Processo di produzione del software

a cura di **Angelo Furfaro**
da "Ingegneria del Software, Fondamenti e Princìpi"
Ghezzi, Jazayeri, Mandrioli

Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: a.furfaro@unical.it
Web: http://siloe.deis.unical.it/angelo

Processo di produzione del software

- Cerca di organizzare il ciclo di vita del software:
  - definendo le attività coinvolte nella produzione del software
  - stabilendo un ordine nell'esecuzione delle attività ed individuando le relazioni tra di esse
- Obiettivi del processo di sviluppo del software:
  - standardizzazione
  - predicibilità
  - produttività
  - elevata qualità del prodotto
  - capacità di pianificare il tempo ed i costi di produzione

I modelli di processo di produzione del software

Code and Fix

- Agli albori dell'informatica, la produzione del software era un lavoro individuale: non c'era distinzione tra programmatore ed utente finale
- Lo sviluppo software consisteva nella scrittura di codice a basso livello
- Il modello inizialmente adottato è il cosiddetto *code and fix* che consiste nell'iterazione di due passi:
  1. Scrittura del codice
  2. Correzione degli errori, miglioramento delle funzionalità, introduzione di nuove caratteristiche
- Dopo una serie di cambiamenti la struttura del codice diveniva disorganizzata rendendo le modifiche sempre più difficoltose
- Man mano che le applicazioni cominciarono ad essere sviluppate per contesti molto diversi, aumentò il divario tra lo sviluppatore ed il fruitore del software
- La complessità delle applicazioni divenne sempre maggiore
- Cominciarono ad essere sviluppati anche "pacchetti" software per un mercato generico

Crisi del software

- Il modello *code and fix* si dimostrò ben presto inadeguato:
  - non era in grado di gestire la crescente complessità delle applicazioni
  - la gestione dell'avvicendamento del personale su un progetto era estremamente difficile (assenza di documentazione)
  - modificare il codice era arduo (nessuna anticipazione del cambiamento)
  - il software realizzato spesso non soddisfaceva le aspettative degli utenti ed occorreva svilupparlo ex-novo per ottenere gli obiettivi desiderati
  - lo sviluppo diventava un processo infinito, non predicibile e non controllabile
- la ragione principale del *code and fix* risiede principalmente nell'errato sfruttamento della duttilità del software
- Il fallimento del modello *code and fix* portò, alla fine degli anni 80, al riconoscimento della *crisi del software*
- Dalla crisi ebbe origine l'ingegneria del software

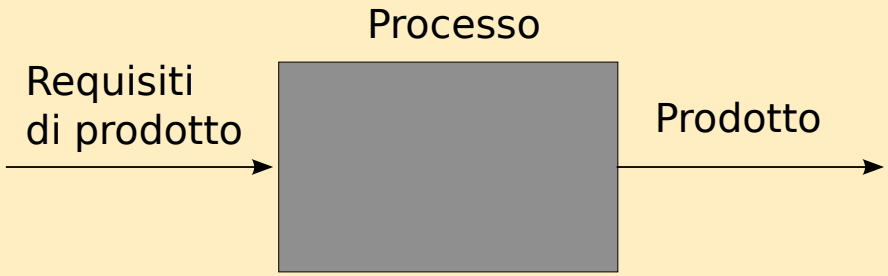
Obiettivi del modello di processo

Boehm, 1988

- determinare **l'ordine degli stadi** coinvolti nello sviluppo e nell'evoluzione del software
- stabilire **criteri di transizione** per progredire da uno stadio al successivo
- Questi includono criteri:
  - di completamento per lo stadio corrente
  - per la scelta e l'ingresso nello stadio successivo
- Un modello si occupa delle seguenti domande:
  - Che cosa faremo dopo?
  - Per quanto continueremo a farlo?

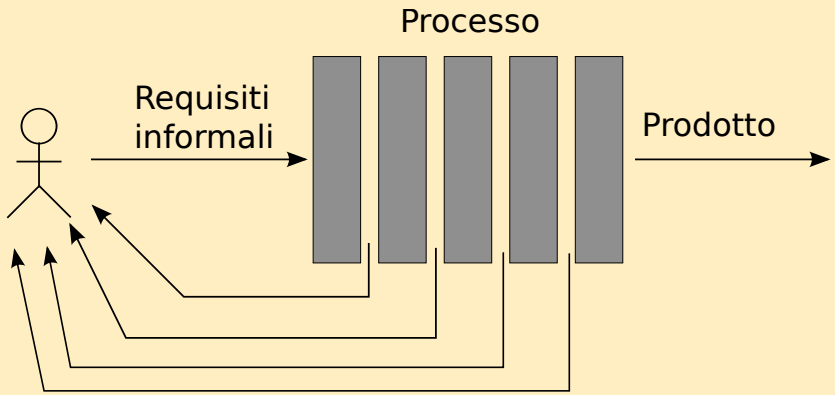
Importanza dei modelli di processo di sviluppo

- i processi di produzione influenzano la qualità dei prodotti
- controllando i processi si possono controllare meglio i livelli raggiunti nelle qualità richieste
- se si segue un processo esplicito lo sviluppo del software procede in modo ordinato e prevedibile riducendo la possibilità di commettere errori
- se non si segue un preciso modello di processo lo sviluppo può essere considerato una *scatola nera* (*black-box*):
  - gli ingressi sono i requisiti del prodotto
  - l'uscita è il prodotto ottenuto
- il prodotto viene completato dopo molto tempo (mesi, anni) e dopo molti investimenti finanziari
- quando il prodotto è terminato è spesso troppo tardi (e troppo costoso) occuparsi della sua qualità



Importanza dei modelli di processo di sviluppo

- All'inizio del processo di sviluppo i requisiti non sono espressi in modo preciso (spesso l'utente non sa precisamente che cosa vuole)
- Il progettista può, in generale, avere scarsa conoscenza del dominio applicativo
- Se il processo di sviluppo è strutturato come una scatola nera il suo progresso è invisibile
- È pericoloso assumere che i requisiti iniziali siano precisi
- È necessario "aprire" la scatola nera per convalidare gradualmente ciò che si sta sviluppando rispetto alle aspettative del cliente
- Un tale tipo di processo può fornire *feedback* agli sviluppatori, riducendo tempi e costi di produzione



Attività principali dei processi di sviluppo

- Sono le attività che devono essere svolte indipendentemente dal modello adottato
- Il modello influenza solo il flusso di informazioni tra tali attività

- Studio di fattibilità
- Acquisizione, analisi e specifica dei requisiti
- Definizione e progettazione dell'architettura software
- Produzione di codice e test dei moduli
- Integrazione e test del sistema
- Rilascio, installazione e manutenzione

Studio di fattibilità

- Viene eseguita prima di cominciare il processo di produzione per decidere se procedere con lo sviluppo
- Il suo risultato è un documento che illustra diversi scenari e soluzioni alternative
- Il progettista deve analizzare il problema a livello globale
- Lo studio di fattibilità consiste in una simulazione del processo di sviluppo sotto vari scenari

Documento risultante

1. Definizione del problema
2. Soluzioni alternative con relativi benefici attesi
3. Risorse richieste, costi e date di consegna di ogni alternativa proposta

Acquisizione, analisi e specifica dei requisiti

Scopo

- capire gli obiettivi del sistema
- documentare o requisiti da soddisfare
- specificare le qualità richieste in termini di:
  - funzionalità
  - prestazioni
  - facilità d'uso
  - portabilità
  - etc.

What vs. How

Questa attività si occupa di descrivere

- le caratteristiche e le qualità che devono essere assicurate dall'applicazione
- **non** come devono essere progettate e implementate

Acquisizione, analisi e specifica dei requisiti

Che cosa è necessario

- Comprendere l'interfaccia tra l'applicazione ed il mondo esterno
- Comprendere il dominio applicativo
- Individuare i principali *stakeholder* (coloro che hanno un interesse nel sistema e che saranno responsabili della sua accettazione)
- Capire cosa essi si aspettano:
  - Stakeholder differenti hanno punti di vista differenti
  - L'ingegnere del software deve cercare di integrarli e riconciliarli

Princìpi cruciali

- Separazione degli interessi
- Astrazione
- Modularizzazione *orizzontale*: strutturare il sistema come collezione di viste allo stesso livello di astrazione

Modularizzazione orizzontale

Separazione dei requisiti funzionali in tre viste

1. modello dei dati su cui opera l'applicazione (Diagrammi ER, Class diagram)
2. modello delle funzioni eseguibili (Diagrammi di flusso, Use Case diagram)
3. modello di come le strutture di controllo disciplinano l'esecuzione di tali funzioni (Reti di Petri, State diagram)

Documento di specifica

- Fornisce una specifica dell'interfaccia tra l'applicazione ed il mondo esterno
- Definisce le qualità da raggiungere
- Possiede delle qualità proprie:
  - comprensibilità
  - precisione
  - completezza
  - consistenza
  - non ambiguità
  - facilità di modifica
- Deve essere analizzato ed approvato dagli stakeholder
- Potrebbe contenere una versione preliminare del manuale dell'utente
- Può essere corredato da un piano di test a cui sottoporre il sistema una volta realizzato

Documento di specifica

Contenuto

1. **Dominio**
   Breve descrizione del dominio applicativo e degli obiettivi da raggiungere.
   - Chi sono gli utenti, quali sono le loro aspettative ed i loro obiettivi?
   - Quali sono le principali entità che caratterizzano il dominio
   - Quali sono le loro relazioni? Che influssi avrà il sistema su di esse?
2. **Requisiti funzionali**
   - Descrivono cosa dovrà fare il prodotto.
   - Devono essere espressi in una notazione opportuna (formale o semiformale)
3. **Requisiti non funzionali**
   - Affidabilità, accuratezza, prestazioni
   - Interfaccia tra sistema e utente
   - Limiti operativi, limiti fisici, portabilità
4. **Requisiti del processo di sviluppo e manutenzione**
   - Procedure per il controllo della qualità
   - Priorità di sviluppo tra le funzioni
   - Possibili cambiamenti del sistema

Classificazione requisiti MUST, SHOULD, MAY

I requisiti sono raggruppabili, in rapporto alla loro priorità/criticità, tra:

- **MUST**: insieme minimo di requisiti per considerare "accettabile" il comportamento del sistema da realizzare
- **SHOULD**: requisiti desiderabili in quanto rendono più completo il sistema, ma la cui omissione non compromette nessuna funzionalità di base. La loro implementazione/integrazione non dovrebbe richiedere modifiche profonde alla struttura del progetto
- **MAY**: sono requisiti la cui presenza nel progetto è facoltativa: in caso di mancanza di risorse (tempo) possono essere tralasciati inizialmente. Tuttavia, la loro integrazione successiva potrebbe comportare modifiche significative al sistema, come il riprogetto o l'aggiunta di nuove classi

Definizione dell'architettura software

- Il risultato di questa attività è la produzione di un *documento di specifica di progetto*
- La forma del documento segue gli standard della compagnia che possono includere l'uso di una notazione standard quale UML

Documento di specifica di progetto

- Descrive il sistema in termini di componenti, delle loro interfacce e delle relazioni tra di esse
- Registra le decisioni significative e le relative motivazioni

Produzione di codice e test dei moduli

- Il risultato di questa attività è la produzione di moduli implementati e testati
- La produzione di codice può essere soggetta a standard aziendali:
  - linee guida da seguire nella strutturazione del codice
  - convenzioni sui commenti, sui nomi di funzioni e variabili, etc.
- Anche il test dei moduli è soggetto a standard aziendali:
  - definizione dei piani di test
  - criteri di test da eseguire (white-box, black-box)
  - definizione di criteri di completamento
  - gestione dei casi di test

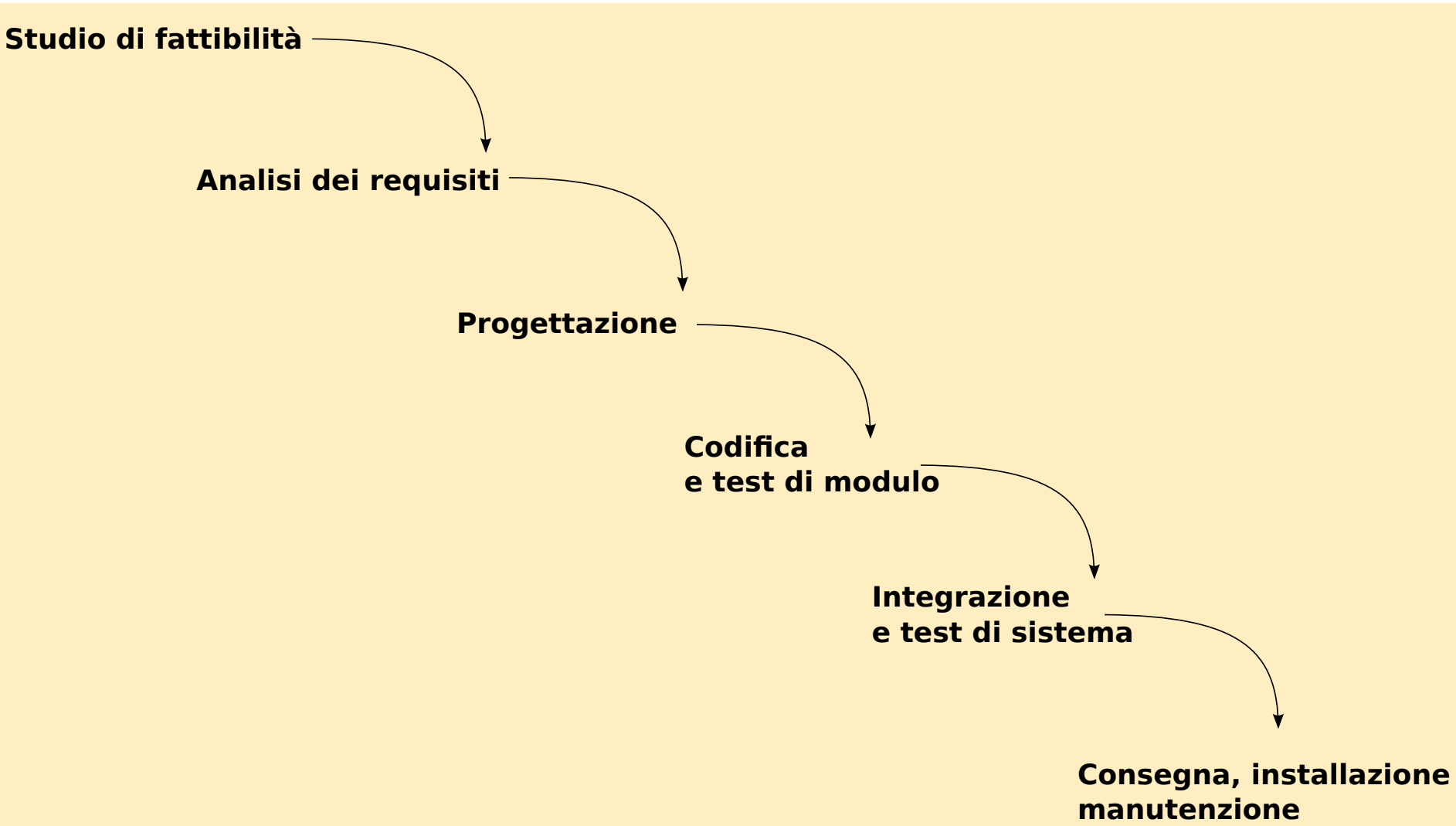
Integrazione e test del sistema

- Consiste nell'assemblare l'applicazione a partire dai singoli componenti sviluppati e testati separatamente
- Alcuni modelli di processo (incrementali) includono quest'attività nella produzione del codice: i componenti sono testati ed integrati man mano che vengono realizzati
- Allo stadio finale si procede al test dell'intera applicazione (*Alpha test*)
- Possono essere adottati standard interni per stabilire
  - come debba essere fatta l'integrazione (top-down, bottom-up)
  - come debbano essere progettati i dati di test e la relativa documentazione

Rilascio, installazione e manutenzione

- Il software viene consegnato ai clienti in due fasi:
  - l'applicazione viene distribuita ad un gruppo selezionato di clienti (*Beta test*) motivati a rilevare la presenza di eventuali errori al fine di correggerli per ottenere un prodotto affidabile
  - rilascio ufficiale
  - L'installazione definisce l'architettura del sistema a tempo di esecuzione ad esempio stabilendo l'allocazione dei vari componenti su i nodi di una rete aziendale
  - La manutenzione, come già visto, è la fase più costosa (60% del totale)
    - il 20% dei costi di manutenzione sono dovuti a manutenzione correttiva ed adattiva
    - oltre il 50% dei costi è imputabile alla manutenzione perfettiva

Modello a cascata



Modelli a cascata

- Furoni inventato alla fine degli anni 50 per sistemi di difesa area di grandi dimensioni
- Tali modelli furono resi popolari negli anni 70
- Essi organizzano le attività in un flusso sequenziale
- Ne esistono diverse varianti che condividono la stessa filosofia:
  - sono sequenziali
  - sono basati su una progressione tra fasi successive
  - sono guidati dalla produzione di documentazione
- Le organizzazioni che usano un modello a cascata:
  - definiscono standard su come devono essere prodotti gli output (i *deliverable*)
  - prescrivono metodi da seguire nella produzione dei deliverable

Critica sui modelli a cascata

Vantaggi

- Il processo di sviluppo è soggetto a disciplina, pianificazione e gestione
- l'implementazione del prodotto dovrebbe essere rimandata fino a quando non sono chiari gli obiettivi

Svantaggi

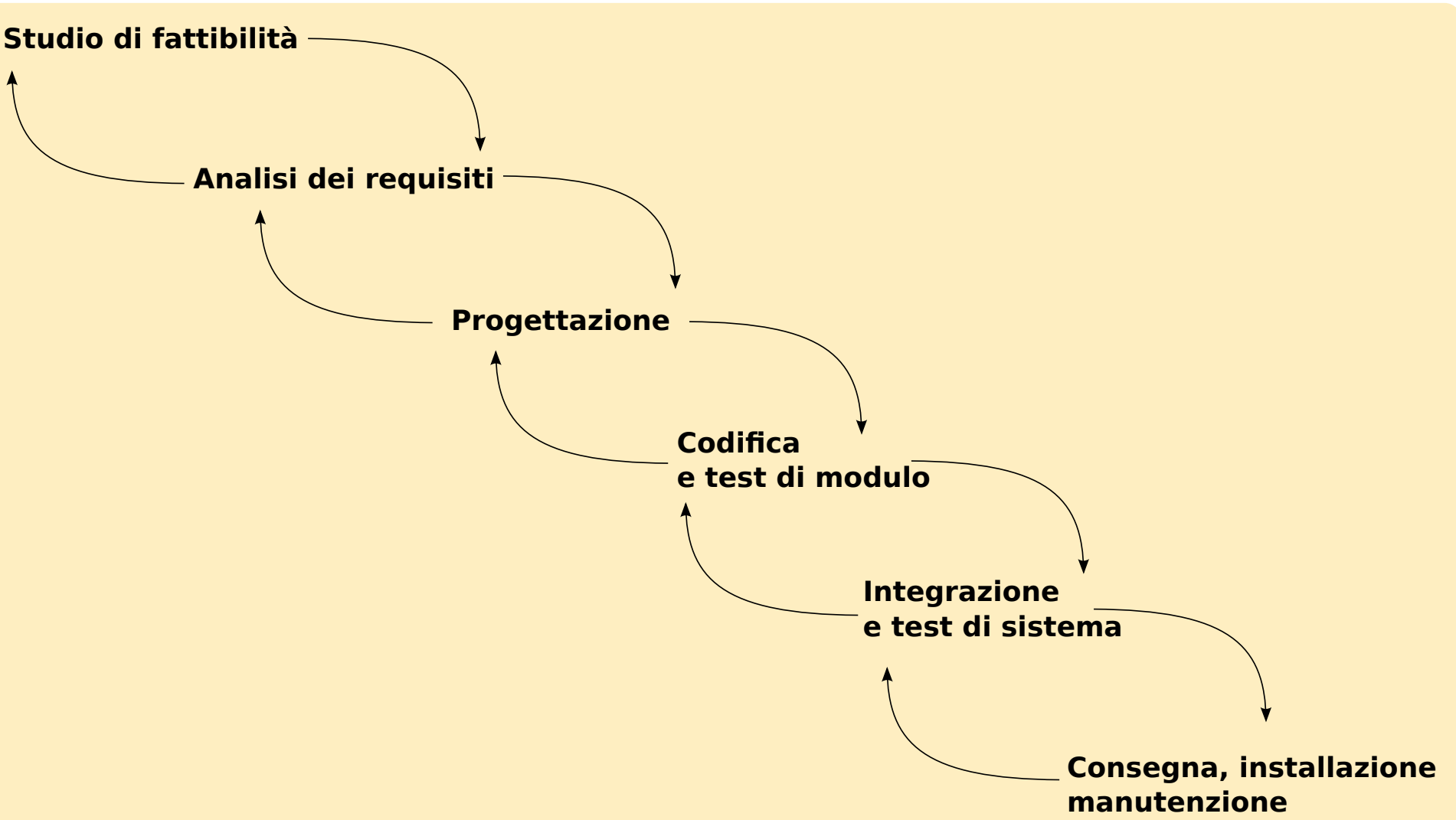
Lineare, rigido, monolitico

- assenza di feedback
- assenza di parallelismo
- unica data di consegna

Introduzione di feedback

- Alcuni modelli a cascata consentono di introdurre del feedback in modo disciplinato
- Il feedback è limitato tra una fase e quella successiva

Modello a cascata con feedback



Problemi del modello a cascata

- Le stime sono effettuate sulla base di informazioni limitate
- È difficile identificare tutti i requisiti assieme in una sola volta:
  - Gli utenti possono non sapere esattamente ciò che vogliono
  - I requisiti non possono essere congelati
- Il modello a cascata non evidenzia sufficientemente la necessità di anticipare possibili cambiamenti
- Obbliga a standard pesantemente basati sulla produzione di una data documentazione in determinati momenti

Modelli evolutivi

- Il progettista dovrebbe accettare il fatto che i fallimenti di una versione dell'applicazione comportino il bisogno di rifare parte dell'applicazione
- Secondo il principio "Do it twice" (Brooks, 1995), la prima versione di un prodotto viene vista come un test
- La prima versione è un *prototipo cestinabile* (throw-away)
- Tale idea è alla base dei modelli *evolutivi* o *incrementali*

Boehm, 1988

Un modello di processo evolutivo è "un modello le cui fasi consistono in versioni incrementali di un prodotto software operazionale con una direzione evolutiva determinata dall'esperienza pratica".

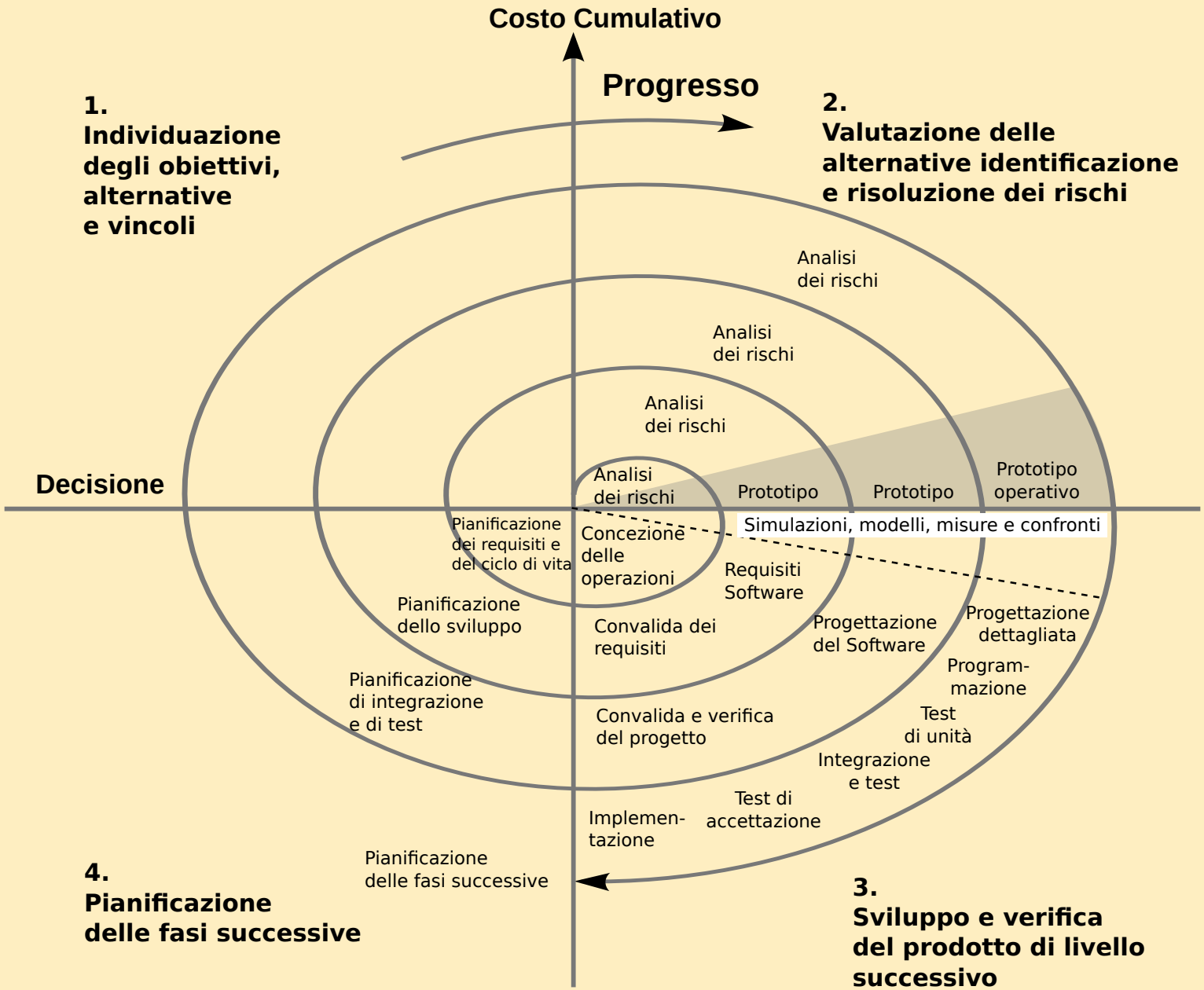
Rilasci incrementali

- Le versioni incrementali possono essere rilasciate al cliente man mano che sono sviluppate
- Si definisce *versione incrementale rilasciabile*:
  - un'unità software funzionale autocontenuta che esegue qualche funzione utile al cliente
  - viene rilasciata corredata da adeguati materiali di supporto (specifiche di requisiti, piani di test, manuale utente, etc.)

La strategia evolutiva può essere riassunta come segue

1. **Rilascio** di una funzionalità all'utente
2. **Misura** del valore aggiunto per il cliente
3. **Aggiustamento** sia del progetto che degli obiettivi

Modello a spirale



Modello a spirale

Struttura

- È un modello ciclico
- Ogni ciclo consiste di quattro fasi ciascuna corrispondente ad un quadrante del diagramma cartesiano
- Il raggio della spirale corrisponde al costo accumulato
- La dimensione angolare rappresenta il progresso nel progetto

Fasi

1. Identifica gli obiettivi della porzione di prodotto che si sta considerando in termini di qualità da ottenere. Identifica le alternative e i vincoli imposti dalla scelta di una di esse.
2. Si valutano le alternative, si identificano i rischi, si pianificano attività quali prototipazione e/o simulazione
3. Si sviluppa e si verifica il prodotto
4. Si effettua la revisione dei risultati ottenuti e si pianifica la prossima iterazione della spirale

Caso di studio: "Synchronize and stabilize" (Microsoft)

- Si basa sulla scomposizione dei progetti in piccole squadre che lavorano in parallelo, ma che si comportano come un unico team di grandi dimensioni
- Il processo di sviluppo comincia con una *fase di pianificazione* che definisce la *visione* del prodotto, la sua specifica e la sua tabella di marcia
- La *visione* indica gli obiettivi del prodotto e stabilisce le priorità tra le caratteristiche
- Il *documento di specifica* fornisce una descrizione precisa di ogni caratteristica, l'architettura globale del sistema e le principali dipendenze tra i componenti
- Le specifiche non sono congelate ma vengono continuamente riviste (circa il 30% delle specifiche viene cambiato)
- Ciascun team è composto da *sviluppatori* e *tester*
- *Sincronizzazione giornaliera*: ogni team deve inserire i risultati prodotti in un database. Il codice sorgente dell'intero prodotto viene compilato ed il "build" trasferito al test.
- *Stabilizzazione del prodotto*: sono definite una serie di date (*milestone*) in cui si controlla la stabilità dello sviluppo (la maggior parte degli errori siano stati corretti)

Caso di studio: approccio open-source

Software Open-Source

- Il successo di prodotti quali i sistemi operativi GNU/Linux, il web server Apache, il linguaggio Pearl hanno dimostrato la validità dell'approccio *open-source* come alternativa allo sviluppo del software tradizionale
- Si basa su una politica di regolamentazione delle licenze:
  - L'*uso* del prodotto software non può essere limitato da alcuna restrizione
  - È possibile effettuare un numero qualsiasi di *copie* senza che ciò comporti alcun pagamento
  - Chiunquè può introdurre liberamente nuove *modifiche*
  - Non ci sono limitazioni sulla *distribuzione* del software o di software da esso derivato
- La maggior parte delle licenze *open-source* specificano la condizione di *copyleft* che impone che versioni modificate debbano essere rilasciate negli stessi termini della licenza originale

Caso di studio: approccio open-source

Processo di sviluppo Open-Source

- Tale modello di sviluppo si è diffuso grazie a Internet, che fornisce una infrastruttura di cooperazione tra i programmatori
- La collaborazione si basa su strumenti quali la posta elettronica e i gruppi di discussione
- Il codice è opportunamente messo a disposizione di tutti
- In alcuni casi si usano strumenti per il controllo delle versioni e delle configurazioni
- L'organizzazione dei progetti open-source è basata su un piccolo gruppo di *sviluppatori* ed una più grande popolazione di *contributori*
- Gli sviluppatori sono coinvolti a lungo termine nel progetto e sono responsabili della ricezione dei contributi, delle loro revisioni e della loro integrazione nel codice di base
- I contributori che si distinguono per la qualità dei loro apporti possono divenire sviluppatori

Organizzazione del processo

- L'organizzazione del processo di sviluppo è un'attività critica che va dalla gestione del personale alla gestione dei prodotti del ciclo di vita
- L'organizzazione del processo definisce dei *metodi* e li combina a formare una *metodologia*

- **Metodo**: modo sistematico ed ordinato per fare qualcosa
- **Metodologia**: insieme di metodi e tecniche

L'impiego di una metodologia:

- migliora la fiducia dei programmatori in quello che stanno facendo
- insegna alle persone poco esperte come risolvere i problemi in modo sistematico
- incoraggia un approccio uniforme e standard alla risoluzione dei problemi

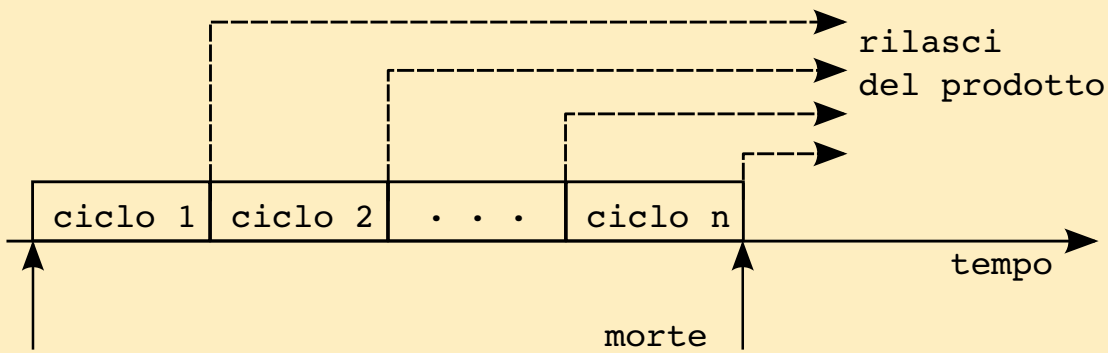
Metodologie esistenti

Critica

- Le metodologie esistenti non hanno solide basi teoriche
- Sono supportate empiricamente dal buon senso
- Talvolta non possono essere applicate in modo meccanico: il progettista deve intervenire con il suo giudizio
- Sono utili in aree applicative specifiche
- L'adozione di una metodologia efficace per una determinata area aiuta ad abbreviare i tempi di sviluppo e facilita la comunicazione tra i membri del team di sviluppo

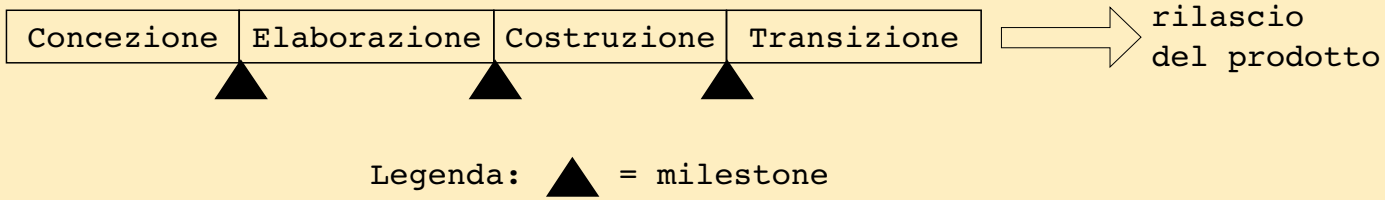
Unified Software Development Process (UP)

- È una metodologia basata sul linguaggio di modellazione UML
- UML fornisce notazioni per la specifica, l'analisi, la visualizzazione, la costruzione e la documentazione degli artefatti prodotti durante il ciclo di vita del software
- UP trasforma lo sviluppo di un sistema orientato agli oggetti in un modello *iterativo* e *incrementale*
- il principio di base è che un progetto grosso dovrebbe essere scomposto in *iterazioni controllate* (mini progetti) che forniscono versioni incrementali del prodotto
- un ciclo di vita UP può essere visto come una sequenza di cicli dalla concezione del progetto al suo termine



Fasi dei processi RUP

- Ogni ciclo viene suddiviso in una sequenza di quattro fasi ciascuna delle quali termina con una *milestone*
- Una *milestone* consiste in un insieme di artefatti che possono essere soggetti a controllo di qualità



1. **Concezione**: Si effettua una valutazione iniziale del progetto (Analisi di fattibilità).
2. **Elaborazione**: Si identificano i casi d'uso e si specifica l'architettura software (*baseline*)
3. **Costruzione**: Si implementa arricchendo la baseline sviluppando e testando il codice fino a supportare una quantità di funzionalità tali da poter essere rilasciato
4. **Transizione**: include varie attività finali condotte iterativamente (es. distribuzione del sistema, formazione degli utenti) (beta testing)

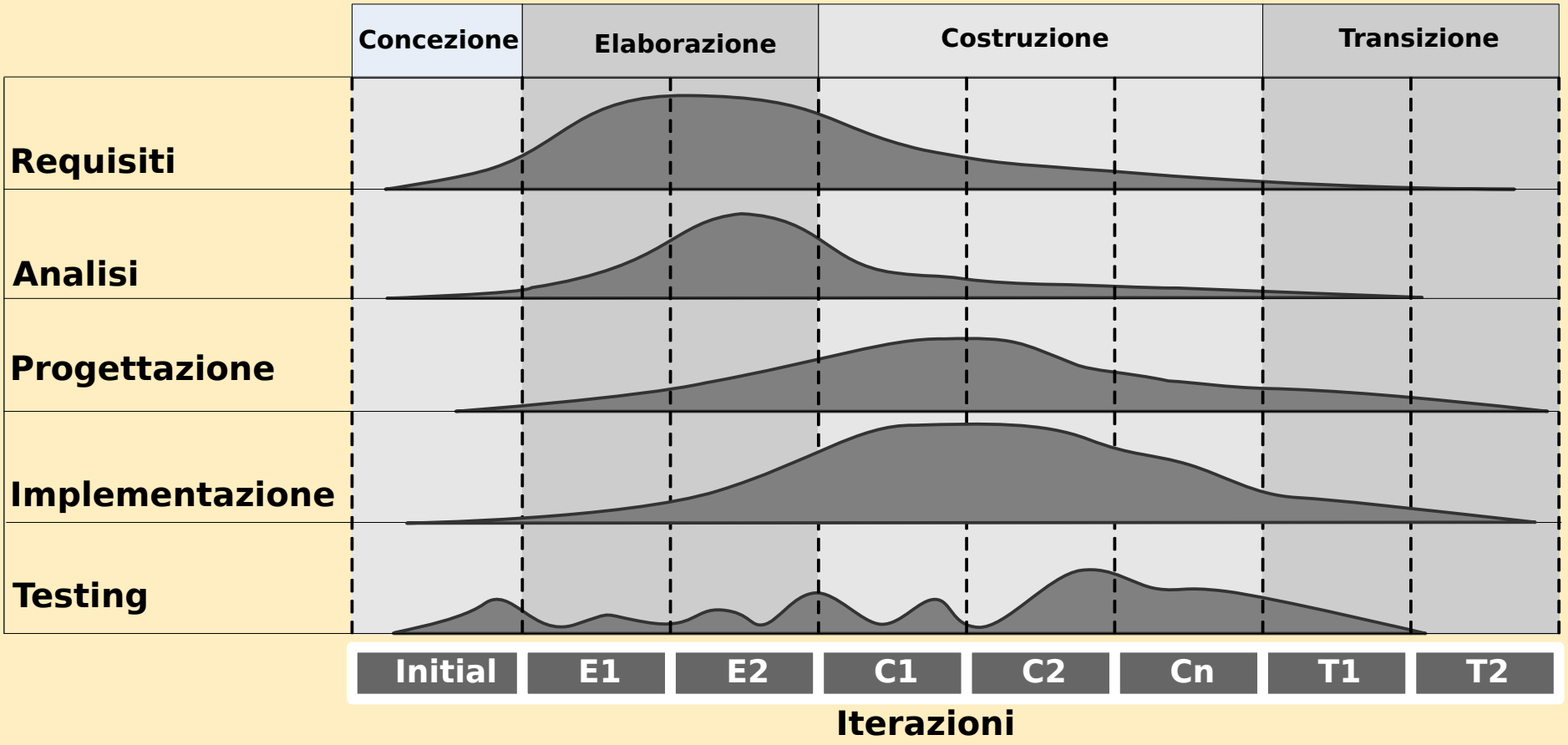
Sommario RUP

- Nessuna fase del ciclo è monolitica
- Ogni fase è suddivisa in una serie di iterazioni controllate
- Le iterazioni producono un incremento che può essere soggetto a valutazione
- Un piano di iterazioni specifica il costo previsto, gli artefatti che saranno prodotti, l'allocazione delle risorse umane e l'assegnazione dei compiti

- In RUP le attività di un processo sono chiamate *workflow*
- I workflow non sono organizzati in fasi sequenziali
- Le varie iterazioni attraversano in genere tutti i workflow principali

- RUP è iterativo ed incrementale: è guidato dagli *use case* e centrato sulle architetture
- Gli use case costituiscono l'input principale dei workflow di analisi, progettazione, implementazione e test
- L'architettura è l'artefatto principale utilizzato per concettualizzare, gestire e fare evolvere il sistema

Attività (workflow) con fasi ed iterazioni



RUP best practices

1. **Sviluppo iterativo del software.**
   Pianificare gli sviluppi incrementali del sistema basandosi sulle priorità del cliente. Sviluppare le funzionalità più importanti nelle prime fasi del processo di sviluppo.
2. **Gestione dei requisiti.**
   Documentare esplicitamente i requisiti e mantenere traccia dei cambiamenti. Analizzare l'impatto dei cambiamenti prima di accettarli.
3. **Uso di architetture component-based.**
   Strutturare l'architettura del sistema in componenti riusabili.
4. **Modellazione visuale.**
   Usare modelli UML per rappresentare le viste statiche e dinamiche del sistema software.
5. **Verifica della qualità.**
   Garantire che il software soddisfi gli standard qualitativi dell'organizzazione.
6. **Controllo del cambiamento.**
   Gestire i cambiamenti nel software adottando procedure e strumenti adeguati.