

*Corso di Laurea in Ingegneria Informatica*

# Corso di Ingegneria del Software

---

## Ciclo di Vita del Software

# Sommario

- Attività fondamentali nel ciclo di vita
- Modelli di Processo Software
- Modello a cascata e sue varianti
- Modello a V
- Modelli evolutivi
- Modello trasformazionale
- Modello a spirale

# Ciclo di Vita del Software (CVS, o SLC)

- ◆ Un modello del ciclo di vita del software (CVS) – o Software Life Cycle, SLC - è una caratterizzazione descrittiva o prescrittiva di come un sistema software viene o dovrebbe essere sviluppato (W. Scacchi - *Encyclopedia of Software Engineering Vol. II pag. 860*)
- ◆ Gli obiettivi sono i) determinare l'ordine delle attività nello sviluppo e nell'evoluzione del software, e ii) stabilire criteri di transizione per progredire da uno stadio di lavorazione al successivo

# **Code and Fix**

- ◆ Approccio “primitivo” alla produzione del software che consiste nello 1) scrivere codice e 2) aggiustarlo per correggere errori, migliorare e/o aggiungere funzionalità
- ◆ Inadeguato per lo sviluppo odierno del software:
  - Le grandi dimensioni dei sistemi rendono difficile la gestione non strutturata della complessità
  - Difficile la gestione non organizzata del personale (ad es. la gestione del turn over resa difficile dalla mancanza di documentazione)
  - Difficile aggiustare/modificare/ristrutturare il codice
  - Interpretazione sbagliate dei requisiti utente
  - Processo non prevedibile (tempi/costi), qualità non misurabile

# Necessità dei modelli di processo

- ♦ Questi problemi portarono a riconoscere la necessità di processi predicibili e controllabili, dunque strutturati
- ♦ Avere un buon processo è fondamentale. La qualità del processo di sviluppo influenza:
  - La **qualità** dei prodotti finali
  - I **tempi** per portare il prodotto sul mercato
  - I **costi**
  - Le **prestazioni** su diversi progetti

# Processi e Modelli di Processo Software: definizioni

## ◆ Processo :

- un insieme di attività concentrate nel tempo finalizzate alla realizzazione di un particolare output.

## ◆ Processo Software:

- Un insieme strutturato di attività necessarie per lo sviluppo di un sistema software.

## ◆ Modello di Processo Software:

- E' una rappresentazione semplificata di un processo. Esso fornisce una descrizione del processo da una particolare prospettiva.
  - ◆ Es. La prospettiva delle attività del processo, o quella dei ruoli del personale coinvolto

# Caratteristiche dei Processi Software

- ◆ Esistono diversi tipi di software e non c' è un processo universale valido per tutti.
- ◆ Tutti i processi software includono comunque le stesse attività fondamentali.
- ◆ Ogni processo si caratterizza per l' organizzazione delle attività che esso include, ma anche per:
  - I prodotti, o deliverables che le sue attività producono.
  - I ruoli assegnati alle persone coinvolte nel processo.
  - Le pre e post-condizioni di ciascuna attività

# Attività fondamentali di un Processo SW

**Studio di fattibilità:** Fornisce un documento con:

Definizione del problema

Valutazione **Costi/Benefici**

- ◆ Risorse finanziarie e umane
- ◆ Soluzioni alternative
- ◆ Tempi di consegna e modalità di sviluppo

**Acquisizione, analisi e specifica dei requisiti:**

Definire cosa il sistema dovrà fare, non come

Specificare le funzionalità e le qualità che deve possedere, senza vincolare la progettazione e l'implementazione

Richiede la conoscenza del dominio

N.B. È un'attività critica. Un errore in questa fase può costare molto

# Analisi e specifica dei requisiti

**L'ingegneria dei requisiti** sviluppa metodi per **raccogliere, documentare, classificare e analizzare** i requisiti

Sono compiti dell'analista:

- ◆ Identificare i portatori di interesse (**stakeholders**)
- ◆ Esplicitare i requisiti
- ◆ Conciliare i vari punti di vista (eventualmente contraddittori)
- ◆ Specificare i requisiti

Il risultato è la produzione di

- ◆ **Documento di Specifica dei Requisiti (DSR)**, *comprendibile, preciso, completo, coerente, non ambiguo, modificabile.*
  - ◆ Contiene una descrizione del **dominio**, dei **requisiti funzionali, non-funzionali**, e **requisiti del processo di sviluppo-mantenzione**
- ◆ **Piano di Test di Sistema (PTS)**

# Progettazione

## Strutturazione del sistema a diversi livelli di dettaglio

- ◆ **Architettura generale** (hardware e software) assegna funzionalità a componenti di alto livello
- ◆ **Architettura dettagliata** arriva alla definizione precisa dei moduli, delle loro funzionalità e delle interfacce
- ◆ Produzione **Documento di Progetto** (DSP)
  - Descrive i componenti del sistema, le loro interfacce, le relazioni tra di loro
  - Registra le decisioni significative e ne spiega le motivazioni
  - Importante per eventuali richieste future di cambiamento
  - La forma del documento è determinata dagli standard adottati dall'azienda

# Codifica, test e rilascio

- ◆ **Produzione** del codice
  - Può essere soggetta a standard aziendali, a convenzioni
- ◆ Test di Unità
- ◆ Test di Integrazione e di Sistema
- ◆ Test di Accettazione

## Manutenzione

- ◆ **Correttiva, Adattativa, Perfettiva**
- ◆ Tutte le attività descritte contengono compiti comuni: documentazione, verifica, gestione

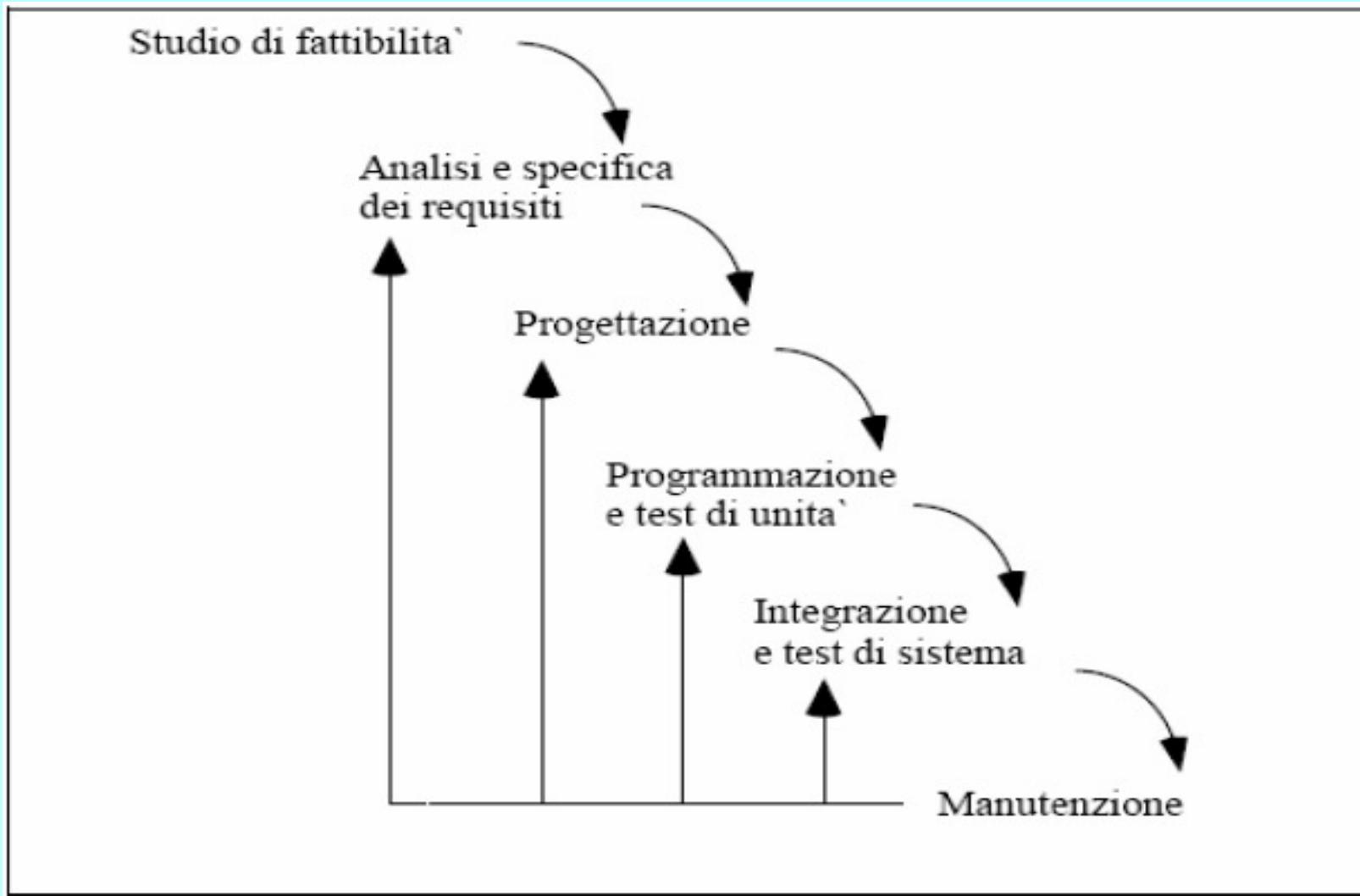
# Modello a cascata *(Waterfall Model)*

# Modello a Cascata

## organizzazione sequenziale delle fasi

- ◆ Ogni fase raccoglie un insieme di attività omogenee per metodi, tecnologie, *skill* del personale, etc.
- ◆ I semilavorati output di una fase sono input alla fase successiva
- ◆ i prodotti di una fase vengono “congelati”, ovvero non sono più modificabili se non innescando un processo formale e sistematico di modifica
- ◆ Il processo è guidato dalla produzione di documentazione

# Modello a cascata



# Modello a cascata

- ❖ La fine di ogni fase è un punto rilevante del processo (**milestone**)
- ❖ L'output di ogni fase è chiamato ***deliverable***
- ❖ La definizione precisa dei ***deliverable*** è importante per misurare il progresso di un progetto

## Criticità del modello a cascata

- ❖ È un modello ideale, che può essere solo approssimato nella pratica
- ❖ È possibile caratterizzarlo mediante tre proprietà: ***linearità, rigidità, monoliticità***
- ❖ Nella pratica sono necessari ***feedback***

# Criticità del modello a cascata

- ◆ A causa della *rigidità*, si assume che i requisiti possano essere congelati nelle prime fasi, quando le conoscenze sono ancora preliminari
- ◆ *Monoliticità*: se si commette un errore nei requisiti, viene fuori solo alla fine, dopo il rilascio, con costi esorbitanti
- ◆ Stime dei costi difficili
- ◆ Difficile anticipare i cambiamenti. Bassa *evolvibilità*
- ◆ Può richiedere una quantità ingente di documenti
- ◆ La difficoltà di produrre specifiche complete causa molta attività di manutenzione (che è particolarmente costosa)

# Una variante del Waterfall Model: V&V e Retroazione (Feedback)



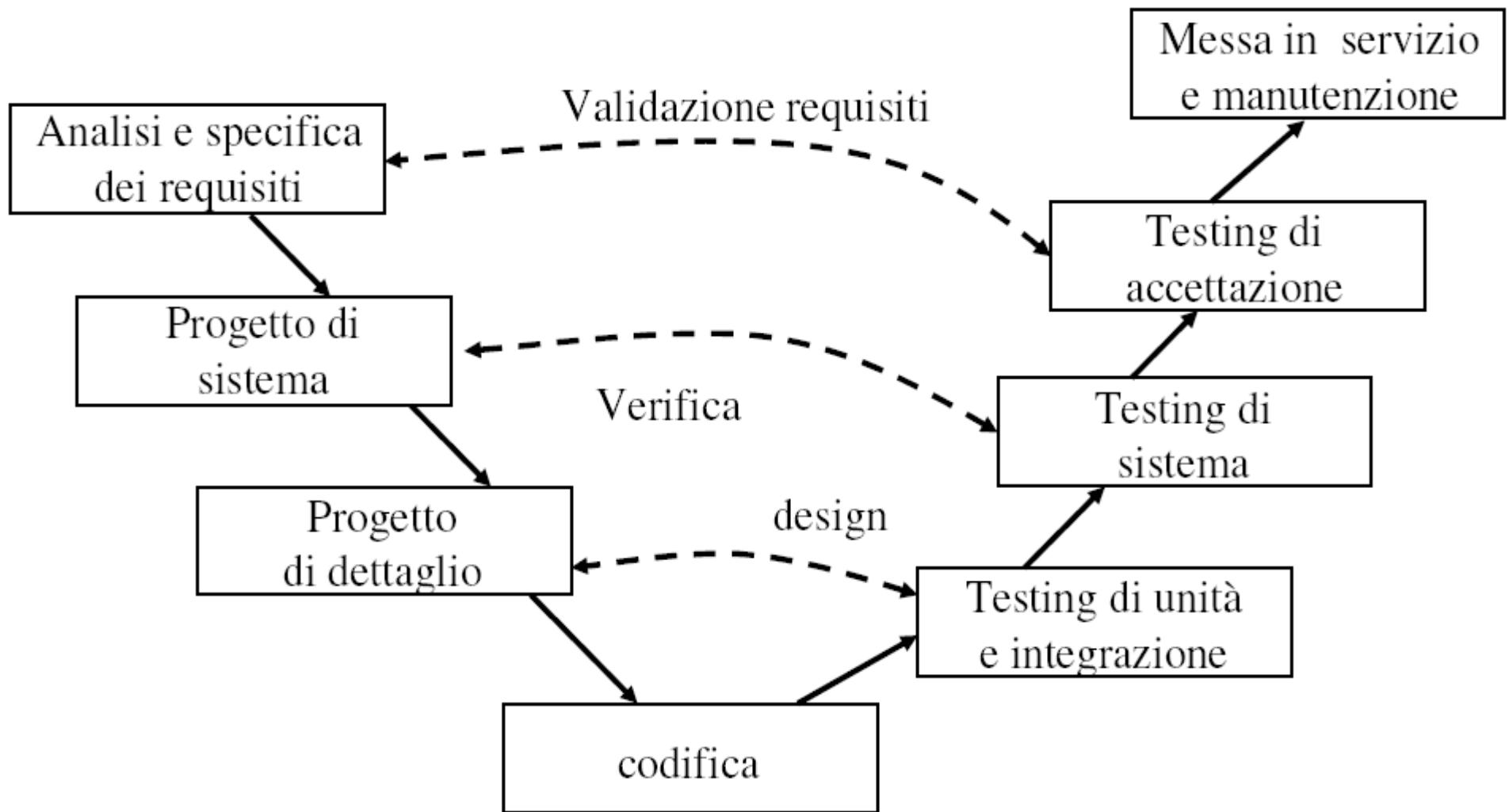
# V&V e Retroazione (Feedback)

- ♦ Variante del modello a cascata che tenta di superarne la monoliticità
- ♦ Introduce dei **feedback** in ogni fase. Si possono così rilevare errori prima del rilascio
  - Tuttavia resta un modello che non “anticipa” i possibili cambiamenti
  - Può essere utile quando si prevede che il sistema sarà poco soggetto a cambiamenti

# Modello a V

## (V-Model)

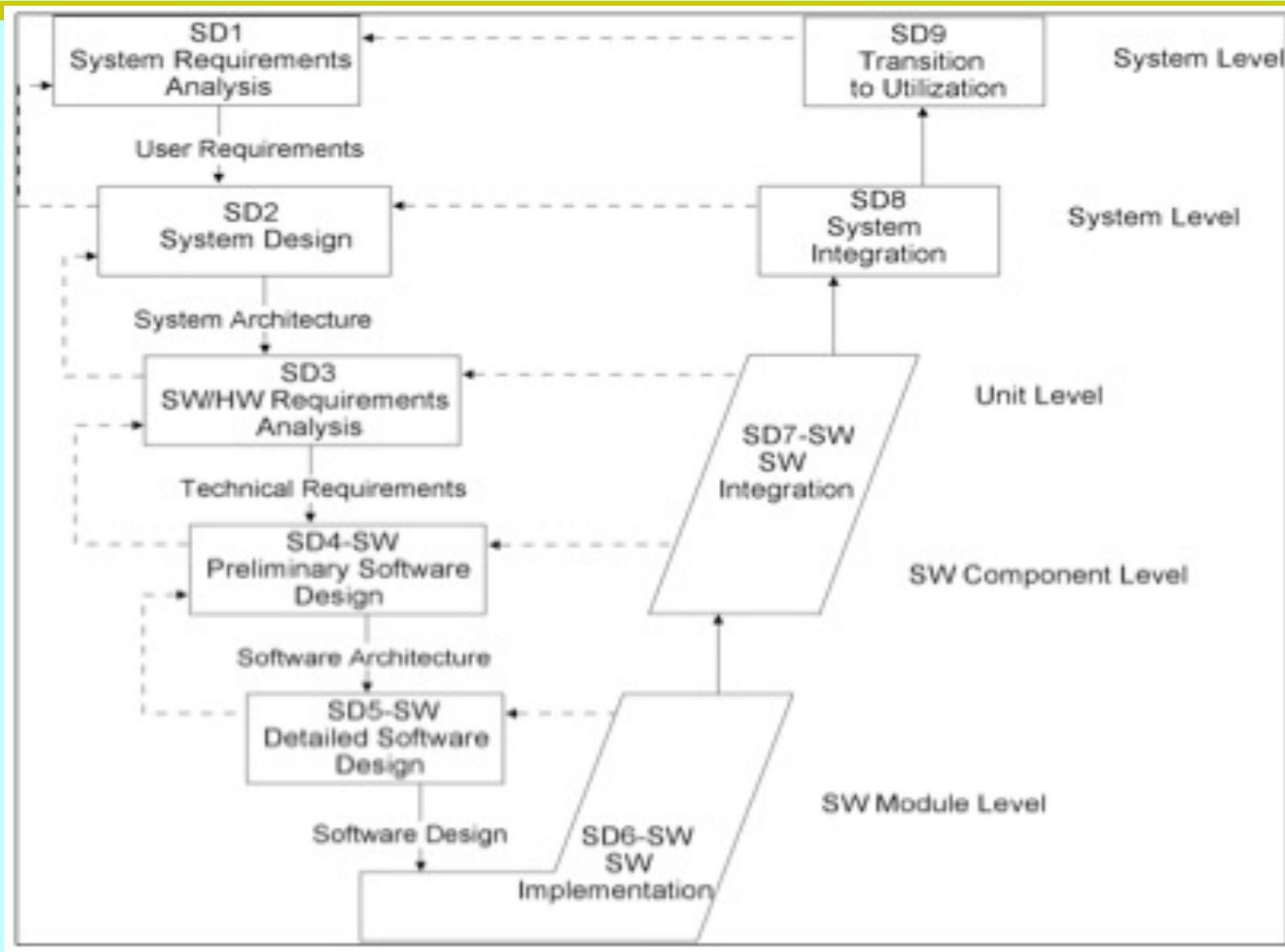
# Il Modello a V



# Strategia del Modello a V

- ◆ Le attività del ramo di sinistra sono collegate a quelle del ramo di destra destra
- ◆ Durante le attività di sinistra, vengono progettati i test della fase a destra corrispondente (ad es., alla specifica dei requisiti corrisponde la progettazione dei test di accettazione)
- ◆ Se si trova un errore in una fase a destra (es. *test* di sistema) si riesegue la fase a sinistra collegata
- ◆ Si può iterare migliorando requisiti, progetto, codice

# Esempio: Standard per sistemi IT della repubblica federale tedesca



# Modello a V

- ◆ Prevede una V per lo sviluppo di sistemi HW/SW, con una doppia retroazione.
  - *N.B. Sviluppo di sistemi, di cui il software è una parte*
- ◆ Esamina dapprima i requisiti utente di **sistema** (sia funzionali sia non funzionali)
- ◆ Identificazione delle unità del sistema e assegnazione dei requisiti alle unità
- ◆ L'analisi dei requisiti hardware/software esamina le risorse di ogni unità, decomponendole in
  - CSCI (Computer Software Configuration Items)
  - HCI (Hardware Configuration Items)

# Modello a V

- ◆ Si passa poi alla progettazione software vera e propria (allocazione di un task set per ogni CSCI)
- ◆ La progettazione software *dettagliata* alloca risorse ai moduli software imponendo requisiti temporali e produce il progetto dettagliato di ogni CSCI
- ◆ Infine si passa all'implementazione, al test ed integrazione software, e al test ed integrazione di sistema prima dell'utilizzo
- ◆ *Le varianti del modello a cascata sono adatte soprattutto quando: a) si prevede che il sistema (e/o l'ambiente) sarà poco soggetto a cambiamenti, b) vi sono requisiti chiari e completi sin da subito con poca possibilità di cambiare (ad es. sistemi non interattivi)*

# Modelli evolutivi

# Sviluppo evolutivo

- ◆ Basato sull'idea di produrre una versione iniziale del software, esporla agli utenti e perfezionarla attraverso varie versioni.
- ◆ Noto anche come Sviluppo Prototipale.

# I Prototipi nei Modelli Evolutivi

- ♦ **Prototipo:** Modello approssimato dell'applicazione, il cui obiettivo è di ricevere feedback per affinare i requisiti
  - Il modello prototipale è dunque adatto quando i requisiti non sono completi e non ambigui
  - Si usa il prototipo per raccogliere feedback dagli opportuni stakeholder
    - ♦ Es: prototipo della GUI per chiarire i requisiti definendo come si deve presentare l'applicazione

# Modelli Evolutivi

- ◆ Due modelli fondamentali:
- ◆ **A) Sviluppo con Prototipo Usa e Getta**
  - L’obiettivo è di comprendere i requisiti o altri aspetti del sistema. Si parte da requisiti poco chiari e si realizzano prototipi per esplorare i requisiti e chiarirli.
- ◆ **B) Sviluppo Esplorativo (Con prototipo evolutivo)**
  - L’obiettivo è di lavorare col cliente per esaminare i requisiti iniziali e farli evolvere fino al sistema finale. Dovrebbe partire da pochi requisiti ben compresi e aggiungere nuove caratteristiche proposte dal cliente.

# (A) Sviluppo con Prototipo Usa e Getta (“Throw Away”)

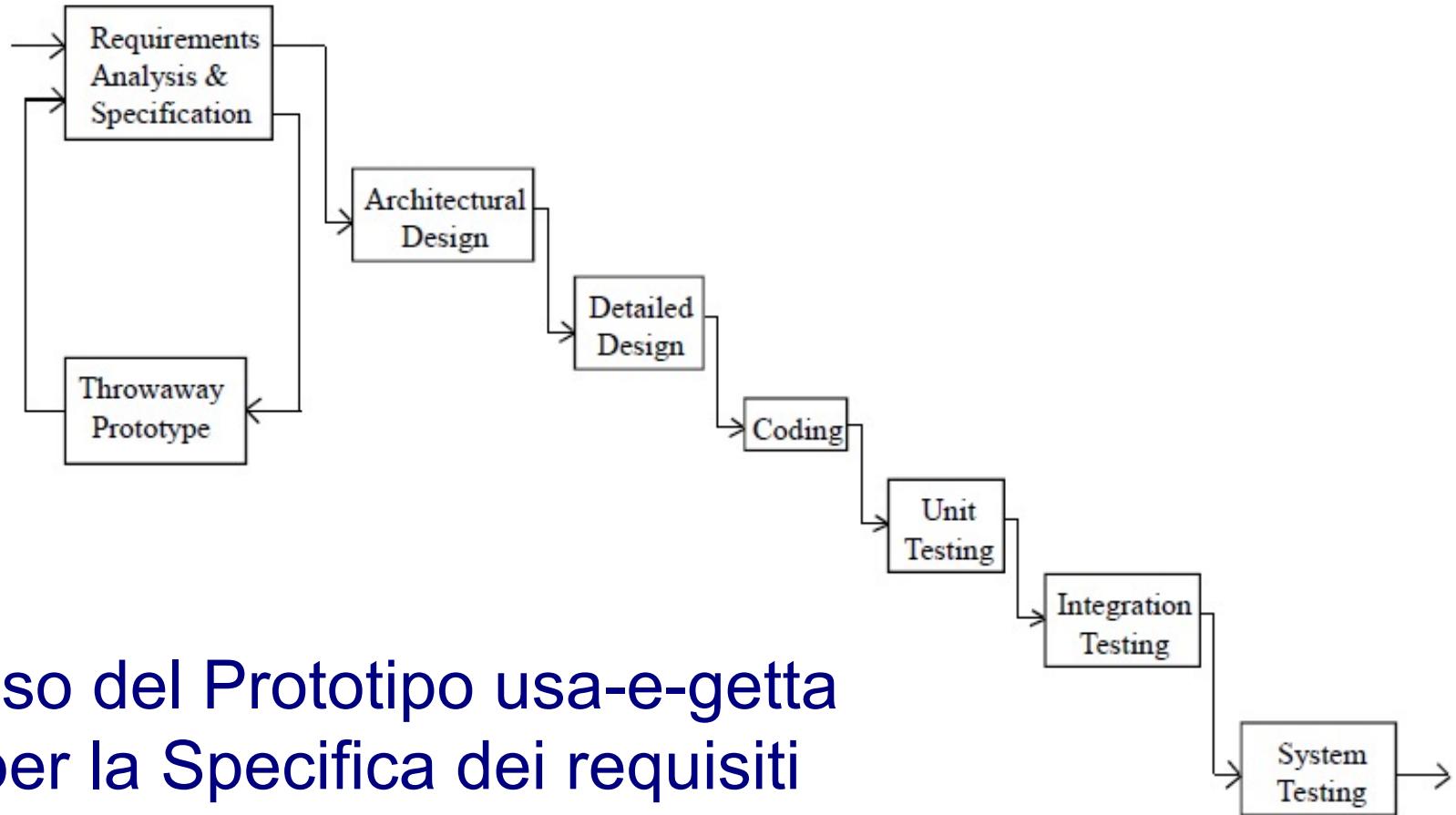
- ◆ Realizzazione di una prima implementazione (prototipo), più o meno incompleta, da considerare come una ‘prova’, con lo scopo di:
  - accettare la Fattibilità del prodotto
  - validare i Requisiti
- ◆ Lo sviluppo effettivo inizia dalla seconda versione (ad es. con un approccio a cascata)
- ◆ Questo approccio (ispirato al principio “**Do it twice**”) consente di ridurre errori sui requisiti

# Modello prototipale con prototipo usa e getta

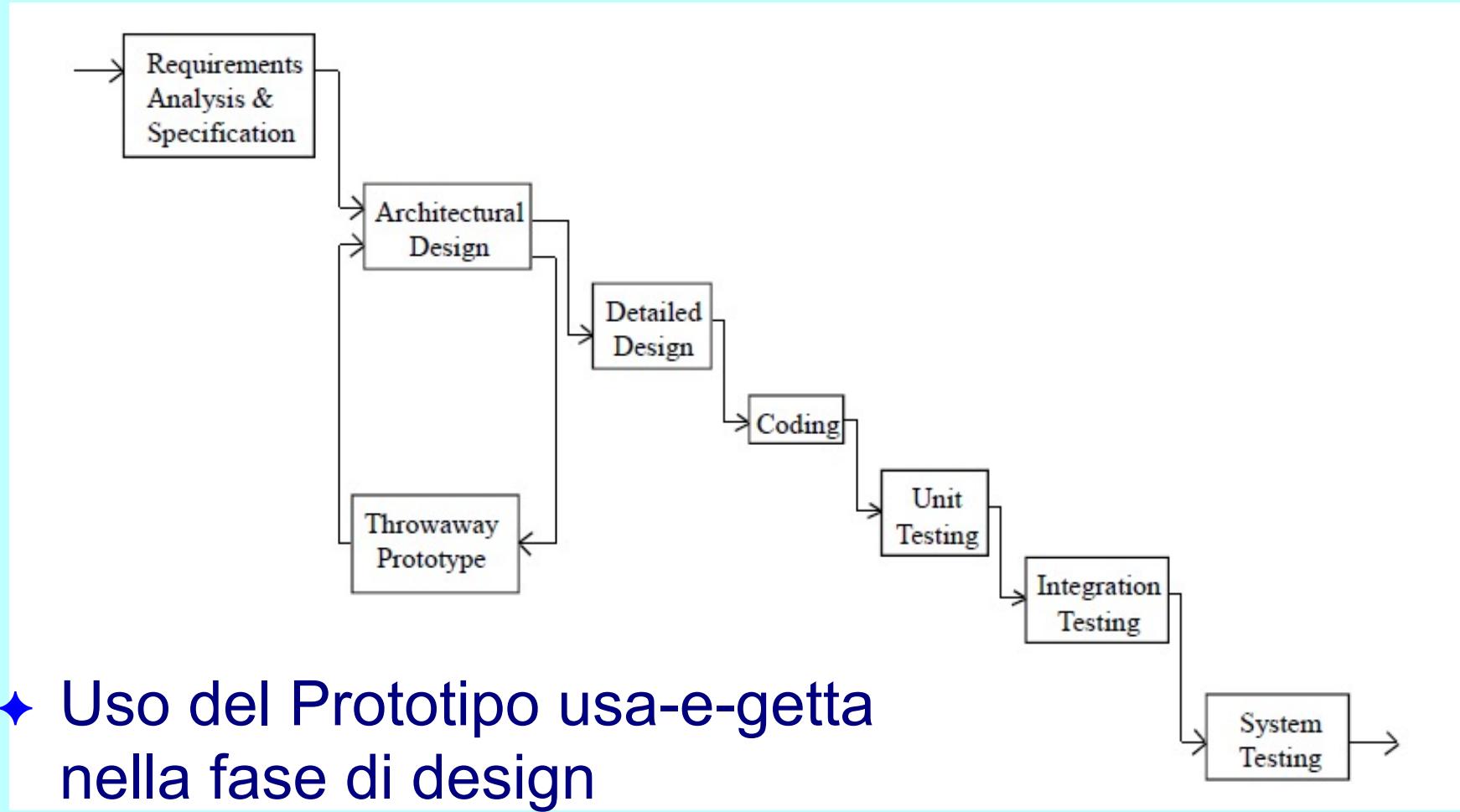


- ◆ Il prototipo è uno strumento di identificazione dei requisiti di utente; è incompleto, approssimativo, realizzato utilizzando parti già possedute o *routines stub*.
- ◆ Il prototipo viene gettato dopo che è servito per chiarire i requisiti

# Esempio di Modello Prototipale

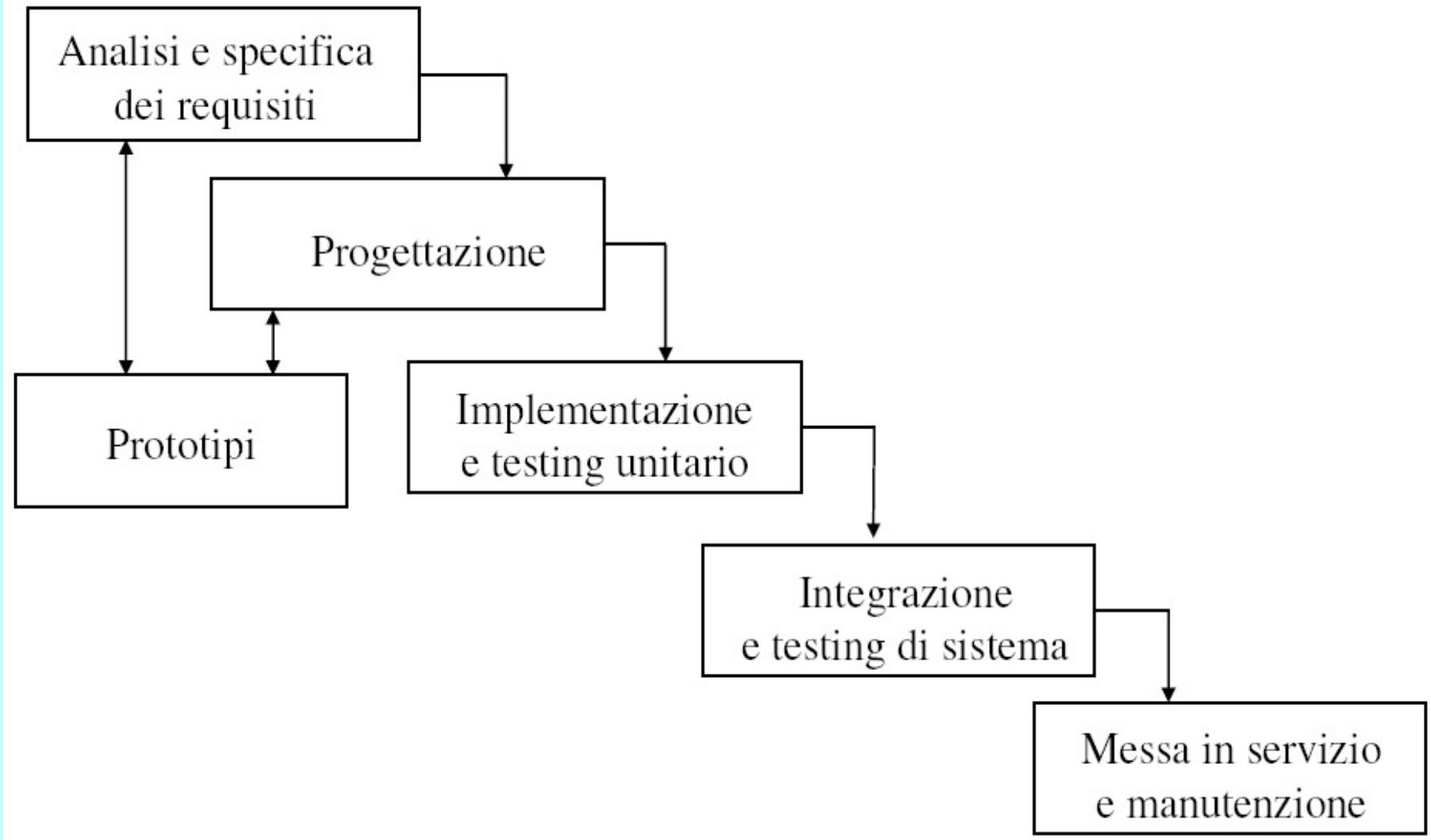


# Variante di Modello Prototipale



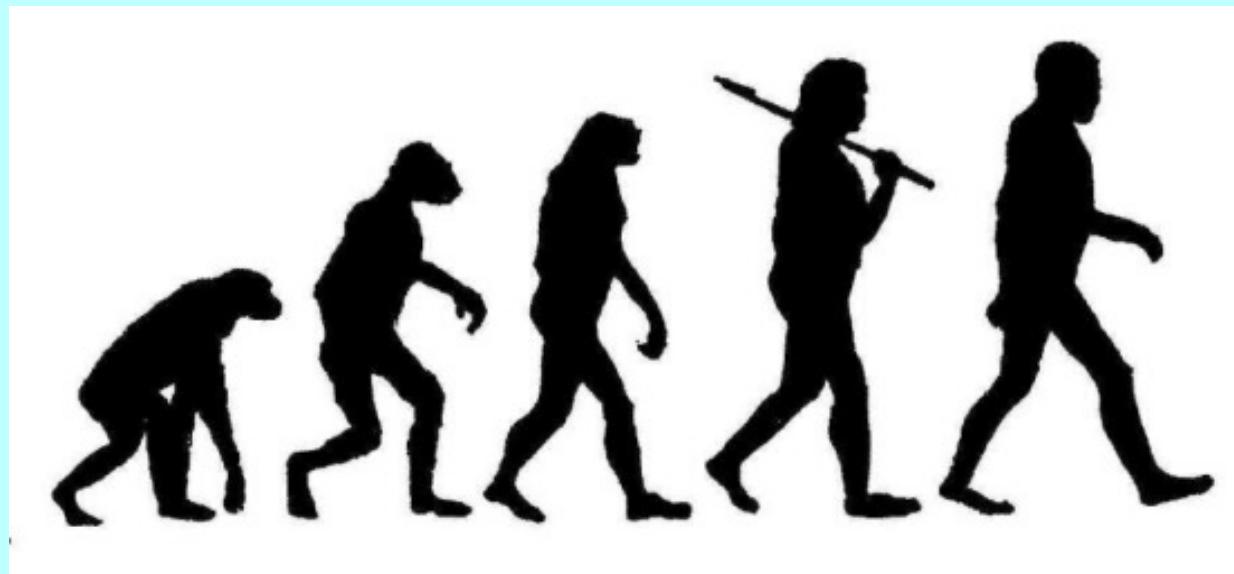
- ◆ Uso del Prototipo usa-e-getta  
nella fase di design

# Altra variante di Modello Prototipale

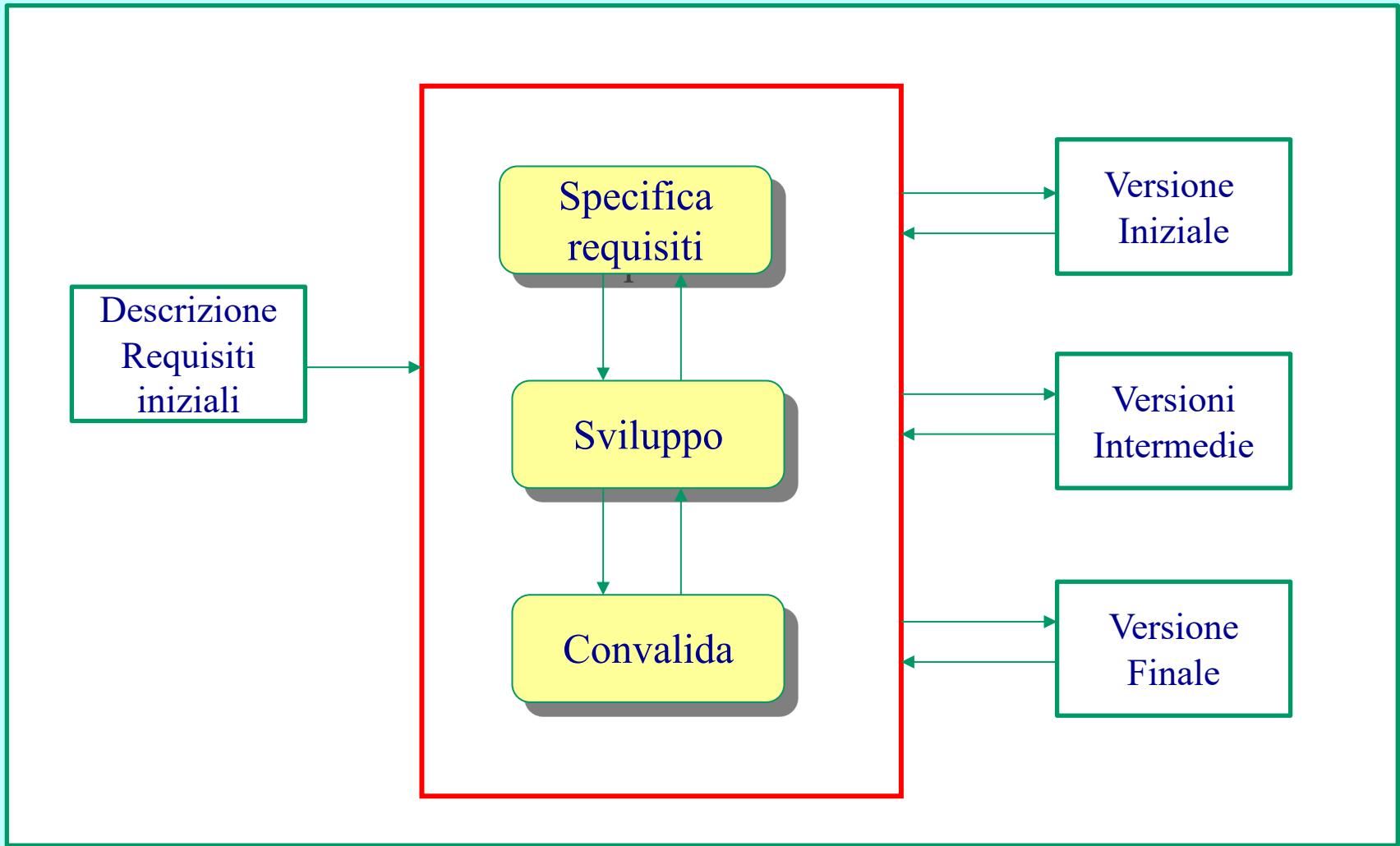


# (B) Sviluppo con prototipo evolutivo

- ◆ Prototipo evolutivo: Le versioni intermedie del prototipo vengono iterativamente raffinate e completate. L' n-esimo prototipo viene rilasciato.



# Sviluppo Evolutivo-Esplorativo



# Sviluppo Evolutivo Esplorativo

---

- ◆ Un modello di processo evolutivo esplorativo è ***“un modello le cui fasi sviluppano versioni incrementali di un prodotto con una direzione evolutiva determinata dall'esperienza pratica”***

# Sviluppo Evolutivo Esplorativo

## ◆ Vantaggi:

- Rapido feedback e possibilità di far cambiare i requisiti

## ◆ Problemi

- Mancanza di visibilità del processo (è anti-economico documentare ogni versione del sistema);
- I sistemi diventano spesso mal strutturati (per i continui cambiamenti);
- Richiedono particolari skills (es. linguaggi di prototipazione rapida)

## ◆ Applicabilità

- A sistemi interattivi di piccole e medie dimensioni (<500.000 LOC);
- Per sviluppare alcune parti di sistemi di grandi dimensioni (per es. l'interfaccia utente);
- Per sistemi destinati a vita breve.

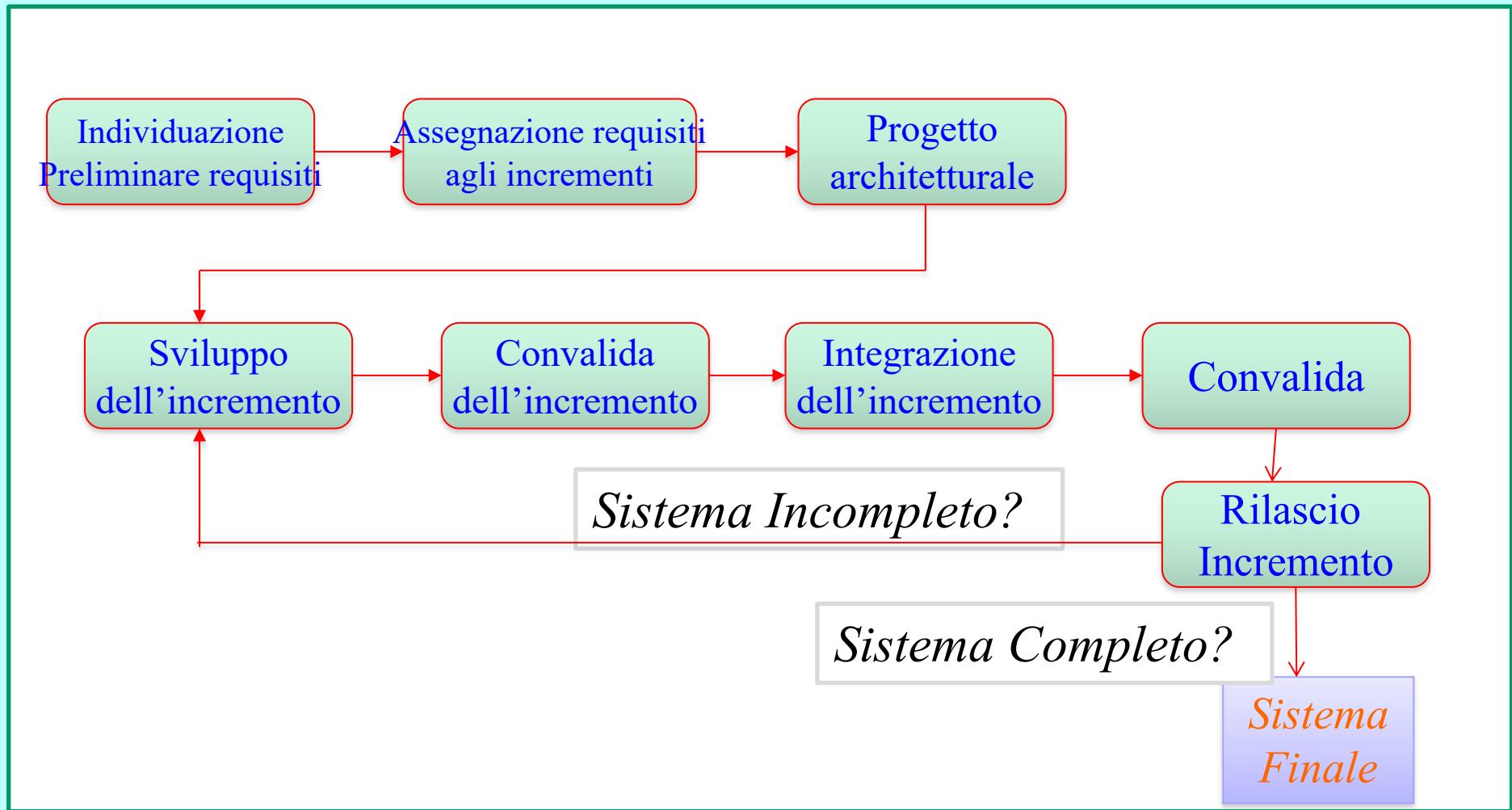
# Modello Incrementale

# Modello con Sviluppo e Consegna Incrementale



- ◆ Piuttosto che consegnare il sistema tutto in una volta, lo sviluppo e la consegna sono eseguiti per incrementi, dove ogni incremento rilascia parte delle funzionalità richieste.
- ◆ In ogni step si usa il modello a cascata
- ◆ L'utente è coinvolto nella definizione dei successivi incrementi
- ◆ Si evita di produrre funzionalità non richieste

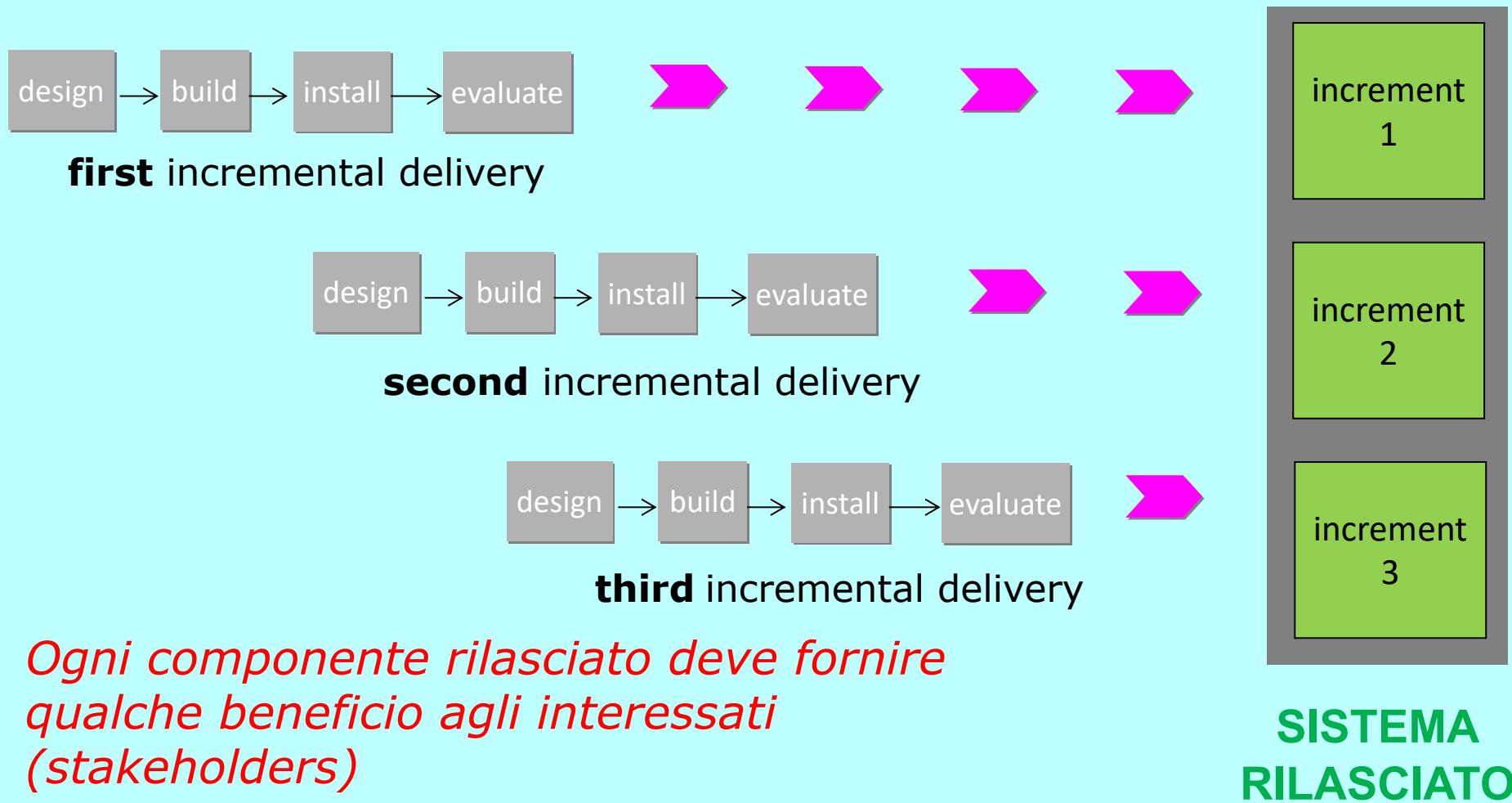
# Processo di Sviluppo e Consegnna Incrementale



# Sviluppo e Consegnna Incrementale

- ◆ Ai requisiti Utente vengono associati livelli di priorità e quelli a priorità maggiore vengono rilasciati con i primi incrementi.
- ◆ Una volta partito lo sviluppo di un incremento, i relativi requisiti devono essere congelati, mentre i requisiti coinvolti in incrementi successivi possono continuare ad evolvere.
- ◆ I servizi comuni possono essere implementati all'inizio del processo, o quando una funzione è richiesta da un dato incremento.

# Rilascio degli incrementi



*Ogni componente rilasciato deve fornire  
qualche beneficio agli interessati  
(stakeholders)*

# Esempio di processo incrementale

- Requisiti: R1, R2, R3
  - Architettura: M1, M2, M3, M4
  - Pianificazione: 3 iterations
- 
- Iteration1**
  - R1, requires M1, M2
  - Develop and integrate M1, M2
  - Deliver R1
- 
- Iteration2**
  - R2, requires M1, M3
  - Develop M3, integrate M1, M2, M3
  - Deliver R1 + R2
- 
- Iteration3**
  - R3, requires M3, M4
  - Develop M4, integrate M1, M2, M3, M4
  - Deliver R1 + R2 + R3

# Vantaggi dello sviluppo incrementale

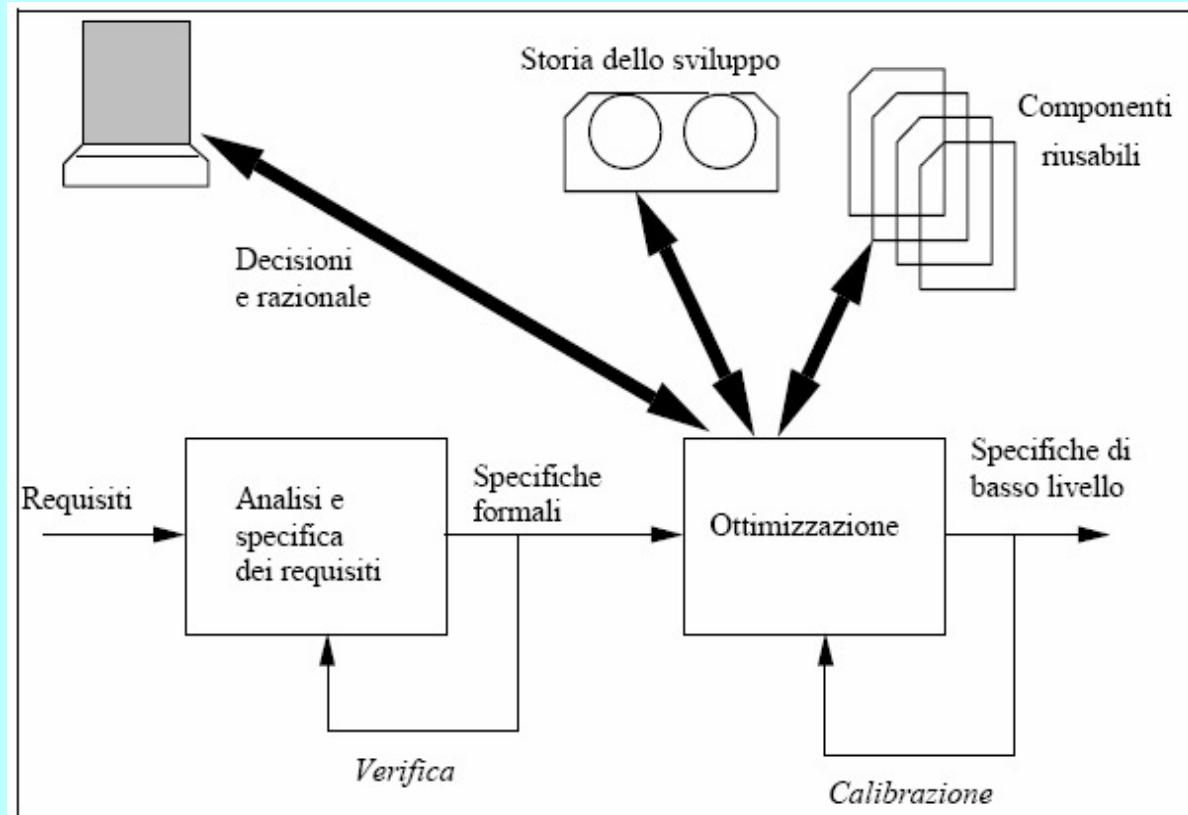
- ◆ I clienti non devono aspettare il sistema completo per la consegna, ma possono disporre al più presto dei requisiti più critici, attraverso i primi incrementi.
- ◆ I primi incrementi possono essere usati come prototipo per aiutare a definire i requisiti degli incrementi successivi.
- ◆ Si riduce il rischio di un fallimento totale del progetto.
- ◆ E' possibile gestire esigenze di cambiamento dei requisiti.
- ◆ I servizi a più alta priorità saranno anche testati più intensamente degli altri.

# Svantaggi dello sviluppo incrementale

- ◆ Lo sviluppo incrementale può essere problematico quando gli utenti hanno bisogno di un sistema funzionante completo che sostituisca un sistema preesistente.
- ◆ Può essere difficile identificare le funzionalità comuni (richieste da tutti i requisiti), giacchè bisogna prima attendere che gli incrementi siano completati per avere ben chiari tutti i requisiti.
- ◆ Quando la specifica completa deve far parte del contratto di sviluppo del sistema, questo modello non è adeguato.

# Modello trasformazionale

# Modello Trasformazionale



**Sviluppo di software come una progressione di passi**

Una descrizione formale viene trasformata in una descrizione meno astratta.

# Modello Trasformazionale

Basato su due concetti

**Prototipazione**  
**Formalizzazione**



Codice eseguibile  
di più basso livello

- ◆ Si specificano i requisiti formalmente
- ◆ Si procede trasformando man mano la descrizione formale in una meno astratta e più dettagliata, fino a che diviene eseguibile da un processore astratto
- ◆ Le specifiche vengono convalidate prima di essere trasformate
- ◆ Le specifiche eseguibili possono essere viste come un prototipo evolutivo

# Modello Trasformazionale

- ❖ Le trasformazioni possono essere eseguite manualmente o supportate da appositi strumenti
- ❖ Il processo di trasformazione può avvantaggiarsi di **componenti riusabili**
- ❖ Il processo si avvale della **storia dello sviluppo**, propriamente immagazzinata, per il supporto a future richieste di cambiamenti
- 
- ❖ Attualmente, è un paradigma praticabile soprattutto per realizzare programmi piccoli e per domini applicativi specifici, ma dovrebbe “scalare” per progetti vasti e complessi
- ❖ è guardato con interesse per l'approccio **formale** allo sviluppo del software

# Modello a spirale

# Obiettivi

- ◆ **Obiettivo:** fornire un quadro di riferimento per la **progettazione dei processi**
- ◆ **Meta-modello**
- ◆ Consentire di scegliere il modello più appropriato in funzione del livello di rischio
- ◆ Il rischio è visto come una circostanza potenzialmente avversa in grado di pregiudicare il processo di sviluppo e la qualità del prodotto
- ◆ **Gestione dei rischi:** “identificare, affrontare ed eliminare i rischi prima che insorgano problemi seri o causa di reimplementazioni costose”

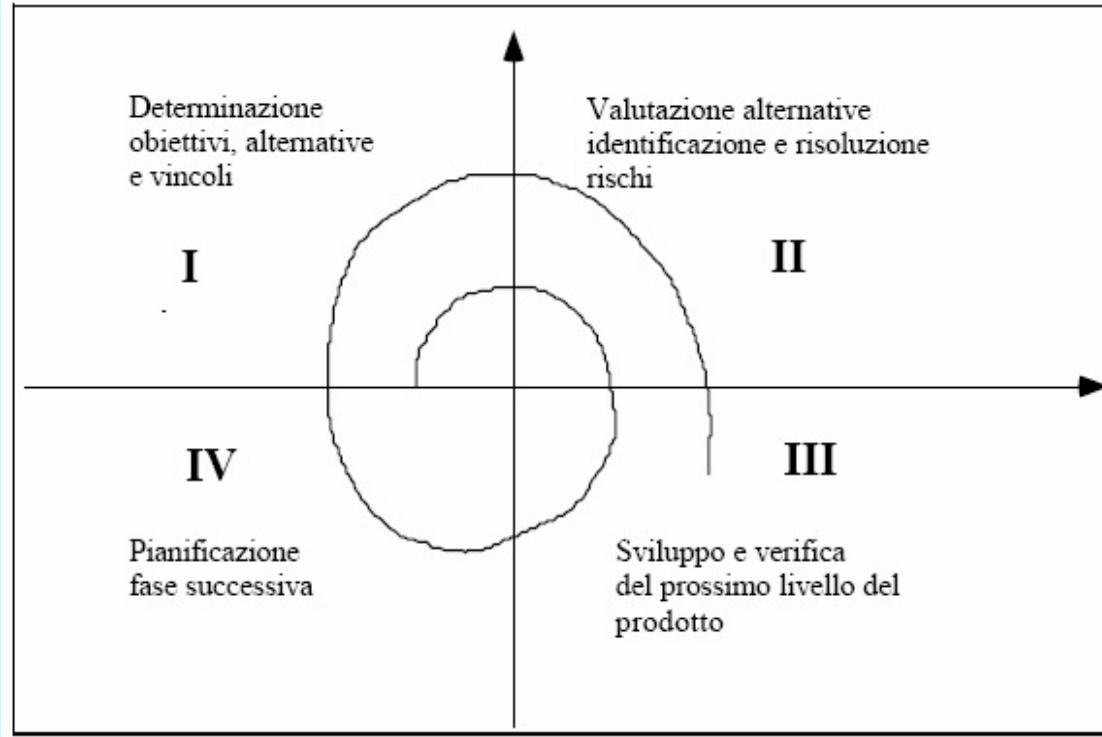


# Caratteristiche

(Boehm, 1998)

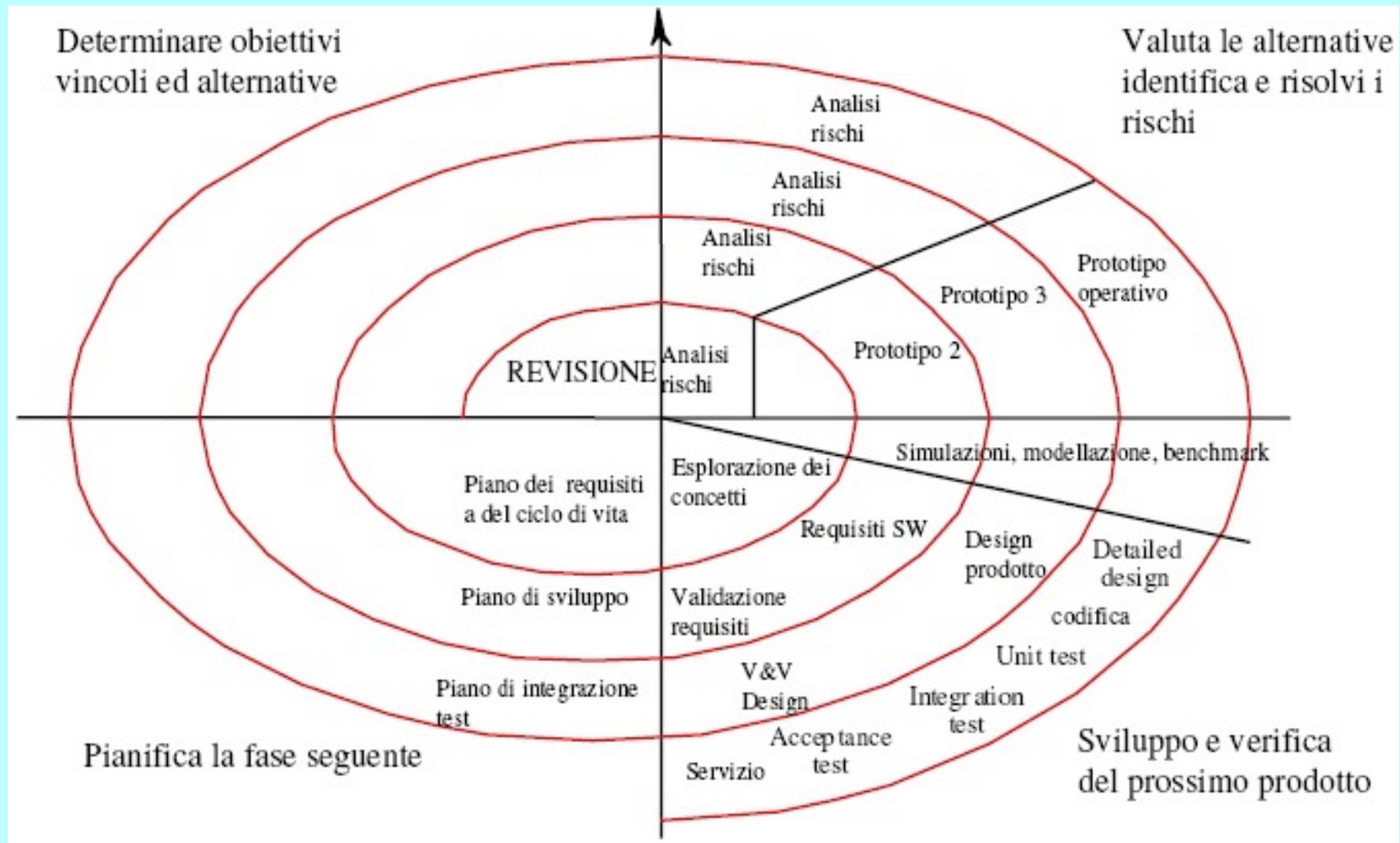
- ◆ Il processo è rappresentato come una spirale, piuttosto che una sequenza di attività con retro-azioni.
- ◆ Ogni giro nella spirale rappresenta una fase del processo.
- ◆ Non prevede fasi prefissate a priori (come la specifica o il design) ma i cicli sono definiti in base al caso specifico.
- ◆ C'è una esplicita gestione dei rischi che vengono valutati e risolti durante tutto il processo.

# Modello a spirale di Boehm



- ◆ **Modello ciclico**
- ◆ **Il raggio del cerchio rappresenta il costo accumulato**
- ◆ **Ogni ciclo è una fase: studio fattibilità, progettazione, etc.**

# Modello a spirale di Boehm



# Settori del Modello a Spirale

- ◆ **Primo settore:** Definizione di obiettivi, vincoli e piano di gestione della fase.
- ◆ **Secondo settore :** Si analizzano i rischi della fase e si scelgono le attività necessarie a gestire i rischi (ad esempio tramite simulazione o prototipazione)
- ◆ **Terzo settore:** Si sceglie un modello di sviluppo per il sistema tra i modelli generici

# Settori

- ♦ Nel terzo settore si può utilizzare:
  - Modello evolutivo, se i requisiti sono incerti
  - A cascata se i requisiti sono chiari e ben definiti
  - Trasformazionale se la sicurezza è un requisito più importante
- ♦ **Quarto settore:** revisione dei risultati e pianificazione della prossima iterazione della spirale

# Unified Process (UP)

- ◆ Tentativo di standardizzazione ed “omogeneizzazione” dei vari processi di sviluppo
- ◆ E’ un framework (schema generale) di processo
- ◆ Approccio iterativo, incrementale
  - Inception
  - Elaboration
  - Construction
  - Transition
- ◆ RUP: Rational UP, definito dalla Rational (oggi IBM)

# Rational Unified Process (RUP)

