Ingegneria del Software

AA 2005-2006

Introduzione

Prof. Giacomo Bucci

Contenuto del corso

- Problematiche relative all'analisi, la progettazione, la produzione e l'uso del software.

Obiettivo del corso

- Acquisire capacità di analizzare, progettare e sviluppare in modo "professionale" sistemi software di complessità non elementare

Indice dei contenuti

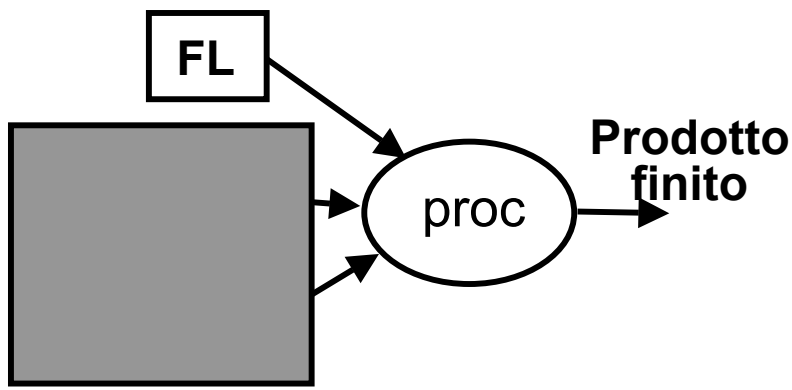
- Introduzione, definizioni, ciclo di vita
- Principi e paradigmi di programmazione
- Metodologie di analisi
  - Analisi e progettazione orientata agli oggetti (OOAD)
  - Analisi e progettazione strutturata
- UML (trasversalmente)
- Analisi/progetto di sistemi - Esempi

Cos'è il software

- Non solo codice
  - documentazione, regole di impiego, …
- Riguardato come *prodotto* di un *processo* industriale
  - rispettando tempi, costi, ..
- E' un prodotto speciale
  - intangibile

Prodotto speciale

- Differente dalla produzione di beni materiali (mezzi di produzione, forza lavoro, materie prime)
- Il software è un prodotto immateriale
  - prodotto di una attività intellettuale (progettazione continua)
  - non c'è un processo manifatturiero
  - totalmente malleabile
  - difficile da descrivere
  - non si consuma
- facilmente copiabile



Legislazione

- In Europa non è brevettabile (negli Usa sì)
  - il Consiglio dei ministri UE aveva emanato una direttiva a favore della brevettabilità
  - il Parlamento europeo ha respinto la direttiva il 6 Luglio 2005
    - 648 contro, 14 favorevoli, 18 astenuti
    - E' la prima volta che una direttiva del Consiglio viene respinta in seconda lettura!!!
- Esperienza americana:
  - Brevettabiltà = Contenziosi legali
  - il 98% dei brevetti si dimostra inutile

In Italia (in Europa)

- Il SW è un'opera di ingegno.
- Regolato dalla legge sul diritto d'autore (248/2000)
  - Copiare sw abusivamente (anche se non per profitto) è un illecito penale
  - l'Art 13 punisce da 6 mesi a 3 anni chi duplica abusivamente SW
- Il compratore non è affatto protetto!!
- E se fa un danno?

Tipi e Acquisizione SW

- Commerciale (Windows XP)
- Shareware (....)
- Freeware
  - Opensource (Linux)
  - Non Opensource (Java)
- Public domain (non ci sono più diritti)

- Comprare, affittare, sviluppare, far sviluppare

Tipi

- Software di sistema
- Software applicativo
- Software embedded
- Prevalente gestione e elaborazione dati
- Prevalente controllo
- ecc.

- Non esiste una soluzione valida per tutti

Tipologia delle applicazioni

- Sequenziali
  - Un solo flusso di controllo
- Concorrenti
  - Più flussi di controllo paralleli
- Dipendenti dal tempo
  - La velocità di esecuzione influenza non solo le prestazioni, ma anche la correttezza (hard real-time)

Dimensione sociale/economica

- Industria SW: qualche % del PIL dei paesi industrializzati
- Spese enormi a volte a vuoto
- I malfunzionamenti possono produrre danni economici e alle persone

- Conviene avere delle teorie/metodi solidi

Software crisis e leggende metropolitane sui progetti software
(un po' datate)

- Oltre il 60% dei progetti fallisce
- L'80% supera i tempi previsti
- Il 50% produce sistemi con funzioni non richieste dall'utente
- Decine di miliardi di $ sprecati

Spesa software mondiale Miliardi di $

Region	2001	2002
U.S.	50,5627	52,8783
Europe	27,4414	28,5391
Asia Pacific	30,1939	31,7036
Latin America	5,7618	6,0499
Rest of the W	7,8467	8,2390
Total	121,8065	127,4099

Sorgente: Computer Economics Inc.

Ingegneria del software

- Termine coniato nel 1968 (a Garmisch Partenkirchen, Germania) durante una conferenza internazionale
  - Vorrebbe essere l'applicazione dei metodi dell'ingegneria al software
  - Approccio sistematico allo sviluppo, all'operatività, alla manutenzione e al ritiro del software(*)
  - Attenzione: Ingegneria NON scienza

(*) Glossario sull'Ingegneria del software IEEE

Programmazione<=> Ing. SW

- Il programma che calcola il MCD di due numeri è di interesse dell'Ing. del SW?
  - No! Perché
    - E' un problema ben formalizzato
    - Nel caso specifico esiste un algoritmo (Euclide) che ne permette il calcolo
  - E' un problema di programmazione

Programmazione<=> Ing. SW

- Un sistema di matematica che consente
  - La scrittura a video di formule, di sistemi di equazioni, …
  - La soluzione di problemi complessi con differenti strategie di soluzione
  - L'analisi dei risultati
  - ecc.

è un problema di Ing. del SW? Sì

- Interfaccia grafica, interpretazione formule, strumenti di analisi, integrazione, …..

L'ingegnere del software

- E' un programmatore in grande (*programming in the large*)
  - capire i requisiti - stendere le specifiche
  - progettare
    - scomporre i problemi, identificare i moduli
  - gestire il processo di sviluppo
  - lavorare in team - ripartire il lavoro

Standard

Sono stati pubblicati molti standard

- IEEE 610 Glossary of Sw terminology
- IEEE 830 Practice for SW req. specification
- IEEE 1233 Developing Sys req. Specification
- IEEE 12207 SW life-cycle processes
- IEEE 1012 SW verification and validation
- IEEE 1471 Practice for architectural description
- Ecc

Processo - prodotto

- Prodotto software:
  - codice; documenti di analisi, di progetto, di test; manuali d'uso, …
- Processo software :
  - sviluppo, evoluzione, … (ciclo di vita)
  - Il processo software è molto simile ad una continua progettazione piuttosto che ad una produzione tradizionale

Misurare il software

"Non si può controllare ciò che non si può misurare"

- Esempi:
  - Quanto costa un dato processo?
  - Qual è la produttività dei programmatori?
  - Quant'è "buono" il codice prodotto?
  - Quanti programmatori si richiedono?
  - Che tempo si prevede per lo sviluppo?
  - Quanto è portabile il software prodotto?

Metriche

- Si usa il termine "metrica" per indicare una misura diretta o indiretta di un qualche attributo di una entità di interesse.
- Le metriche software si riferiscono ad attività di misurazione riguardanti:
  - Misure e modelli di produttività
  - Stima dei costi e dell'*effort*
  - Misure e modelli di affidabilità
  - Misure di complessità

➡ **Qualità**

Esempi di "metriche"

Formali <=> informali (soggettive)

- Affidabilità
  - Informale: quanto si conta sul prodotto
  - Formale: probabilità di assenza di malfunzionamenti su un periodo di tempo

- Portabilità
  - Informale: quanto è facile portare il prod.
  - Formale: 1 - NStatNuovi/NStatTotale

Misurazione

- *Misurazione:* il processo di assegnazione di simboli, normalmente numeri, per rappresentare un attributo dell'entità di interesse, secondo regole definite

- Quattro elementi:
  - Entità: l'oggetto o l'evento su cui si indaga (un tavolo, un viaggio..)
  - Attributo: caratteristica dell'oggetto (altezza, durata, costo, ..)
  - Forma (della rappresentazione): l'altezza si misura in centimetri; gli indumenti con S (small) , M (medium), L (large), XL (extra large); la benzina con "normale", "super".
  - Regole: per arrivare a determinare il valore dell'attributo (in modo da rendere il processo ripetibile, non soggettivo)
- Il metro è stato definito nel 1889 (a Parigi c'è il metro campione)

Misurazione

- L'altezza di una persona si misura in cm
- L'intelligenza come si misura?
- Del vino si può misurare la gradazione alcoolica, ma il suo sapore come si misura?
- L'accuratezza di una misura dipende dallo strumento e dalla forma della misura (cm, QI)
- Misura diretta: quantificazione diretta (p.e.: misura dell'altezza)
- Misura indiretta: richiede un calcolo (p.e.: misura dell'intelligenza)

Misura indiretta

"Rendere misurabile ciò che non è direttamente misurabile"

- Si fanno misure dirette e si combinino per dare una quantificazione di un attributo non direttamente misurabile.
  - Esempio: nel decathlon si misurano tempi e lunghezze; le misure vengono poi combinate e pesate in modo da dare luogo ad uno "score" che identifica il "miglior atleta"

Attributi (e relative metriche)

- Esterni (visibili all'utilizzatore)
  - Prestazioni (tempo risposta)
  - Costo (€)
  - Affidabilità (MTBF)
  - Usabilità (???)
- Interni (visibili al progettista/sviluppatore)
  - Dimensione (SLOC)
  - Effort di produzione (MU)
  - Funzionalità (FP)
  - Complessità (Num ciclomatico)
  - Modularità (???)

La dimensione

- La maniera più semplice consiste nel misurare il numero di linee di codice (SLOC).
  - Facile da misurare (salvo mettersi d'accordo su ciò che si misura)

- Si ritiene che questa semplice misura sia insufficiente rispetto alla quantificazione di effort, produttività e costo.

  (l'altezza non permette di dire se una persona è anche in sovrappeso)

Linee di codice (sorgente)

- E' la misura software più antica (anni settanta).
- Grande confusione: Occorre stabilire come si contano:
  - linee vuote
  - linee di commento
  - dichiarazioni e altri comandi
  - linee con più istruzioni
  - linee programmate o generate da tools
  - copiate da altra parte
- Di norma in SLOC entrano tutte le linee di programma eccetto le linee vuote e le linee di commento

- Si derivano altre misure

Autodocumentazione: $\frac{NC\_SLOC}{SLOC}$

COCOMO

$$E = aS^{b}F$$

E: Sforzo (mesi/persona)

S: Dimensione (migliaia di linee di codice consegnate , KDSI)

a,b: costanti che variano a seconda del modello e in base alla modalità di sviluppo

F: fattore correttivo.

$$T = cE^{d}$$

T: tempo solare in mesi necessario allo sviluppo

c,d: costanti che variano a seconda del modello e in base alla modalità di sviluppo

.. COCOMO

- KDSI include tutte le istruzioni sorgente consegnate all'utente, *escludendo eventuali istruzioni usate appositamente per il debugging, testing, o altro supporto.*
- *Esclude tutte le linee di commento.*
- Sforzo e tempo solare coprono il periodo intercorrente dall'inizio dell'analisi del progetto (fine raccolta dei requisiti utente) fino alla fase di consegna del sistema.
- I valori calcolati comprendono le attività di *management* e la stesura della documentazione, *ma escludono i tempi di addestramento dell'utente, i tempi di installazione ed eventuale manutenzione.*
- La stima comprende tutti i costi diretti (programmatori, librerie, strum di supporto). *Sono esclusi i costi indiretti (costi segreteria, costi top management).*

SLOC

- NASA space shuttle flight control
  - 430 K (shuttle) 1,4 M (ground)
- Microsoft Windows 3.1 (1992) 3 M
- Microsoft NT (1992) 4 M
- Sun Solaris (1998) 7 M
- Microsoft Windows 95 14 M
- Microsoft Windows 98 18 M
- Microsoft NT5.0 (1998) 20 M
- RedHatLinux 6.2 (2000) 17 M
- RedHatLinux 7.1 (2002) 32 M
- Microsoft Windows XP 40 M

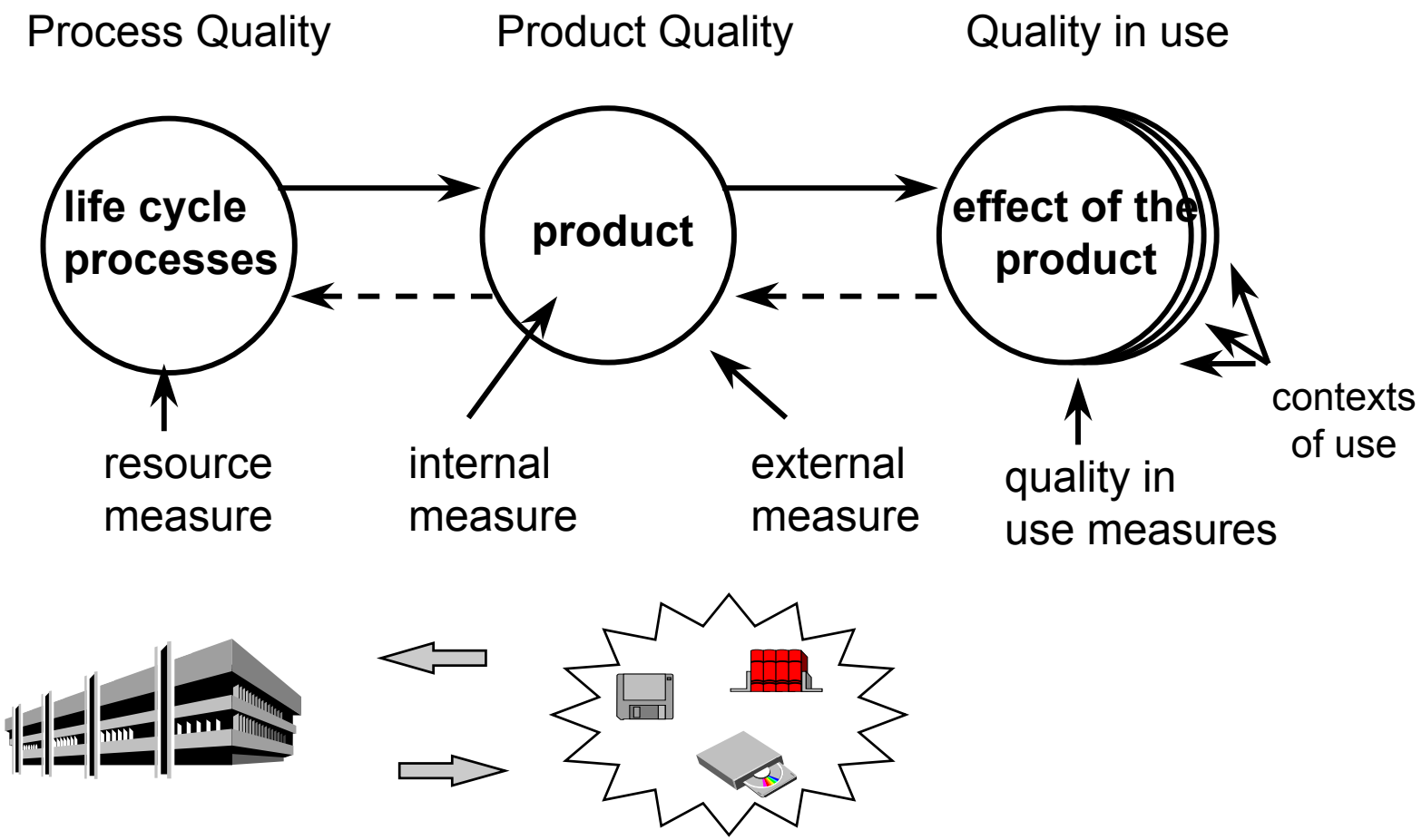
Qualità di un prodotto

- La sua adeguatezza all'uso

- ISO 8402 (glossario) cita:

  "The totality of the features and characteristics of a product or service that bear on its ability to satify stated or implied needs"

Qualità

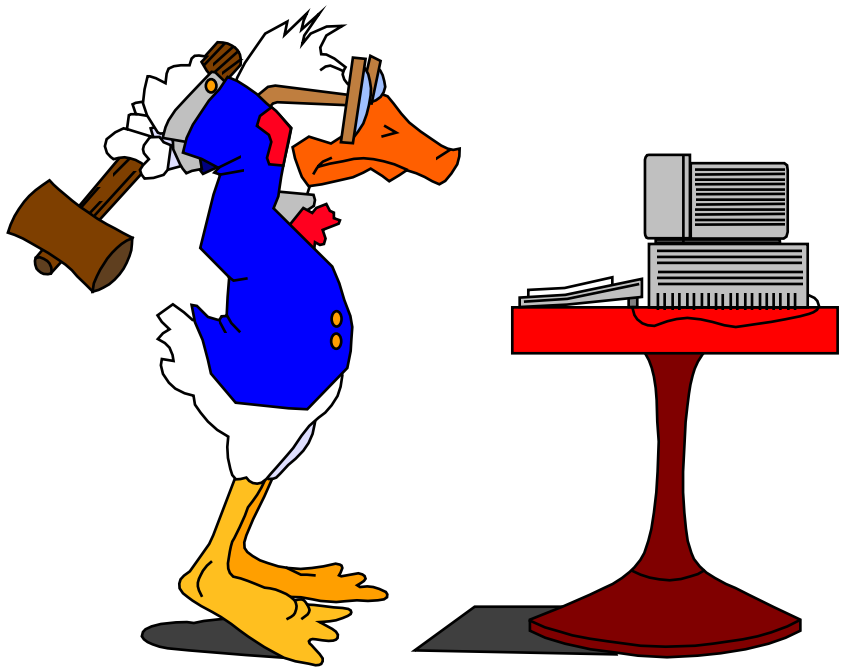

Process Quality
Product Quality
Quality in use
life cycle processes
product
effect of the product
resource measure
internal measure
external measure
quality in use measures
contexts of use

Qualità processo software

- Produttività
  - efficienza del processo (# linee/giorno)
- Struttura
  - processi/metodi di sviluppo ben codificati
- Puntualità (temporale)
  - rispetto dei tempi

Qualità del software

- Qualità soggettiva
  - Soddisfacimento dei requisiti
- Qualità oggettiva
  - Prestazioni
  - Funzionalità
  - Documentazione



Percezione della qualità

- Utilizzatore: Facilità di uso, efficacia
- Cliente: Soluzione di un problema a costo accettabile
- Sviluppatore:Facilità di comprensione, manutenzione, riuso
- Manager: resa della vendita

Indici diversi in ambiti diversi

- Sistemi Informativi: integrità dei dati, sicurezza accessi
- Sistemi Tempo reale: tempo medio di risposta, latenza massima

Qualità del software

- Interna - Esterna
  - Interna: interessa lo sviluppatore
  - Esterna: interessa l'utente
  - La qualità interna ha influenza sulla qualità esterna

- La qualità del processo ha influenza sulla qualità del prodotto
- Da un buon processo ci si aspetta un buon prodotto

Qualità prodotto software

- Affidabilità
- Robustezza
- Riusabilità
- Usabilità
- Portabilità
- Manutenibilità
- Prestazioni
- .....

Oggettiva
Soggettiva

Standard di *qualità*

- ISO 9000: standard internazionale di qualità
- ISO 9001: Qualità del processo (generico)
- ISO 9126: Qualità del prodotto software
  - definisce 7 *Quality characteristics*: Functionality, Reliability, Usability, Efficiency, Maintainability, Portability

Il ciclo di vita (processo software)

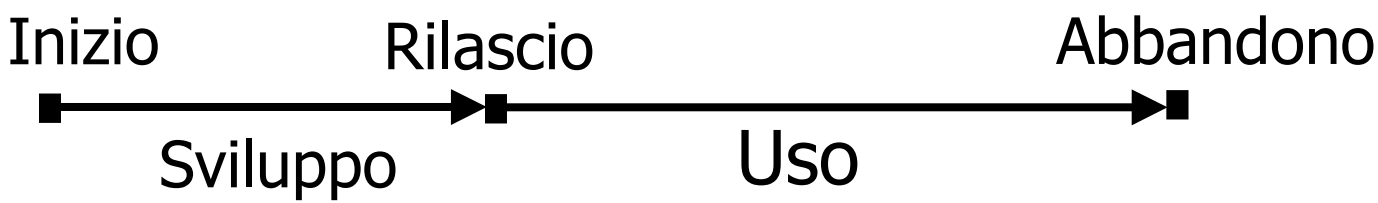
Da ANSI / IEEE Std 729-1983

- Un ciclo di vita è un periodo di tempo che:
  - inizia quando un prodotto software viene concepito
  - termina quando il prodotto software non viene più usato

- Contesto organizzativo per lo sviluppo di un progetto

Il ciclo di vita (processo software)

- Organizzato in fasi.
  - Grossolanamente: *sviluppo* e *uso*
  - La fase di uso è molto più lunga della fase di progetto e sviluppo
  - durante la fase d'uso è normale apportare



Leggenda metropolitana:

- Il costo della fase di uso (costo maintenance) è il 70% del totale

Ciclo di vita

- Ci sono due classi estreme di modelli di ciclo di vita:
- Modelli Prescrittivi
  - fasi ben codificate in successione
  - organizzazione e metodi
- Modelli Agili
  - privilegiano l'adattabilità
  - riducono la burocrazia

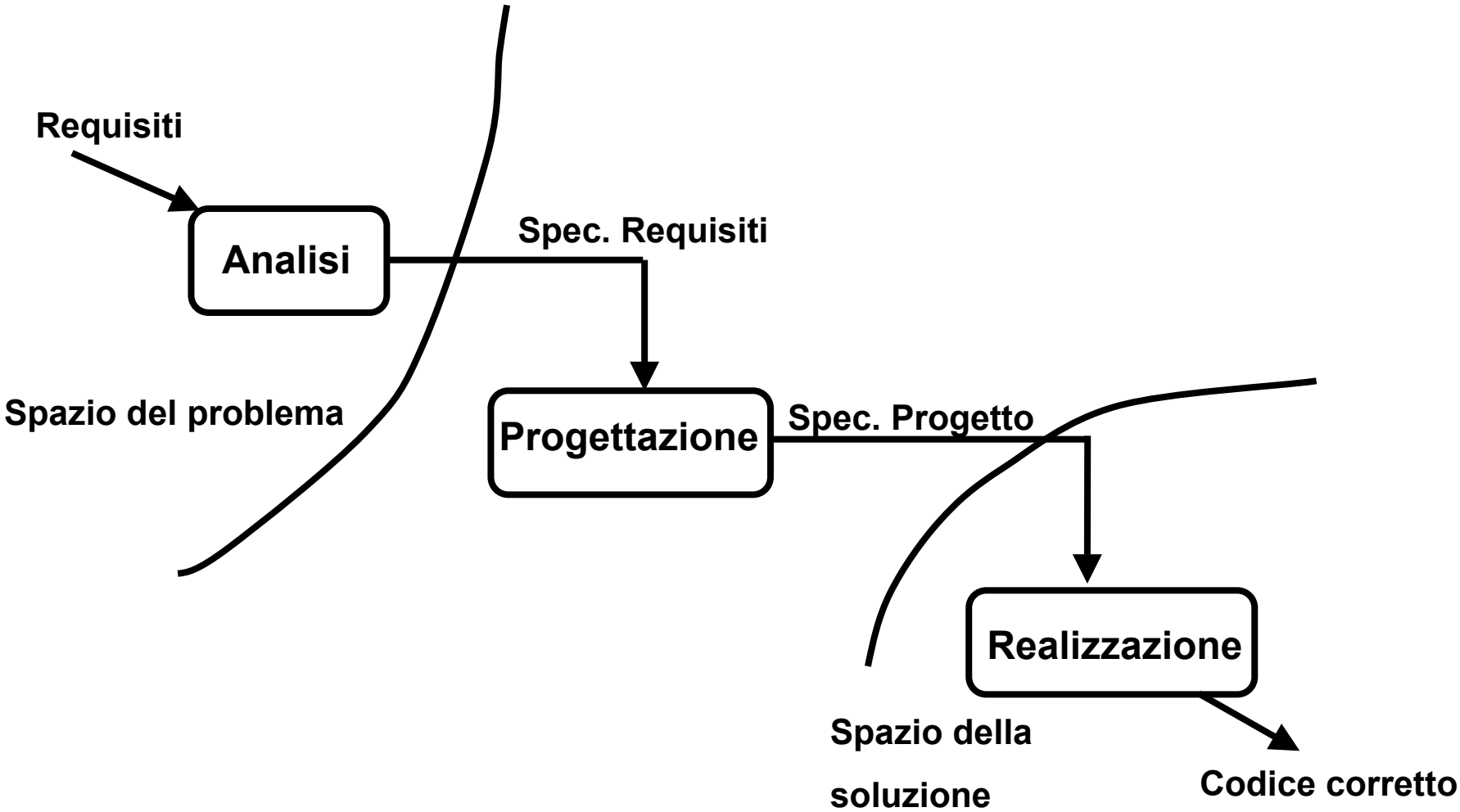
Ciclo di vita - Esempi

- *Code & Fix*
  - Non è niente !!!
- A cascata
  - Tradizionale (prescrittivo)
- Extreme programming
  - In gran voga (agile)

Ciclo di vita

- Serve un modello che possa essere:
  - controllato
  - diviso in fasi
  - che consenta di identificare i *semilavorati* lungo il processo di produzione
  - che permetta di identificare *milestones*
  - etc.

Schema preliminare



Analisi/specifica dei requisiti

- L'analisi è lo studio di un problema prima di intraprendere qualsiasi azione (De Marco)
- L'analisi è lo studio del dominio di un problema, che porta alla specifica di un comportamento esternamente osservabile, a una descrizione coerente e fattibile di ciò che occorre realizzare

Requisiti

- Un requisito è:
  - qualcosa che il software deve fare;
  - una caratteristica che esso deve possedere.
- Un requisito esiste sia perché:
  - la natura dell'applicazione lo richiede;
  - il committente vuole che esso sia parte del prodotto (software).

Scopo - requisiti

- **Scopo**: obiettivo che il prodotto si prefigge
- **Requisiti**: insieme delle caratteristiche che il prodotto deve possedere per soddisfare al proprio scopo di uso.

- Ovviamente ci si riferisce al prodotto software

Scopo - requisiti

- Esempio: Un sistema per il controllo del traffico aereo
  - Scopo: il controllo del traffico aereo di una certa regione.
  - Requisiti:
    - riconoscere le situazioni che possono portare a collisioni
    - registrare la rotta seguita da ciascun aereo
    - effettuare un ciclo di controllo di n aerei in x secondi

Categorie

- Requisiti funzionali
  - ciò che il sistema deve fare
- Requisiti non funzionali
  - proprietà (qualità) che il sistema deve possedere
- Vincoli
  - sono requisiti di carattere generale; definiti prima che inizi l'effettiva raccolta dei requisiti

Requisiti funzionali

- *R1 - Il sistema deve consentire ad un generico utente dal proprio PC di vedere i movimenti sul suo CC*
- *R2 - Deve consentire di effettuare operazioni di deposito dagli sportelli abilitati all'operazione. Il dettaglio dell'operazione e quello di seguito specificato:*
  - *R2.1 - L'utente inserisce il codice da tastiera numerica*
  - *R2.2 - .....bla, bla*

Requisiti

- Non funzionali
  - Il sistema deve essere portabile tra gli ambienti X,Y e Z
  - La risposta non deve superare 20 s
- Vincoli
  - Il sistema deve operare sotto Linux
  - I programmi devono essere scritti in Java

"Specifica" e "Requisiti"

- Il documento che contiene la specifica dei requisiti del software viene indicato come SRS (Software Requirements Specification)
  - Dovrebbe dire **COSA** fa il sistema
  - NON dovrebbe dire **COME**. Il "come" è lasciato alla fase di progetto

  - *SRS rappresenta la "specifica esterna"*

Modello per SRS

- Vincoli
- Requisiti funzionali
- Requisiti non-funzionali
- Altri aspetti del progetto

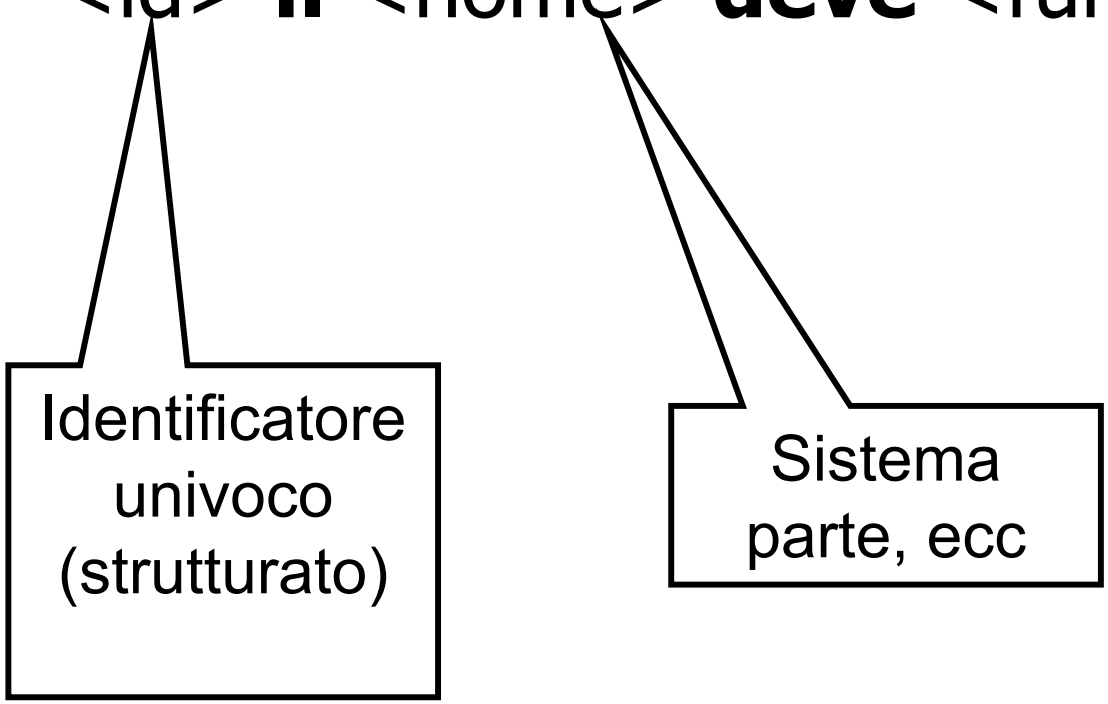
- Esistono degli standard
  - **IEEE Std 830-1998**

Come si fa un SRS

- In forma testuale elencando ordinatamente i requisiti (funzionali)
  - La specifica tende a diventare un racconto voluminoso, quasi sempre pieno di ambiguità, inconsistenze, etc.
- Ricorrendo a qualche "linguaggio" di specifica (+ o - formale)
  - Evita i problemi della specifica discorsiva
  - Riduce la possibilità di errore

Schema per i requisiti (testuali)

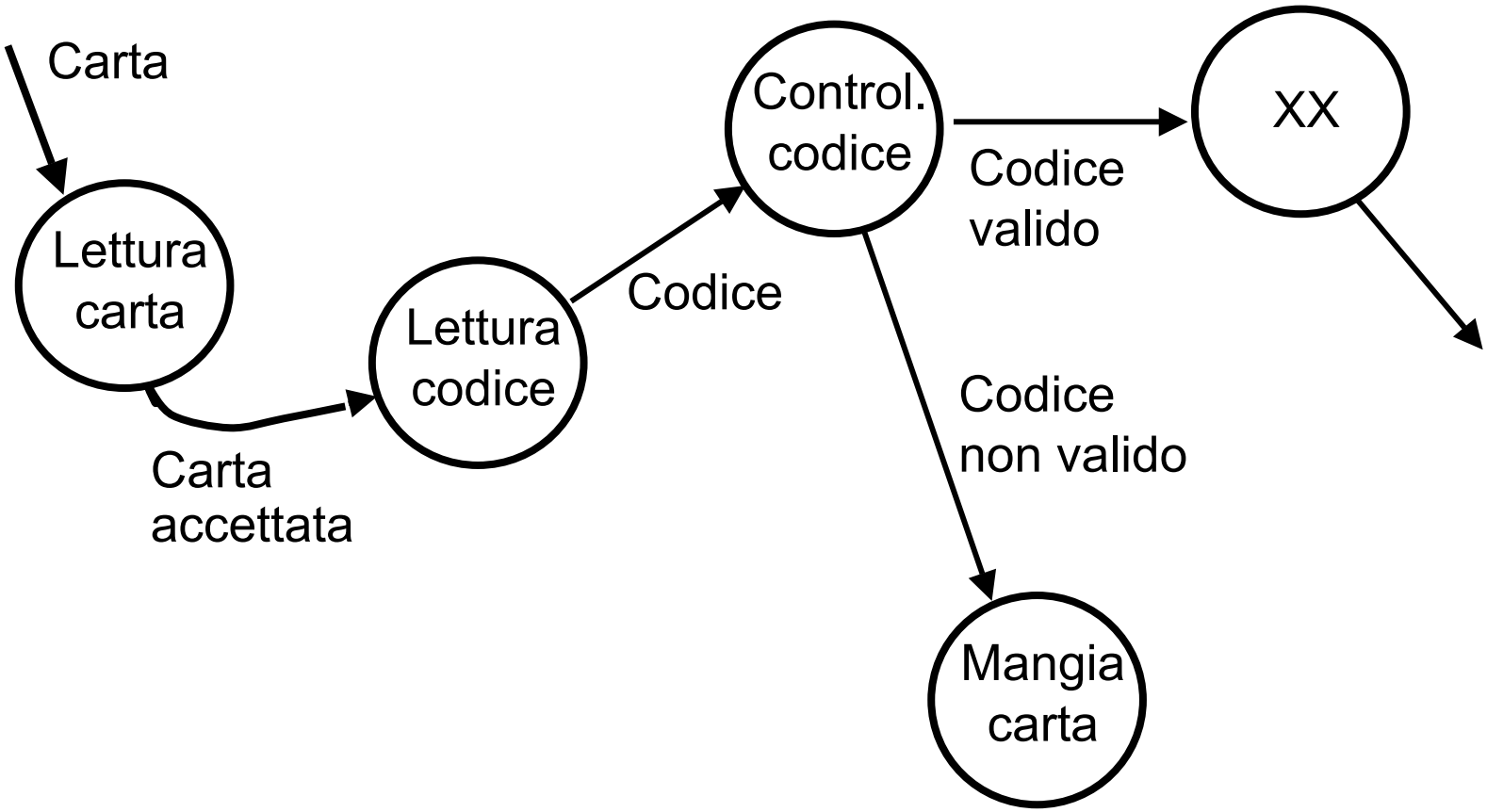
<id> **il** <nome> **deve** <funzione>



Esempio (ATM) testuale

- R1: il sistema deve leggere carte di credito
  - R1.1: accetta solo quelle appartenenti ai circuiti Visa e Master card
- R2: il sistema deve leggere il codice utente battuto da tastiera
- R3: il sistema deve effettuare il controllo di validità del codice
  - R3.1: invita a ribattere se inaccettabile
  - R3.2: "mangia" la carta dopo 3 tentativi sbagliati

Esempio (ATM)

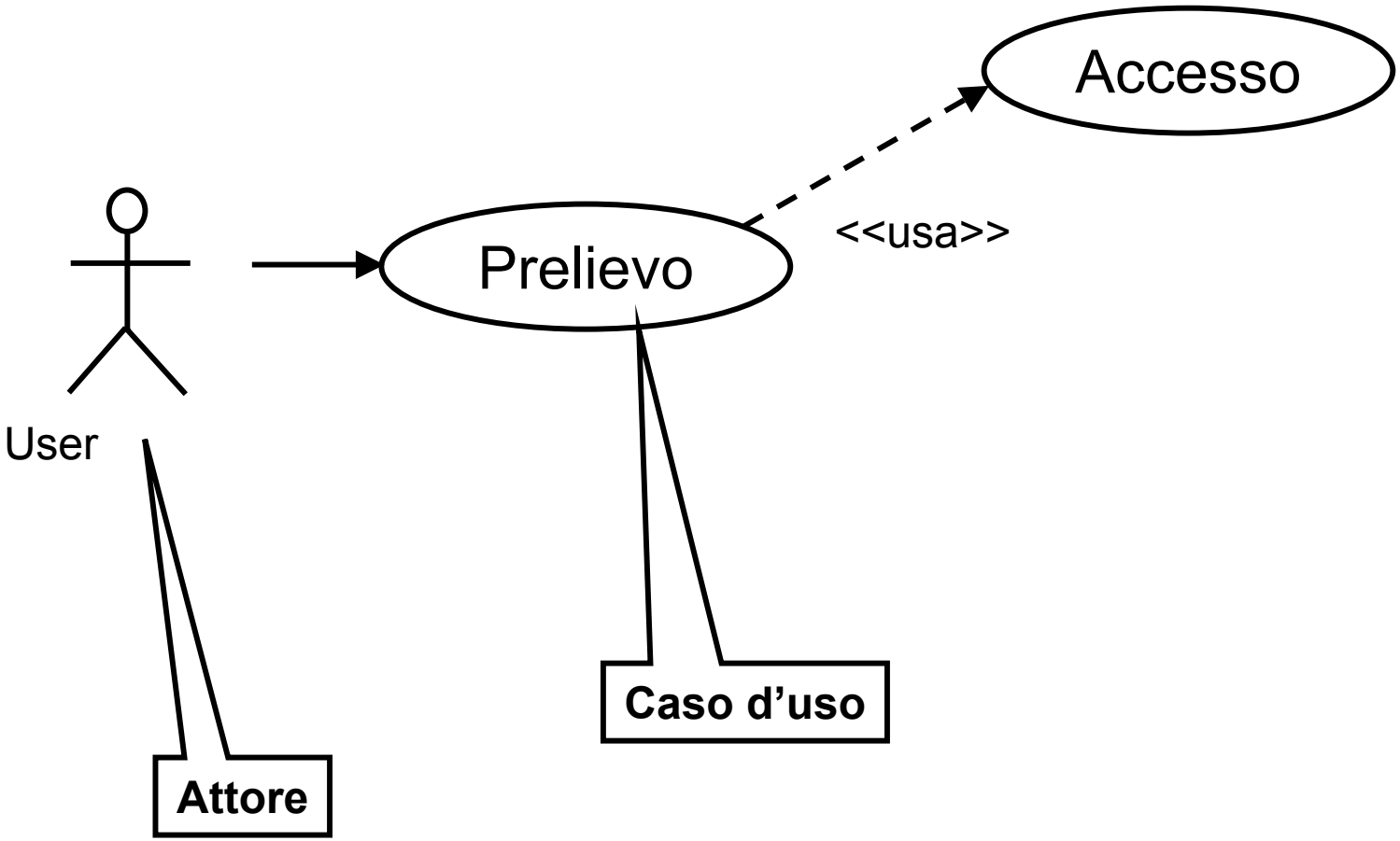

Carta
Lettura carta
Carta accettata
Lettura codice
Codice
Control. codice
Codice valido
XX
Codice non valido
Mangia carta

Esempio (ATM)

- La specifica precedente usa i DFD
  - Palle: attività (o processi)
  - Archi: flussi di dati

- Attenzione: è una specifica di *cosa* fa il sistema, **non** di *come* lo fa
- **E' il risultato dell'analisi**

Esempio (ATM)



Esempio (ATM)

- Lo schema precedente è un diagramma dei Casi di uso (*Use Case*)
  - E' la tecnica usata in UML per catturare i requisiti funzionali
- I casi d'uso mostrano il modo in cui gli utenti usano il sistema
  - I casi d'uso vengono descritti testualmente elencando i passi svolti al loro interno

Casi d'uso

- Dicono cosa fa il sistema a livello (molto) alto/aggregato
- Centrati sull'utente
- Informali (non rigorosi)
- Non specifici dell'Object Oriented
- Servono a stimare il lavoro (le parti di lavoro) da fare

Metodo

- Identificare gli attori
- Definire cosa un attore vuol fare => caso d'uso
- Per ogni caso d'uso una descrizione:
  - L'attore fa <xx> il sistema fa <yy>
  - L'attore fa <aa> il sistema fa <bb>

Descizione caso d'uso

- Numero e titolo
- Breve descrizione
- Attori
- Precondizioni per l'esecuzione del CU
- Flusso principale
- Flussi alternativi
- Postcondizioni

SRS

- **IEEE Std 830-1998** IEEE Recommended Practice for Software Requirements Specification

- Revisione di due precedenti versioni: la prima dell'84 la seconda del 93
- Pesante da digerire (risente dell'epoca in cui è stato concepito)

A cosa deve rispondere (IEEE Std 830-1998)

- **Funzionalità**: Cosa fa il sistema software?
- **Interfacce esterne**: Come interagisce con l'esterno, con gli utenti, con l'hardware, con altro software?
- **Prestazioni**: Qual'è il tempo di risposta, che livello di disponibilità deve presentare?

A cosa deve rispondere (IEEE Std 830-1998)

- **Attributi**: A quali condizioni circa la portabilità, sicurezza, etc deve soffisfare?
- **Vincoli realizzativi**: Ci sono vincoli su standard da rispettare, sulle risorse, sui mezzi di sviluppo, sui linguaggi, etc ?

SRS (IEEE Std 830-1998)

- Un SRS è "buono" se è:
  - Corretto
  - Non ambiguo
  - Completo
  - Consistente
  - Con requisiti in ordine di importanza e/o stabilità (ranking)
  - Verificabile
  - Modificabile
  - Tracciabile

Esempio: tracciabilità

Che vorrà mai dire questo requisito??

- Un SRS è tracciabile se
  - è chiara l'origine di ogni requisito (tracciatura all'indietro, ai documenti precedenti)
  - ogni requisito ha un nome o un numero (tracciatura in avanti, per i requisiti futuri)

Struttura (IEEE Std 830-1998)

- Lo standard fornisce la struttura a capitoli/paragrafi del documento
- Prevede che la specifica dei requisiti funzionali, oltre che a parole possa essere fatta anche con altri metodi (DFD, SM, ...)

- (E' una pizza tremenda !!!!)

Struttura (IEEE Std 830-1998)

- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions, acronyms, and abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall description
  - 2.1 Product perspective
  - 2.2 Product functions
  - 2.3 User characteristics
  - 2.4 Constraints
  - 2.5 Assumptions and dependencies
- 3. Specific requirements
- Appendixes
- Index

Conclusioni

- Usare lo standard come modello, soprattutto per la parte descrittiva
  - Scopo, elenco requisiti, vincoli, ecc.
- Ricorrere a un qualche linguaggio per la specifica
  - Use case, DFD, SM, SDL….

Lavoro individuale

- Esaminare/studiare lo IEEE Std 830-1998 (si trova sul CD)
- Stendere SRS per un distributore di bibite automatico seguendo i dettami dello standard

Modello del ciclo di vita a Cascata

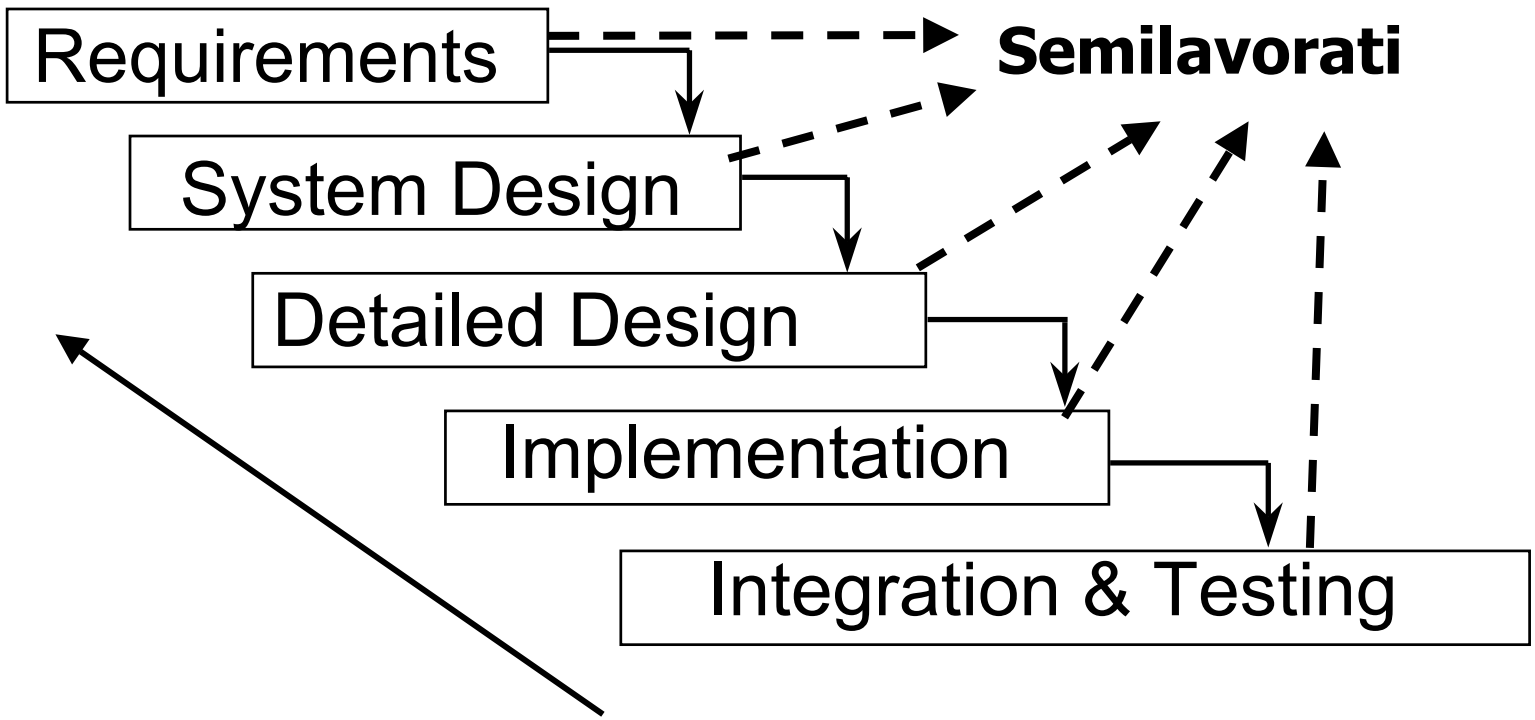
W. W. Royce, Winston W., *Managing the Development of Large Software Systems*, IEEE 1970

Modello a cascata

- Capostipite di tutti i modelli di ciclo di vita
  - risente dell'epoca in cui è stato formulato
    - albori dell'industria del software
    - macchine/strumenti scadenti
  - E' un po' il modello "industria pesante" o "piano quinquennale"

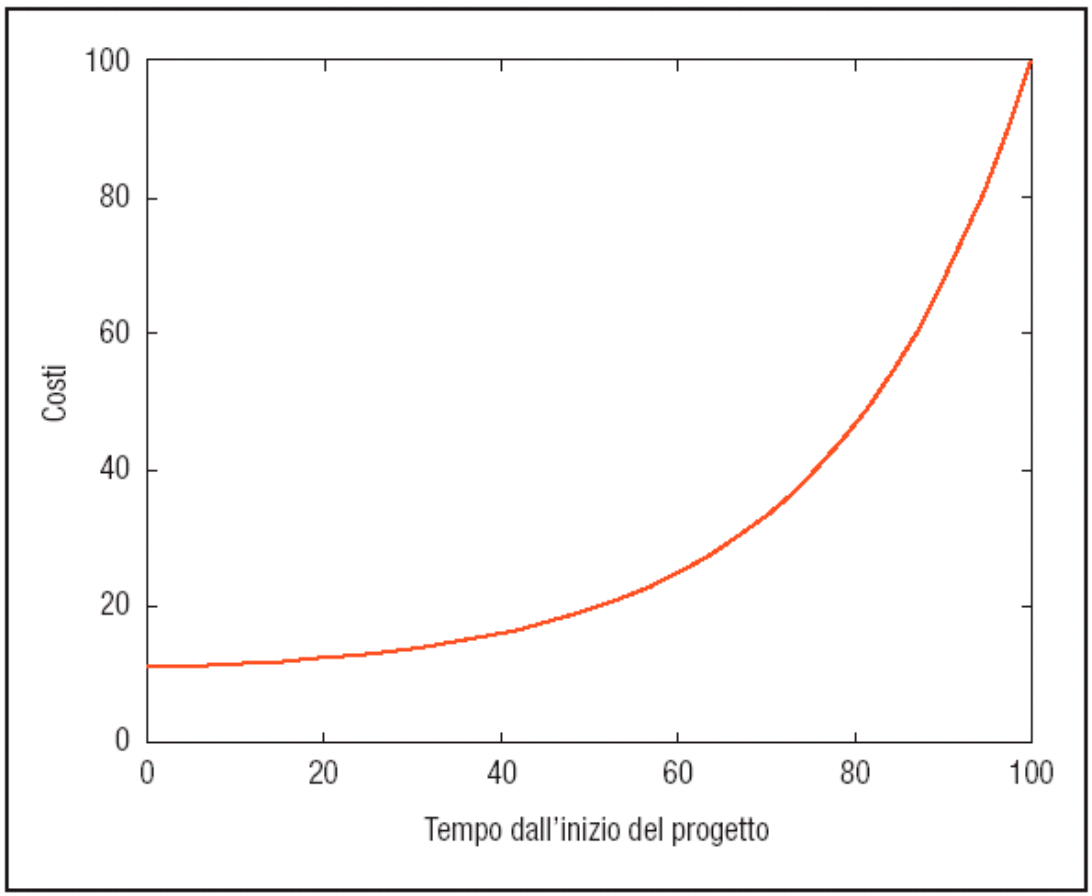
Modello a cascata

Fasi svolte in sequenza



Costi modifiche (tradizionale)

Il costo dei cambiamenti cresce esponenzialmente nel tempo



Prevenire è meglio che curare !

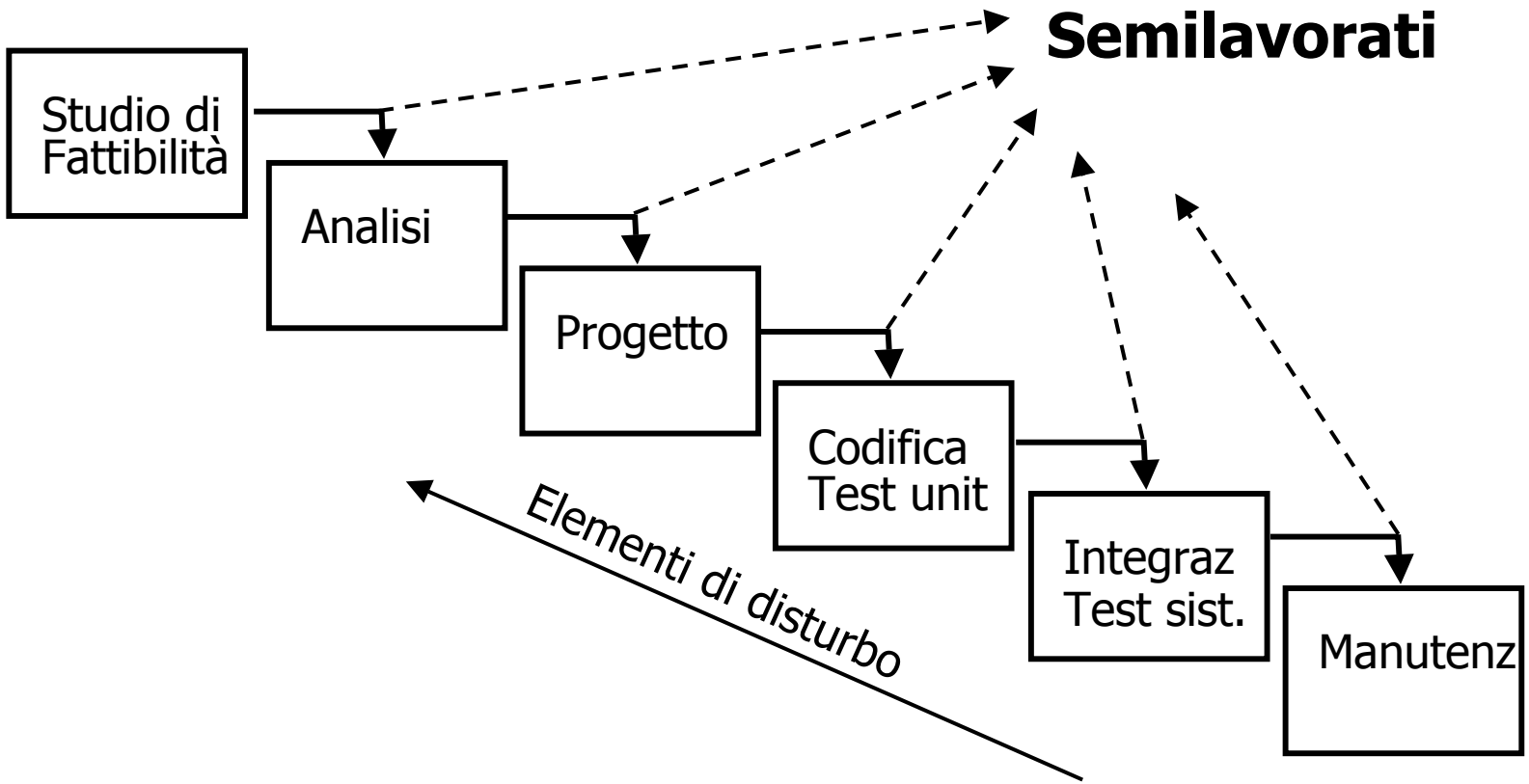
Se i costi crescono esponenzialmente

- Bisogna pianificare bene prima
- Scoprire un errore subito fa salvare denaro
- Modificare il modello è meglio che modificare il codice

=>Investire sulle fasi di analisi

=>BDUF (big design up-front)

Modello a cascata (variazione)



Fasi

- Ogni fase
  - ha un obiettivo definito;
  - si basa sui risultati della fase precedente
- Le fasi corrispondono ad attività che
  - transformano i loro ingressi in uscite;
  - presuppongono la verifica ai milestones

Milestone

- E' un evento predefinito
  - che serve a misurare il progredire dello sviluppo
- Un milestone prevede normalmente:
  - la consegna di prodotti (intermedi) - detti *deliverable*
  - un esame (*review*) formale
  - (l'emissione di un documento)

Documentazione

- Documentazione tecnica interna
- Documentazione tecnica esterna
- Documentazione per l'utente finale

  - *Per ogni versione e rilascio (!!!???)*
  - *Basata su standard (IEEE, ANSI)*

Documentazione

- Documentazione tecnica interna
  - risultati di ogni fase
  - motivazioni per le assunzioni fatte e le decisioni prese in ogni fase
  - schemi e risultati dei test
  - errori e log degli interventi

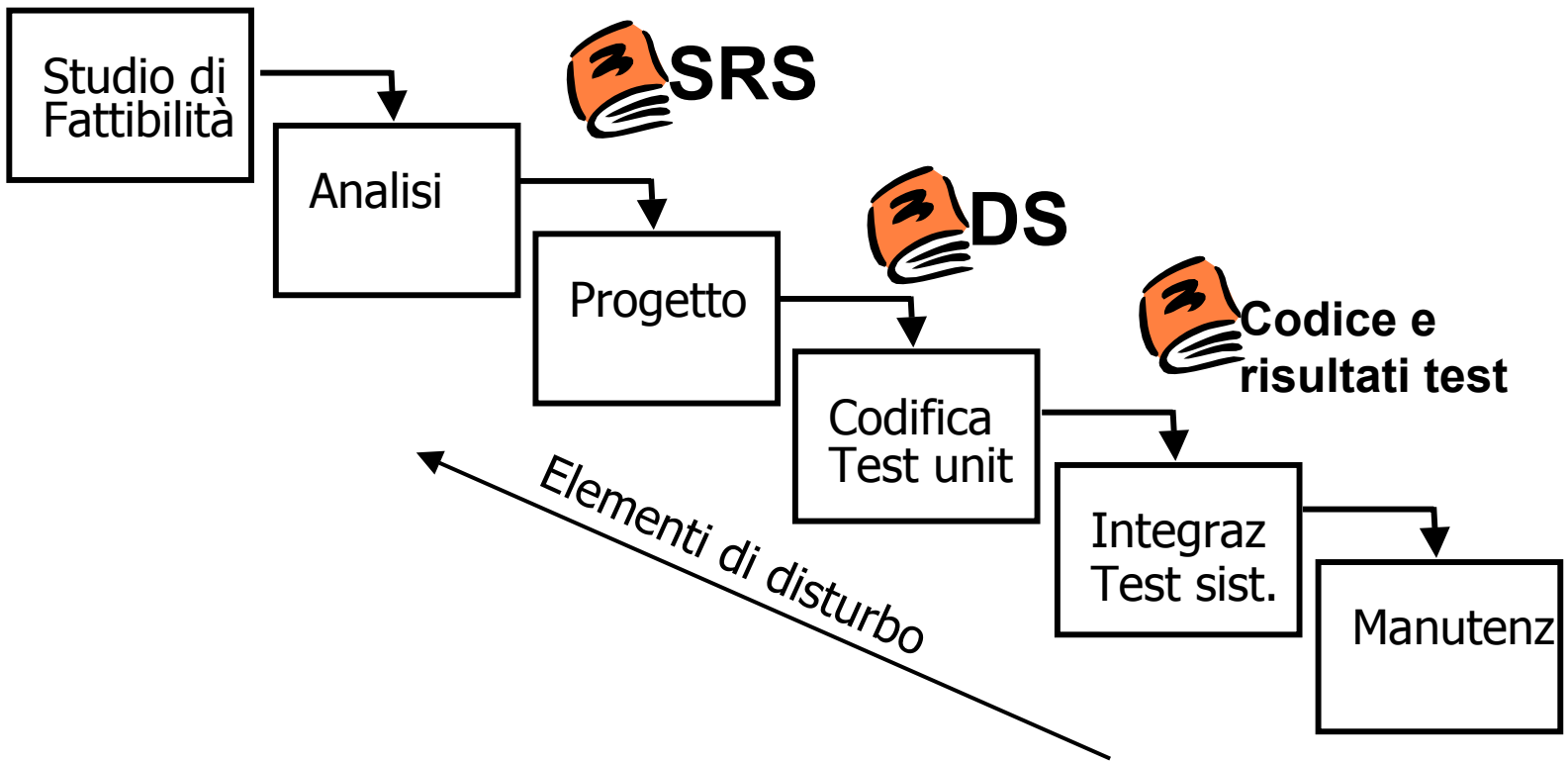
Documentazione

- Documentazione tecnica esterna
  - struttura del sistema
  - ambiente richiesto
  - istruzioni per l'installazione
  - maintenance & operation
  - guida alla soluzione dei problemi

Documentazione

- Documentazione per l'utente finale
  - Manuale utente
  - prontuario (quick reference)
  - guided tour
  - ...

Fasi e deliverables



SRS Software Requirement Specification

- ***SRS dovrebbe essere sottoscritto dal committente e formare la base contrattuale***

- L'analisi dovrebbe produrre anche il Manuale Utente e il Piano di test del sistema

Analisi

- Dovrebbe produrre anche il Manuale Utente e il Piano di test del sistema
  - Il manuale utente dice meglio di qualunque altra cosa come si comporterà il sistema;
  - Fornisce una base di confronto e di accordo contrattuale con il committente;
  - Il piano di test dice come verranno condotti i test del sistema.
- Grande approfondimento

Progettazione

- Specifica del Progetto (DS) contiene l'architettura del software:
  - Scomposizione del sistema in sottosistemi
  - Identificazione dei moduli e delle relazioni tra i moduli (struttura del sistema secondo una conveniente rappresentazione)
  - Interfacce
  - Convenzioni di chiamata
  - ...

Realizzazione

- Codifica
  - I singoli moduli vengono sviluppati in base alla specifica di progetto, nel linguaggio di programmazione scelto
- Test individuali

Integrazione

- Il fatto che i singoli moduli siano individualmente testati non è sufficiente a garantire il corretto funzionamento del sistema
- Spesso si scoprono incongruenze porgettuali
- Normalmente è la fase più dolente
- Si parla di alfa test, beta test.

Manutenzione

- I costi per la manutenzione sono di norma superiori ai costi di sviluppo
- Si parla di un 70% del costo totale (sviluppo più manutenzione)
  - Correttiva (eliminazione errori)
  - Adattativa (alle condizioni esterne)
  - Perfettiva (nuove funzionalità, eliminazione di funzionalità inutili)
    - di norma oltre il 50% della manutenzione

Manutenzione

- Sui costi di manutenzione hanno grande incidenza:
  - la modalità di progetto e sviluppo;
  - la qualità della documentazione

Problemi del ciclo a cascata

- Tende a spostare in avanti le verifiche:
  - Ci si può accorgere troppo tardi di errate concezioni o di errori
- Produce disallineamento tra ciò che viene "fatto" e ciò che era "desiderato"
  - Il prodotto risultante finisce per soddisfare le esigenze degli sviluppatori, anziché quelle dei committenti.

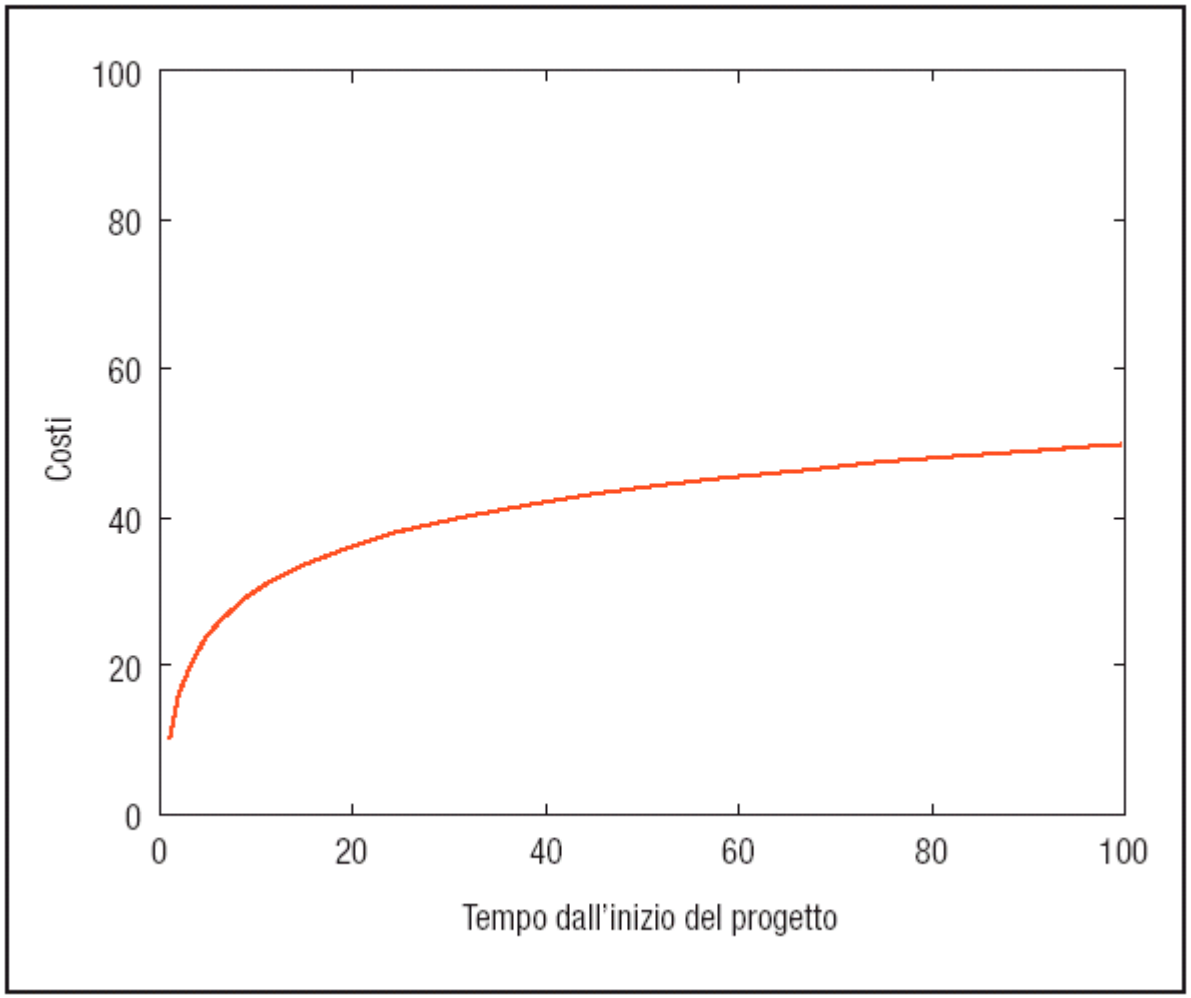
Il modello a cascata

- Presume che sia possibile definire esattamente quali sono i requisiti del sistema
- Presume che i requisiti siano stabili
- E' la traduzione dei processi manifatturieri:
  - La Fiat deve pianificare esattamente la catena di montaggio.
  - Il SW può essere cambiato sempre

Le cose cambiano

- I requisiti non sono stabili durante il ciclo di sviluppo
- L'industria SW non è più la stessa
  - Calcolatori più potenti meno cari
  - Nuove tecniche/linguaggi di programmazione (in particolare OOP)
  - Disponibilità di sistemi di sviluppo, basi dati, CASE tools ecc.

Cosa ci si aspetta



Dall'uso di strumenti, tecniche e metodi attuali

Se i costi sono questi

- Perché investire tanto nella fase iniziale quando non si sa bene dove si va a cascare?
- Perché investire su cose che possono rivelarsi del tutto inutili?

  ...Il tempo dirà al momento opportuno cosa fare e cosa non fare

Inoltre

- Il modello a cascata
  - Tende a irreggimentare
  - Non si adatta allo spirito del programmatore (un progettista/inventore piuttosto che un produttore)
  - E' burocratico
  - impone la produzione di una quantità di semilavorati che disturba il programmatore
- Occorrono modelli meno rigidi

Prototipazione

- Un modo per ridurre i problemi
- **Prototipo**: una realizzazione parziale di un sistema, costruita allo scopo di permettere a committenti, utenti e sviluppatori una miglior conoscenza del problema e delle sue soluzioni
  - Usa e getta
  - Prototipo evolutivo

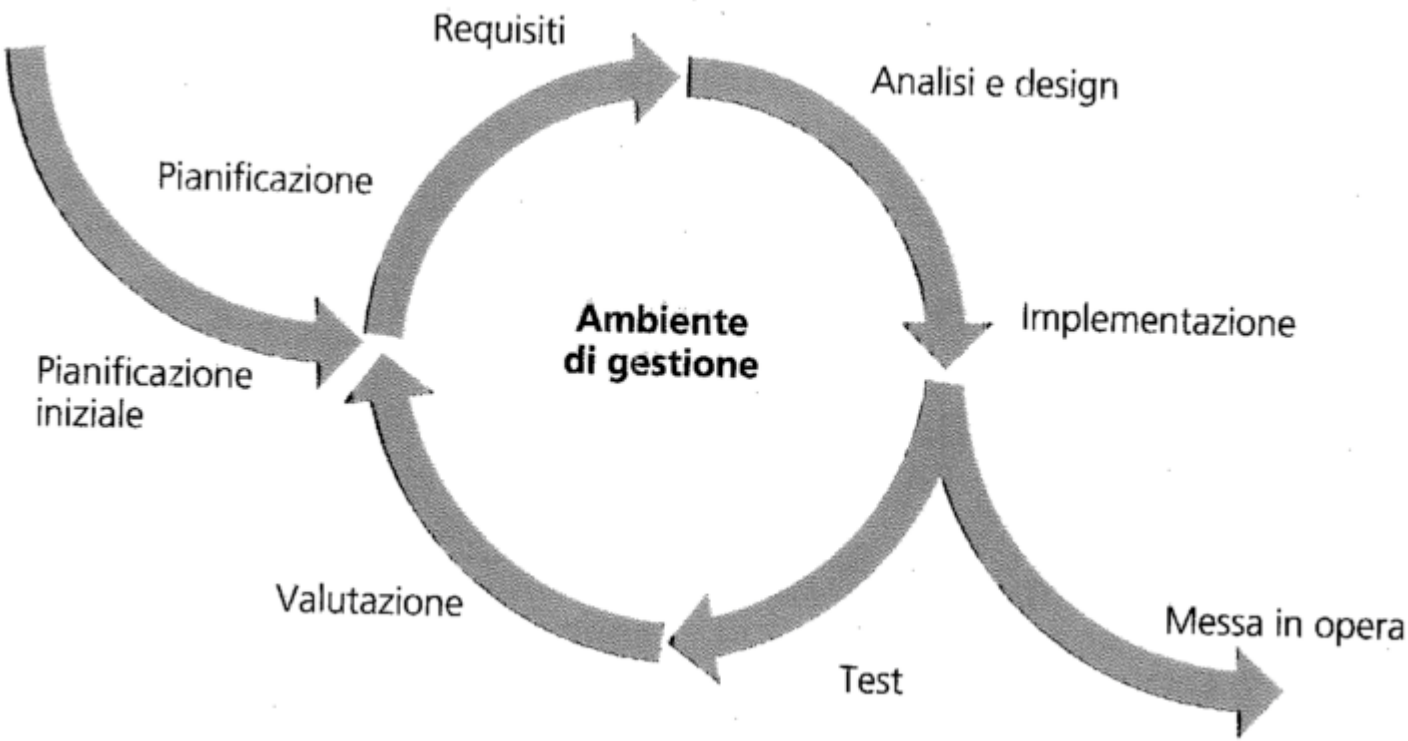
Prototipazione

- L'uso di prototipi aiuta nella scrittura di SRS
  - Il committente preferisce vedere un prototipo piuttosto che un documento scritto
  - Il prototipo mette in evidenza aspetti non previsti
  - Alcuni requisiti (p.e., i formati delle videate) possono essere estratti direttamente dal prototipo e portati in SRS.

Prototipazione

- Il processo deve essere ordinato: successivi rilasci di SRS e relativi prototipi, fino ad arrivare a SRS definitivo.

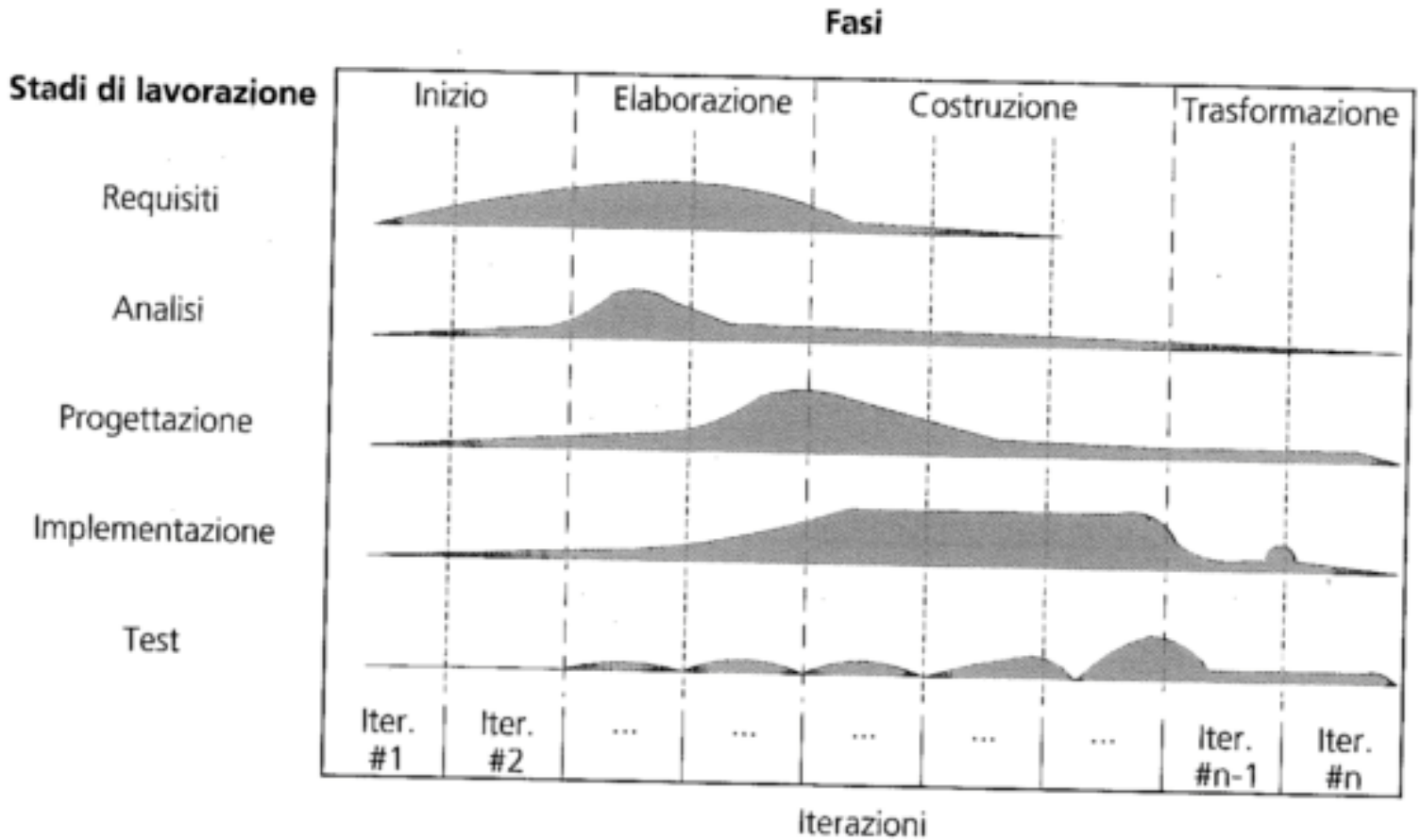
Modello iterativo



UP Unified Process

- Modello proposto da chi ha definito UML
- E' iterativo e incrementale
  - Ogni iterazione è fatta di:
    - Requisiti, analisi, progettazione, implementazione, test
- E' pilotato dai casi d'uso (requisiti)

UP


Fasi
Stadi di lavorazione
Inizio
Elaborazione
Costruzione
Trasformazione
Requisiti
Analisi
Progettazione
Implementazione
Test
Iter. #1
Iter. #2
...
...
...
...
...
Iter. #n-1
Iter. #n
Iterazioni

UP

- Sostanzialmente ci riferiremo ad esso
  - W. Zuser, S. Biffl, T. Grechenig, M. Kohle,, “Ingegneria del software con UML e Unified process”, McGraw-Hill, 2004

Modelli *leggeri/agili*

- Nati in opposizione (estrema) ai metodi tradizionali
- I metodi tradizionali sono inadeguati specialmente quando:
  - i requisiti sono incerti o variabili
  - si richiede una rapida risposta ai cambiamenti di mercato

Modelli *leggeri/agili*

- Ogni applicazione pone problemi diversi
- E' irrealistico definire un processo universalmente valido, senza cadere nei generici richiami al buon senso
- Occorre adattare il processo alle specifiche esigenze del caso

Le tecnologie moderne aiutano

Modelli *leggeri/agili*

- Il lavoro aggiuntivo per i semilavorati intermedi risulta in una burocrazia inutile, che influisce in maniera negativa su flessibilità e efficienza.
- ridurre il lavoro di supporto, convogliando il massimo dello sforzo sulla mera produzione dell'applicazione

Manifesto (2001)

http://www.agilealliance.org/

- Privilegiare
  - ***gli individui e le loro interazioni*** rispetto a processi e strumenti
  - ***la disponibilità di sw funzionante*** rispetto a un'ampia documentazione
  - ***la collaborazione col cliente*** rispetto alla negoziazione dei contratti
  - ***la pronta risposta ai cambiamenti*** rispetto all'esecuzione di un piano

Assunzioni

- Dettagliate specifiche formali, approfondite analisi e puntigliosa documentazione sono considerate attività troppo costose rispetto ai benefici apportati
  - limitano la flessibilità del processo che deve poter modificare i propri obiettivi in ogni momento.

Assunzioni

- La produzione di un'applicazione non è un'attività che possa essere analizzata e precisamente pianificata a priori
  - come quando si guida l'auto: la condotta complessiva è il risultato di un gran numero di minimi cambiamenti di rotta che il pilota decide in base alla sua istantanea percezione di curve ed ostacoli.

What is XP?

- XP is a lightweight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements.

  -- Kent Beck.

  - XP is:
    - Humane.
    - Honest.
    - Productive.
    - Professional.
    - Fun.

What is XP?

- XP is a discipline of software development.
- There are certain things you must do.
  - You must write tests before code.
  - You must program in pairs.
  - You must integrate frequently.
  - You must be rested.
  - You must communicate with the customer daily.
  - You must follow the customer's priorities.
  - You must leave the software clean and simple by the end of the day.
  - You must adapt the process and practices to your environment.

eXtreme Programming

- Evita la produzione di semilavorati diversi da quelli necessari alla realizzazione delle applicazioni
- Mira a fornire meccanismi che, in corso d'opera, danno agli sviluppatori la consapevolezza che ciò che stanno costruendo soddisferà pienamente le esigenze del committente/utente

Attività

- Quattro attività fondamentali che si ripetono per tutto il progetto:
  - scrittura del codice dell'applicazione (coding);
  - verifica delle funzionalità (testing);
  - osservazione dell'ambiente inteso come desideri del committente, opportunità tecnologiche, sviluppi di mercato (listening);
  - progetto dell'applicazione (design).

Criteri

- I programmatori sono responsabili non solo della codifica dell'applicazione, ma anche della sua verifica
- L'enfasi è sulla verifica:
  - Prima di scrivere le istruzioni pensare a come si testano
  - nessuna istruzione dovrebbe essere considerata veramente parte dell'applicazione finché non ne sia stato verificato l'effetto secondo le attese

What makes XP familiar?

- XP matches the behavior of successful programmers in the wild.
  - Tests.
  - Refactoring.
  - Evolutionary delivery.
  - Incremental planning.
  - Low overhead.

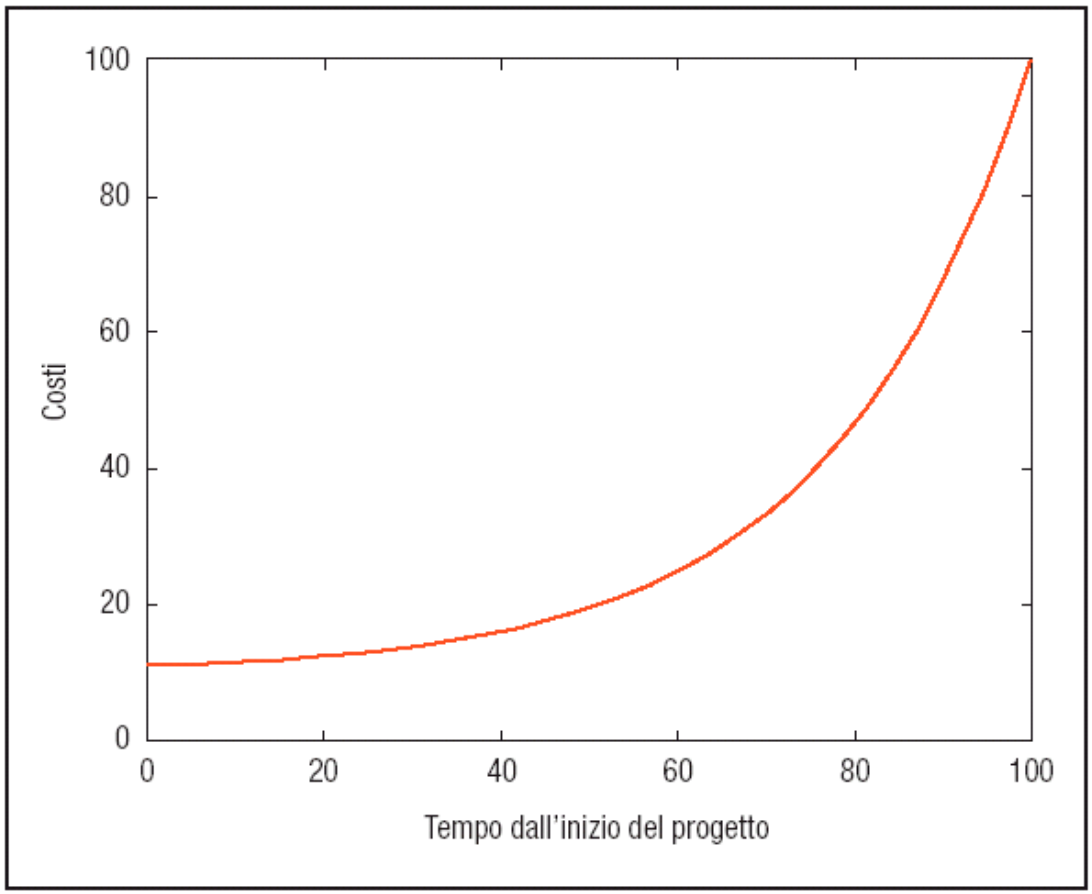
Processo

- La costruzione dell'applicazione procede in maniera iterativa
  - cogliendo le reazioni dei committenti e riprogettando, in maniera adeguata, le sue funzionalità e i meccanismi adottati per ottenerle

Extreme programming

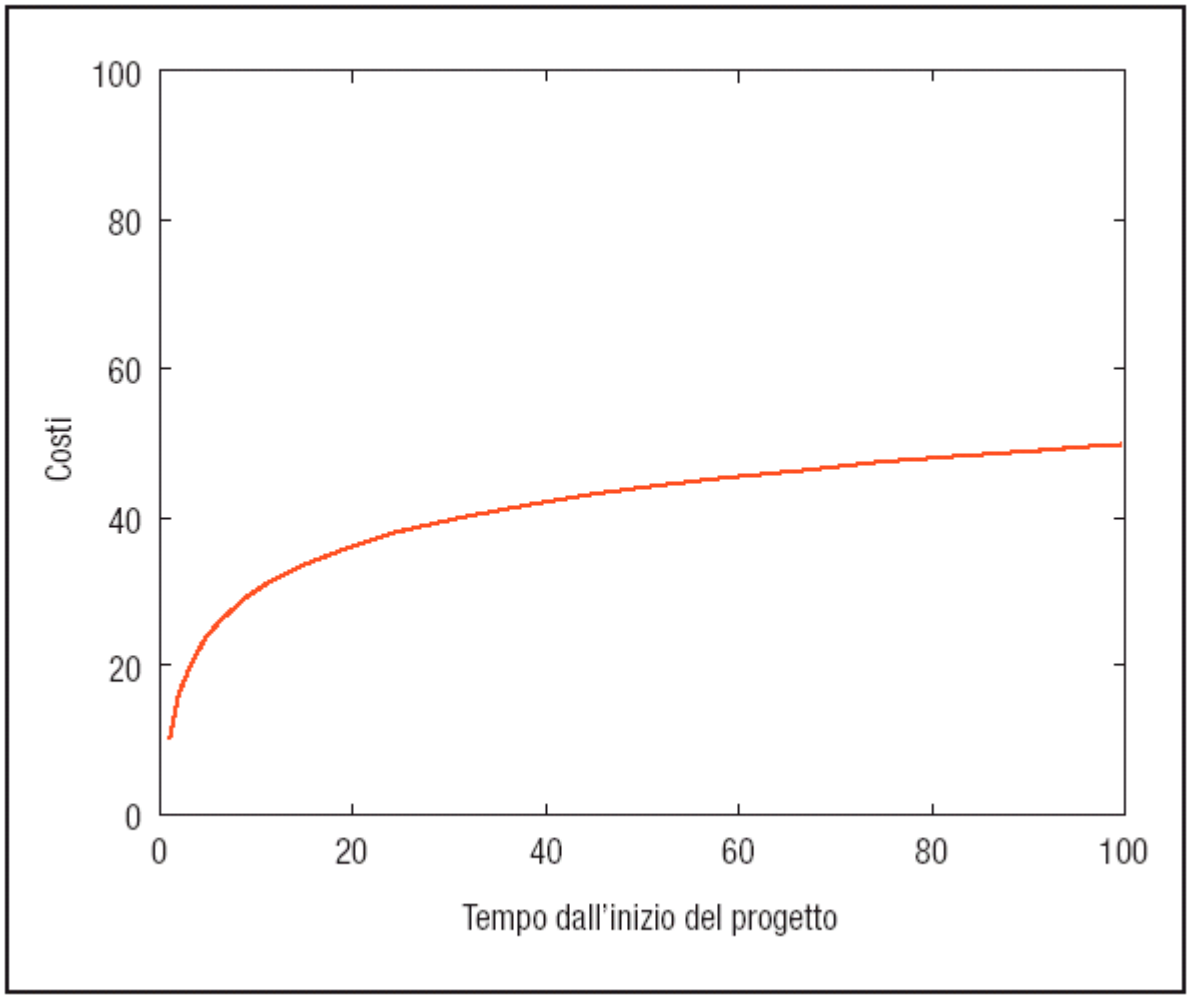
- Pianificazione attività (planning the game)
- Brevi Iterazioni, Rilasci frequenti (Short releases)
- Progetti semplici (simple design)
- Integrazione continua (continous integration)
- Ristrutturazione del codice (refactoring)
- Verifica di ogni funzionalità (testing)
- Programmazione a coppie (pair programming)
- Proprietà collettiva del codice (collective ownership)
- Partecipazione del committente (onsite costumer)
- Uso di standard di codifica (coding standards)
- Rinuncia la lavoro straordinaro (40 h/w)

Costi modifiche (tradizionale)



Prevenire è meglio che curare !

Costi con XP (pretesi)



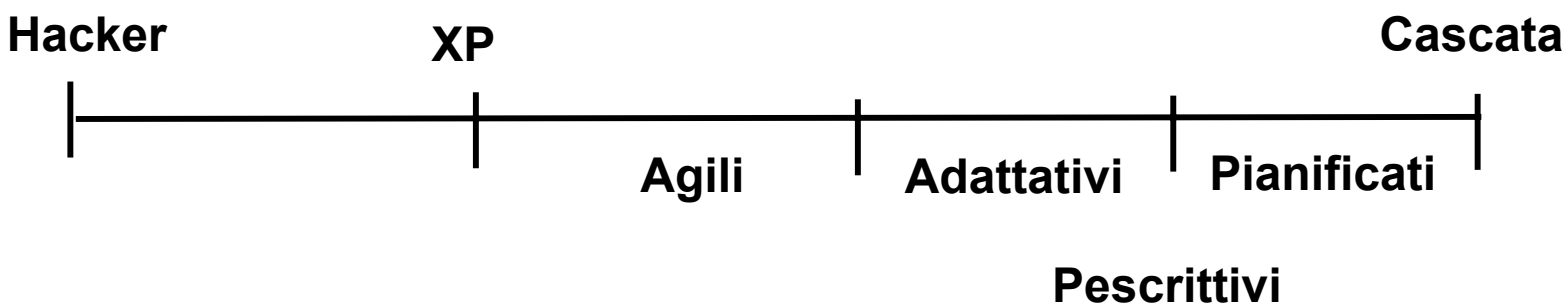
Sarà vero? Quasi sicuramente no!

Vantaggi XP

- Coinvolgimento del cliente
- Attività più gratificante per il programmatore
- Riduzione dei costi
- Maggior adeguamento al cambiamento in corso d'opera

Problemi XP

- C'è il rischio di cadere nel CODE & FIX
  - Contrario a ogni pratica ingegneristica
- Conta troppo sulla qualità delle persone
  - Chi fa software non sempre è un genio!!!



Lavoro individuale

- Approfondire il modello XP
  - andare al sito dell'XP
    www.extremeprogramming.org
- Visitare il sito

  http://www.agilemodeling.com/

Riferimenti per metriche

- N.E. Fenton e S.L. Pfleeger “Software Metrics, a rigorous approach, second edition”, PWS publishing company, 1997
- L. Buglione, “Misurare il software, seconda edizione” Franco Angeli, 2003

Bibliografia

- W. Zuser e altri, “Ingegneria del software con UML e Unified process”, McGraw-Hill, 2004
- R.S. Pressman, “Principi di ingegneria del software” McGraw-Hill 2004
- Kent Beck “Extreme Programming Explained: Embrace Change”, Addison-Wesley (1999)
- www.extremeprogramming.org
- B. Boehm, “Get Ready with Agile methods, with Care”, IEEE Computer Genn. 2002, pp.64-69