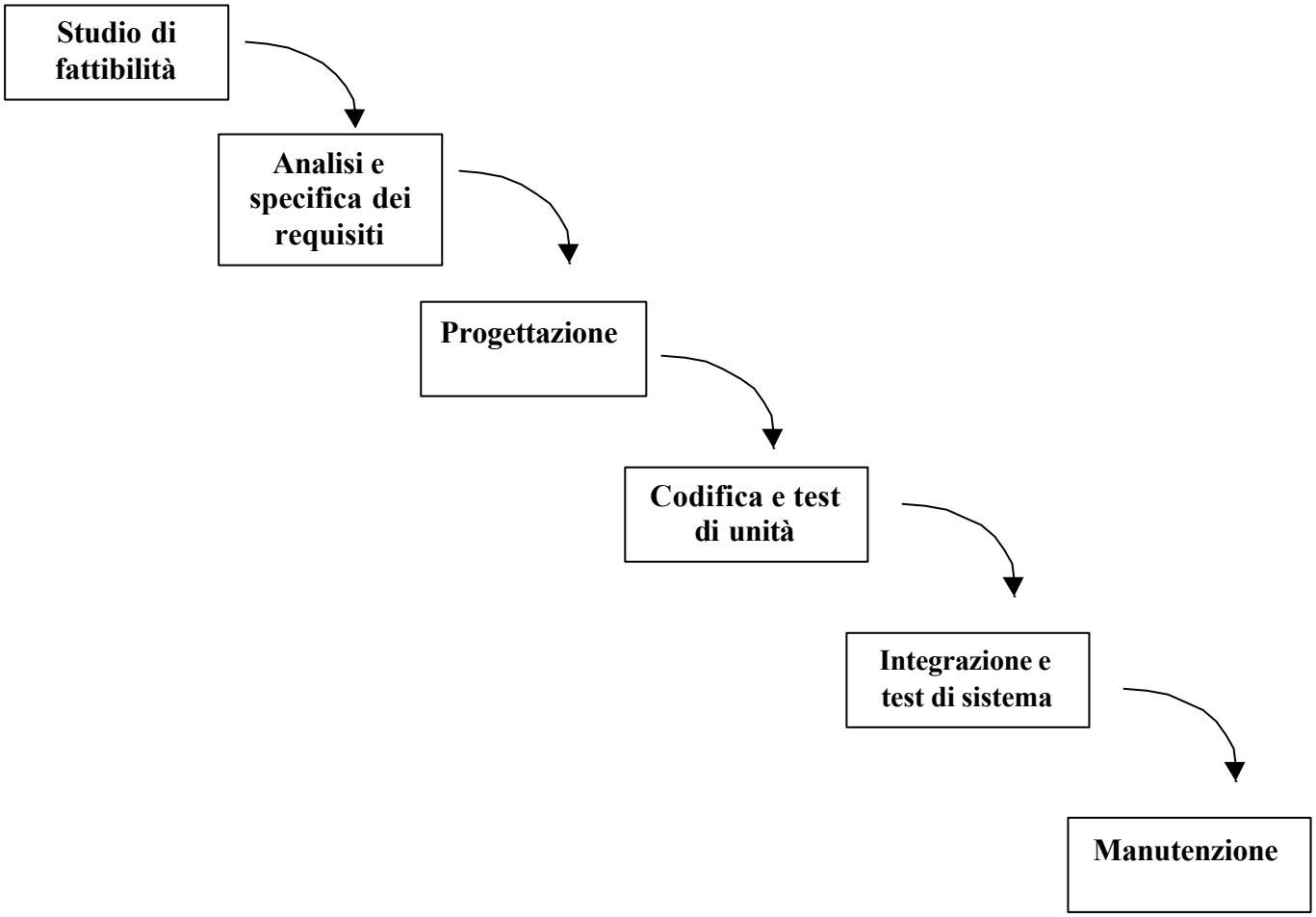
Modelli di processo

Modello a cascata (Royce 1970)



Modello a cascata (cont.)

Fasi:

1) Studio di fattibilità: definizione preliminare del problema, valutazione a priori di costi e benefici; obiettivo: stabilire se lo sviluppo debba essere avviato, evidenziare le risorse disponibili per il progetto, elencare e comparare le alternative
2) Analisi e specifica dei requisiti: analisi completa del problema dell'utente e della sua realtà applicativa al fine di specificare le caratteristiche di qualità dell'applicazione, fra cui i requisiti funzionali (*cosa* il sistema deve fornire, non *come*). I risultati di tale analisi (spesso incompleti/inconsistenti/ambigui) devono essere sottoscritti dal committente
3) Progettazione: definizione dell'architettura sw
4) Codifica e test di unità: programmazione (distinzione sfumata rispetto alla progettazione) + test per verificare il soddisfacimento delle specifiche di progetto

Modello a cascata (cont.)

Fasi:

5) Integrazione e test di sistema: collaudo dell'intero sistema + (opzionalmente) alfa test (rilascio entro l'organizzazione del produttore) e beta test (rilascio a pochi utenti selezionati)
6) Manutenzione

Modello a cascata (cont.)

Definisce con precisione contenuti e struttura dei semilavorati e, in fase di pianificazione, le scadenze entro cui devono essere prodotti e superare i controlli di qualità

Pro:
Le fasi, pur essendo l'astrazione di un flusso continuo, indirizzano l'attività del progettista e consentono il controllo dello svolgimento del progetto

Contro:

- Le fasi non sono formalmente definite, né passibili di svolgimento o controllo automatici
- I ricicli (retroazioni) sono nascosti
- Manutenzione ed evoluzione sono di difficile previsione ed attuazione

Modello a cascata (cont.)

Rischi:

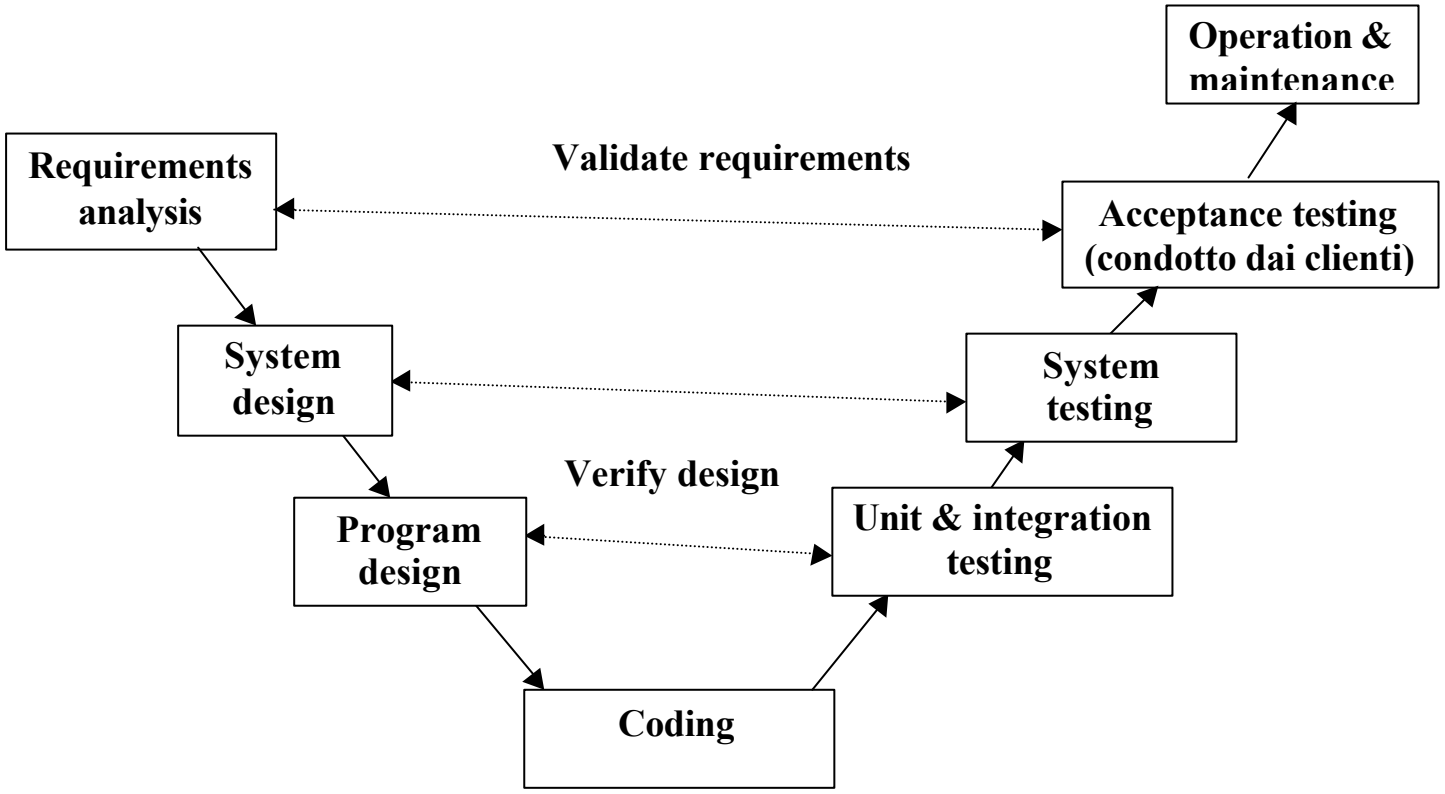
- Individuare scelte ottimali solo per l'applicazione attuale, senza pensare all'evoluzione futura dell'applicazione
- Non pianificare l'attività di manutenzione ed eseguirla solo sul sw, non sugli altri semilavorati

Tutto ciò è miope e si paga a caro prezzo →

Reingegnerizzazione: riportare sw destrutturato e non documentato in uno stato dal quale si possa ripartire per una manutenzione sistematica

Conclusione:
se il maggiore rischio è l'affidabilità dell'applicazione, mentre i requisiti sono estremamente stabili e ben noti, è il modello più ragionevole, con rigorosi controlli per il passaggio da una fase all'altra

Modello a V (Ministero della difesa tedesco, 1992)



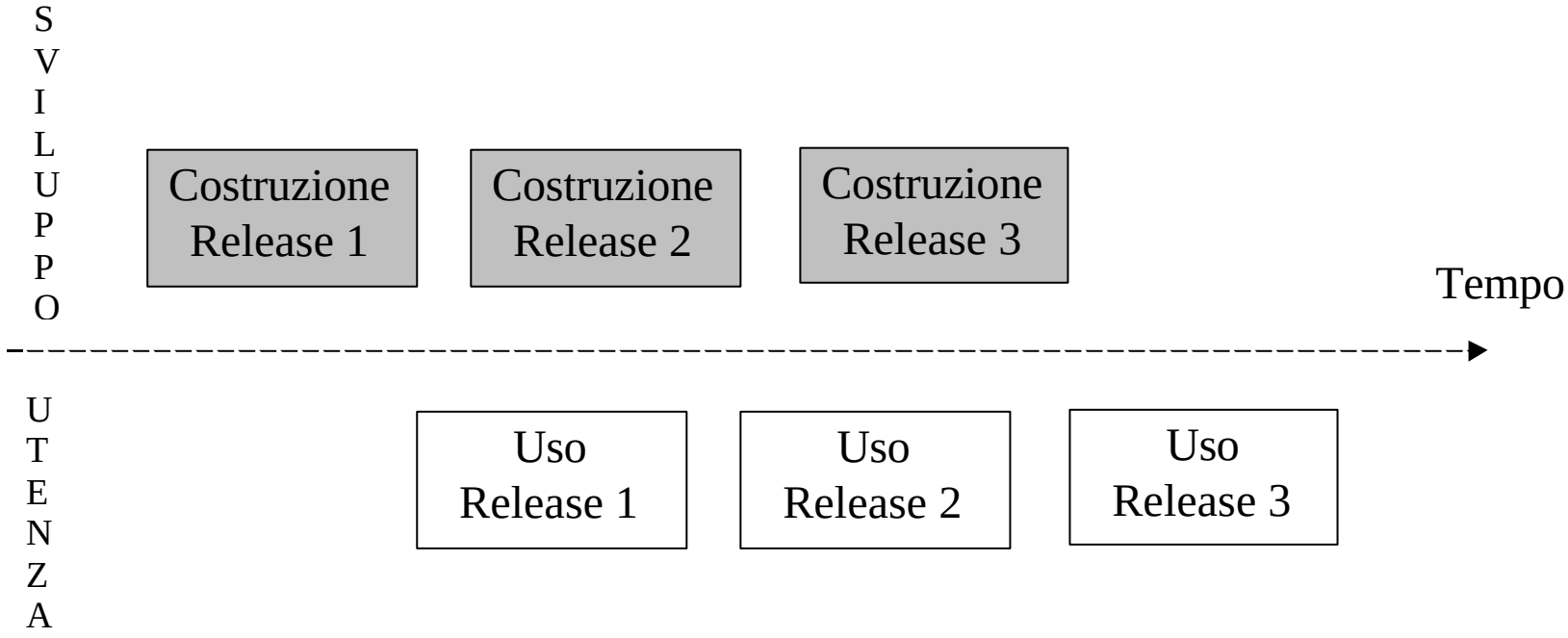
Modello a V (cont.)

- È una variante del modello a cascata che rende esplicita la necessità di effettuare iterazioni
- Mostra come le attività di testing siano collegate ad analisi dei requisiti e progetto
- I test di unità e di integrazione, oltre a occuparsi della correttezza dei programmi, possono essere usati per assicurarsi che tutti gli aspetti del progetto del programma siano stati implementati correttamente
- Il test di accettazione convalida l'aderenza ai requisiti associando un passo del test a ciascun elemento della specifica
- I problemi scoperti sul lato destro della V determinano la riesecuzione di attività sul lato sinistro

Modelli incrementali / iterativi (detti anche evolutivi)

Sono una risposta alla necessaria evoluzione del sw, alternativa interessante soprattutto quando i requisiti sono imperfetti o instabili: il sistema evolve man mano che i requisiti vengono progressivamente compresi

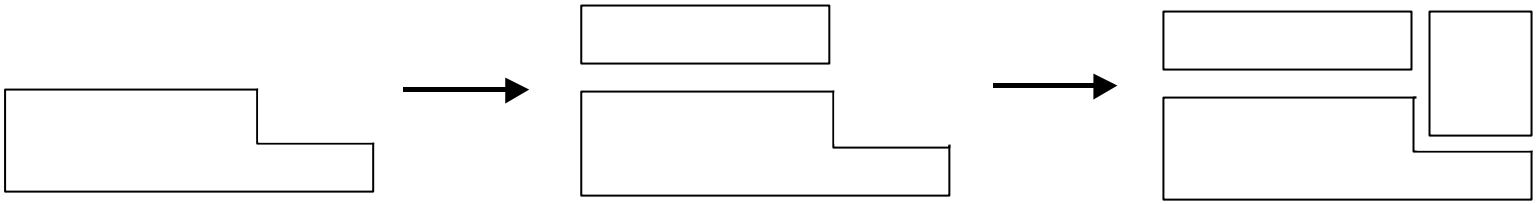
Sviluppo a stadi: mentre è operativa la release *n* del sistema, si lavora alla release *n+1*



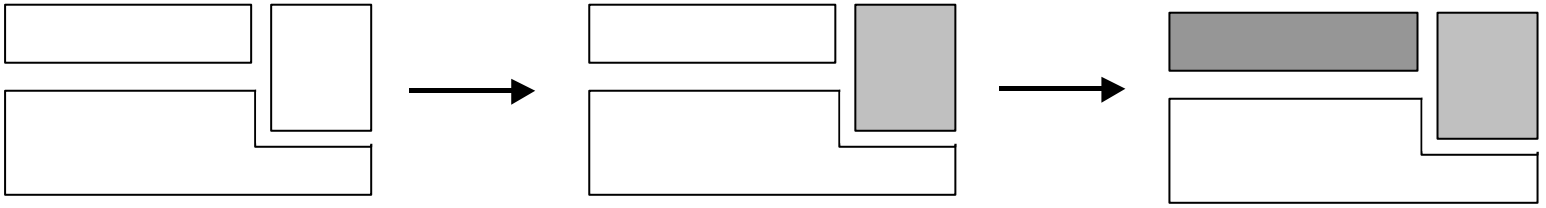
Modelli incrementali / iterativi (cont.)

Lo sviluppo a stadi è supportato da due approcci:

- sviluppo incrementale



- sviluppo iterativo

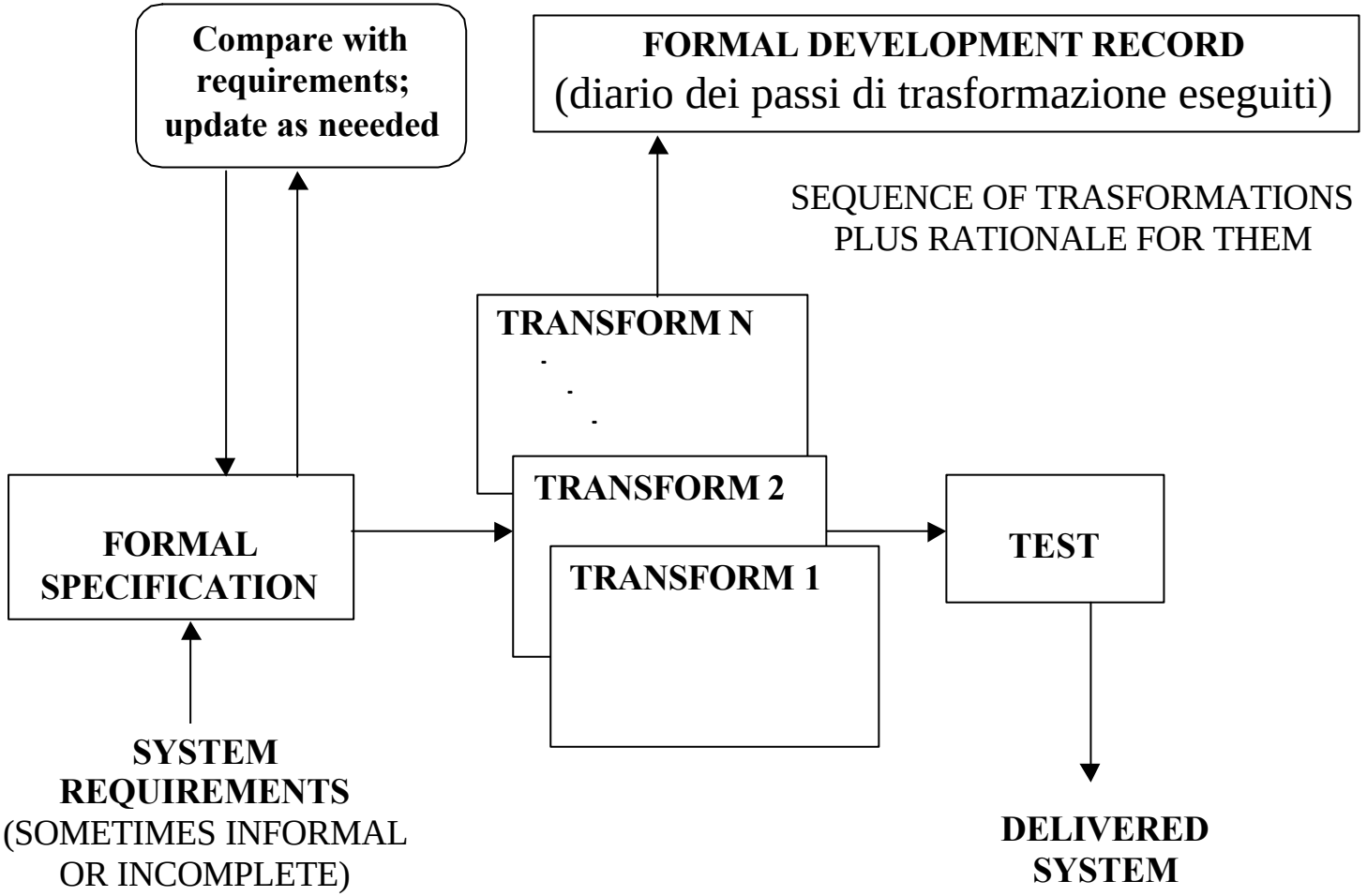


Modelli incrementali / iterativi (cont.)

- Le funzionalità individuate nell'analisi dei requisiti sono allocate a iterazioni diverse; ad ogni release segue la consegna di una nuova versione operativa con qualità e funzionalità aumentate
- Dall'incremento corrente si devono trarre indicazioni su come effettuare il successivo attraverso modifiche semplici e affidabili → necessità di metodi di progettazione opportuni
- Uso della prototipazione
  Prototipo (= modello operativo dell'applicazione da mostrare al committente)
  - ✓Usa e getta (ad es.,
    - per migliorare la comprensione da parte degli sviluppatori
    - per la convalida dei requisiti da parte di utenti/clienti)
  - ✓Evolutivo (= primo incremento)

Conclusione: sono i modelli preferibili quando i rischi maggiori risiedono nell'instabilità e incertezza dei requisiti

Modello trasformazionale (Balzer 1981)



Modello trasformazionale (cont.)

- Basato sull'uso di specifiche formali
- Progressione di passi che trasformano una descrizione formale in una descrizione formale meno astratta
- Verifica delle specifiche formali basata sulla loro eseguibilità ($\rightarrow$ le specifiche rappresentano un prototipo che può interagire con l'utente)
- Creazione di un eventuale catalogo di moduli e/o passi di trasformazione riusabili

Modello a camera bianca (cleanroom) (Mills, Dyer, Linger, 1987)

- Sviluppato da IBM ed evolutosi fino a oggi
- Filosofia: sostituire il testing di unità con ispezioni statiche tese a controllare la consistenza dei componenti rispetto alle specifiche

1. Specifiche formali: il sw da sviluppare viene definito formalmente. Per esprimere le specifiche viene usato un modello stimolo-risposta (stati-transizioni).
2. Sviluppo incrementale: il sw è decomposto in parti che vengono sviluppate separatamente usando trasformazioni ben definite che cercano di preservare la correttezza nel passaggio da una rappresentazione più astratta a una più dettagliata. Tali parti sono individuate, con la collaborazione del cliente, a uno stadio iniziale del processo. Le funzionalità più critiche sono consegnate prima, in modo che il cliente possa fornire feedback che possono determinare revisioni.

Modello a camera bianca (cont.)

3. Programmazione strutturata: vengono usati solo un numero limitato di costrutti di controllo di flusso e di astrazione dei dati. Il processo di sviluppo del programma è basato sul raffinamento progressivo delle specifiche.
4. Verifica statica: il sw sviluppato è verificato staticamente usando prove di correttezza basate sulla matematica (si dimostra che l'uscita della trasformazione è consistente con l'ingresso). I componenti del codice non sono necessariamente eseguiti o testati.
5. Test statistico: ogni nuova release del sw viene testata statisticamente per determinare la sua affidabilità. Questi test statistici sono basati su un profilo operativo che viene sviluppato in parallelo con le specifiche di sistema. Viene costruito, magari da un generatore automatico, un insieme di dati di test che riflette il profilo. Il sistema viene testato con questi dati e viene registrato il numero dei fallimenti nonché i tempi degli stessi. Dopo un numero statisticamente significativo di fallimenti, si calcola l'affidabilità del sistema.

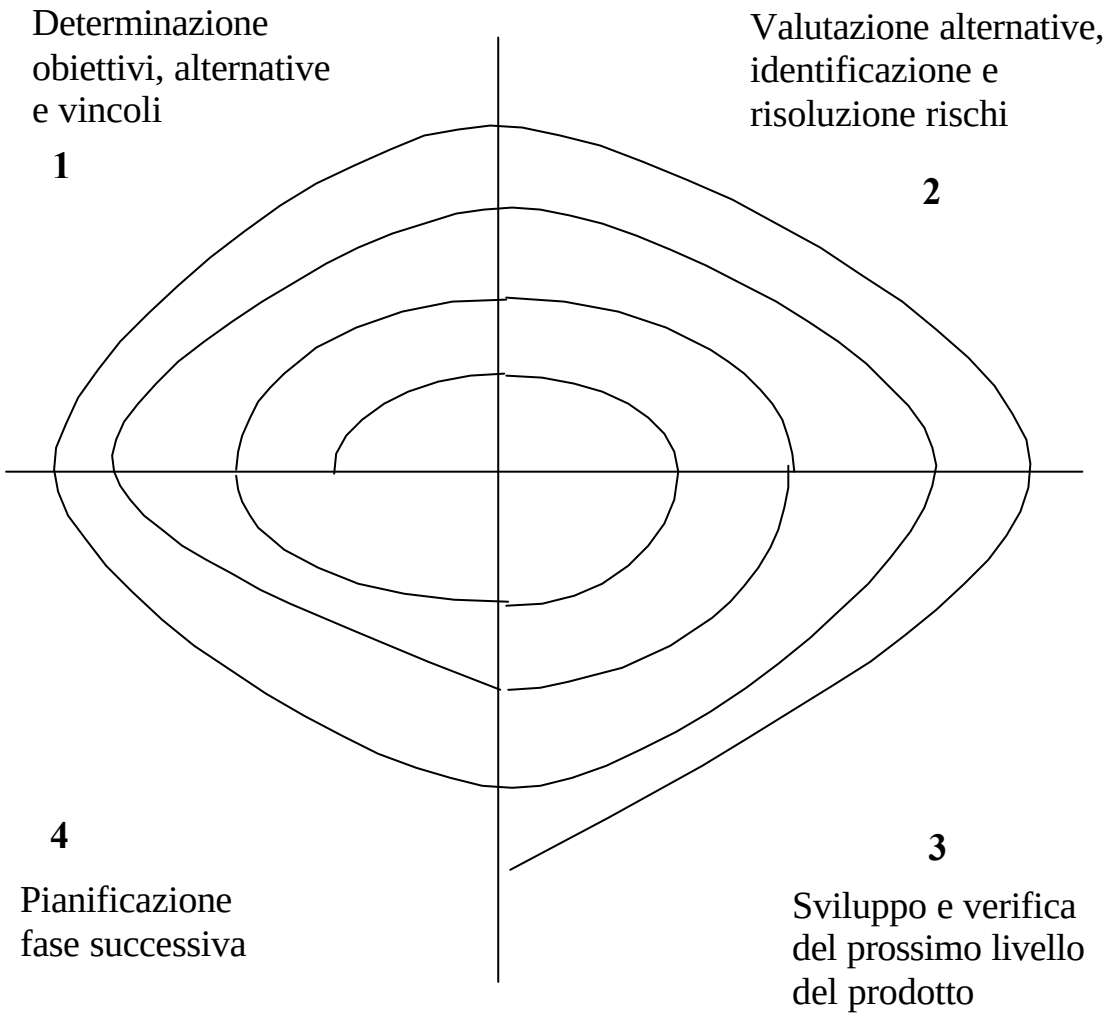
Modello a camera bianca (cont.)

- Profilo operativo: identifica diverse classi di input del sistema e la loro probabilità. Si può ottenere in base a db d'uso di sistemi dello stesso tipo
- Problemi: il profilo è difficile da costruire se il sw è innovativo (non sostituisce un sistema manuale o automatico esistente) e può cambiare durante la vita operativa del sistema

Team coinvolti nel processo:

1. Gruppo specifiche: è responsabile dello sviluppo e del mantenimento delle specifiche del sistema (comprese le specifiche matematiche per la verifica)
2. Gruppo sviluppo: ha la responsabilità di sviluppare e verificare il sw
3. Gruppo certificazione: è responsabile dello sviluppo di un insieme di test statistici finalizzati a esercitare il sw dopo che è stato sviluppato. Lo sviluppo dei casi di test è parallelo allo sviluppo del sw

Modello a spirale (Boehm, 1988)



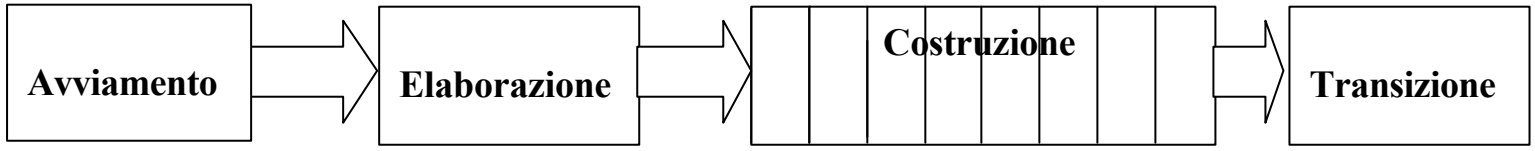
Modello a spirale (cont.)

- È un meta-modello dei processi sw
- Dati gli obiettivi di ciascuna iterazione, se ne valutano in dettaglio i rischi e si decide di conseguenza che alternativa adottare
- Quattro volute i cui prodotti sono (dalla più interna alla più esterna):
  - ✓concept of operation (descrizione di alto livello di come il sistema dovrebbe funzionare)
  - ✓requisiti
  - ✓progetto
  - ✓codifica e testing
- In ogni voluta è previsto l'uso di prototipi per valutare la fattibilità o desiderabilità di un'alternativa
- Raggio della spirale = costo accumulato nel progetto

Unified Software Development Process
(o Rational Unified Process, RUP)
(Jacobson, Booch, Rumbaugh, 1999)

Iterativo e incrementale: l'iterazione è la chiave per uno sfruttamento efficace dell'OO

Visione di alto livello



Avviamento: valutazione commerciale (costi/guadagni) del progetto (lo sponsor del progetto si prende l'impegno a sostenere una fase di analisi approfondita)

RUP: Elaborazione

- Occupa 1/5 della durata del progetto
- Consiste in:
  - ✓esame del dominio del problema
  - ✓analisi dei requisiti (ovvero individuazione del maggior numero di casi d'uso possibile, idealmente tutti)
  - ✓modellazione del dominio guidata dai casi d'uso, mano a mano questi vengono identificati
  - ✓valutazione dei rischi
  - ✓scelte architetturali di alto livello

↓

piano di progetto

Modello di dominio: qualsiasi modello il cui soggetto principale è il mondo supportato dal sistema; descrizione dei termini della questione e delle loro relazioni reciproche
Il modello è sviluppato da un piccolo gruppo che include sia sviluppatori, sia esperti di dominio

RUP: Elaborazione (cont.)

Si dovrebbe realizzare un prototipo di ogni parte dei casi d'uso che presenta dei rischi, al fine di

- capire meglio il funzionamento delle situazioni più dinamiche
- comprendere l'entità del rischio
- stimare il tempo di realizzazione

Fine della fase di elaborazione quando

- Gli sviluppatori ritengono di sapere stimare, con precisione di una settimana-uomo di lavoro, il tempo di realizzazione di ogni caso d'uso
- Tutti i rischi significativi sono stati identificati e compresi fino al punto di sapere come affrontarli

RUP: Elaborazione (cont.)

Rischi	Gestione
Relativi ai requisiti: quali sono i requisiti del sistema? Se questi non sono compresi si costruisce un sistema che non fa quanto richiesto dal cliente	• Usare i casi d'uso (non dettagliati) • Realizzare il modello del dominio insieme agli esperti
Tecnologici: la tecnologia scelta è in grado di svolgere il compito richiesto? Ricordare che i rischi maggiori sono quelli che riguardano l'aggregazione dei vari componenti di un progetto	• Costruire prototipi per mettere alla prova ogni tecnologia che si intende utilizzare • Provare più strumenti e valutare quale è il più adatto • Prendere tutte le decisioni circa l'architettura del sistema • Aggregare tutti i componenti che si intendono usare

RUP: Elaborazione (cont.)

Rischi	Gestione
Legati alle risorse umane: si può contare sulla squadra e sull'esperienza necessarie?	• Dedicare tempo alla formazione • Organizzare la formazione in piccoli blocchi, nel momento in cui è necessaria, e metterla subito in pratica • Fare partecipare al progetto (o a ciascuna area specifica dello stesso) un mentore (sviluppatore esperto), oppure • Fare revisionare il progetto ogni due mesi ca. • Organizzare gruppi di lettura
Politico/aziendali: il progetto verrà sostenuto o intralciato?	

RUP: Pianificazione della fasi di costruzione
(secondo le tecniche della programmazione estrema)

Passo 1: i clienti (cioè le persone che possono stimare il valore commerciale di ogni caso d'uso) suddividono i casi d'uso in tre categorie, a seconda della loro priorità (ovvero, del valore commerciale)

Passo 2: gli sviluppatori suddividono i casi d'uso a seconda del rischio di sviluppo ("alto rischio" = difficile da implementare / non ben compreso)

Passo 3: gli sviluppatori stimano il tempo necessario allo sviluppo di ogni caso d'uso (analisi, progetto, implementazione, testing, integrazione e documentazione) con la precisione di una settimana-uomo

Controllo: se gran parte del tempo corrisponde ai casi d'uso a rischio massimo, ritornare alla fase di elaborazione

RUP: Pianificazione della fasi di costruzione (cont.)

Passo 4: determinare la lunghezza di ogni iterazione (che rimarrà fissa per l'intero progetto) = tempo di sviluppo di un piccolo # di casi d'uso (dipende dal linguaggio usato)

Passo 5: calcolo della quantità di sviluppo che è possibile compiere in una iterazione, tenendo conto del # degli sviluppatori e il carico esterno degli stessi

Passo 6: assegnazione dei casi d'uso alle iterazioni (prima quelli ad alta priorità e/o a maggior rischio)

Passo 7: stesura del piano di consegna, le cui scadenze vanno rispettate (eventualmente si possono posporre dinamicamente dei casi d'uso)

RUP: Costruzione

Costruzione iterativa (riscrittura di una parte del codice esistente) e incrementale (a ogni iterazione si aggiungono nuove funzionalità) del prodotto comprensiva di analisi, progettazione, implementazione e test (sw rilasciato pezzo per pezzo, dove ogni pezzo è di qualità industriale). Demo per il cliente alla fine di ogni iterazione

Scopo del processo: ridurre il rischio, che spesso deriva dal rimandare i problemi più difficili (quali le fasi di testing e integrazione) alla fine del progetto

RUP

Regole empiriche:

- se per 2 o 3 iterazioni consecutive si pospongono una gran quantità di casi d’uso, ripianificare
- se a ogni iterazione si scarta meno del 10% del codice esistente, insospettirsi
- per evitare appesantimenti del testing, sviluppare sw che si autoverifichi
- per evitare problemi di integrazione aggiuntivi, eseguire una ricompilazione completa del sistema ogni giorno
- usare tecniche di Refactoring

REFACTORING

Insieme di tecniche che riducono lo sforzo della riprogettazione (che nel RUP è richiesta a ogni iterazione), ovvero del cambiamento della struttura interna di un programma (senza alternarne la funzionalità) per renderlo più facile da comprendere e modificare → modifiche di refactoring, che sono piccole (ad es. cambiare il nome di un metodo, spostare un campo da una classe a un'altra, unificare due metodi simili in una superclasse)

Principi di refactoring

- Non effettuare un refactoring mentre si aggiungono funzionalità al programma
- Avere pronti dei buoni test prima di cominciare il refactoring
- Compiere piccole modifiche, verificando il funzionamento dopo ogni passo (così da risparmiare il debugging)

RUP: Transizione

Occupa un tempo pari a 10-35% del tempo di costruzione; consiste in: consegna, correzione errori (eventualmente anche mediante beta-testing), ottimizzazioni (che riducono la chiarezza e l'estensibilità del sistema per aumentarne la velocità di esecuzione) e possibile formazione utenti

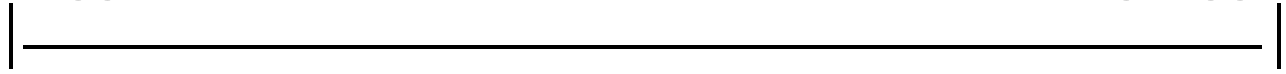
RUP: Caratteristiche del processo

- iterativo
- centrato sulla architettura
- supportato da tecniche orientate agli oggetti (OO)
- configurabile
- dotato di controllo qualità
- attrezzato per la gestione del rischio
- basato su modelli (specificati in UML)
- guidato dai casi d'uso

UML

- Linguaggio di modellazione grafico usato per esprimere le caratteristiche di un progetto, indipendente dalla metodologia di progettazione adottata
- È uno standard OMG

Spettro dei linguaggi di specifica



impreciso — preciso ma troppo dettagliato

Necessità di disporre di un certo grado di precisione, evidenziando solamente i dettagli importanti → UML

UML: Motivazioni

Si usa la notazione semiformale UML perché:

- favorisce la comunicazione fra i membri del gruppo di sviluppo
- favorisce la comunicazione con i clienti (in particolare, grazie ai casi d'uso)
- è di grande aiuto nella fase di elicitazione e analisi dei requisiti (collocata entro la fase di Elaborazione del RUP)
- aiuta a sfruttare i vantaggi dell'OO (i linguaggi OO permettono tali vantaggi ma non li forniscono)

RUP e UML

Fase	Modelli impiegati
Avviamento	
Elaborazione	Contestualmente: a) Identificazione dei casi d'uso e valutazione del rischio di ciascuno b) Creazione, guidata dai casi d'uso, del modello di dominio (come mezzo per interagire con gli esperti al fine di elicitarne la conoscenza): • Diagramma delle classi (dal punto di vista concettuale; serve per definire un vocabolario rigoroso) • Diagrammi delle attività (se nel dominio vi è una forte componente di workflow) ↓ Aggregazione dei diagrammi delle classi e delle attività + Diagrammi di stato (eventualmente)

RUP e UML (cont.)

Fase	Modelli impiegati
Elaborazione	N.B. Il modello di dominio è uno scheletro (ovvero sta alla base ed è il fondamento del resto del modello, è dettagliato ma è solo una piccola parte del tutto, ovvero illustra solo i dettagli importanti), non un modello di altro livello, in cui si illustra tutto, omettendo i dettagli Al fine di gestire i rischi tecnologici, eventuali brevi schizzi di: • diagrammi delle classi e diagrammi delle interazioni, per mostrare come comunicano i componenti • diagrammi dei package, per mostrare un'immagine ad alto livello dei componenti • diagrammi di deployment, per mostrare la distribuzione dei pezzi

RUP e UML (cont.)

Fase	Modelli impiegati
Costruzione	Si possono impiegare • schede CRC • diagrammi delle interazioni e riportare responsabilità e operazioni da essi evidenziate sul • diagramma delle classi (a livello di specifica) usando quest'ultimo come strumento per discutere dei vari approcci alla progettazione Costruito il sw, usare UML come ausilio per la documentazione dei concetti più importanti (la documentazione dettagliata dovrebbe essere generata automaticamente a partire dal codice): • diagrammi dei package, per comprendere come il sistema si suddivide in parti e come queste interagiscono • diagramma delle classi (a livello implementativo) di ciascun package, solo per elencare tali classi e i loro attributi chiave

RUP e UML (cont.)

Fase	Modelli impiegati
Costruzione	• diagramma di deployment, per mostrare la struttura fisica del sistema ad alto livello • diagramma di stato delle (sole) classi che hanno un comportamento complesso (alcuni ritengono che lo siano quelle dedicate all'interfaccia utente e al controllo del sistema) • diagramma delle interazioni, se complesse • diagramma delle attività dei (soli) algoritmi particolarmente complicati e solo se questo migliora la comprensione del codice (in questo caso il diagramma delle attività non è altro che un flowchart) • diagramma delle attività per documentare applicazioni multithread • pattern, se gli stessi concetti emergono ripetutamente
Transizione	