisiti (#requisiti modificati durante lo sviluppo)
  - Numero di task pianificati e completati
- Organizzazione che sviluppa software
  - Livelli dei difetti rilasciati (*critical, major, average, minor, exception*)
  - Tempo di ciclo di sviluppo del prodotto
  - Accuratezza della pianificazione e dello sforzo stimati
  - Riuso effettivo
  - Costo preventivato e reale

# Misurare la qualità del prodotto sw

La qualità di un prodotto software è la misura in cui il prodotto soddisfa la sua specifica

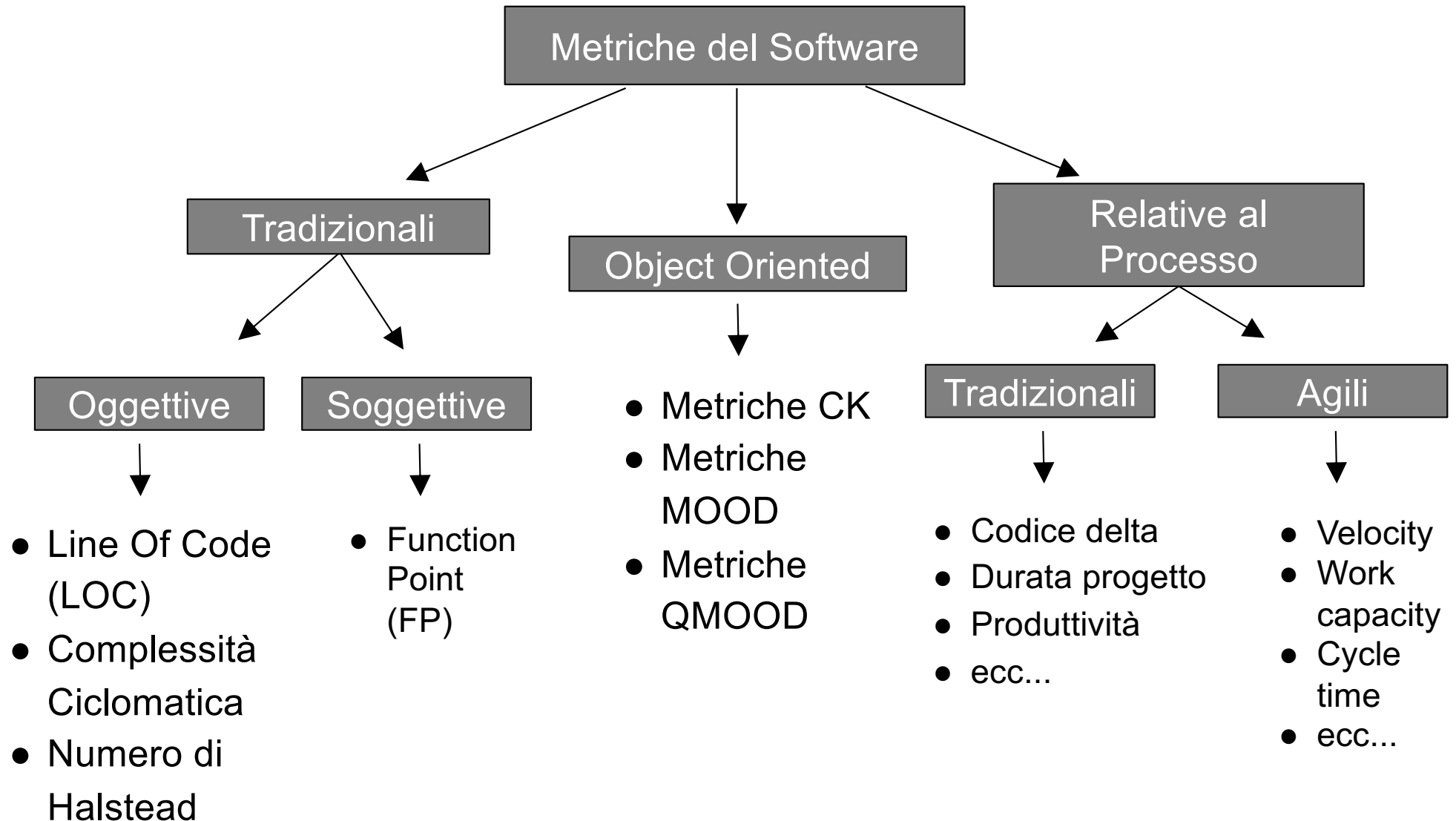
- **Attributi di qualità interni ed esterni**  
attributi **esterni**: visibili all'utente  
attributi **interni**: visibili ai costruttori



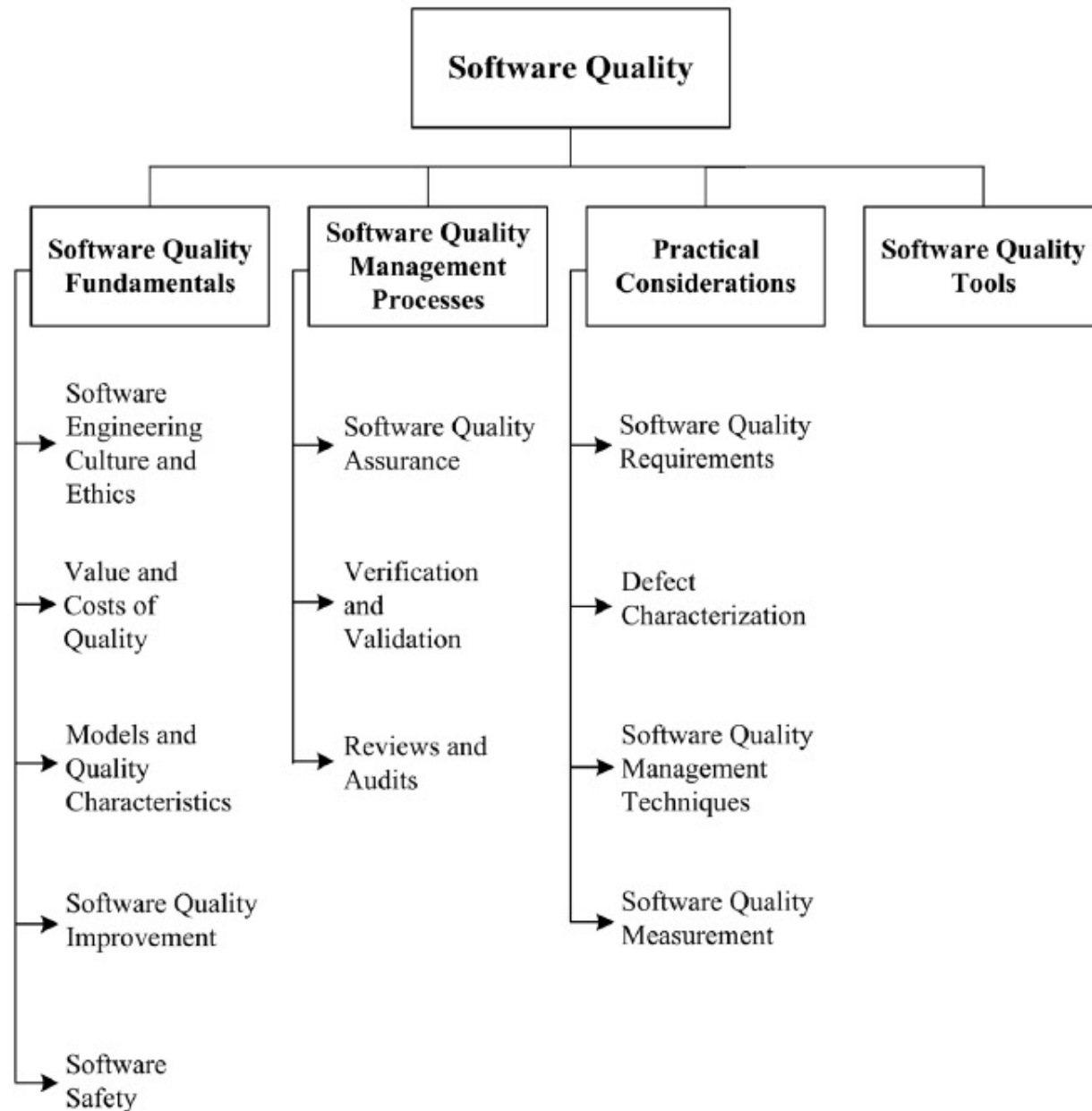
# Esempi di misure usate in Sw Eng

codice	lunghezza	LOC
	funzionalità	Function Points
	complessità	Indice McCabe
specifiche	lunghezza	#pagine
	riuso	#pagine
codifica	sforzo	Mesi/persona
testing	Fase rilevazione	%difetti trovati
	volume	#test schedulati
manutenzione	Costo medio	€/difetto

# Classificazione



# La qualità del sw nel SWEBOK



# Come introdurre la qualità

- Qualsiasi valutazione di qualità inizia dallo *scopo* che ha chi la vuole valutare
- Chi valuta la qualità di un prodotto o processo dovrebbe
  - avere chiari i propri **obiettivi**,
  - legarli a **domande** specifiche sui prodotti o processi oggetto di analisi, e
  - definire **metriche** capaci di analizzare e quantificare le qualità richieste ai prodotti rispetto agli obiettivi

# Goal Question Metric (GQM)

- Metodo di definizione di metriche software
- Sviluppato per valutare i difetti software in progetti NASA, e poi generalizzato
- Passi:
  - Comprendere e analizzare gli obiettivi del progetto o organizzazione
  - Per ciascun obiettivo definire le domande cui occorre dare risposta onde poter capire se gli obiettivi sono stati raggiunti o meno
  - Stabilire quali attributi occorre misurare per poter rispondere alle domande

# GQM

Il metodo **Goal-Question-Metric** (GQM) si usa per definire misure di progetto, processo e prodotto sw in modo che

- La misurazione sia semplice
- I dati di misurazione possano essere usati in modo costruttivo e condiviso
- Le metriche e la loro interpretazione riflettano i valori ed i punti di vista delle diverse parti interessate

<https://www.geeksforgeeks.org/goal-question-metric-approach-in-software-quality/>

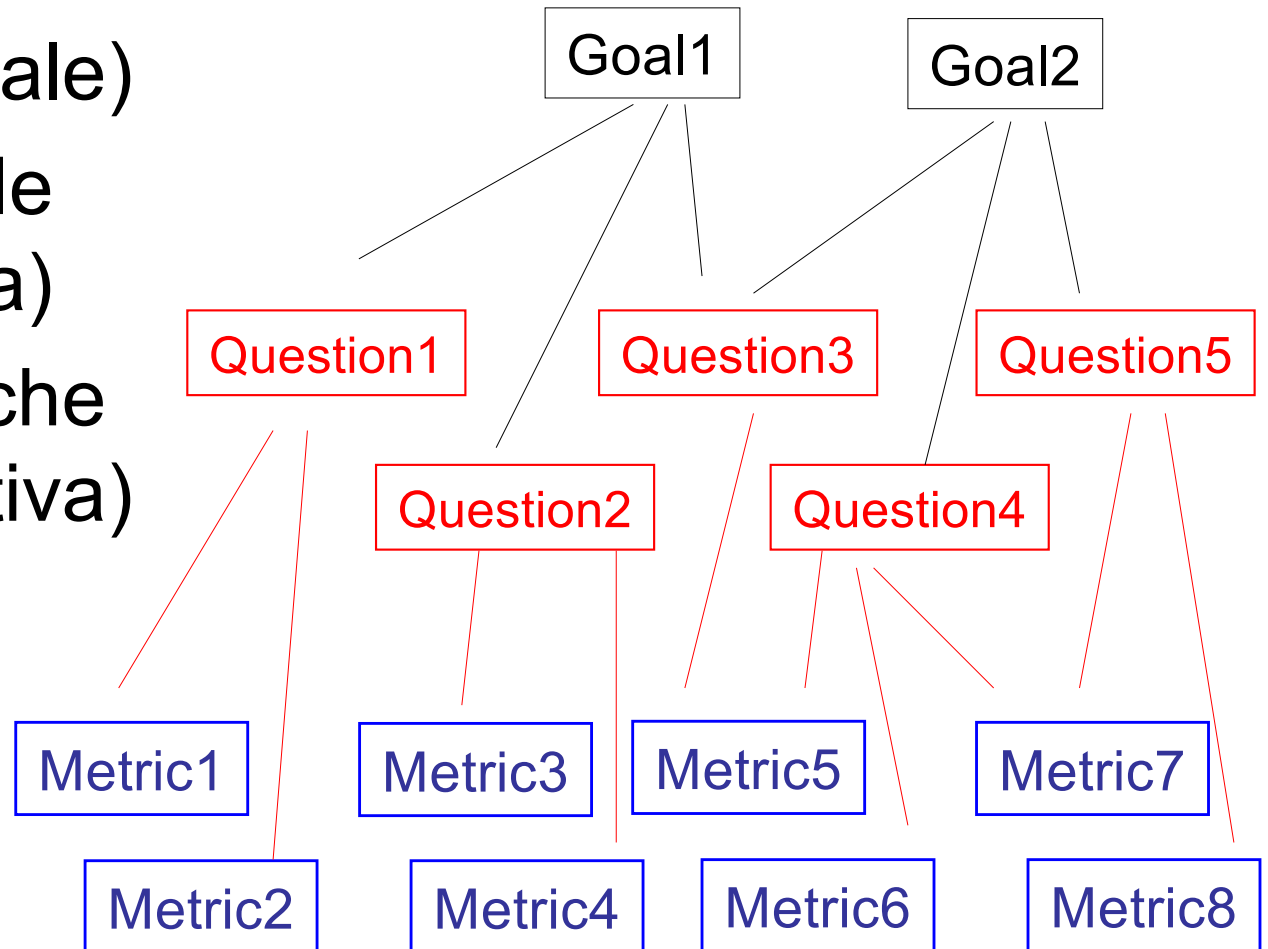
# GQM

GQM si basa su tre livelli:

- Livello concettuale (**goal**): un obiettivo (goal) per un oggetto di studio viene definito rispetto a vari modelli di qualità e da vari punti di vista, relativamente ad un ambiente particolare
- Livello operativo (**question**): Si usa un insieme di domande per definire modelli dell'oggetto di studio atti a valutare l'obiettivo richiesto
- Livello quantitativo (**metrics**): Un insieme di metriche basate sui modelli viene associato ad ogni domanda in modo da caratterizzare le risposte in modo misurabile

# GQM: gerarchia di modellazione

- Prima i goal (fase concettuale)
- Poi le domande (fase operativa)
- Infine le metriche (fase quantitativa)





# Usare GQM. Esempio 1



# Usare GQM: Esempio 2

## Goal:

Valutare l'affidabilità  
del prodotto

## Questions:

Il codice soddisfa  
lo standard di codifica?

Qual è la distribuzione  
degli errori?

Le review  
sono efficaci?

## Metrics:

Per ogni modulo:  
Aderenza allo standard  
(metrica soggettiva)

#errori  
(failures)

Per ogni errore:  
classificazione della gravità

Per ogni errore:  
classificazione del rilevatore

#errori  
(faults)

Per ogni modulo:  
Rivisto? (sì/no)

# Usare GQM

- GQM si può usare in tutte le le fasi di sviluppo sw
- Si può applicare ai progetti, ai processi ed ai prodotti
- Le metriche che vengono definite debbono correlarsi agli obiettivi dell'organizzazione
  - Le misurazioni dovrebbero essere utili e costruttive, perché l'organizzazione deve imparare analizzandoli
  - Le metriche e le loro definizioni dovrebbero riflettere il punto di vista di diverse parti interessate (es. sviluppatori, utenti, progettisti, ecc.)

# Pianificare un modello di qualità con GQM

1. Sviluppo dei goal e delle misure associate di qualità
2. Generazione di domande che definiscono i goal, quantificandoli
3. Specificare le misure da collezionare in conformità ai goal
4. Sviluppare i meccanismi operativi di collezione delle misure
5. Raccogliere i dati e analizzarli onde sviluppare azioni correttive
6. Analizzare i dati postmortem per raccomandazioni sul futuro

# Esercizio



Pianificare con GQM

- Le misure di qualità di una festa
- Le misure di qualità di chi organizza una festa
- Le misure di qualità di chi partecipa ad una festa

# Discussione

Come usare GQM per valutare la qualità della presentazione che mi manderete?



# Cosa si intende per “qualità”?

## I punti problematici per un sistema software

- La qualità non è soltanto assenza di difetti del prodotto finale
- Spesso i requisiti sulle qualità esterne (es. efficienza, affidabilità) e quelli sulle qualità interne (es. manutenibilità, riusabilità) sono in antitesi, e vanno bilanciati
- Alcuni requisiti di qualità sono difficili da specificare
- Le specifiche sono spesso **incomplete** e a volte **inconsistenti**, ma non possiamo aspettare che le specifiche migliorino prima di preoccuparci della qualità del prodotto finale

# Obiettivi della verifica e della validazione

- **Verifica:**

*Confronto di un prodotto con la sua specifica*

(ovvero, confronto di un prodotto con i suoi requisiti)

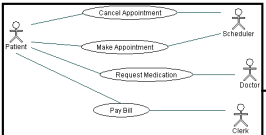
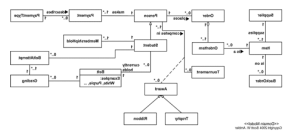
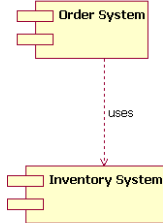


- **Validazione:**

*Accettazione del prodotto da parte del committente*

**Nota:** Secondo alcuni, la “verifica” determina se una certa attività è stata effettuata correttamente, mentre la “validazione” certifica che un prodotto soddisfa i suoi requisiti; non NON usiamo queste accezioni



# Attività di sviluppo del software

Raccolta dei requisiti	Analisi dei requisiti	System Design	Object design	Implementazione	Testing
	espressi mediante	mappati su	realizzati con	definiti da	verificati con
Modello dei casi d'uso				<div data-bbox="1473 1074 1711 1241"> <p>Class A...</p> <p>Class B...</p> <p>Class C...</p> </div>	
	Oggetti del dominio	Sotto-sistemi	Oggetti del sistema	Sorgente	Casi di test

# Verifiche

- L'attività di *verifica*, così come quella di *documentazione*, non è una fase separata del processo
- Tuttavia è opportuno che sia effettuata da persone diverse da quelle coinvolte nel design o nella codifica
- Ogni documento prodotto dovrebbe essere controllato (possibilmente da persone diverse dagli autori del documento) e sistematicamente documentato esso stesso
- Esistono due tipi di verifica: quella basata sull'analisi (**ispezione**) e quello basata sull'esecuzione (**testing**)

# Attività di sviluppo legate alla qualità

- **Testing:** processo di investigazione sui rischi connessi all'esecuzione di un sistema software
- **Misurazione:** di indicatori di qualità, sia mediante ispezione sia mediante esecuzione
- **Verifica:** analisi delle funzioni rispetto alla specifica
- **Validazione:** accettazione da parte degli stakeholder
- **Certificazione:** analisi delle funzioni rispetto ai requisiti di legge da certificare

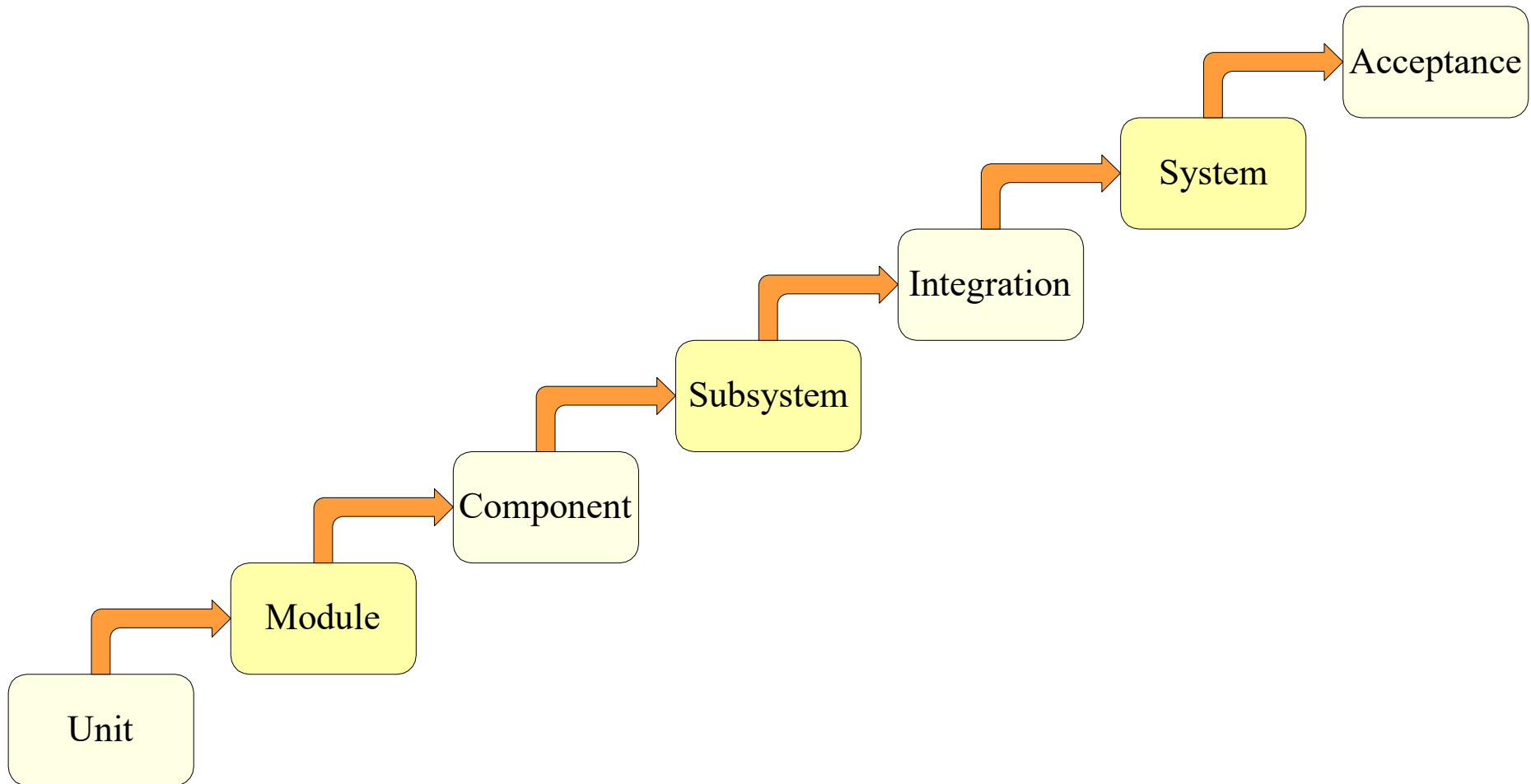
# Testing

- Il testing di un prodotto software è un'attività di processo che ha lo scopo di misurare i rischi connessi all'uso di un prodotto software in esecuzione
- Gli artefatti prodotti dal testing sono i piani di test che includono i casi di test, le suite di test che automatizzano il testing, e i rapporti di test che contengono i risultati dell'attività

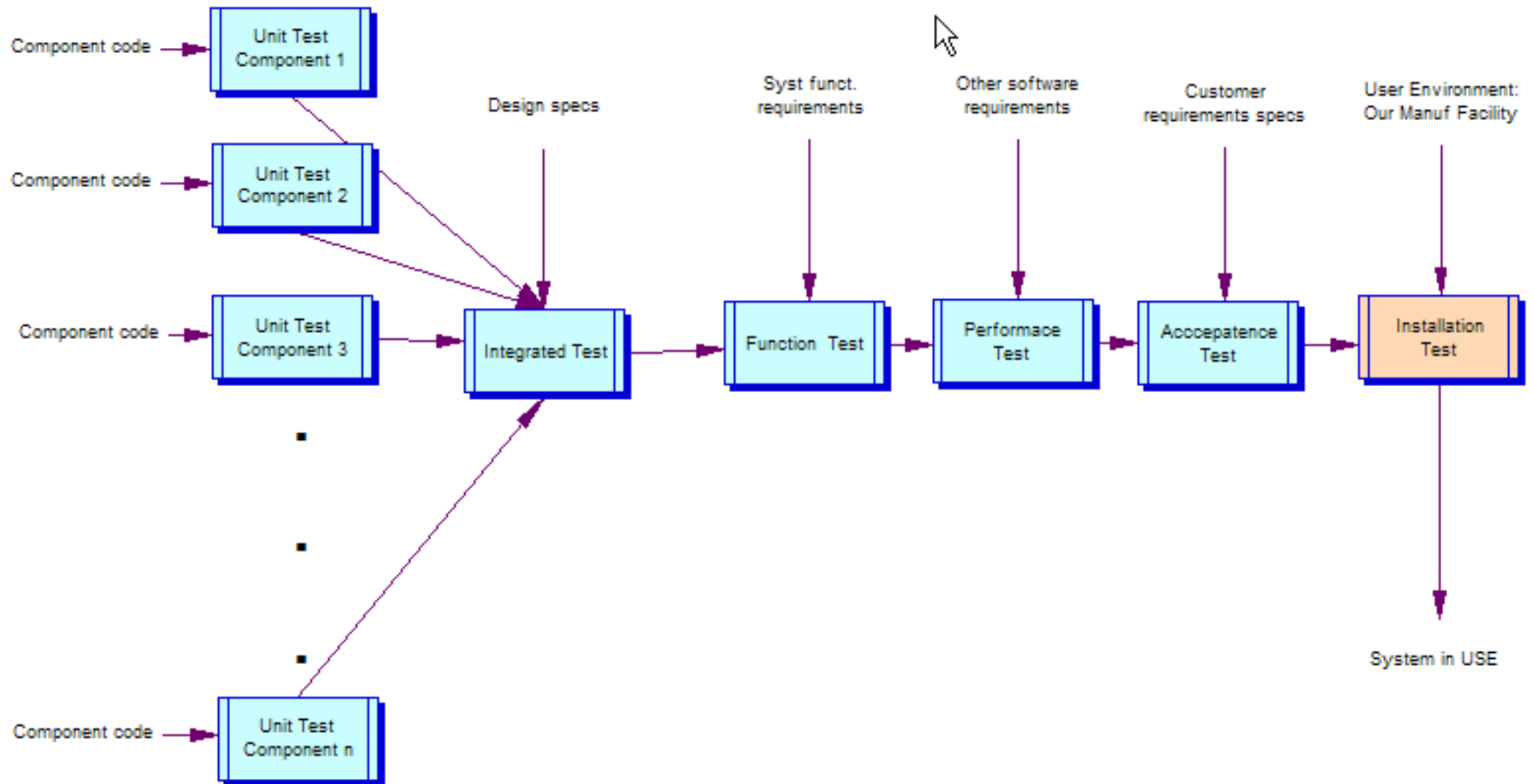
# Fasi/tipi di testing

- **Unit testing**
  - Controllo di componenti individuali
- **Module testing**
  - Controllo di collezioni di componenti correlati
- **Sub-system testing**
  - Controllo di moduli integrati: attenzione alle interfacce
- **System testing**
  - Controllo dell'intero sistema. Controllo delle proprietà “emergenti” (non attribuibili a singole componenti)
- **Acceptance testing**
  - Testing con dati (e presenza) del cliente per controllare che il comportamento del sistema sia accettabile

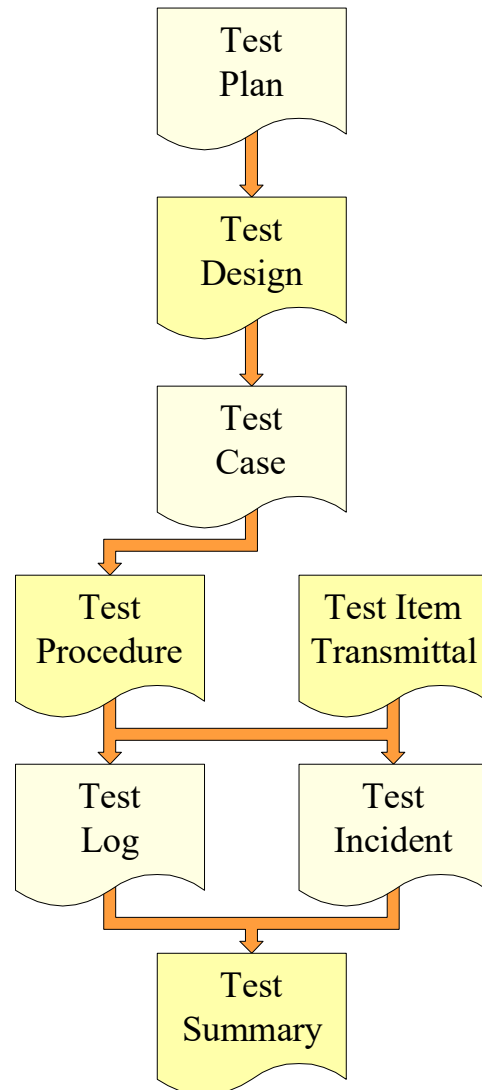
# Tipi di testing



# Tipi di testing



# Artefatti di testing



**Test plan** artefatto che descrive in dettaglio obiettivi, risorse e processi di test specifici per un prodotto software

**Test design** artefatto che guida la realizzazione dei test

**Test case** insieme di condizioni o variabili sotto le quali un tester determina se una applicazione o sistema software risponde correttamente o meno

**Test procedure** specifica formale dei casi di test da applicare ad un prodotto

**Test item transmittal report** artefatto che identifica stato e metadati degli elementi di un test

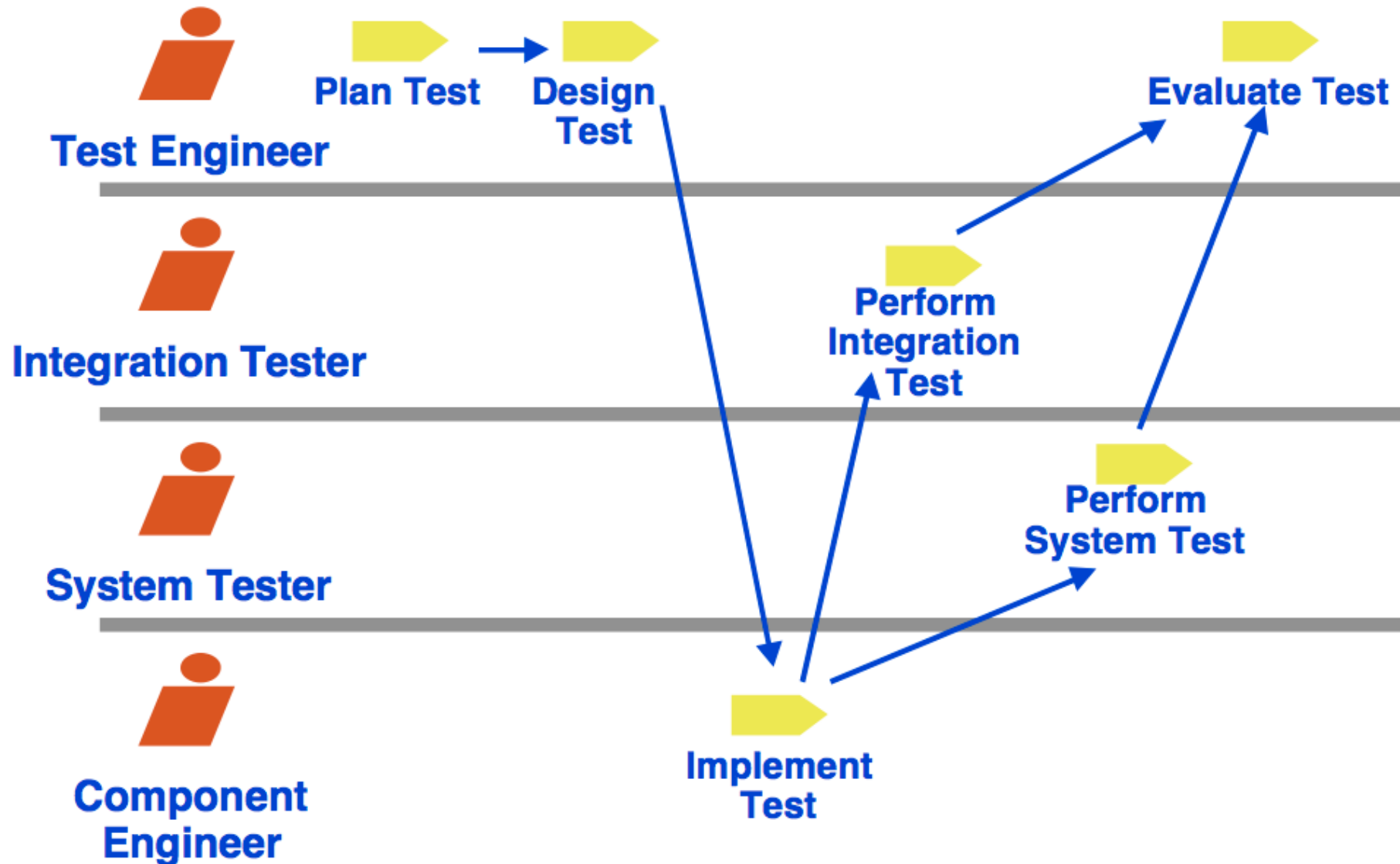
**Test log** registrazione dei risultati di un test, con il risultato: passato o fallito

**Test incident** Quando il risultato del test è inatteso, allora prende il nome di incidente (può essere un errore, un difetto, un problema)

**Test summary** Riassunto dei risultati di un test



# Workflow di testing



# Testing: c'è sempre un altro errore

“Il testing si può usare per provare la presenza di errori in un programma, mai per dimostrarne l'assenza.”

—Edsger W. Dijkstra, 1972

# Testing

“Se l’obiettivo è mostrare l’**assenza**  
di errori, ne troveremo pochi;  
Se l’obiettivo è mostrare la **presenza**  
di errori, ne troveremo molti.”

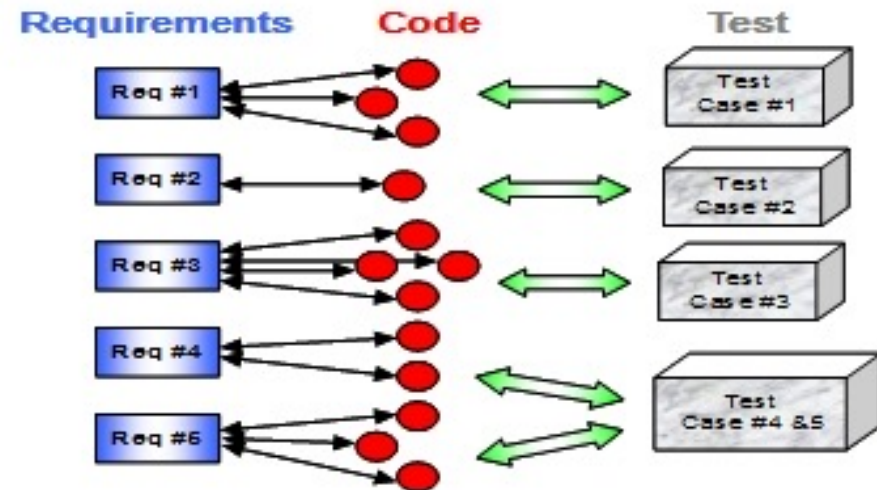
—G. J. Meyers, 1979

# Testing in SWEBOK (cap 5)



# Requisiti e test

- Il testing ha lo scopo di verificare che il codice soddisfi i requisiti



# Il testing è difficile

- Non sempre i requisiti sono chiari
- Molti sviluppatori non conoscono né le tecniche di testing né i relativi strumenti
- Il testing viene ritenuto “noioso” o “costoso”
- Spesso non c'è tempo per testare bene “tutto”

**Nota bene:** Il testing sistematico ed esaustivo è computazionalmente *intrattabile*; dobbiamo però fare ogni sforzo per eliminare ogni possibile difetto o fattore di rischio

# Noi non scriviamo i test...

- Noi non scriviamo i test.
- Perché?
- Perché non abbiamo tempo.
- Perché?
- Perché c'è troppo lavoro e molta pressione per finire.
- Perché?
- Perché non siamo abbastanza veloci.
- Perché?
- Perché modificare il software è difficile e rischioso.
- Perché?
- Perché noi non scriviamo i test.

# Definire un piano di test

- Completo (ma non eccessivo)
  - Usare una matrice di test
  - Includere test positivi e negativi
- Trovare i casi di test “interessanti”
- Eseguire ogni test almeno due volte
- Aggiornare il piano durante lo sviluppo
  - Aggiungere test di regressione
  - Sfruttare il feedback degli utenti
- Guardarsi dall'obsolescenza del prodotto e dei suoi test

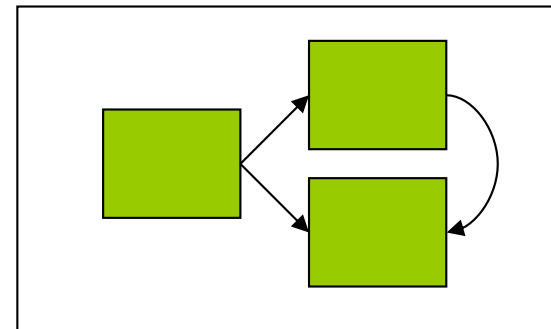
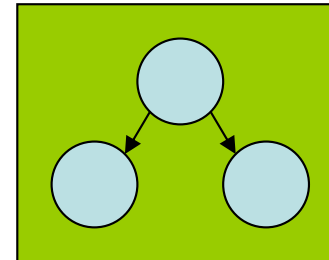


### Sample traceability matrix

[illegible]

# Tipi di test

- Unità o modulo
  - Classi o tipi individuali
- Componente
  - Gruppo di classi correlate
- Integrazione
  - Interazione tra componenti

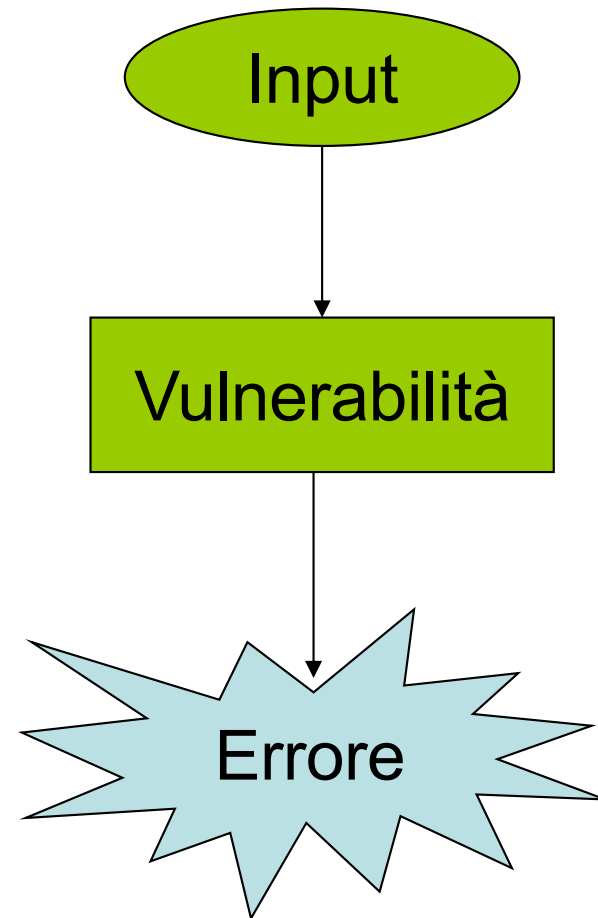


# Progettare i test

- I test sono “attacchi” al software condotti per vedere se ci sono difetti o rischi
- Metodi di test
  - Diretti
  - Indiretti
- Cercare e colpire i punti vulnerabili
  - Black box
  - White box

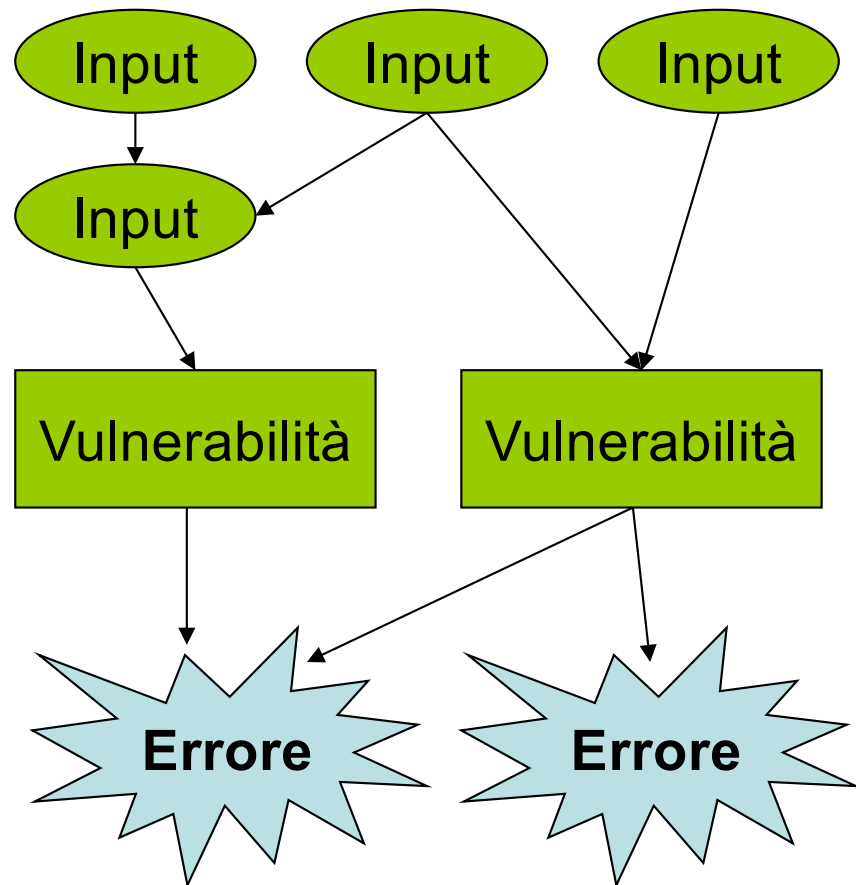
# Testing diretto

- Di solito automatico
- Di basso livello
- Solo funzionalità base
- Sfrutta le specifiche



# Testing indiretto

- Di solito manuale
- Di alto livello
- Scenari “realistici”
- Scopre molti errori imprevisti

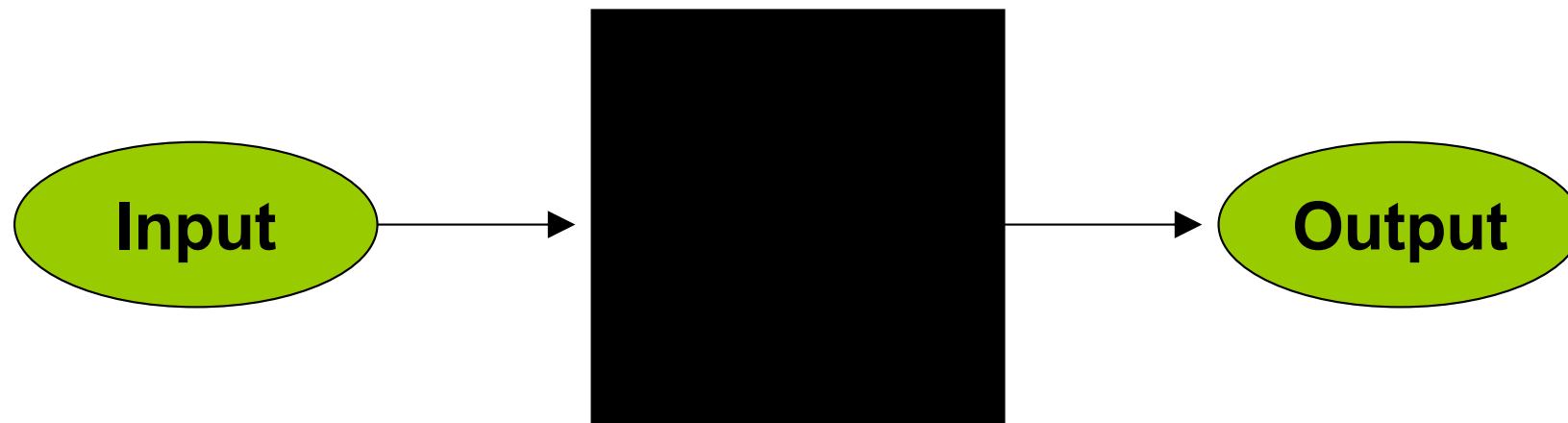


# Testing di un modulo

- **Black-box testing** (funzionale, data-driven, I/O-driven): i casi di test sono definiti nel documento di specifica; il codice del prodotto viene ignorato durante il testing.
- **White-box testing** (glass-box, logic-driven, path-oriented): i casi di test sono definiti sul codice sorgente.
- Nota bene: *Eseguire il testing esaustivamente (cioè in tutti i casi possibili) è impossibile*

**Esempio (I/O-driven testing):** se l'applicazione da testare ha 10 parametri di ingresso, e ciascuno può assumere 5 valori, occorre testare  $10^5$  casi di input.

# Black Box



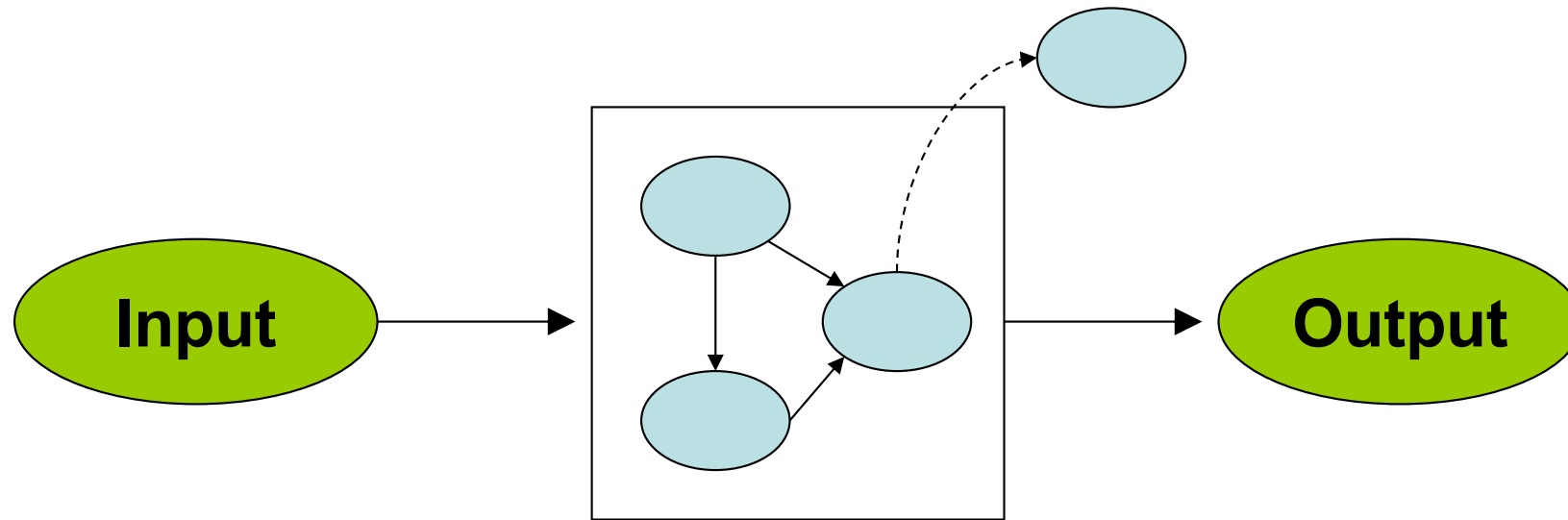
- Si sfrutta la semplicità apparente del software
  - Si fanno assunzioni sull'implementazione
  - Adatto per testare le interazioni tra componenti
- Testa le interfacce ed il comportamento

# Tecniche di black-box testing

- Il black-box testing si effettua **senza** conoscere il sorgente.
- Consiste nell'usare la specifica del prodotto per predisporre un insieme di casi di test che massimizzi la probabilità di trovare un errore e minimizzi quella che due test diversi trovino lo stesso errore
- **Equivalence testing with boundary value analysis:** i possibili dati di input e/o di output (definiti dalla specifica) possono talvolta essere partizionati in classi di equivalenza; in questo modo si diminuiscono i casi di test necessari e si possono usare i “boundary values” (valori di confine di partizione) per guidare il test.
- **Functional testing:** dopo aver identificato nella specifica tutte le funzioni di un modulo, si definiscono i dati di test necessari per controllare ciascuna funzione separatamente.



# White Box

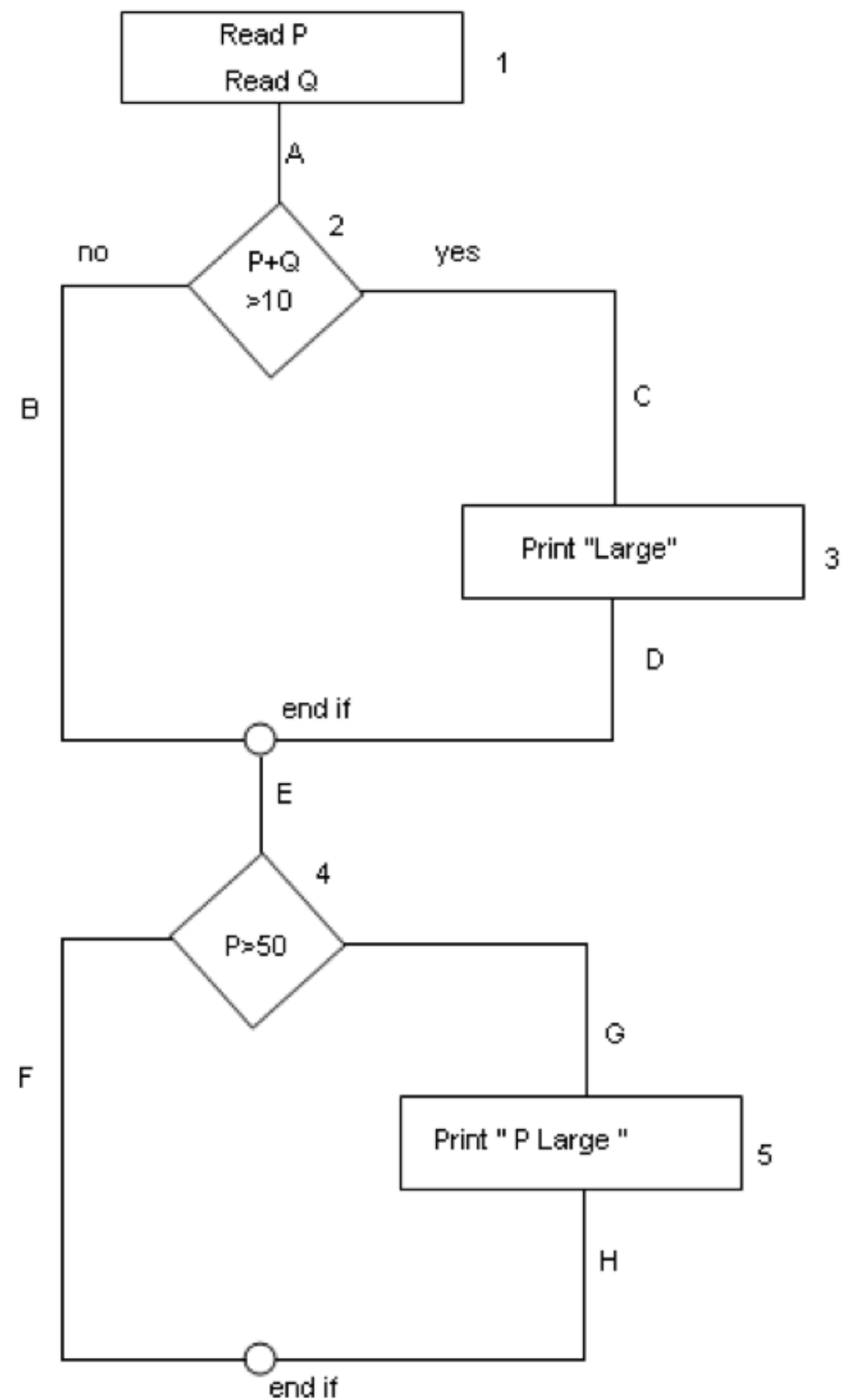


- Sfrutta la complessità interna del software
  - Occorre completa conoscenza dell'implementazione
  - Adatta a testare funzioni singole
- Testa l'implementazione ed il progetto

# Tecniche di white-box testing

- **Statement coverage:** ogni comando viene eseguito almeno una volta (ma non c'è garanzia che ogni condizione venga testata).
- **Branch coverage:** ogni comando viene eseguito almeno una volta ed ogni condizione viene testata.
- **Path coverage:** ogni cammino possibile viene testato almeno una volta.

*read P*  
*read Q*  
*if  $P+Q > 10$  then*  
*print "Large"*  
*endif*  
*if  $P > 50$  then*  
*print "P Large"*  
*endif*



# Teorema di Weyuker

Dato un generico programma  $P$  sono indecidibili:

- Esiste un dato di input di  $P$  che causa l'esecuzione di un particolare comando? **indecidibile**
- Esiste un dato di input che causa l'esecuzione di una particolare condizione (branch)? **indecidibile**
- Esiste un dato di input che causa l'esecuzione di un particolare cammino in  $P$ ? **indecidibile**

È però possibile individuare sottoproblemi decidibili

# Criteri di copertura del codice

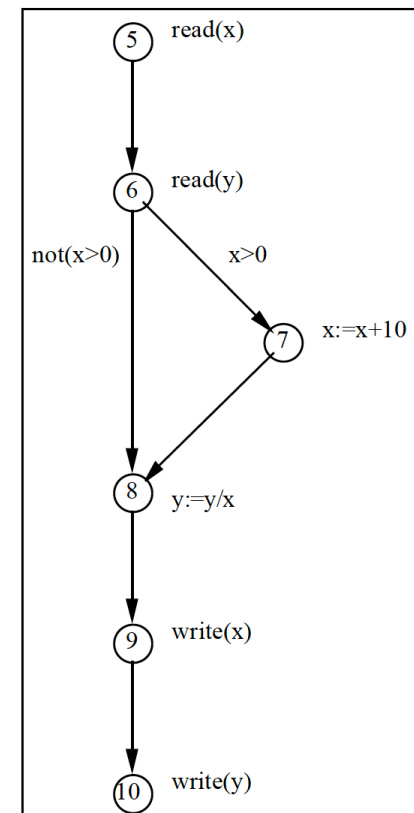
- copertura dei comandi
- copertura delle decisioni
- copertura delle condizioni
- copertura delle decisioni e delle condizioni
- copertura dei cammini
- di n-copertura dei cicli

# Copertura dei comandi

Criterio di copertura dei comandi (statement test): un test  $T$  soddisfa il criterio di copertura dei comandi se e solo se ogni comando eseguibile del programma è eseguito in corrispondenza di almeno un dato di test in  $T$

Nel grafo di controllo del programma, ogni nodo corrispondente ad un comando eseguibile deve essere percorso almeno una volta

```
1 Program statement (input, output);  
2   var  
3     x,y : real;  
4   begin  
5     read(x);  
6     read(y);  
7     if x > 0 then x:=x+10;  
8     y:=y/x;  
9     write(x);  
10    write(y);  
11  end.
```



# Sviluppo guidato dal testing: XP

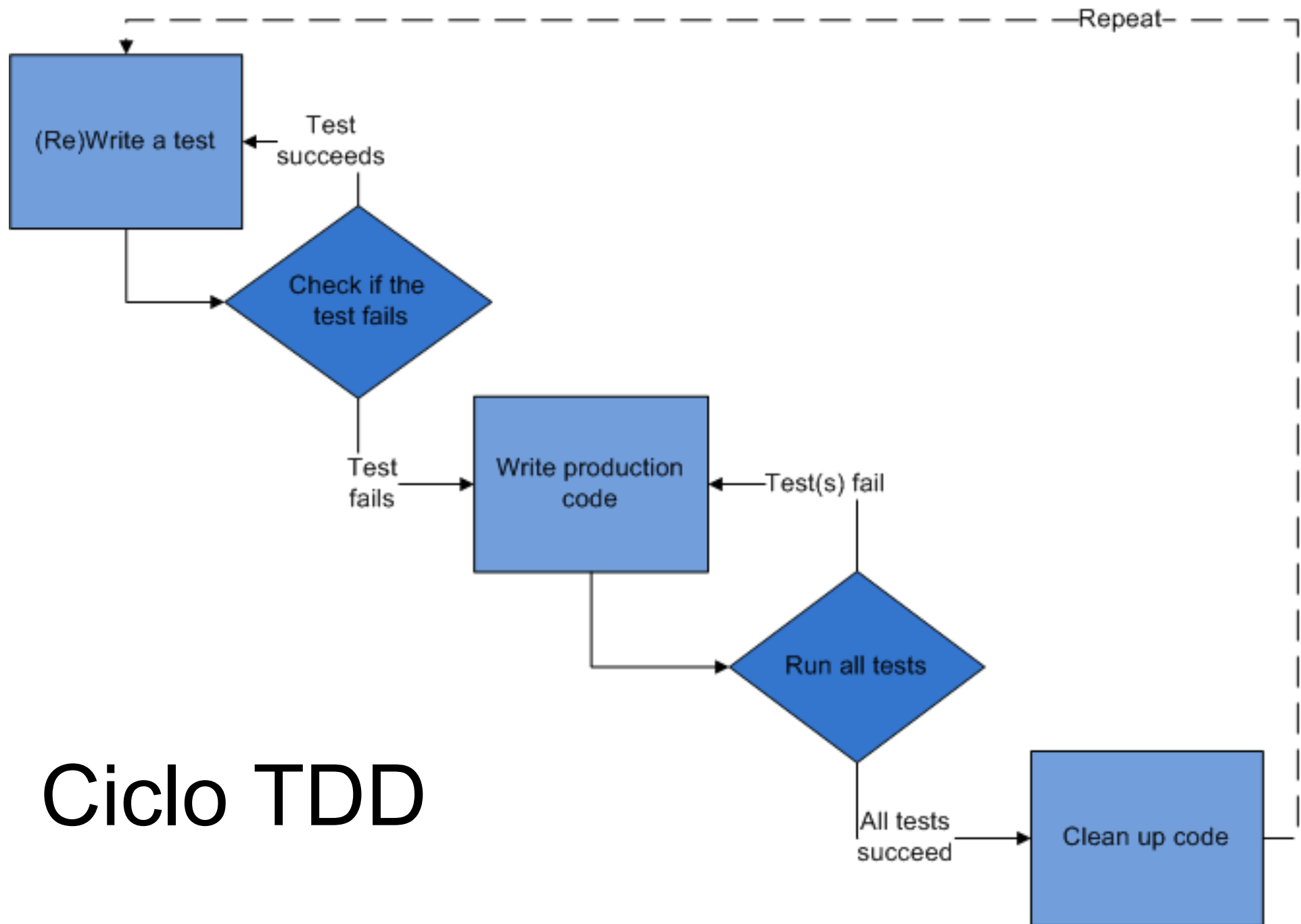
Il metodo XP mette il testing **prima** della codifica (*Test Driven Development: TDD*): nessun modulo viene sviluppato prima che sia stato definito almeno un test di correttezza

## Ciclo:

Test **rosso**: test di nuova funzione vuota; test che fallisce perché la funzione non esiste;

Test **verde**: codice necessario per passare il test rosso;

**Refactoring**: modifiche al codice di test verde per semplificarlo



# Ciclo TDD

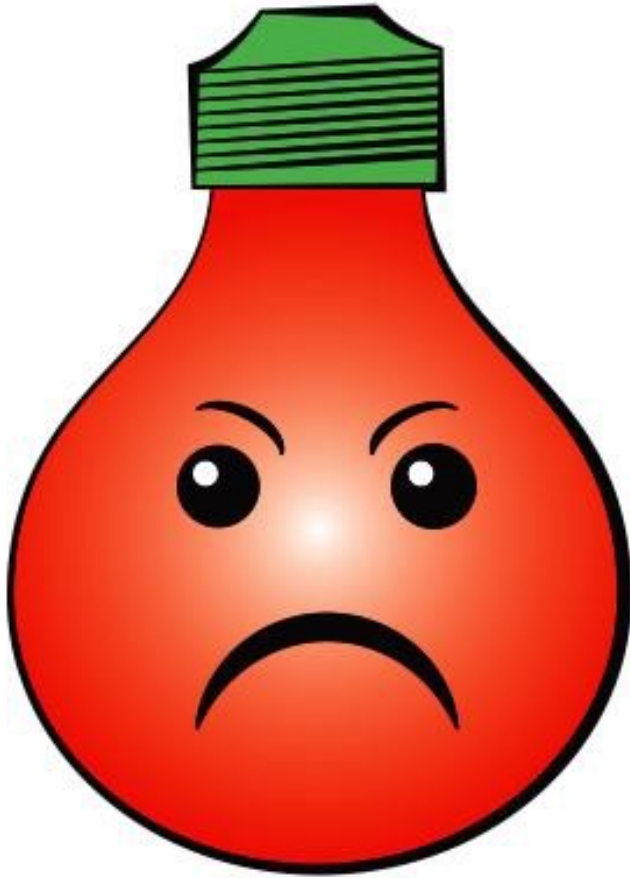


# Riferimenti

Vedi libri:

Beck, *Test Driven Development by Example*

Freeman, *Growing OO Software, guided by tests*

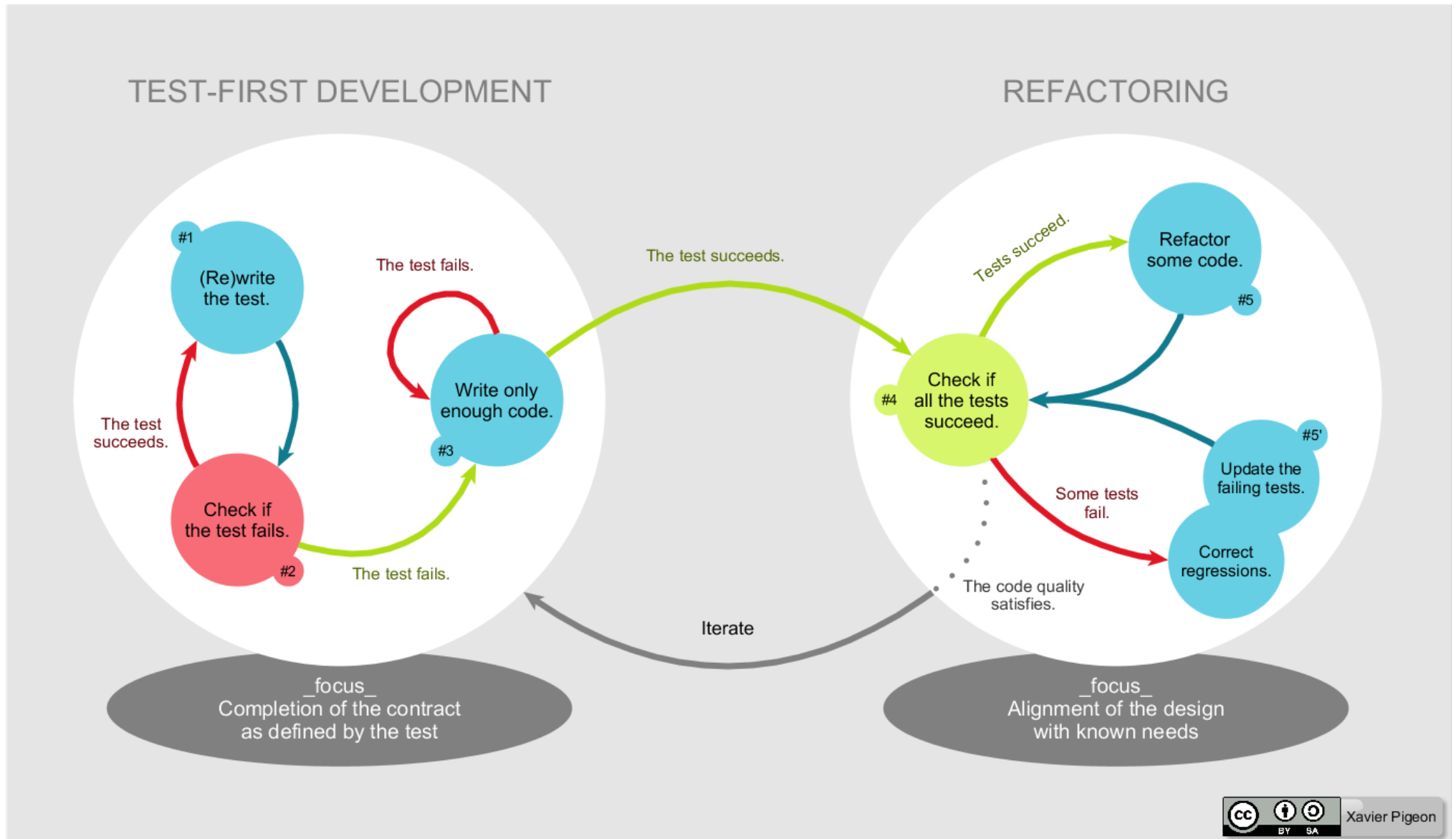


**Debugging  
Sucks!**



**Testing  
Rocks!**

# Test-first + refactoring



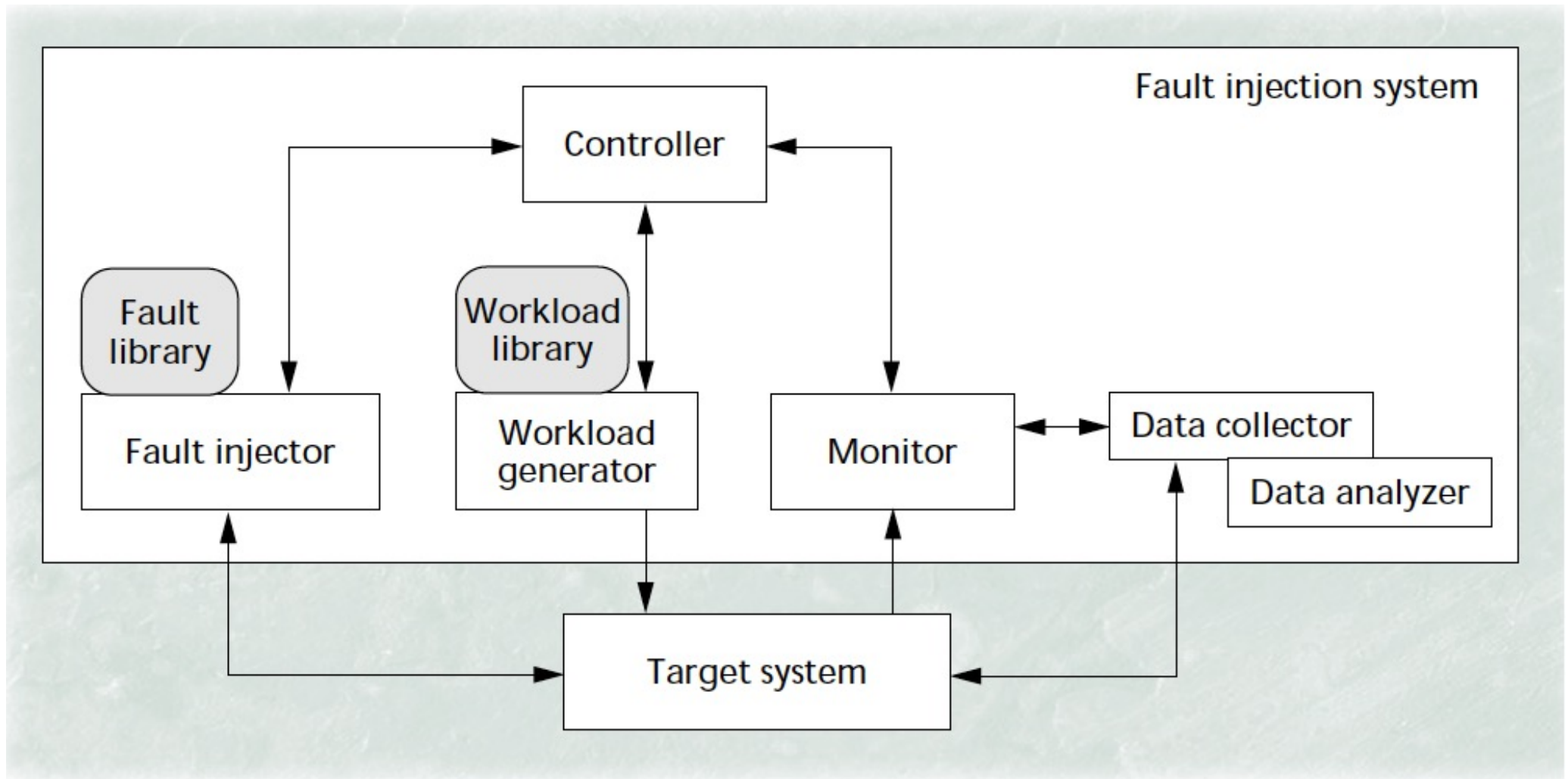
# Crowdsourced testing

- Test gestito da non specialisti mediante infrastruttura cloud
- Utilizzato per software user-centric specie per valutarne l'usabilità
- Si pagano solo gli errori trovati

# Fault injection

- Una tecnica per migliorare il test coverage introducendo errori per testare in particolare il codice che gestisce tali errori, che altrimenti non sarebbe eseguito
- Esempio: nel test di un kernel di sistema operativo si può inserire un driver che intercetta le system calls e ritorna a caso un errore per alcune delle calls.

# Fault injection



Hsueh, Mei-Chen, Timothy K. Tsai, and Ravishankar K. Iyer. "Fault injection techniques and tools." *Computer* 30.4 (1997): 75-82.

# Strumenti di testing

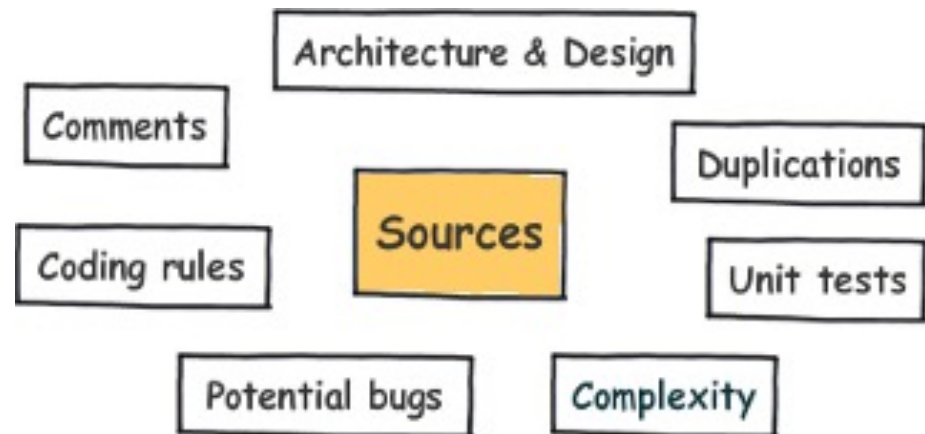
- Junit, xunit
- Jenkins
- Selenium
- sonarqube

# SonarQube: analisi statica

SonarQube è una piattaforma aperta, estendibile con plugin

Serve per analizzare grosse codebase, segnalando bugs e smells

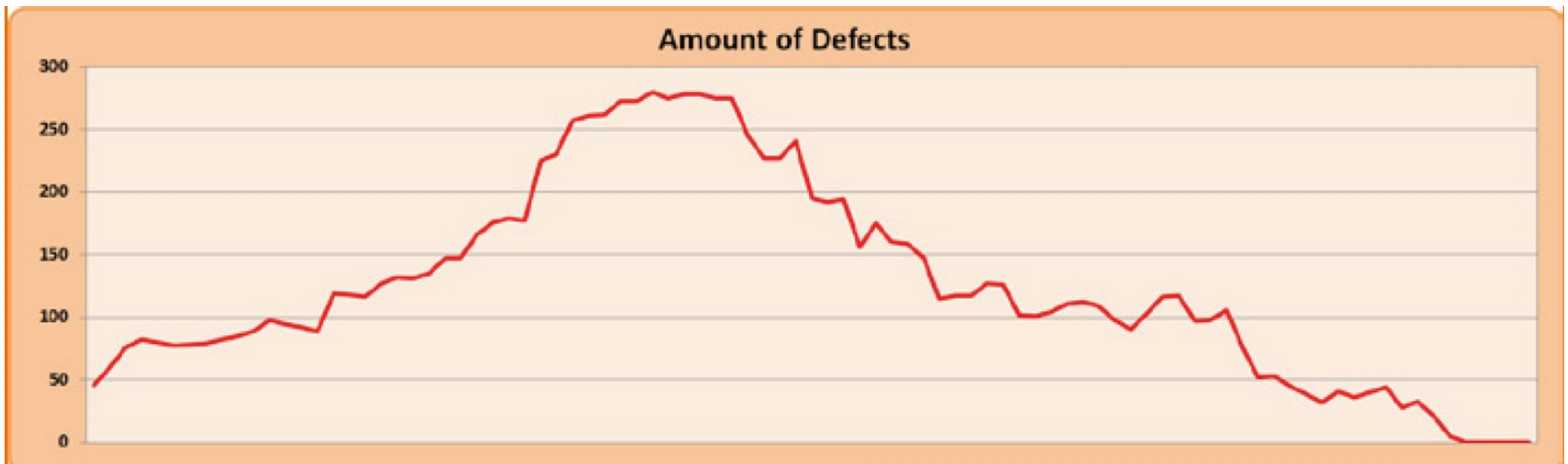
Copre più di 20 diversi linguaggi





# Il debito tecnico

*Debito tecnico* è un'espressione che designa le conseguenze di accettare deliberatamente una soluzione di sviluppo non ottimale per risparmiare tempo, che però sarà necessario "spendere" prima o poi, dopo il primo deployment, magari con gli interessi

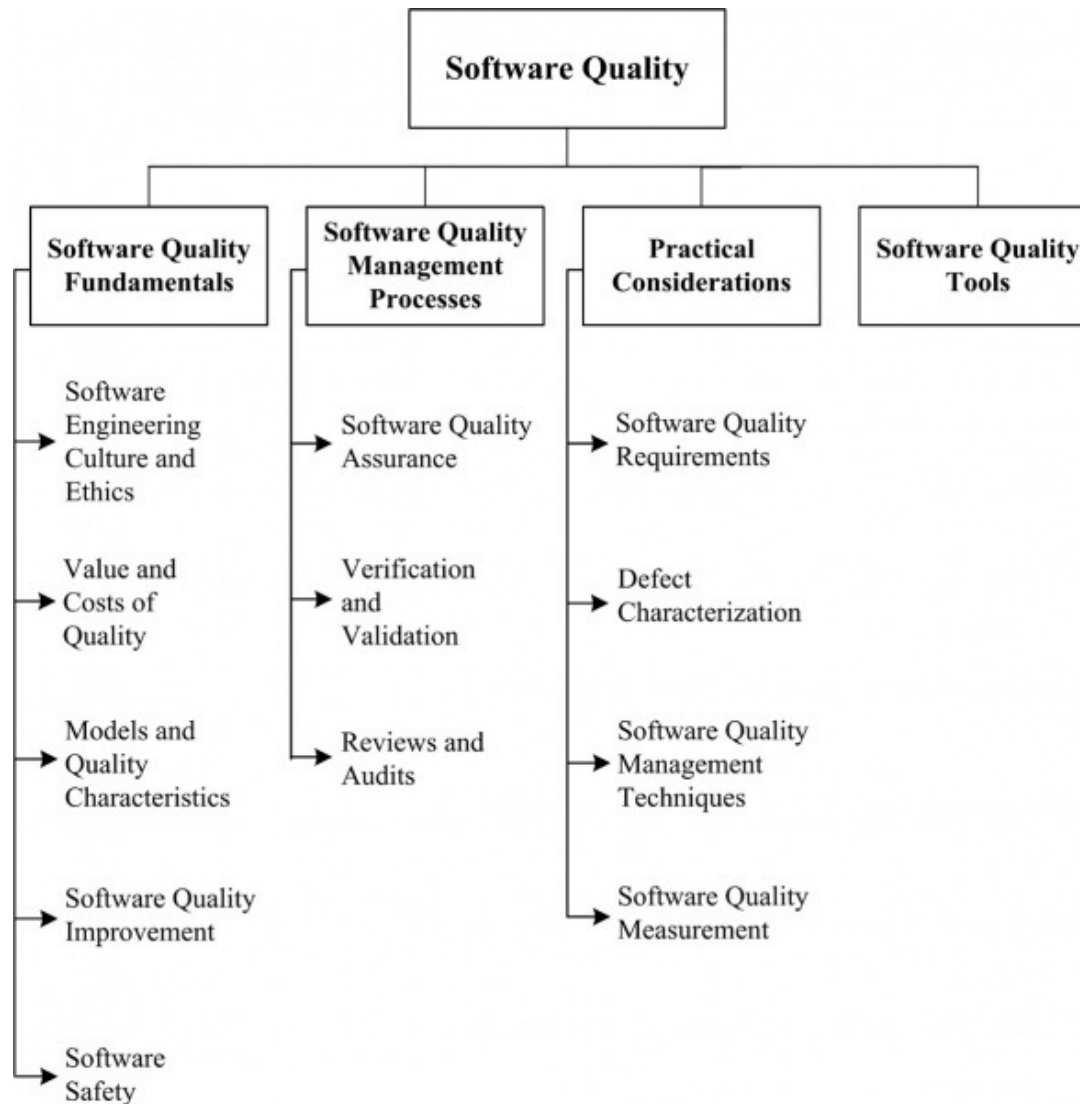


Esempio di andamento del debito tecnico in un anno di sviluppo

# Conclusioni

- La qualità del software si misura sia rispetto ai requisiti funzionali sia a quelli extrafunzionali, inclusa la qualità del sorgente
- Il testing misura soprattutto la qualità funzionale rispetto ai requisiti
- Le qualità extrafunzionali misurabili sono moltissime: includono per es. le prestazioni, la modificabilità, la manutenibilità, ecc.
- La qualità del codice (es. leggibilità, modificabilità, ecc.) si misura con strumenti appositi, capaci anche di calcolare il debito tecnico

# Sw quality in SWEBOK



# Domande di autotest

- Qual è la prima cosa da fare quando qualcuno segnala che un prodotto software ha un bug?
- Come si definisce la “qualità” di un prodotto software?
- Come si possono misurare gli attributi di qualità del software, quali la correttezza, la manutenibilità, l'affidabilità?
- Che differenza c'è tra verifica e validazione?
- Usando l'approccio GQM, come si potrebbe controllare la qualità di un processo di sviluppo? e quella di una specifica UML?
- Quali sono le tecniche di “white box” testing?

# Riferimenti

- Capitolo 5 del SWEBOK. “Software testing”, 2014
- Capitolo 11 del SWEBOK. “Software quality”, 2014
- Hutcheson, *Software Testing Fundamentals*, Wiley, 2003
- McConnell, *Code Complete*, MS Press 2004
- McGregor & Sikes, *A Practical guide to Testing OO Software*, Addison Wesley, 2001
- V.Basili, G.Caldera, D.Rombach: The Goal Question Metric Approach, *The Encyclopedia of Software Engineering*, 1994
- Galin, *Software Quality*, IEEE 2018
- N.Davis, W.Humphrey et al.: "Processes for producing secure software", *IEEE Security and Privacy Magazine*, 2004

# Siti utili

- [asq.org](https://asq.org) community
- `JUnit.org`, `xunitpatterns.com` Unit testing
- `Sonarqube.org` strumento di analisi statica
- [cwe.mitre.org/top25](https://cwe.mitre.org/top25) i 25 errori più comuni
- `docs.seleniumhq.org/` testing automatico per applicazioni web
- [www.aivosto.com](https://www.aivosto.com) strumenti di analisi di errori

# Publicazioni di ricerca sul testing e la qualità del software

- IEEE Int. Conf. on Empirical Software Engineering and Measurement
- IEEE Int. Conf. on Software Testing, Verification and Validation
- Int. Conf. on Software Quality
- Software Quality Journal

# Domande?





# L'evoluzione del software



*Prof. Paolo Ciancarini*  
*Corso di Ingegneria del Software*  
*CdL Informatica*  
*Università di Bologna*

# Obiettivo della lezione

- Tipi di evoluzione del software
- La manutenzione nel ciclo di vita del sw
- Impatto economico della manutenzione
- Strumenti di manutenzione
- Metriche utili per la manutenzione

# Evoluzione del sw: sinonimi

- Reingegnerizzazione
- Manutenzione
- Aggiornamento
- Adattamento
- Refactoring
- Trasformazione architetturale

# Modificare il codice

Per modificare un pezzo di software,  
bisogna poterlo leggere

Leggere un programma è più difficile che  
scriverlo

Modificarlo **\*senza introdurre errori\*** è  
ancora più difficile, se non si hanno i test  
di regressione

# Cosa fa questo codice?

```
#include "stdio.h"
#define e 3
#define g (e/e)
#define h ((g+e)/2)
#define f (e-g-h)
#define j (e*e-g)
#define k (j-h)
#define l(x) tab2[x]/h
#define m(n,a) ((n&(a))==a))

long tab1[]={ 989L,5L,26L,0L,88319L,123L,0L,9367L };
int tab2[]={ 4,6,10,14,22,26,34,38,46,58,62,74,82,86 };

main(m1,s) char *s; {
    int a,b,c,d,o[k],n=(int)s;
    if(m1==1){ char b[2*j+f-g]; main(l(h+e)+h+e,b); printf(b); }
    else switch(m1-=h){
        case f:
            a=(b=(c=(d=g)<<g)<<g)<<g);
            return(m(n,a|c)|m(n,b)|m(n,a|d)|m(n,c|d));
        case h:

for(a=f;a<j;++a)if(tab1[a]&&!(tab1[a]%((long)l(n))))return(a);
        case g:
            if(n<h)return(g);
            if(n<j){n-=g;c='D';o[f]=h;o[g]=f;}
            else{c='\r'-' \b';n-=j-g;o[f]=o[g]=g;}
            if((b=n)>=e)for(b=g<<g;b<n;++b)o[b]=o[b-h]+o[b-g]+c;
            return(o[b-g]%n+k-h);
        default:
            if(m1-=e) main(m1-g+e+h,s+g); else *(s+g)=f;
            for(*s=a=f;a<e;) *s=(*s<<e)|main(h+a++,(char *)m1);
    }
}
```

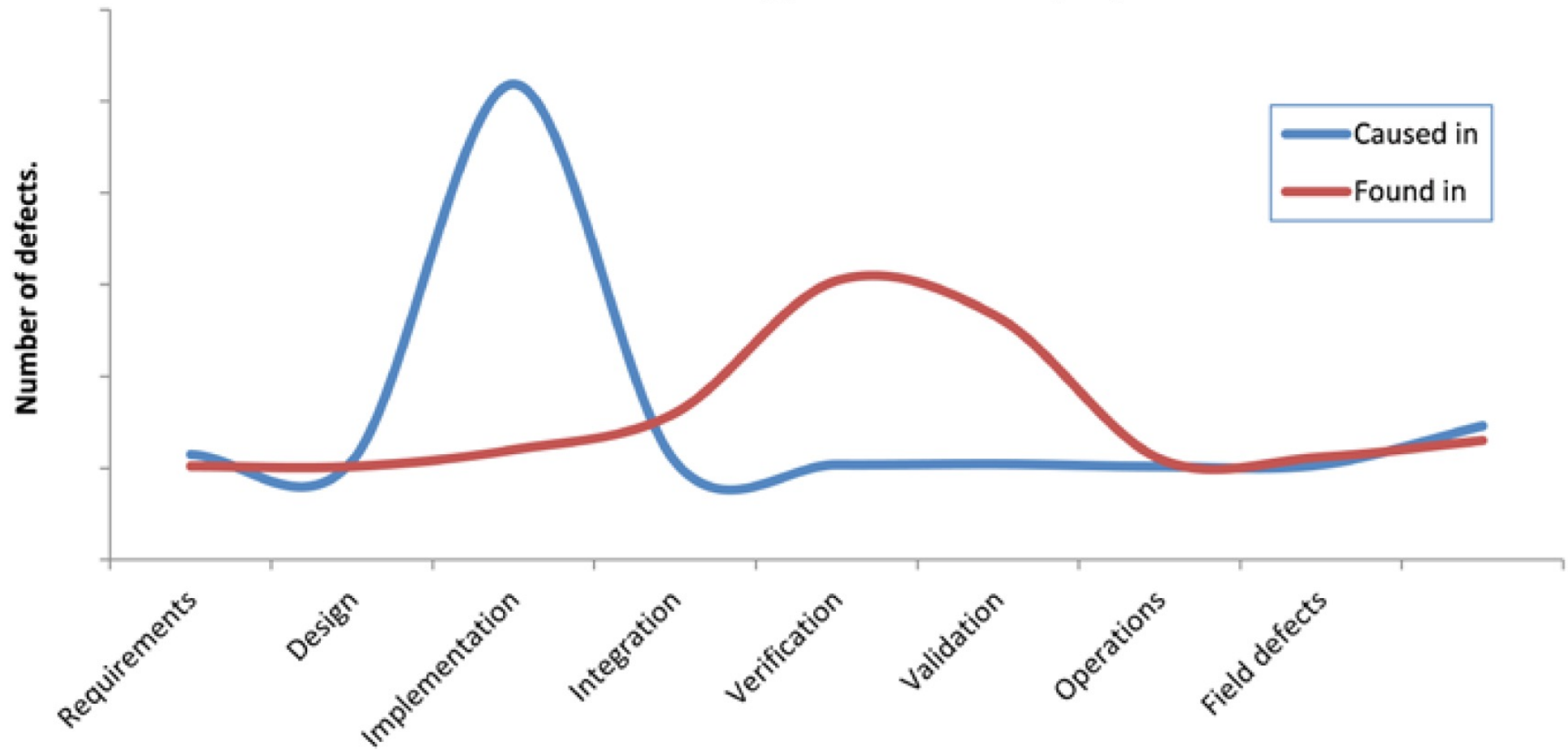
# Programmazione “offuscata”

- Questo programma stampa “Hello World”
- The International Obfuscated C Code Contest  
[www.ioccc.org/](http://www.ioccc.org/)
- Questo è un esempio di codice “altrui” che potremmo dover modificare

# L'evoluzione del software

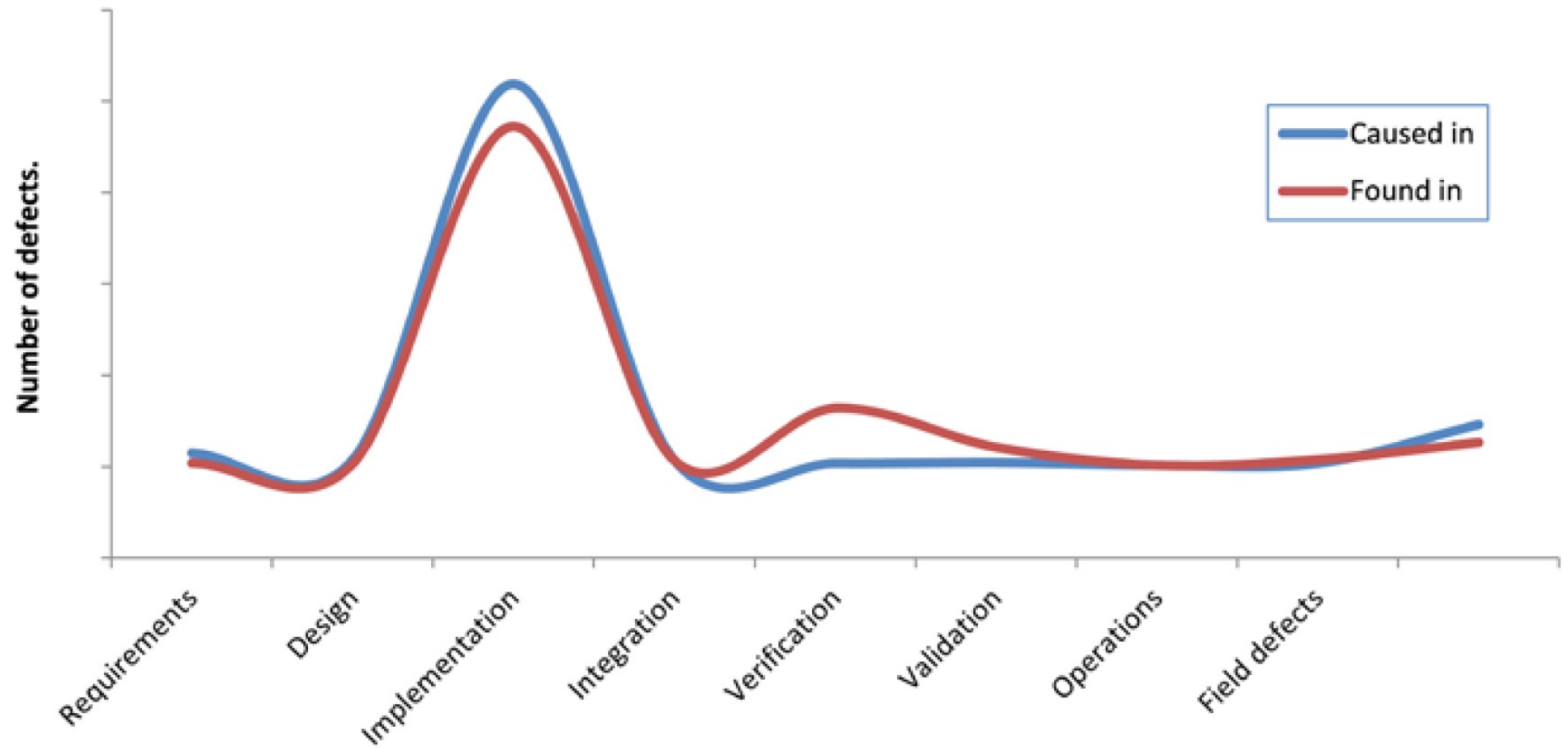
- La necessità di cambiamenti prima (durante lo sviluppo) e dopo il deployment è l'unico fattore costante nei diversi cicli di vita del sw
- I prodotti software che hanno successo sono quelli che si adattano ai continui mutamenti dei requisiti richiesti dagli stakeholder
- Notabene: Progettare l'evoluzione del software significa spesso progettare l'evoluzione di una **famiglia** di prodotti o sistemi, invece che un unico sistema

**Defect detection in a typical Waterfall project**





**Defect detection in a typical Agile project**



# Come evolve un prodotto sw: i rilasci

I prodotti software esistono in genere in più versioni di sviluppo, dette **rilasci** (*release*, in inglese)

- **Rilascio corrente**: contiene miglioramenti e aggiustamenti incrementali, in base ai rapporti degli utenti che segnalano difetti

- **Rilascio in campo** (*fielded*): contiene riparazioni urgenti fatte presso l'utente per mantenere operativo il sistema software

- **Rilascio su richiesta** (es. *prodotto in versione alfa*): prodotto che contiene nuove funzioni da testare e convalidare prima di inserirle nel rilascio corrente

- **Rilascio in itinere** (es. *prodotto in versione beta*): prodotto completo ma non ancora testato del tutto, e non necessariamente da distribuire a tutti gli utenti

# Dopo lo sviluppo

- Il software viene rilasciato, e inizia la sua vita operativa con gli utenti
- Il software operativo potrà essere soggetto richieste di modifica, per diversi motivi
- Le modifiche dopo il rilascio costituiscono attività di manutenzione

# Tipi di manutenzione del sw

- Manutenzione correttiva

- Manutenzione perfetta

- Manutenzione adattiva

- Manutenzione preventiva

- Manutenzione d'emergenza

# Tipi di manutenzione del sw

- Modificare un prodotto sw dopo il deployment è normale
  - Durante l'uso emergono nuovi requisiti (m. **perfettiva**)
  - L'ambiente operativo del sw cambia oppure viene aggiunto nuovo hardware oppure occorre migliorare prestazioni e affidabilità (m. **adattiva**)
  - Gli errori emersi durante l'uso vanno riparati (m. **correttiva**)
- Tutte le organizzazioni hanno il problema di gestire i propri sistemi legacy, cioè i sistemi software la cui utilità persiste nel tempo - anche decenni!
- La gestione di famiglie di prodotti software (Software Product Lines) anch'essa necessita di tecniche efficaci per gestirne l'evoluzione

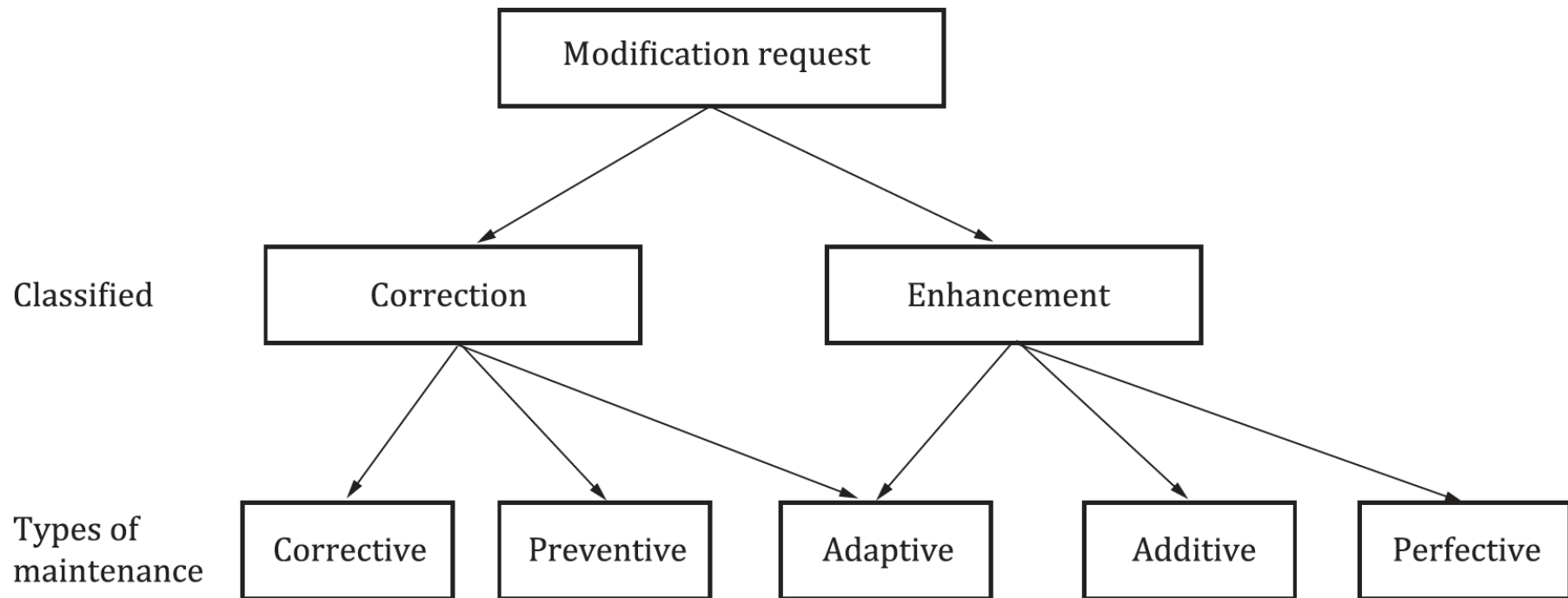
# Correggere gli errori

- Manutenzione **preventiva**
  - Identifica e rimuove errori latenti, prima che vengano riscontrati
  - Adatta a sistemi con problemi di sicurezza
- Manutenzione **correttiva**
  - Identifica e rimuove i difetti riscontrati durante l'uso
  - Corregge gli errori residui dopo il testing pre-delivery
- Manutenzione **d'emergenza**
  - Manutenzione correttiva imprevedibile e urgente
  - Rischiosa perché prevede testing ridotto

# Manutenzione di sviluppo

- Manutenzione **perfettiva**
  - Migliora le prestazioni, l' affidabilità, la manutenibilità
  - Aggiunge nuove funzionalità su richiesta del cliente
- Manutenzione **adattiva**
  - Sviluppo o migrazione
  - Adatta ad un nuovo ambiente (hw, sistema operativo, middleware)

# Tipi di manutenzione (standard IEEE 14764-2022)





# Ciclo di manutenzione: fasi



# Il costo della manutenzione

L'incidenza della manutenzione sul costo complessivo di sviluppo di un sistema è crescente nel tempo (seconda Legge di Lehman) e può giungere sino all'80%, di cui

- Manutenzione correttiva: 20%
- Manutenzione adattiva: 25%
- Manutenzione **perfettiva**: 55%

La maggioranza dei costi è attribuibile ai mutamenti imposti dall'ambiente (1° Legge di Lehman)

# Fattori di costo della manutenzione

- Instabilità del personale
  - I costi di manutenzione aumentano se il personale addetto cambia spesso
- Irresponsabilità dei progettisti
  - Se gli sviluppatori del sistema non hanno responsabilità di manutenzione non c'è incentivo per *“design for change”*
- Inesperienza del personale
  - Il personale addetto alla manutenzione è spesso poco esperto e con scarsa conoscenza del dominio applicativo
- Età e struttura del software
  - Quando il programma invecchia la sua struttura si degrada e diventa più difficile da capire e modificare

# Complessità ciclomatica di un modulo

- La complessità ciclomatica di un modulo è un ottimo indicatore della sua testabilità
- Un' applicazione tipica della complessità ciclomatica di un modulo è la seguente **scala di valori di rischio**

Complessità ciclomatica	Valutazione del rischio
1-10	programma semplice, rischio minimo
11-20	rischio moderato
21-50	programma complesso, rischio alto
maggiore di 50	programma non testabile, rischio altissimo

# Complessità ciclomatica

- La complessità ciclomatica si può usare per:
  - **Analisi del rischio durante la stesura del codice**
  - **Analisi del rischio di cambiamento durante la manutenzione.** La complessità del codice aumenta nel tempo a causa delle manutenzioni. Misurando la complessità ciclomatica prima e dopo un cambiamento si può monitorare la configurazione del sistema e decidere come minimizzare il rischio di ulteriori cambiamenti
  - **Reingegnerizzazione.** L'analisi della complessità ciclomatica valuta la struttura di un sistema: dunque il rischio di reingegnerizzazione di una parte del codice è correlato alla sua complessità.

# Complessità ciclomatica

- La complessità ciclomatica si può usare durante il testing:
  - Definisce il numero di cammini indipendenti da testare
  - *Path coverage set* = insieme dei cammini che eseguiranno tutti i comandi e valuteranno tutte le condizioni logiche almeno una volta
  - Obiettivo: definire un insieme minimale di casi di test che «coprano» tutte le istruzioni
  - Il *Path coverage set* non è unico!

# Metriche di manutenibilità

- La manutenibilità è la misura in cui un sistema software ammette modifiche per estendere il suo tempo di uso
- Prospettive sulla manutenibilità:
  - **Interna:** Dipende solo dalle caratteristiche del sistema
  - **Esterna:** Dipende anche dal personale di manutenzione, dall'ambiente di supporto, dalla documentazione e dagli strumenti disponibili

# Metriche di manutenibilità

Valori bassi sono migliori:

- Quantità di risorse umane necessarie (manutentori)
- # persone necessarie (manutentori) / dimensione del codice
- Tempo per costruire, eseguire e testare
- Complessità ciclomatica del codice
- Numero di versioni da gestire
- Tempo medio per fare una modifica
- Costo medio per correggere un difetto dopo il rilascio
- # di file oggetto ricompilati dopo una modifica ad un file sorgente
- Quantità di codice morto, % di codice morto su LOC totali
- Grado di accoppiamento tra i moduli



# Metriche di manutenibilità

Valori alti sono migliori:

- Numero di design pattern usati
- Livello del linguaggio di programmazione
- Rapporto tra linee commento e linee codice
- Numero di test di regressione
- Percentuale dei manutentori che dicono che il codice è facilmente modificabile
- Rapporto tra numero di moduli e dimensione del codice
- Numero di moduli ad alta coesione sul totale dei moduli

# Strumenti

- Esistono parecchi strumenti in grado di supportare la manutenzione del sw
- Molti registrano misure utili per valutare l'indice di manutenibilità
- Esempio: plugin *metrics* di Eclipse  
<http://metrics.sourceforge.net>

# Riferimenti

Capitolo 6 del SWEBOK: “Software maintenance”, 2004

IEEE Standard 1219-1998 for Software Maintenance

Mens, *Software Evolution*, Springer, 2008

Lehman & Belady, *Program evolution : processes of software change*, AP 1985

Pigoski, *Practical Software Maintenance*. Wiley 1996

Lehman e altri, Metrics and Laws of Software Evolution - The Nineties View, *Int. Conf. on Sw Metrics*, 1997

Oman e Hagemeister, Metrics for assessing a software system's maintainability, *Int. Conf. on Software Maintenance*, 1992

Welcher, The Software Maintainability Index Revisited, *Crosstalk* 2001

# Siti

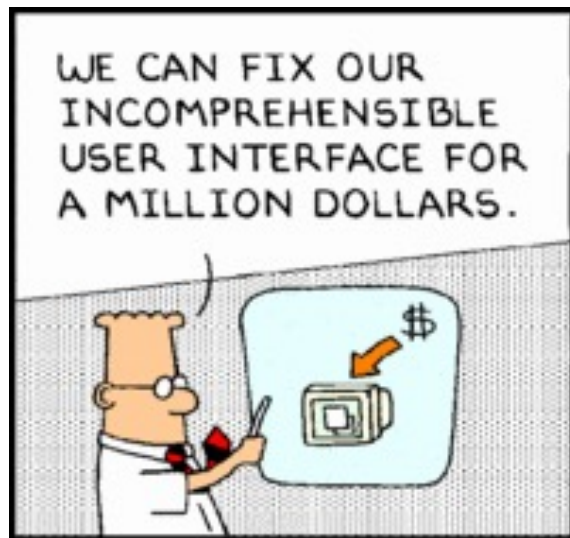
`www.stateofflow.com/projects/16/eclipsemetrics`

`pmd.sourceforge.net`      **strumento open source per analisi codice Java**

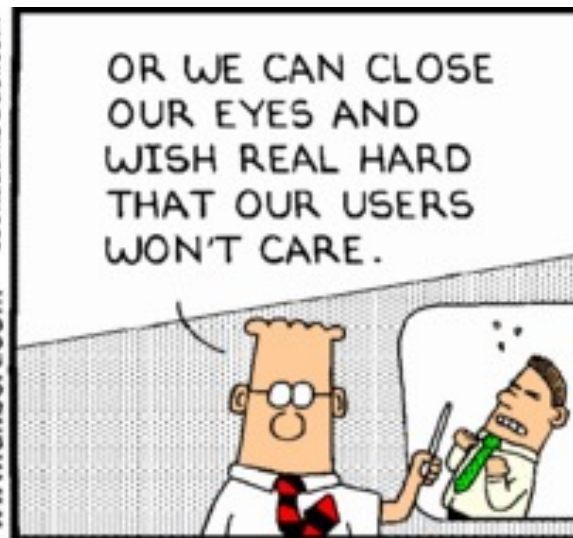
`radon.readthedocs.org/en/latest/index.html`      **altro strumento**

`www.moosetechnology.org`      **piattaforma per analisi del sw**

# Domande?



www.dilbert.com scottadams@aol.com



5/11/02 © 2002 United Feature Syndicate, Inc.



# Conclusioni del corso



*Prof. Paolo Ciancarini*  
*Corso di Ingegneria del Software*  
*CdL Informatica Università di Bologna*

# Scopo del corso

Presentare e sperimentare *metodi e strumenti* di

- analisi,
- modellazione,
- progettazione, e
- misura

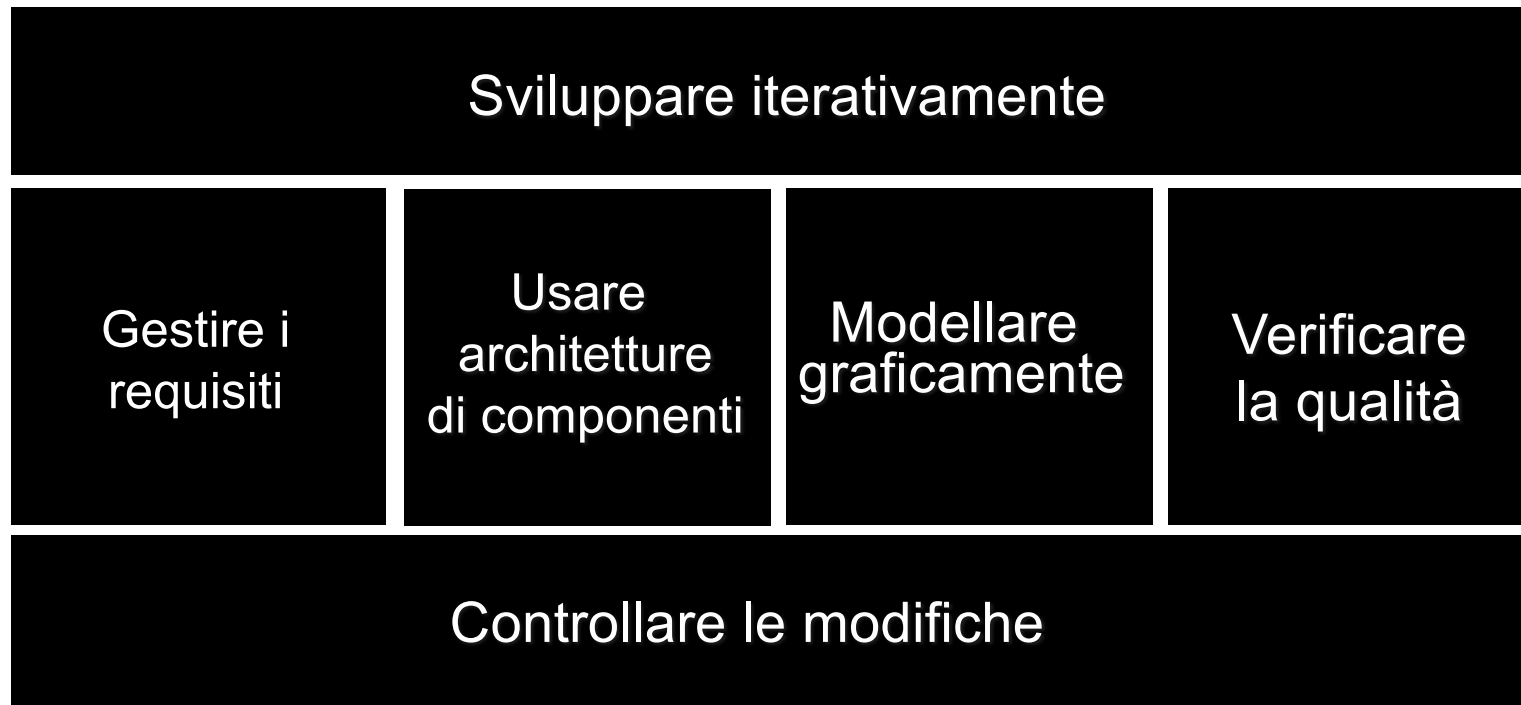
di sistemi e applicazioni software

# Cosa abbiamo visto

- Gli standard di produzione del software
- Il ciclo di vita dei prodotti software
- I modelli agili e Scrum
- La modellazione del software con UML
- L'analisi e la specifica dei requisiti
- La progettazione del software
- I design pattern
- La gestione dei progetti software: tempi e costi
- Controllare e misurare la qualità del software



# Principi guida dello sviluppo software



# Il software è un costrutto sociale

- Sviluppare da soli?
- Sviluppare in due?
- Sviluppare in gruppo?
- Sviluppare in tanti?

Stiamo scoprendo modi migliori di costruire il software facendolo e **aiutando altri a farlo**. Attribuiamo valore a

**Individui e interazioni**

**Software che funziona**

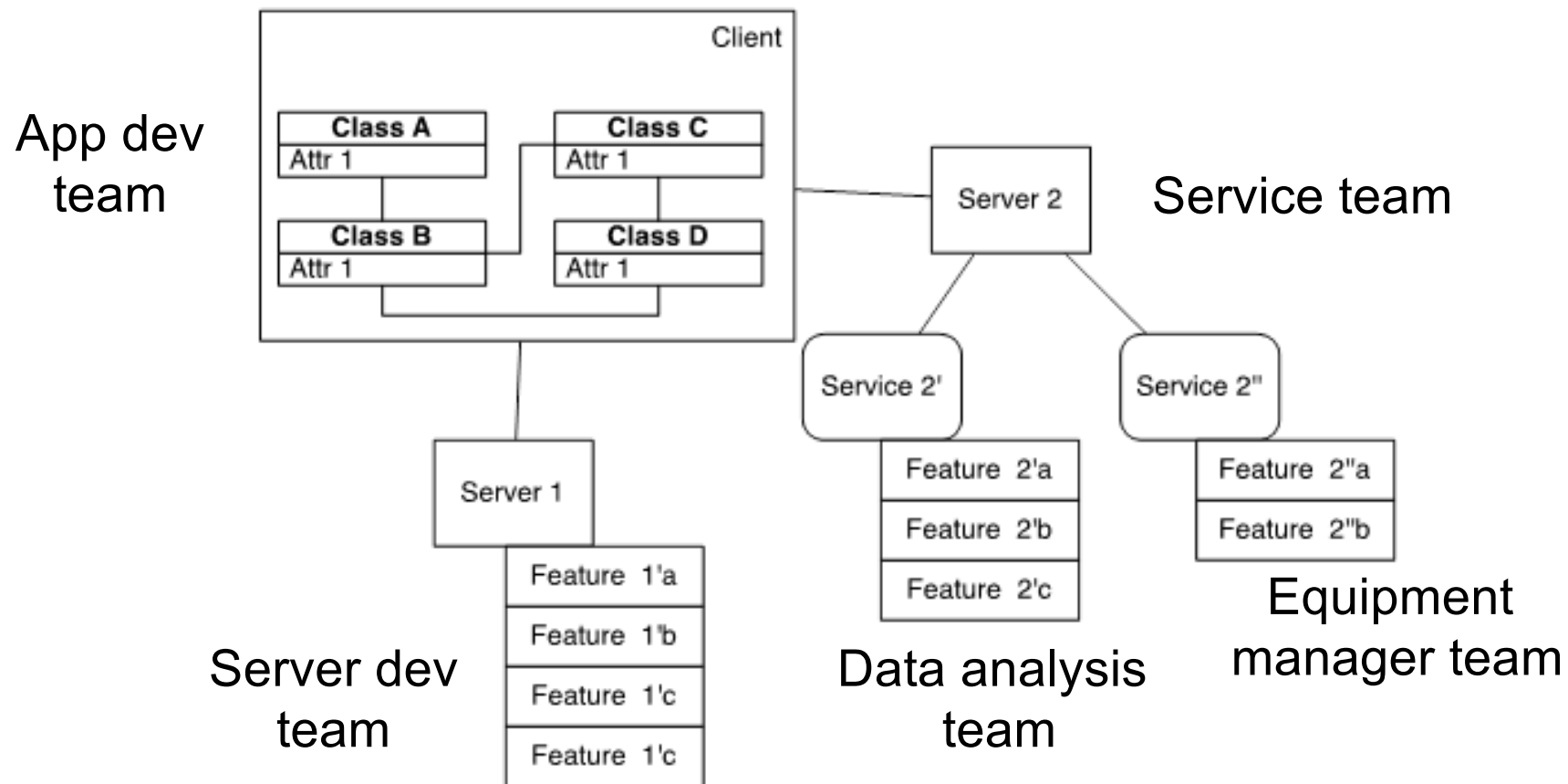
**Collaborazione col cliente**

**Reagire al cambiamento**

Manifesto agile

[https://www.youtube.com/watch?v=7CVfTd-\\_qbc](https://www.youtube.com/watch?v=7CVfTd-_qbc)

Legge di Conway: L'architettura riflette la struttura dell'organizzazione che costruisce un sistema



# Legge di Conway

(versione di Paolo)

Il software è il prodotto di un  
processo sociale,  
e ne incorpora la struttura

# Adesso sapete rispondere?

- Come si *modella* il software?
- Come si *sviluppa* il software?
- Come si *riusa* il software?
- Quali *strumenti* sono disponibili per chi costruisce software?
- Quanto *costa* costruire il software?
- Quanto *tempo* ci vuole?
- Come si valuta la *qualità* del software?

*Le persone che amano le salsicce e il software non dovrebbero mai chiedersi come vengono fatti né le une né l'altro*

David Lee Todd, Product Manager

# Domande?

