Gestione progetti e processi di sviluppo

Ingegneria del software

Vincenzo Bonnici
Corso di Laurea in Informatica
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Università degli Studi di Parma

2025-2026

Gestione progetti

- La gestione di un progetto software (management) descrive le **attività** necessarie affinché il prodotto software sia finito in tempo e in accordo ai requisiti delle organizzazioni di sviluppo e di acquisto

- Attività di gestione
  - Scrittura della proposta (proposal)
  - Pianificazione e scheduling progetto
  - Costi di progetto
  - Monitoraggio e revisioni progetto
  - Selezione e valutazione personale
  - Scrittura report e presentazioni

Personale (Staff)

- Non sempre è possibile avere le persone ideali per lavorare su un progetto
  - Il budget potrebbe non consentire di usare il personale molto costoso
  - Il personale con l'esperienza appropriata potrebbe non essere disponibile
  - Una organizzazione potrebbe voler formare del personale lavorando su un progetto
- I manager devono lavorare con questi vincoli specialmente quando (come oggi) vi è carenza di personale ben qualificato

Pianificazione del progetto

- La pianificazione (compreso il monitoring) è probabilmente l'attività che prende più tempo
  - E' continuativa dall'inizio alla consegna del prodotto software
  - I piani devono essere revisionati (arricchiti di dettagli, corretti, aggiornati) quando nuove informazioni diventano disponibili
- Piani di tipo differente possono essere sviluppati a supporto del piano principale che si concentra su scheduling e budget

Tipi di piani del progetto

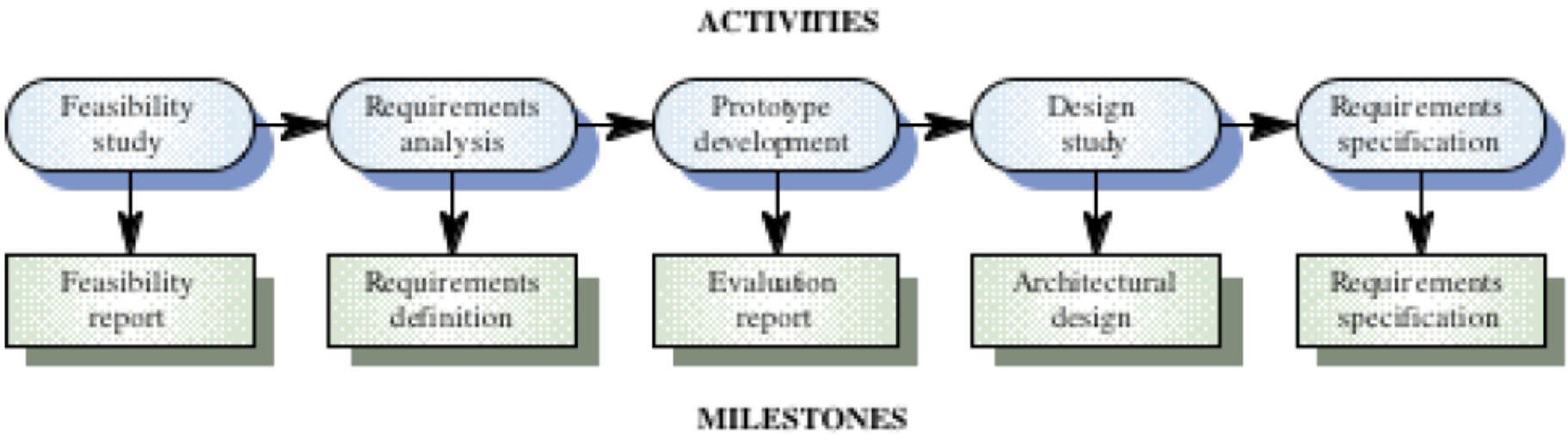
- **Piano della qualità** - Descrive le procedure di qualità e gli standard che dovranno essere usati
- **Piano di validazione** - Descrive l'approccio, le risorse e lo scheduling usati per validare il sistema
- **Piano di gestione configurazione** - Descrive le procedure e le strutture di gestione configurazione che devono essere usate
- **Piano di manutenzione** - Predice i requisiti di manutenzione del sistema, i costi di manutenzione e lo sforzo richiesto
- **Piano di sviluppo del personale** - Descrive come le abilità e l'esperienza del personale sarà sviluppato

Organizzazione attività

- Le attività del progetto dovrebbero essere organizzate per produrre risultati tangibili che i manager possono esaminare per stabilire i progressi raggiunti
- **Milestones** sono il punto finale di una attività del processo software
- **Deliverables** sono i risultati del progetto che sono consegnati ai clienti
- Il processo a cascata permette direttamente la definizione di milestones

Milestones

Esempio di attività e milestones nella fase di specifica dei requisiti



Scheduling del progetto

- Dividere il progetto in task e stimare tempo e risorse necessarie a completare ciascun task
  - Ogni task dovrebbe durare almeno 1 settimana e non dovrebbe superare le 10 settimane
- Organizzare i task in modo concorrente per fare un uso ottimale della forza lavoro
- Minimizzare le dipendenze tra task per evitare ritardi dovuti ad un task che aspetta il completamento di un altro
- Lo scheduling dipende dall'intuizione e dall'esperienza dei manager del progetto

Problemi nella schedulazione

- E' difficile valutare la difficoltà dei problemi e quindi il costo per lo sviluppo di una soluzione
- La produttività non è proporzionale al numero di persone che lavorano per un task
  - Aggiungere persone ad un progetto in ritardo fa aumentare il ritardo, a causa della comunicazione necessaria
- Qualcosa di inaspettato può capitare
  - Prevedere un piano per le emergenze
- E' possibile stimare la durata di un progetto
  - Ricorrendo all'esperienza (propria o altrui)
  - Facendo affidamento che niente vada male
  - Aggiungendo un 30% per i problemi che possono essere intravisti
  - Aggiungendo un ulteriore 20% per tener conto di ciò che non è stato immaginato

Diagrammi

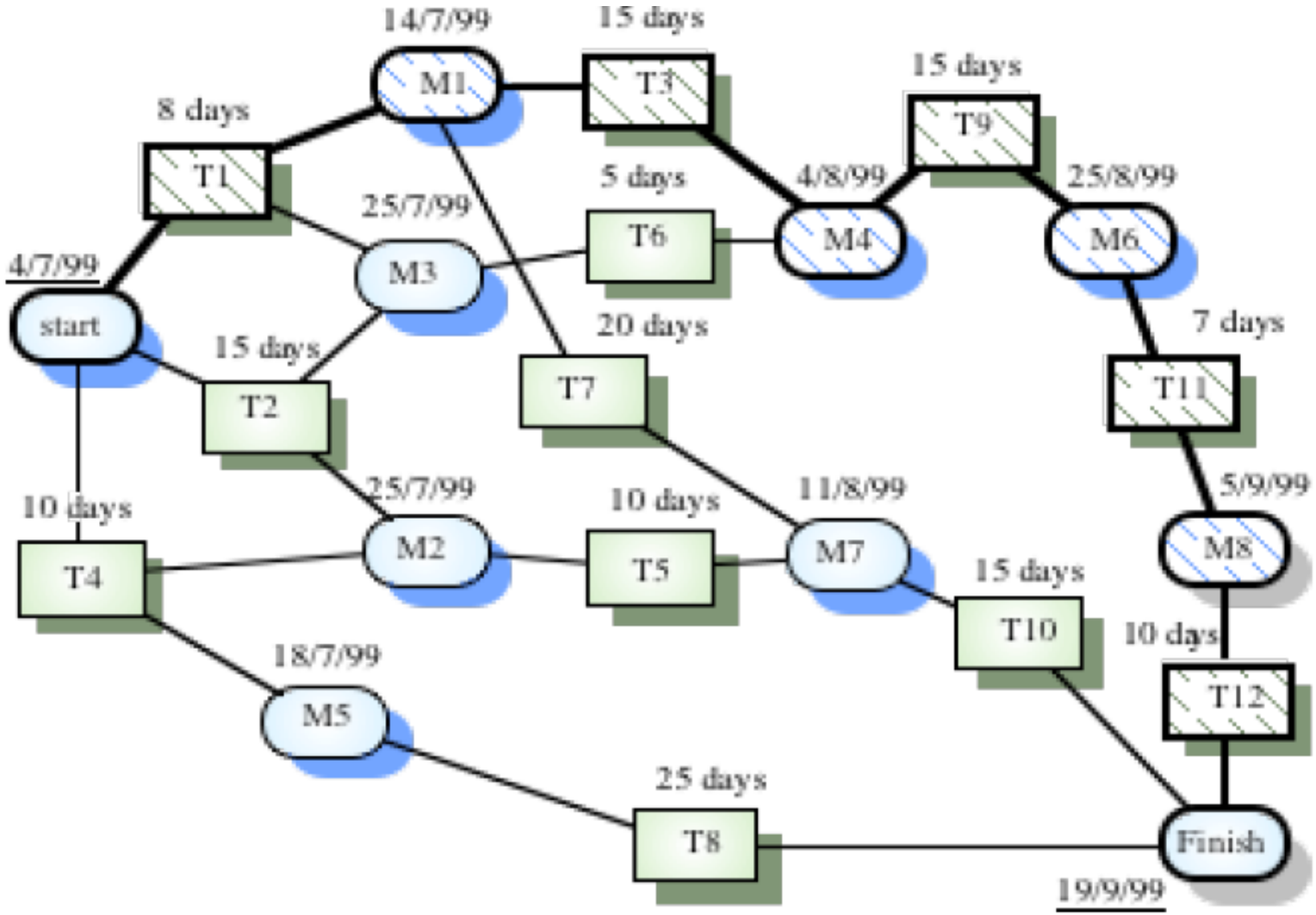
- Uso di notazioni grafiche per illustrare lo scheduling del progetto
- Mostrare le divisioni in task
  - I task non dovrebbero essere troppo piccoli, la loro durata dovrebbe essere di una o due settimane
- I diagrammi delle attività mostrano le dipendenze tra i task ed i percorsi critici
- I diagrammi a barre mostrano lo scheduling su un calendario

Durata e dipendenza dei task

Task	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

Rete delle Attività

Diagramma PERT.



Il tempo minimo richiesto per completare il progetto è dato dal più lungo percorso del grafo (detto percorso critico)

Temporizzazione attività

Diagramma di Gantt.

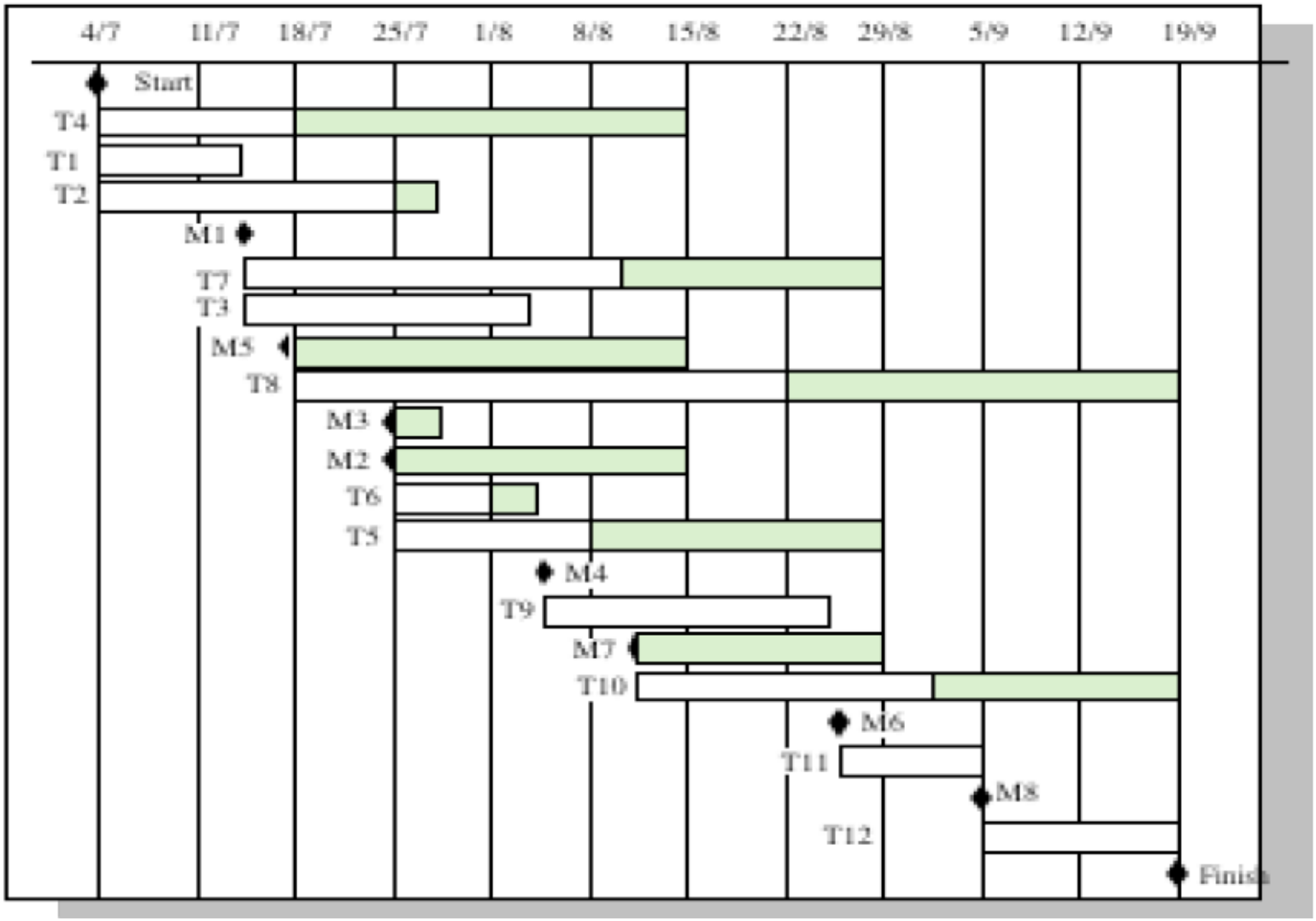


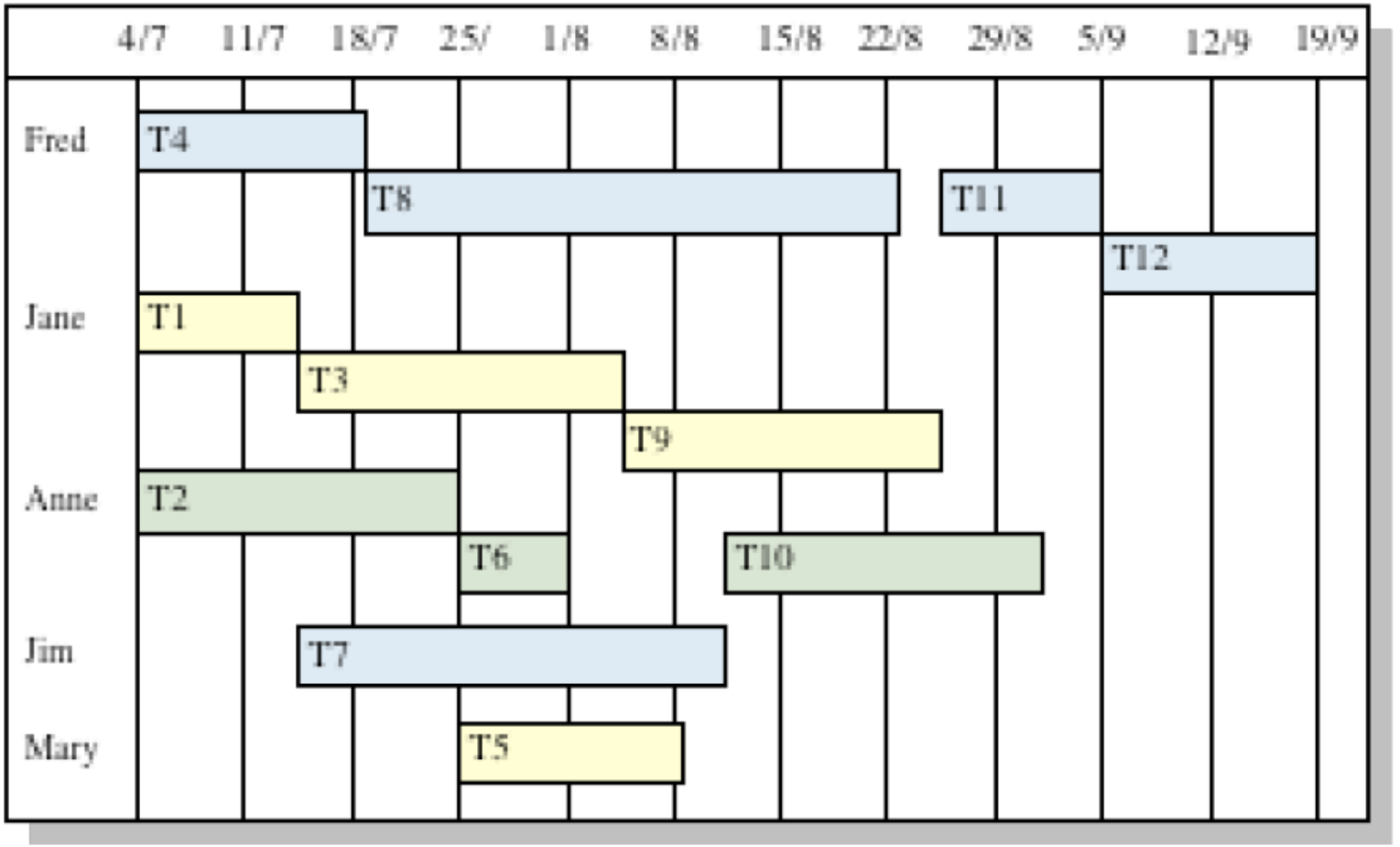
Diagramma di Gantt

Il diagramma di Gantt è costruito partendo da un asse orizzontale - a rappresentazione dell'arco temporale totale del progetto, suddiviso in fasi incrementali (ad esempio, giorni, settimane, mesi) - e da un asse verticale - a rappresentazione delle mansioni o attività che costituiscono il progetto.

Delle barre orizzontali di lunghezza variabile rappresentano le sequenze, la durata e l'arco temporale di ogni singola attività del progetto. Queste barre possono sovrapporsi durante il medesimo arco temporale ad indicare la possibilità dello svolgimento in parallelo di alcune delle attività.

Un diagramma di Gantt permette dunque la rappresentazione grafica di un calendario di attività, utile al fine di pianificare, coordinare e tracciare specifiche attività in un progetto; di contro, uno degli aspetti non tenuti in considerazione in questo tipo di diagrammazione è l'interdipendenza delle attività, caratteristica invece della programmazione reticolare, cioè del diagramma PERT.

Allocazione del personale



Gestione del rischio

- La gestione del rischio consiste nell'identificare i rischi e progettare piani che minimizzano gli effetti dei rischi sul progetto

- Un rischio è la probabilità che una circostanza negativa si verifichi
  - Rischi di **progetto** hanno effetto sullo scheduling o sulle risorse
  - Rischi di **prodotto** hanno effetto sulla qualità o sulle performance del prodotto che si sta sviluppando
  - Rischi di **business** hanno effetto sulla organizzazione che sviluppa o richiede il software

Rischi

Tipo	*Rischio*	*Descrizione*
Progetto	Turnover del personale	Personale esperto lascia il progetto prima che sia finito
Progetto	Cambio della gestione	Cambiamenti nell'organizzazione della gestione con differenti priorità
Progetto	Indisponibilità hardware	Hardware essenziale non è consegnato in tempo
Progetto e prodotto	Cambiamento dei requisiti	Più cambiamenti di requisiti rispetto alle previsioni
Progetto e prodotto	Ritardo nelle specifiche	Le specifiche su interfacce essenziali non sono disponibili in tempo
Progetto e prodotto	Dimensione sottostimata	La dimensione del sistema è stata sottostimata
Business	Cambio di tecnologia	La tecnologia su cui il sistema è costruito è sorpassata da una nuova tecnologia

Gestione del rischio

- **Identificazione** dei rischi - Di progetto, prodotto e di business
- **Analisi** dei rischi - Valutazione della probabilità e delle conseguenze dei rischi identificati
- **Pianificazione** rischi - Progettare un piano per evitare o minimizzare gli effetti dei rischi
- **Monitoraggio** rischi - Monitoraggio durante il progetto

Che cos'è l'ingegneria del software?

- L'ingegneria del software è l'applicazione di un approccio sistematico, disciplinato e quantificabile allo sviluppo, l'esercizio e la manutenzione del software

- Dunque
  - una disciplina ingegneristica
  - interessata allo sviluppo del software
    - interessata a tutti gli aspetti della produzione del software – dalla specifica alla manutenzione, dagli aspetti tecnici a quelli economici e di gestione del progetto e delle persone
    - interessata allo sviluppo efficiente di software di alta qualità
  - diversa da altre discipline ingegneristiche
    - a causa delle caratteristiche distintive del software

10 aree di conoscenza dell'ingegneria del sw

- **Requisiti del software**
  - un requisito è una proprietà che deve essere esibita nella risoluzione di un problema del mondo reale
- **Progettazione del software**
  - la progettazione (design) è il processo di definizione dell'architettura, dei componenti, delle interfacce e di altre caratteristiche di un sistema o componente
  - il progetto (design) è il risultato del processo di progettazione
- **Costruzione del software**
  - la costruzione del software si riferisce alla creazione dettaglia di software funzionante e significativo, attraverso una combinazione di codifica, verifica, test unitari, test di integrazione e debugging
- **Verifica (testing) del software**
  - il test del software consiste nella verifica dinamica del comportamento di un programma sulla base di un insieme finito di test case, scelto opportunamente da un dominio di esecuzione solitamente infinito
- **Manutenzione del software**
  - manutenzione del software in "operazione", per gestire anomalie, cambiamenti nell'ambiente operativo e nuovi requisiti utente

10 aree di conoscenza dell'ingegneria del sw

- **Gestione delle configurazioni del software**
  - disciplina che riguarda l'identificazione delle configurazioni del software in momenti temporali diversi, al fine di controllare in modo sistematico cambiamenti della configurazione e di mantenere l'integrità e la tracciabilità della configurazione durante tutto il ciclo di vita del sistema
- **Gestione dell'ingegneria del software**
  - riguarda la gestione e la misurazione dell'ingegneria del software
- **Processi dell'ingegneria del software**
  - riguarda definizione, implementazione, valutazione, misurazione, gestione, cambiamento e miglioramento del processo di ingegneria del software stesso
- **Strumenti e metodi dell'ingegneria del software**
  - riguarda metodi (informali, formali e basati su prototipazione) dell'ingegneria del software e strumenti di supporto alle diverse aree dell'ingegneria del software
- **Qualità del software**
  - riguarda considerazioni sulla qualità del software che trascendono i processi per la gestione del ciclo di vita del software

Che cos'è un processo di sviluppo software?

- Un **processo di sviluppo software** è l'insieme delle attività e dei risultati che creano un prodotto software
  - ad es., XP, UP
- Un processo definisce **chi fa che cosa, quando e come** per raggiungere un certo obiettivo

Che cos'è un processo di sviluppo software?

- Attività fondamentali comuni a tutti i processi di sviluppo software
  - **specifica** – definizione di che cosa deve fare il software e dei vincoli per il suo sviluppo
  - **sviluppo** – progettazione e programmazione
  - **validazione** – verifica che il software prodotto sia conforme alle richieste del suo committente
  - **evoluzione** – modifica del prodotto per adeguarlo ai requisiti dell'utente e del mercato che cambiano
- Processi diversi organizzano queste attività in modi diversi
- Perché esistono diversi processi di sviluppo software?
  - perché contesti diversi (tipo di sistema, tipo di committente, caratteristiche degli sviluppatori e del team di sviluppo, tempo, budget) richiedono tipi differenti di processi di sviluppo

Che cos'è un modello di processo di sviluppo software?

- Un **modello di processo software** è una descrizione semplificata di un processo (o di un suo aspetto) – osservato da un determinato punto di vista

- Alcuni punti di vista possibili
  - **modello a flusso di lavoro (workflow)** – il processo è una sequenza di attività (azioni), con relativi input, output e dipendenze

  - **flusso di dati o modello di attività** – il processo è un insieme di attività (trasformazioni), che modificano delle informazioni
    - ad es., come trasformare le specifiche in progetto

  - **modello ruolo/azione** – rappresenta i ruoli delle persone coinvolte e le attività di cui sono responsabili

Modelli generici di processi di sviluppo software

- Molti modelli di processo sono basati sui seguenti modelli generici (paradigmi) per lo sviluppo del software
  - **approccio a cascata** – le attività sono fasi di lavorazione separate – un'attività inizia quando le precedenti sono concluse e "firmate per accettazione"
  - **sviluppo iterativo** – il processo è organizzato in iterazioni, in cui vengono eseguite varie attività
  - **ingegneria del software basata su componenti** – tecnica che presuppone l'esistenza di alcune parti del sistema, e che concentra lo sviluppo sull'integrazione di queste parti, anziché sullo sviluppo dal nulla

Che cos'è un metodo dell'ingegneria del sw?

- Un **metodo dell'ingegneria del software** è un approccio strutturato relativo ad uno o più aspetti (tecnici e focalizzati) dello sviluppo del software, che ha l'obiettivo di facilitare la produzione di software di alta qualità
  - i metodi costituiscono il sapere tecnico relativo allo sviluppo del software
    - ad es., il metodo OO di Larman, progettazione di basi di dati con ER
  - metodi diversi hanno aree di applicabilità diverse
  - non esiste un metodo ideale
- I metodi dell'ingegneria del software comprendono
  - modelli (linguaggi) – con la loro sintassi e semantica
  - raccomandazioni e linee guida – a vari livelli di dettaglio

L'ingegneria del software è una tecnologia stratificata

- un processo software è un framework
  - che definisce un contesto in cui applicare i metodi, per creare prodotti ed elaborati intermedi
  - con il supporto di opportuni strumenti
- con un impegno alla qualità

Quali le caratteristiche di un buon sw?

- Un prodotto software ha – oltre alle funzionalità – diverse caratteristiche associate che ne riflettono la qualità
  - **affidabilità** – il software deve effettivamente fornire le funzionalità richieste
  - **disponibilità** – il software deve essere funzionante in modo continuato nel tempo
  - **efficienza** – il software deve fornire le prestazioni desiderate
  - **sicurezza** – il software deve fornire protezione alle sue funzionalità ed ai suoi dati
  - **mantenibilità** – il software deve poter evolvere per soddisfare i requisiti che cambiano dei suoi clienti
  - **verificabilità** – deve poter essere possibile verificare le funzionalità e le altre qualità del software
  - **usabilità** – il software deve essere facilmente utilizzabile dagli utenti per quale stato sviluppato
  - ...

Quali le sfide chiave per l'ing. del sw?

- L'ingegneria del software del XXI secolo deve affrontare diverse sfide chiave
  - **eterogeneità** – far interagire sistemi distribuiti che sono eterogenei – per piattaforma, rappresentazione dei dati, ambiente di esecuzione, ... – consentendo un accesso ubiquitario e pervasivo
  - **consegna e flessibilità** – sviluppare tecniche per rispondere con velocità alla consegna del software ed alle richieste di cambiamento
  - **fiducia** – è essenziale potersi fidare del software
  - ...
  - **agilità di business** – poter creare o modificare dinamicamente processi di business – e le applicazioni software che li supportano in modo da reagire in modo competitivo nel proprio settore di business

Processi (di sviluppo del) software

- Un processo software descrive le attività (o task) necessarie allo sviluppo di un prodotto software e come queste attività sono collegate tra loro
- Attività o fasi dello sviluppo
  - Analisi dei requisiti (specifiche)
  - Progettazione (design)
  - Codifica o implementazione (codice)
  - Convalida o testing (approvazione)
  - Manutenzione

Fase di Analisi dei Requisiti

- L'**analisi dei requisiti** è la fase che porta a definire le specifiche, stabilisce i servizi richiesti ed i vincoli del software
  - **Requisito**: ciascuna delle caratteristiche che il software deve avere
  - **Specifica**: descrizione rigorosa delle caratteristiche del software
    - I requisiti tendono ad essere granulari (ovvero: molti e piccoli)
  - **Feature**: un insieme di requisiti correlati tra loro. Una feature permette all'utente di soddisfare un obiettivo

- Passi per l'**ingegneria dei requisiti**: (1) Studio di fattibilità, (2) Analisi dei requisiti, (3) Specifica dei requisiti, (4) Convalida requisiti
- Requisiti
  - Funzionali: Cosa il sistema deve fare (funzionalità)
  - Non-funzionali: Come il sistema lo fa (es. affidabilità, efficienza, prestazioni, manutenibilità, etc.)

Esempi di Feature e Requisiti

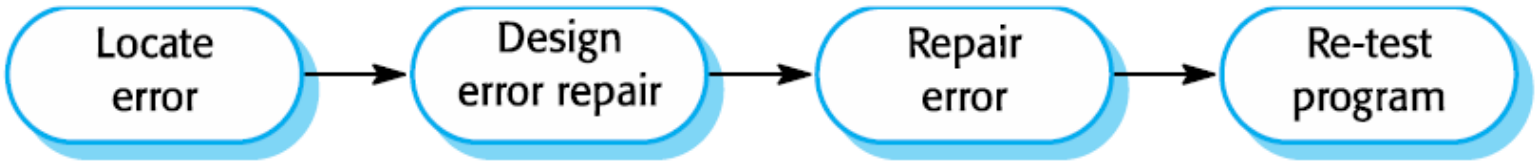
- Feature di Firefox 3.6
  - Browsing privatamente: navigazione del web senza lasciare tracce
  - Password manager: ricordare le password dei siti, senza usare pop-up
  - Awesome Bar: trovare i siti preferiti in pochi secondi
  - One-click bookmark: bookmark, cerca e organizza siti web velocemente e facilmente
- Requisiti (sintetici) di Firefox 3.7
  - Eseguire i plug-in in un processo separato per migliorare la stabilità dell'applicazione e diminuire i tempi di risposta
  - Migliorare i tempi di startup
  - Ottimizzare caricamento delle pagine

Progettazione ed implementazione

- Fase di Progettazione: stabilisce la struttura software che realizza le specifiche
- Attività della progettazione
  - Suddivisione dei requisiti
  - Identificazione sottosistemi, ovvero progettazione architettura software
  - Specifica delle responsabilità dei sottosistemi
  - Progettazione di: interfacce, componenti, strutture dati, algoritmi
- Ognuna delle attività suddette produce un documento corrispondente (o integra un documento già esistente) che descrive un modello
  - Modello degli oggetti, di sequenza, di transizione stati, strutturale, data-flow
- Fase di Implementazione: produce un programma eseguibile a partire dalla struttura stabilita
- Progettazione ed implementazione sono attività correlate e spesso sono alternate

Fase di implementazione

- Consiste nella programmazione ovvero nella traduzione dei modelli del progetto in un programma (codice) e della rimozione degli errori dal programma
- I programmatori effettuano alcuni test sul programma prodotto per scoprire errori (o bug) e rimuoverli
- Per rimuovere gli errori
  - Localizzare l'errore nel codice
  - Rimuovere l'errore nel modello e poi nel codice
  - Effettuare nuovamente il test nel programma



Fase di Convalida (Verifica & Validazione)

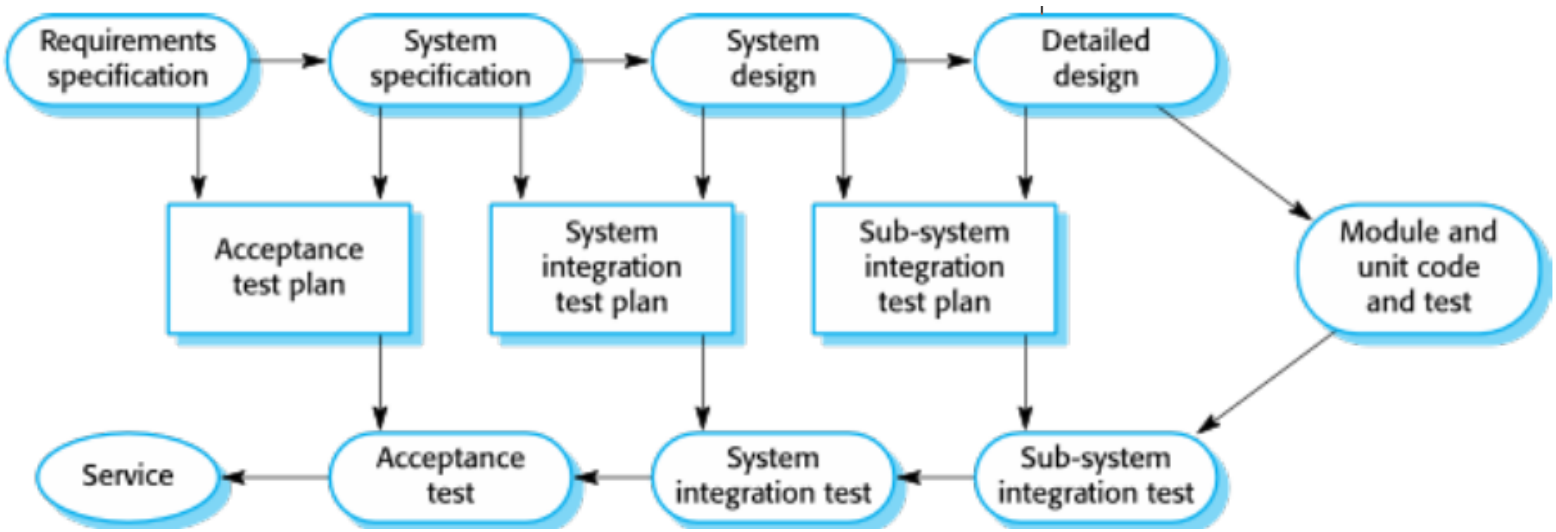
- La fase di convalida o Verifica e Validazione (V & V) del sistema software intende mostrare che il sistema software è conforme alle specifiche e che soddisfa le richieste (aspettative) del cliente
  - Viene condotta tramite processi di revisione e test del sistema software
  - I test mirano ad eseguire il sistema software in condizioni derivate dalle specifiche di dati reali che il sistema software dovrà elaborare

Fase di Test

- Test di componenti o unità (unit test): i singoli componenti sono testati indipendentemente
  - Componenti potranno essere funzioni, o oggetti, o loro raggruppamenti
- Test di sistema: l'intero sistema è testato, dando speciale importanza alle proprietà emergenti
- Test di accettazione (alpha test): test condotti dagli sviluppatori con dati del cliente per verificare che il sistema soddisfi le esigenze del cliente
- Beta test: test condotti da alcuni clienti sul prodotto quasi completo
- Prodotti software corrispondenti alle varie fasi di test
  - Versione alfa, versione beta, versione gold
  - Alfa: versione integrata (con difetti e parti mancanti)
  - Beta: versione che include tutte le feature (ma con difetti

Quadro riassuntivo di sviluppo

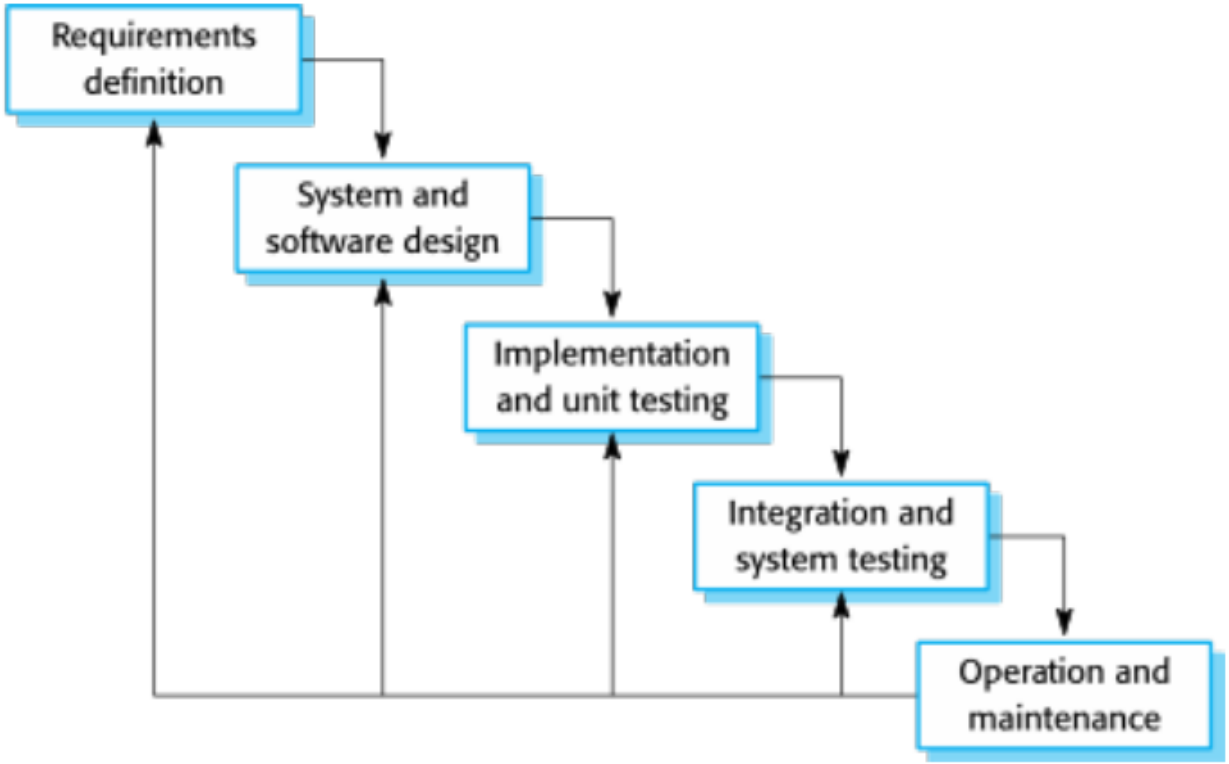
- Dai requisiti (R) otteniamo il documento della specifica dei requisiti (SRS)
- Dall'SRS ricaviamo il design del sistema (DS)
- Dal DS ricaviamo il design dettagliato (DD)
- Da DD ricaviamo codice e test
- Da DS e da DD ricaviamo come integrare i sottosistemi e come fare i test di sistema
- Da R e SRS ricaviamo come fare i test di accettazione



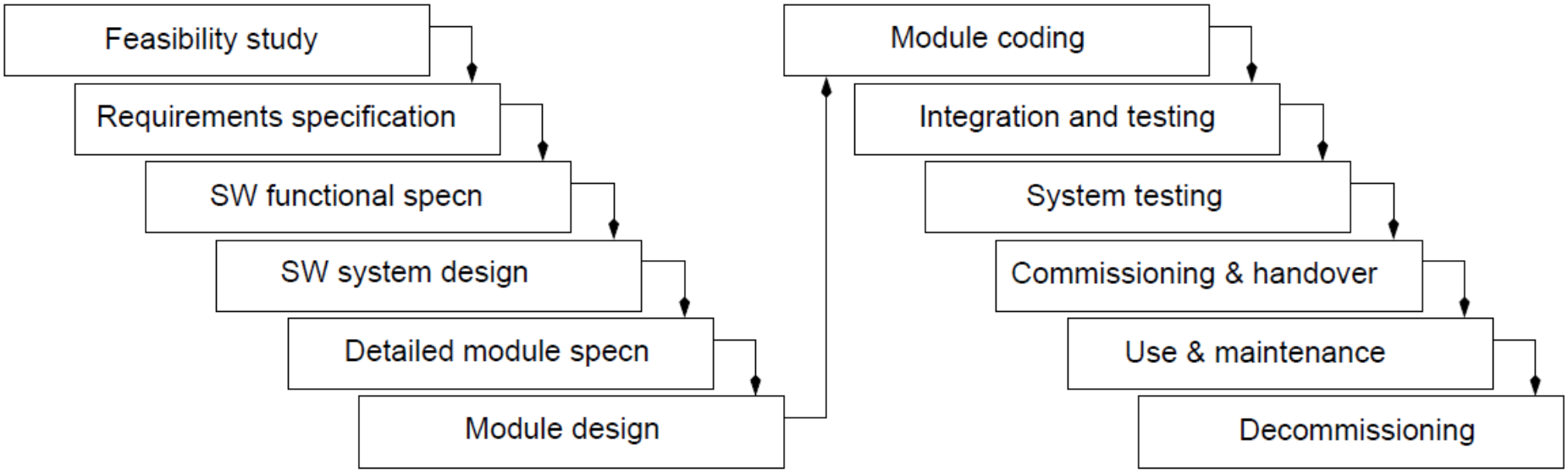
Modello a cascata (Waterfall)

- Il primo dei processi (anni ‘70), derivato da altri processi di ingegneria
- Focalizza sul prodotto completo
- Si comincia la fase successiva solo se la fase precedente è completa: prima specifica tutto, poi produci tutto, poi testa tutto, ...
- Processo statico con tanta documentazione
- — Lungo tempo per ottenere il prodotto
- — Poche interazioni con i clienti (solo nella fase iniziale)
- — Difficoltà ad introdurre i cambiamenti richiesti dal cliente
- + Consistenza tra artefatti
- + Ampia documentazione
- + Utile se i requisiti sono stabili e chiaramente definiti
- + Usato principalmente per sistemi grandi, complessi, critici, per gestire team numerosi
- + Alta qualità del codice prodotto

Modello a cascata (Waterfall)



Modello a cascata (Waterfall)



International Atomic Energy Agency, Manual on Quality Assurance for Computer Software Related to the Safety of Nuclear Power Plants, Tech. Reports Series No. 282, IAEA, Vienna, 1988.

Modello a cascata (Waterfall)

É un processo articolato nelle seguenti fasi

- studio di fattibilità;
- analisi e specifica dei requisiti, suddivisa in
  - analisi (definizione e specifica) dei requisiti dell'utente, e
  - specifica dei requisiti del software;
- progetto, suddiviso in
  - progetto architetturale, e
  - progetto in dettaglio;
- programmazione e test di unità;
- integrazione e test di sistema;
- manutenzione.

Modello a cascata (Waterfall)

Studio di fattibilità

- stabilire se il prodotto può essere realizzato;
- stabilire se il è conveniente realizzarlo;
- possibili strategie di realizzazione alternative;
- tipo e quantità di risorse necessarie, quindi stima dei costi.

In base allo studio di fattibilità, il committente decide se firmare o no il contratto per la fornitura del software.

Gli errori di valutazione possono causare ritardi e inadempienze contrattuali, con perdite economiche per il fornitore o per il committente.

lo studio di fattibilità può essere fornito come prodotto finito, indipendente dall'eventuale prosecuzione del progetto:

- A chiede lo studio a B e affida il progetto a C.

Analisi e specifica dei requisiti

- capire e descrivere nel modo piú completo e preciso possibile che cosa vuole il committente;
- risultati usati da persone con diversi ruoli; quindi
- diversi livelli di astrazione;
- analisi del dominio: descrive il contesto in cui opera il SW;
- definizione dei requisiti e specifica dei requisiti: descrivono i servizi richiesti a diverso livello di dettaglio;
- specifica dei requisiti del software: descrive le caratteristiche del SW;
  - non l'implementazione!
  - rappresentazione esterna dell'informazione, interfaccia utente, comportamento osservabile. . .

Analisi e specifica dei requisiti: semilavorati

- **Documento di specifica dei requisiti (DSR)**: fondamentale; ha valore legale se incluso nel contratto;
  - analisi del dominio: parti in causa, entità, concetti, relazioni;
  - gli scopi dell'applicazione;
  - i requisiti funzionali;
  - i requisiti non funzionali;
  - i requisiti sulla gestione del processo di sviluppo.
- **Manuale utente**: Descrive il comportamento del sistema dal punto di vista dell'utente;
- **Piano di test di sistema**: Definisce come verranno eseguiti i test finali per convalidare il prodotto rispetto ai requisiti; può avere valore legale.

Modello a cascata (Waterfall)

Progetto

- decidere come deve essere fatto il sistema definito dai documenti di specifica;
- scelte fra le soluzioni possibili;
- architettura software:
  - moduli;
  - funzioni;
  - relazioni;
- progetto architetturale: moduli ad alto livello (sottosistemi);
- progetto in dettaglio: moduli a basso livello (moduli unità);

Progetto: semilavorati

- **Documento delle specifiche di progetto (DSP)**: definizione testuale e grafica dell'architettura SW;
- **Piano di test di integrazione**: come collaudare l'interfacciamento fra i moduli nel corso della costruzione del sistema.

Modello a cascata (Waterfall)

Programmazione (codifica) e test di unità

- implementazione e collaudo dei singoli moduli;
  - scelta di strutture dati ed algoritmi;
- gestione delle versioni: Subversion (SVN), git;
- configurazione, compilazione e collegamento automatici (in ambienti "a linea di comando"): Make, Automake, Autoconf, Libtool;
- ambienti di sviluppo integrati: MS Studio, Eclipse, . . . ;
- documentazione del codice: Doxygen, Javadoc, . . . ;
- test di unità: CppUnit, mockpp, ...

Programmazione (codifica) e test di unità:: semilavorati

- **Codice sorgente**: dei moduli e dei programmi di prova;
- **Documentazione del codice sorgente**: dei moduli e dei programmi di prova;
- **Documentazione dei test**: compresi i dati di test.

Integrazione e test di sistema

- vengono assemblati i sottosistemi a partire dai moduli componenti;
- effettuando parallelamente il test di integrazione;
- pianificazione e coordinamento fra gruppi di lavoro;
- test di sistema;
- alfa test;
- beta test.

Manutenzione

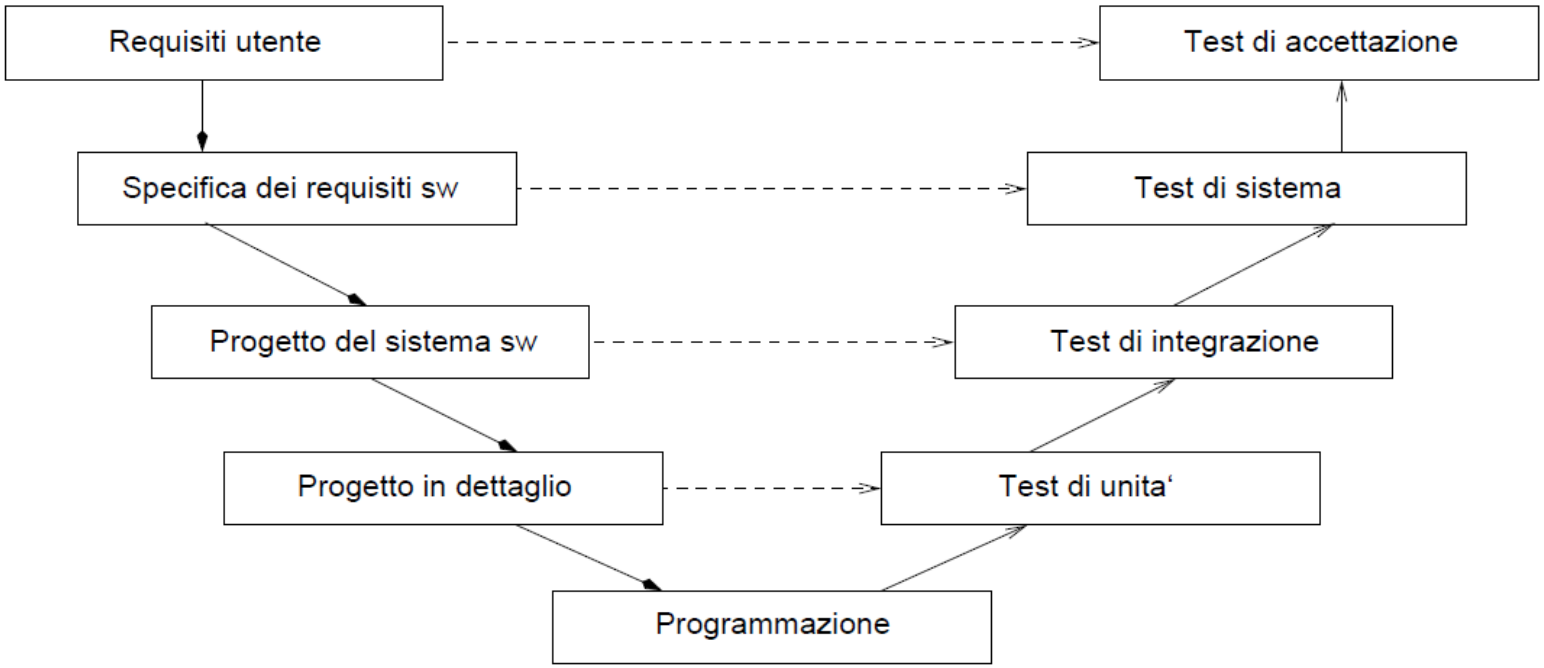
- correzione di errori presenti nel prodotto consegnato al committente; oppure
- aggiornamento del codice allo scopo di fornire nuove versioni;
- riprogettazione del codice;
- design for change: anticipare la necessità di modificare il codice;
- tipi di manutenzione:
  - correttiva: individuare e correggere errori;
  - adattativa: cambiamenti di ambiente operativo (porting) (piattaforma hardware, leggi, lingue, . . .);
  - perfettiva: aggiunte e miglioramenti.

Attività di supporto

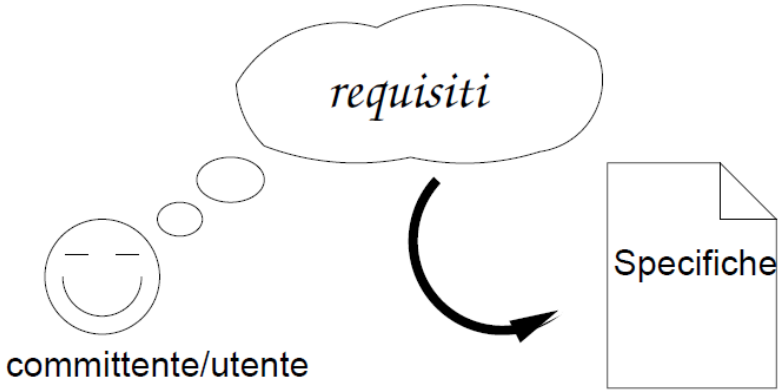
- gestione: pianificazione, sviluppo, allocazione delle risorse, flussi di informazione . . . ;
- documentazione: semilavorati, rapporti sull'avanzamento dei lavori, linee guida per gli sviluppatori, minute delle riunioni, . . . ;
- convalida e verifica: accertare che il prodotto ed i semilavorati soddisfino i requisiti;
- controllo di qualità: standard (p.es., ISO 9000) e procedure.

Il modello a V

Il modello di processo a V è una variante del modello a cascata in cui si mettono in evidenza le fasi di collaudo e la loro relazione con le fasi di sviluppo precedenti.



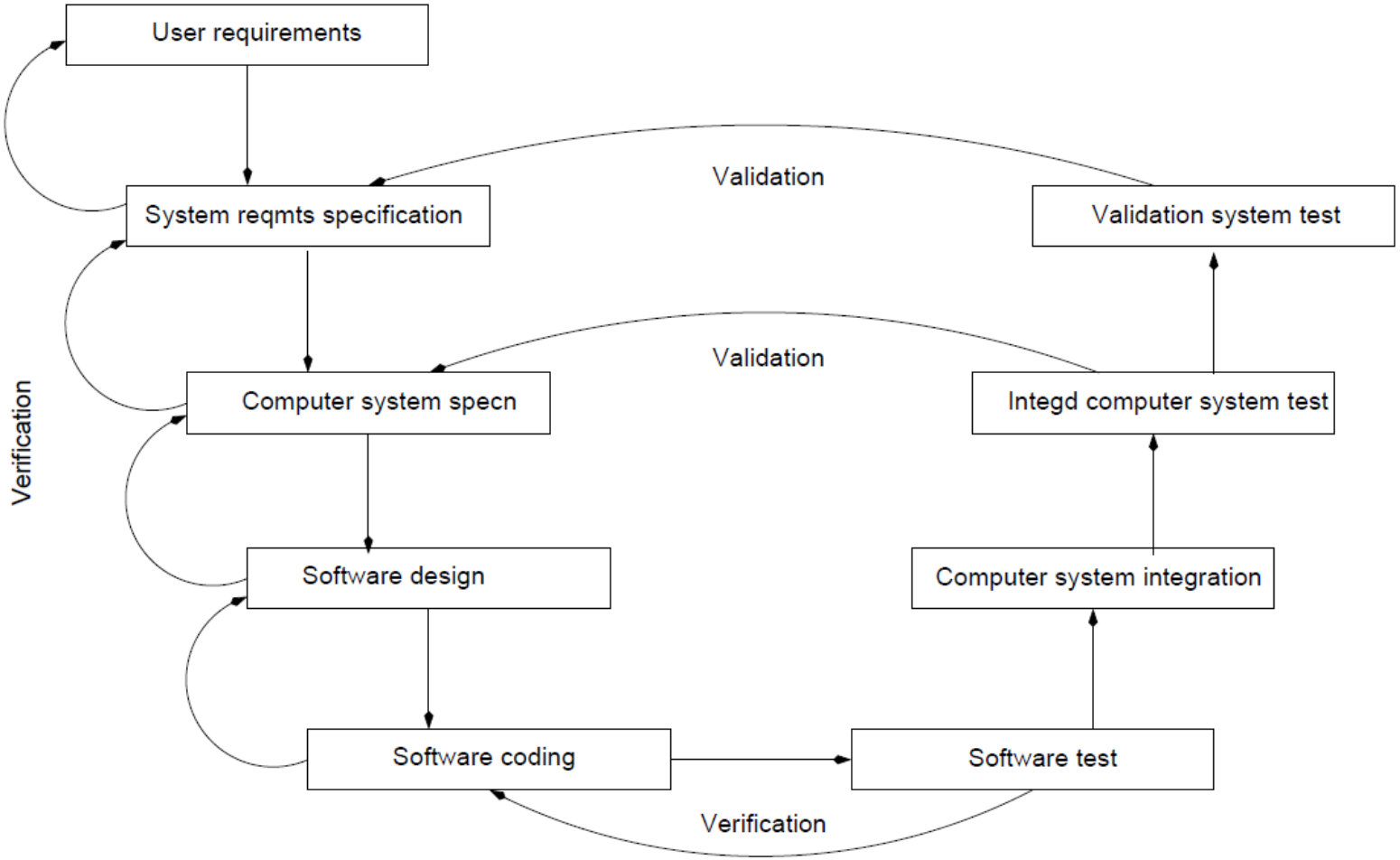
Il modello a V



- Definizione 1:
  - convalida = valutare prodotto e semilavorati rispetto ai requisiti;
  - verifica = valutare prodotto e semilavorati rispetto alle specifiche;
- Definizione 2:
  - convalida = valutare solo il prodotto rispetto ai requisiti;
  - verifica = valutare i semilavorati rispetto alle specifiche;

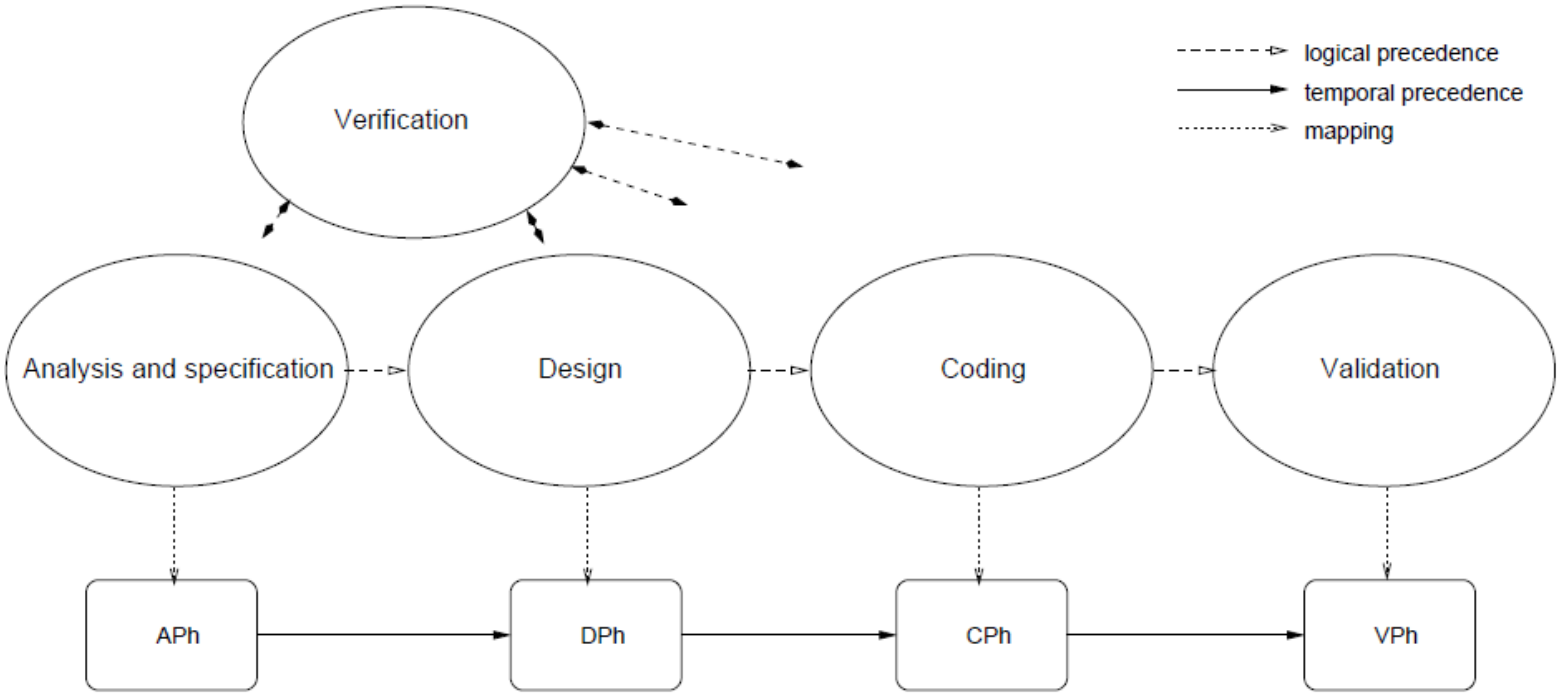
Il grado di formalità della verifica dipende da quello delle specifiche.

Il modello a V



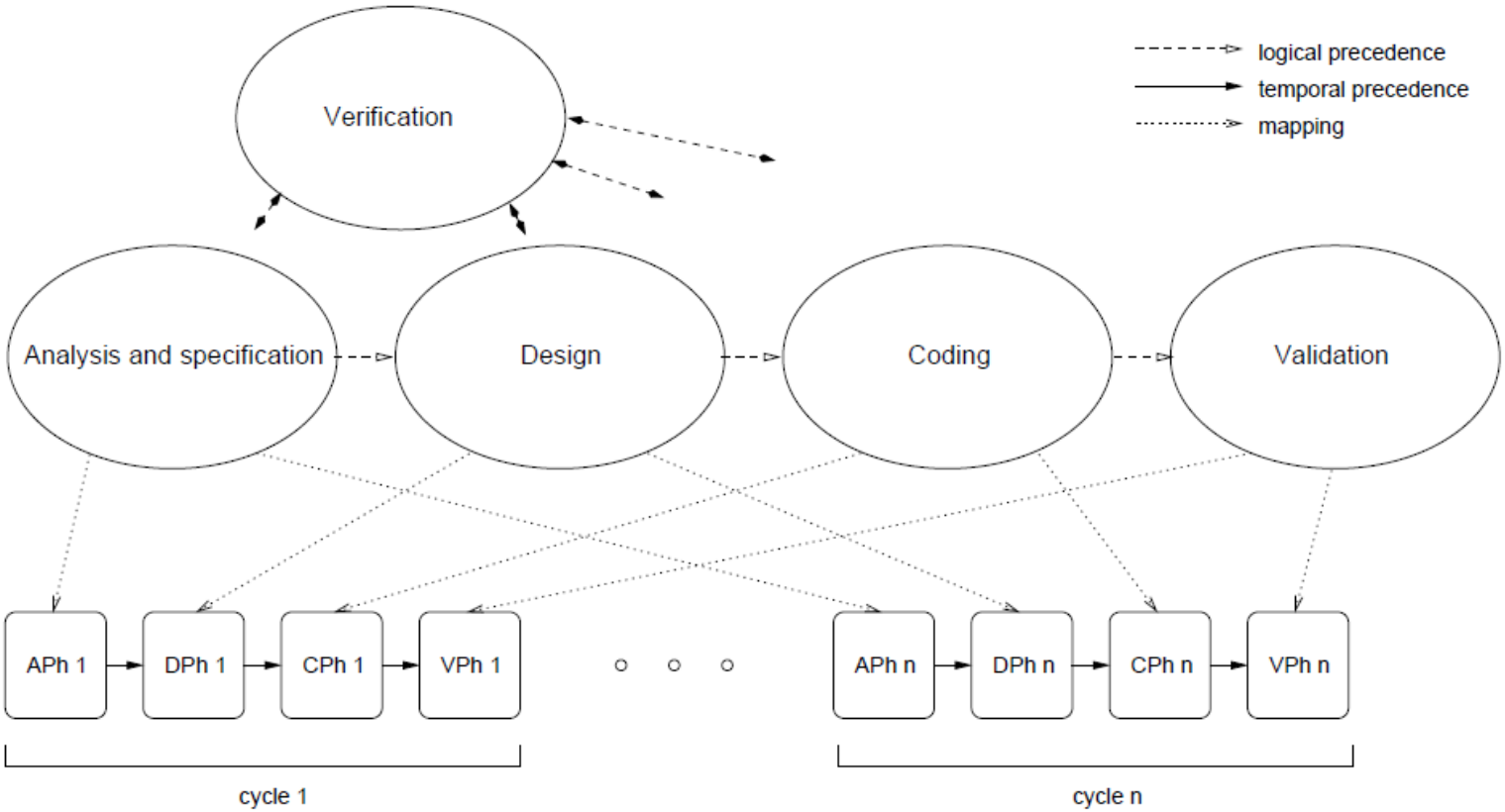
The V-model (as in IAEA TRS 384)

Il modello a V



La famiglia di processi a cascata associa ciascuna attività a una fase distinta

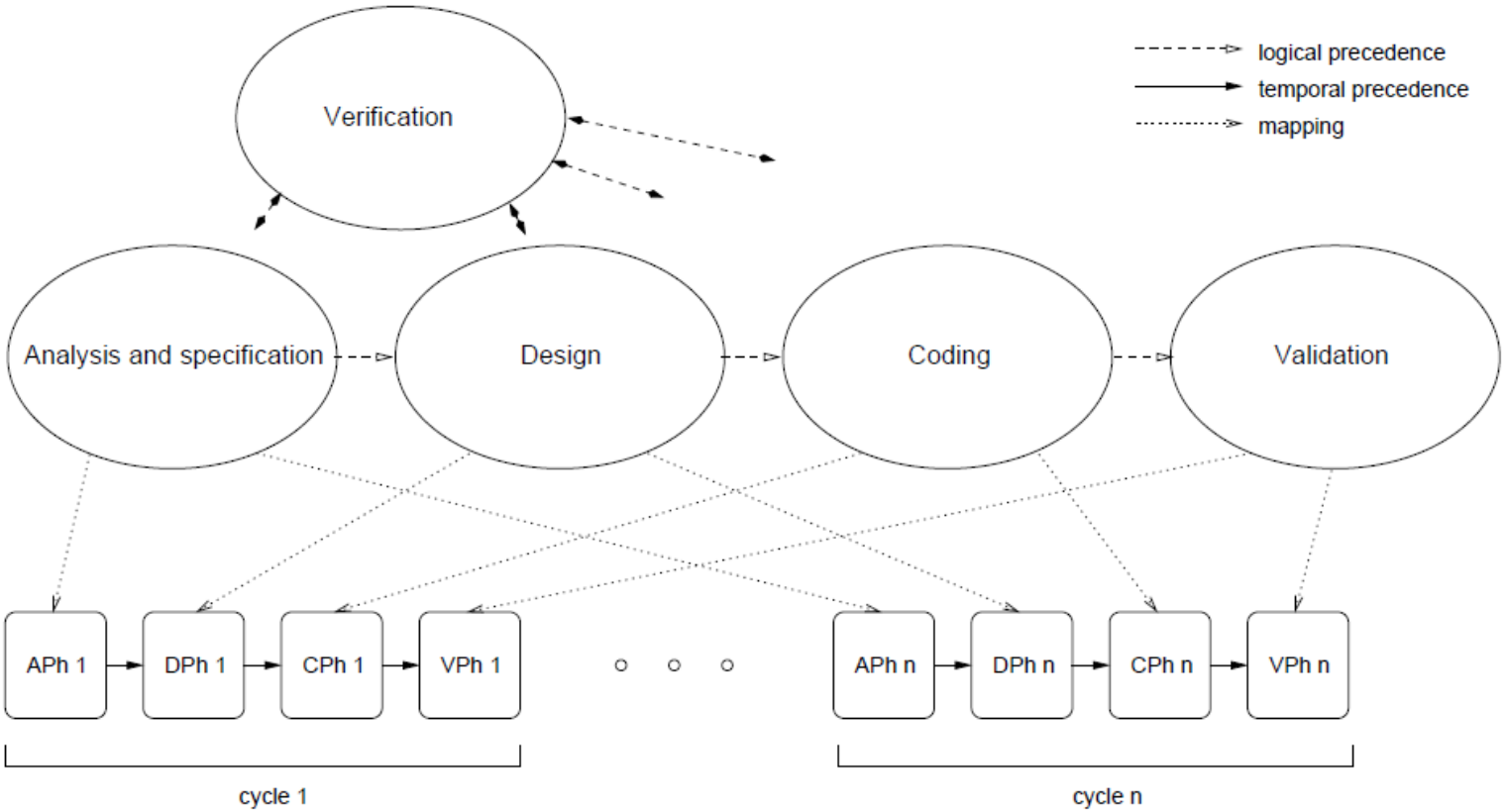
Il modello a V



Altre famiglie di processi, come i processi iterativi, sono più complessi mappature dalle attività alle fasi.

I processi a cascata, tuttavia, sono generalmente preferiti (o obbligatori). software con requisiti di sicurezza.

Processi evolutivi



- Rilevare tempestivamente la necessità di cambiamenti:
  - applicazioni o tecnologie nuove;
  - requisiti instabili;
- **sviluppo incrementale**.

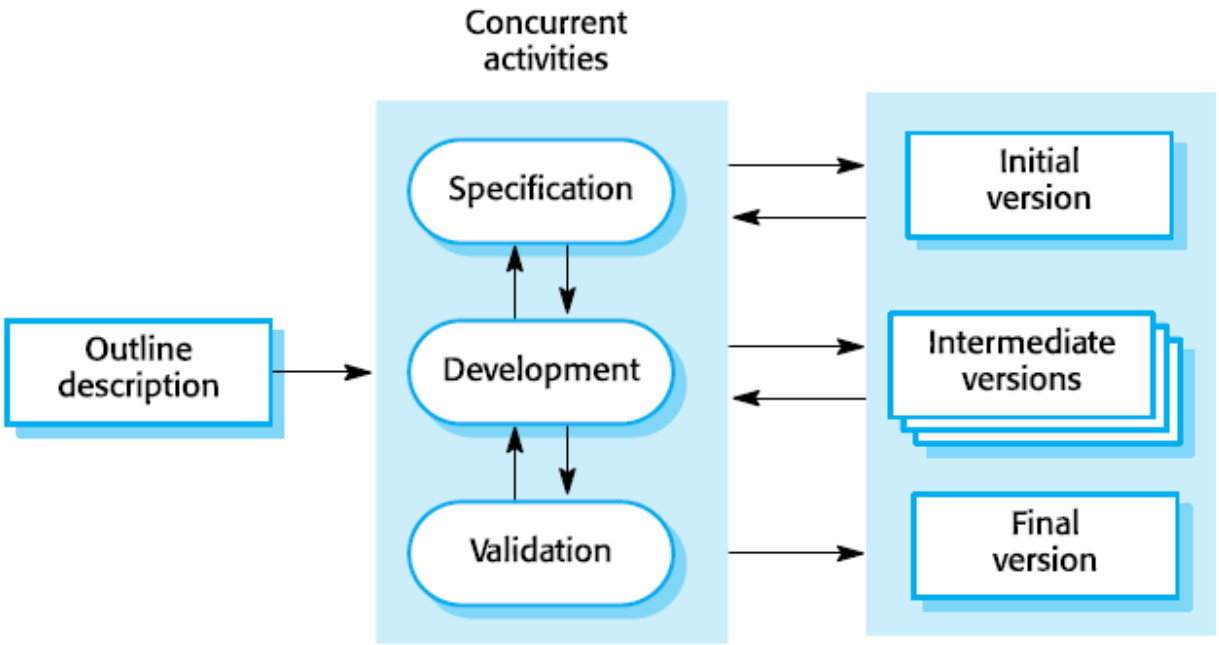
Processi evolutivi

Il processo evolutivo ha due varianti: esplorazione e Build and Fix..

- Sviluppo per esplorazione
  - Gli sviluppatori lavorano con i clienti
  - Dalle specifiche iniziali si arriva per mezzo di trasformazioni successive (evoluzione) fino al sistema software finale
  - Dovrebbe partire da requisiti ben chiari ed aggiungere nuove caratteristich definite dal cliente

- Sviluppo Build and Fix
  - Documentazione inesistente o quasi
  - Comprensione limitata del sistema da produrre
  - Costruire la prima versione e modificarla fino a che il cliente è soddisfatto
  - Fase di design pressoché inesistente
  - Codice prodotto di bassa qualità

Processi evolutivi



Processi evolutivi

- Problemi
  - Tempi lunghi
  - Sistemi difficilmente comprensibili e modificabili, probabilmente non corretti
  - Mancanza di visione d'insieme del progetto
- Applicabilità
  - Sistemi di piccole dimensioni
  - Singole parti di sistemi grandi (es. interfaccia utente)
  - Sistemi con vita breve (es. prototipi)

Processi evolutivi: lo Unified Process

- l'arco temporale del processo di sviluppo è suddiviso in quattro fasi successive;
- ogni fase ha un obiettivo e produce un insieme di semilavorati chiamato milestone ("pietra miliare");
- ogni fase è suddivisa in un numero variabile di iterazioni;
- nel corso di ciascuna iterazione possono essere svolte tutte le attività richieste (analisi, progetto. . .), anche se, a seconda della fase e degli obiettivi dell'iterazione, alcune attività possono essere predominanti ed altre possono mancare;
- ciascuna iterazione produce una versione provvisoria (baseline) del prodotto, insieme alla documentazione associata.

Attività (workflow)

- raccolta dei requisiti (requirement capture);
  - modello dei casi d'uso;
- analisi (analysis);
  - modello di analisi;
- progetto (design);
- implementazione (implementation);
- collaudo (test).

Fasi

- **Inizio (inception)**:
  - studio di fattibilità;
  - analisi dei rischi di varia natura (tecnica, economica, organizzativa. . .)
  - prototipi
  - modello dei casi d'uso: descrizione sintetica delle possibili interazioni degli utenti col sistema, espressa mediante la notazione UML;
- **Elaborazione (elaboration)**:
  - estendere e perfezionare i risultati della fase precedente
  - baseline architetturale eseguibile: prima versione eseguibile anche se parziale;
  - non è un prototipo, ma una base per lo sviluppo successivo.
  - milestone: codice della baseline, modello UML, versioni aggiornate dei documenti.

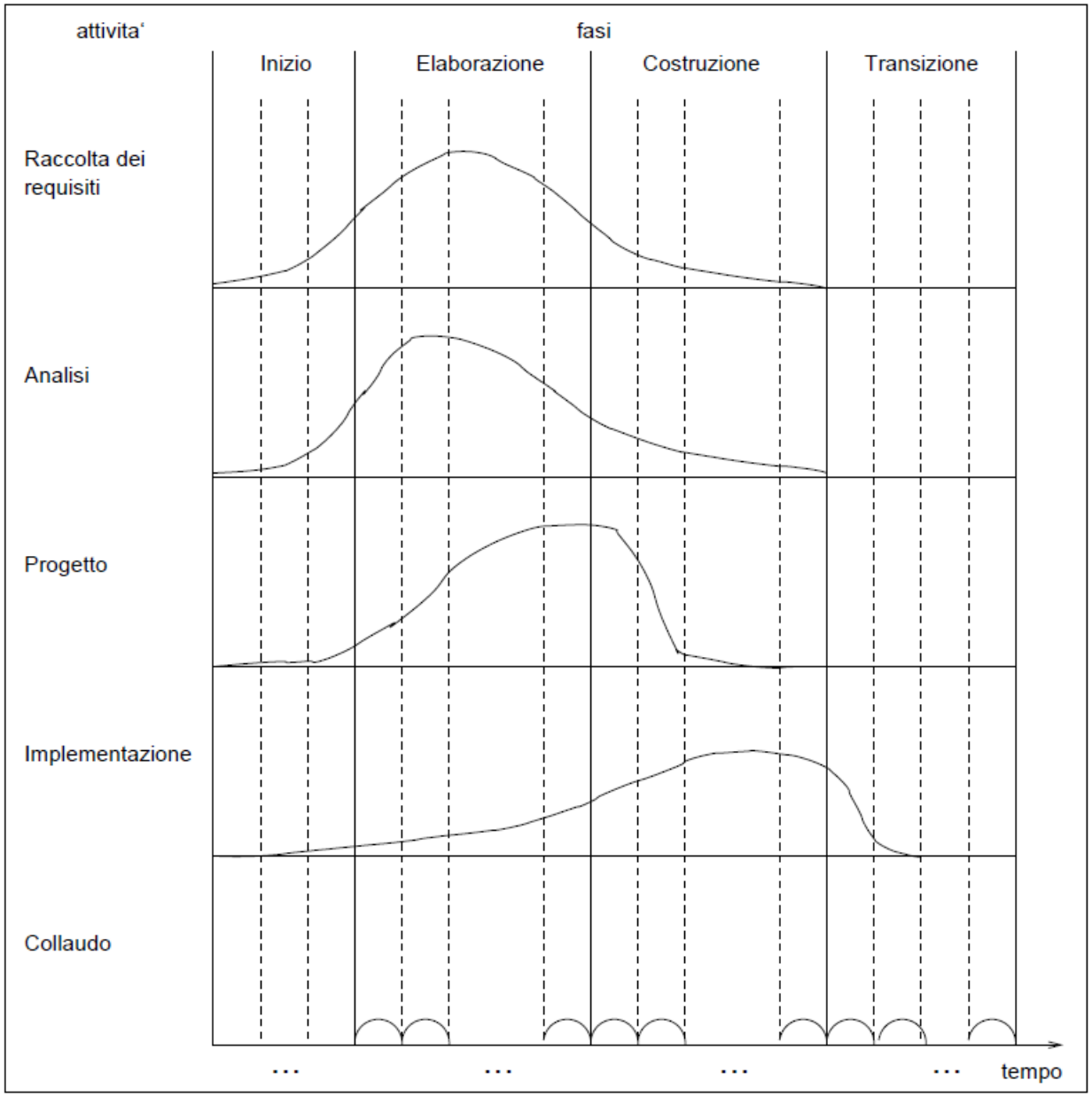
Fasi

- **Costruzione (construction)**:
  - produce il sistema finale, partendo dalla baseline architetturale;
  - completa le attività di raccolta dei requisiti, analisi e progetto;
  - milestone: sistema completo, documentazione in UML, test suite, manuali utente. . . ;
  - beta-test.
- **Transizione (transition)**:
  - correzione degli errori trovati in corso di beta-test, consegna e messa in opera;
  - milestone: versione definitiva del sistema, manuali utente, piano di assistenza tecnica.

Distribuzione delle attività nelle fasi

- **Inizio**:
  - raccolta e analisi dei requisiti;
  - progetto di architettura ad alto livello, non eseguibile;
  - implementazione, se usato un prototipo.
- **Elaborazione**:
  - sforzo maggiore per raccolta e analisi dei requisiti, decrescente verso la fine;
  - sforzo maggiore per elaborazione ed implementazione, crescente verso la fine.
- **Costruzione**:
  - progetto ed implementazione;
  - raccolta e analisi dei requisiti, per aggiornamenti dei requisiti.
- **Transizione**:
  - progetto ed implementazione per correggere errori.

Processi evolutivi: lo Unified Process



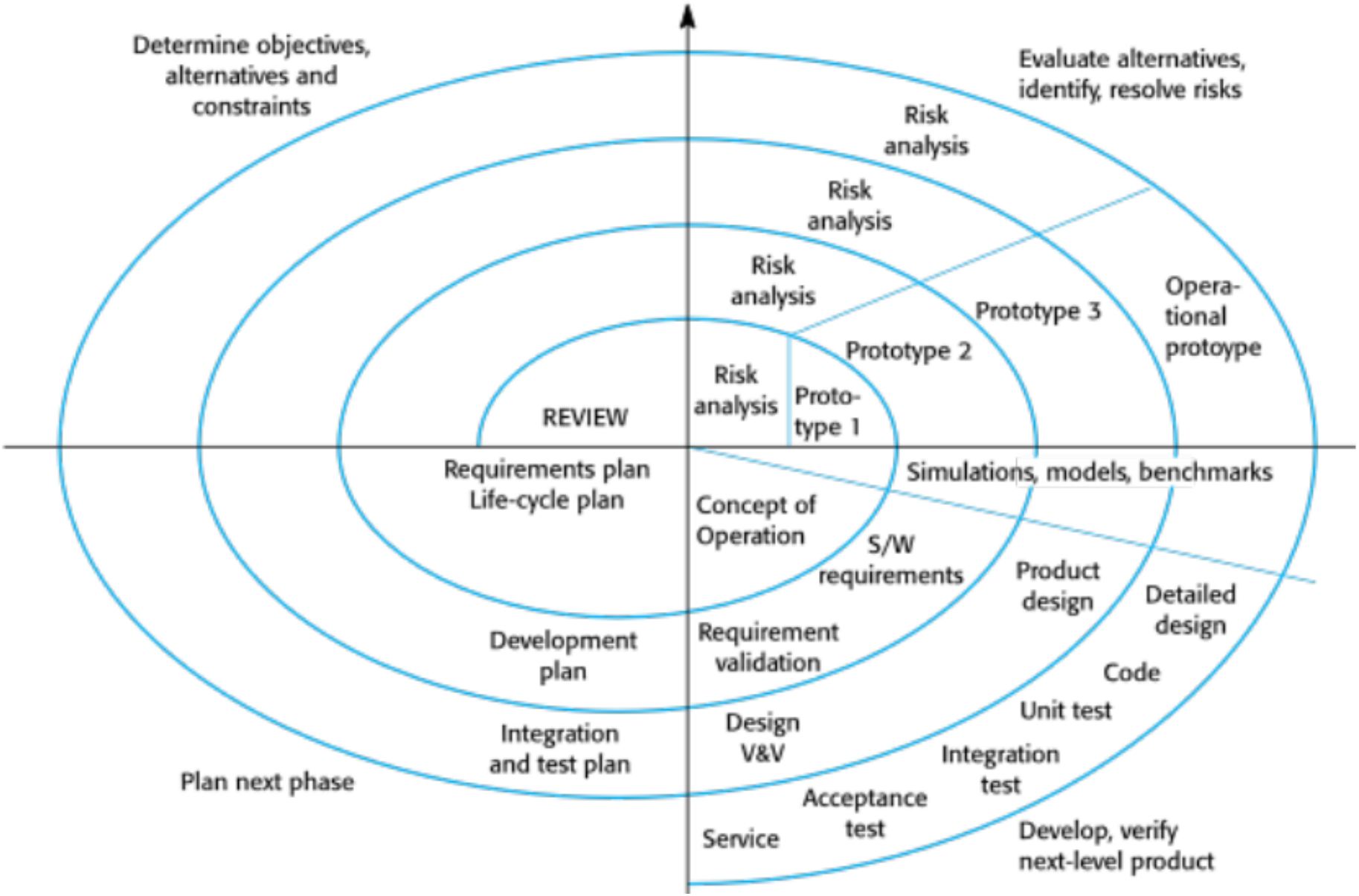
Altri processi

- Processo di Sviluppo Incrementale
  - Sono implementate prima le funzionalità di base (o prioritarie)
  - Al codice sviluppato in precedenza è aggiunto altro codice per un altro gruppo di funzionalità
  - Si ripete il passo precedente, fino a completamento
- Modelli trasformazionali
  - Modelli formali;
  - trasformazioni successive dal modello di analisi ad un modello di progetto dettagliato;
  - generazione automatica di codice dal modello dettagliato;
  - analogia con soluzione di equazioni: un processo è un'espressione che rappresenta un'insieme di sequenze di azioni;
- Processo CBSE o basato su COTS
  - COTS = componenti esistenti (Components Off The Shelf)
  - Analisi dei componenti esistenti
  - Modifica dei requisiti (?)
  - Progettazione tramite riuso
  - Sviluppo ed integrazione

A Spirale [Boehm 1988]

- Focalizza su tanti **prodotti parziali** (sottosistemi funzionali)
- Ogni ciclo (loop) della spirale è una fase (es. ciclo per requisiti)
- Ogni ciclo consiste dei seguenti settori
  - **Identificazione obiettivi specifici** per la fase corrente
  - **Valutazione rischi** del progetto
    - Rischio: qualcosa che può impedire il successo e che è sconosciuta
    - Successo: soddisfare tutti i requisiti. Attributi del rischio: (i) probabilità di occorrenza; (ii) impatto sul progetto (ovvero gravità, danno peggiore)
- **Produzione** di una parte e convalida della parte
- **Revisione** del progetto e **pianificazione** fase successiva
- Processo agile
  - + Poco tempo per la prima versione del prodotto
  - + Opportunità di interagire con il cliente
- Ogni fase produce un codice testato e integrato nel sistema complessivo

A Spirale [Boehm 1988]



A Spirale [Boehm 1988]

- Vantaggi
  - Concentra l'attenzione sulle possibilità di riuso
  - Concentra l'attenzione sull'eliminazione di errori
  - Pone al centro gli obiettivi
  - Integra sviluppo e mantenimento
  - Costituisce un framework di sviluppo hardware/software
- Limiti
  - Per contratto di solito si specifica a priori il modello di processo e i "deliverables"
  - Richiede esperienza nella valutazione dei rischi
  - Richiede raffinamenti per un uso generale

Il Manifesto agile (http://agilemanifesto.org):

Persone e interazioni	sono più importanti dei processi e degli strumenti;
Un software funzionante	è più importante della documentazione;
Collaborare con i clienti	è più importante del contratto;
Aderire ai cambiamenti	è più importante che aderire al progetto.

Ovvero, fermo restando il valore (e quindi la necessità) delle entità a destra, consideriamo più importanti le entità a sinistra.

I processi agili

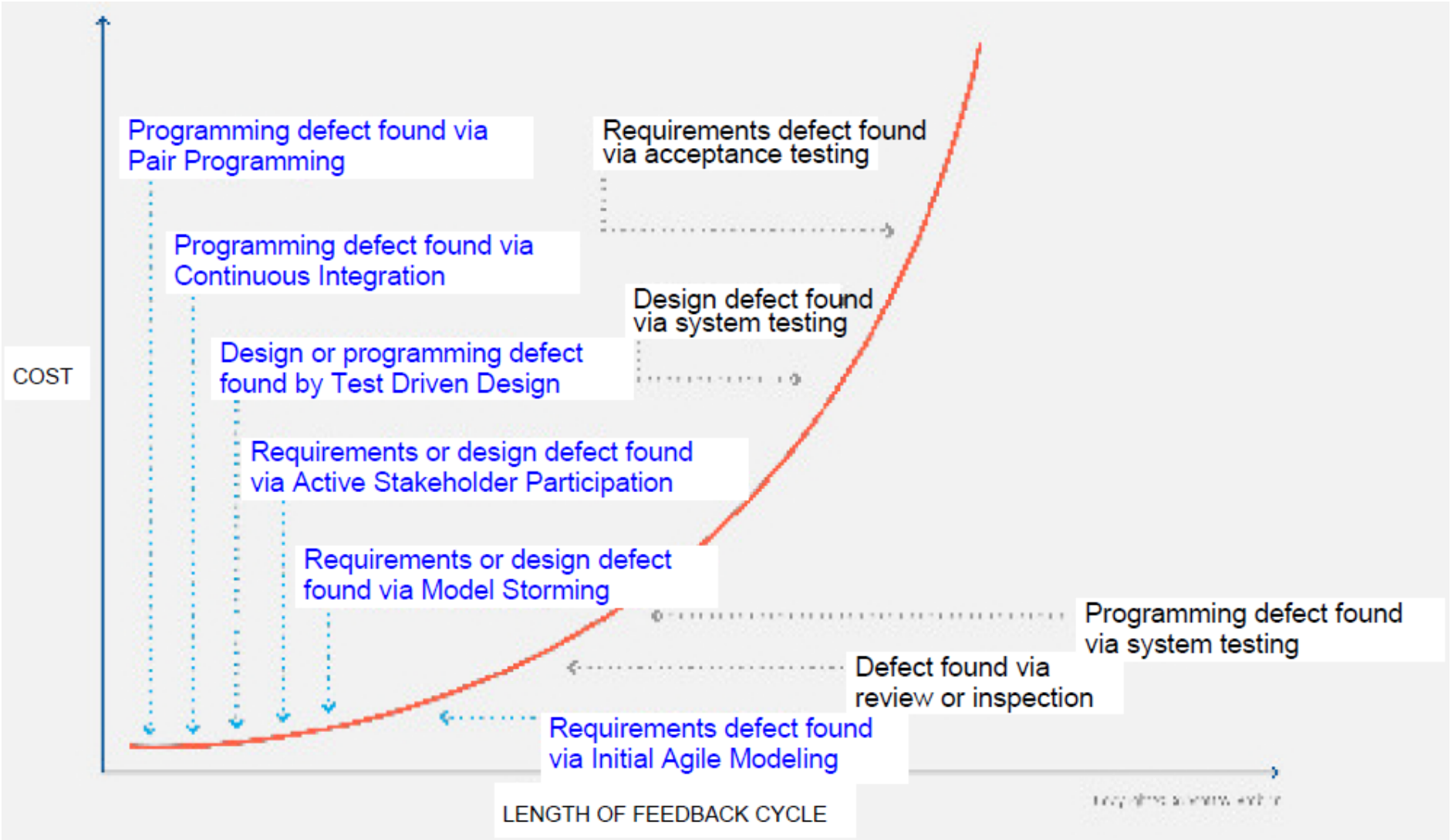
- Soddisfazione del cliente;
- Accogliere favorevolmente i cambiamenti anche se si è avanti con lo sviluppo;
- Usare scale temporali brevi (feedback piú rapido possibile);
- Eccellenza tecnica e buon design;
- Team che si auto regolano e persone motivate;
- Lavorare insieme quotidianamente (giocare per vincere);
- Semplicità del progetto e software funzionante;
- Iterazioni regolari e valutazione dell'attività (Sprint: 2 Settimane);
- Fornire a chi lavora al progetto l'ambiente, il supporto e la fiducia necessari al completamento del lavoro.

I processi agili

Principali metodologie agili:

- Extreme Programming (XP);
- SCRUM;
- CRYSTAL;
- LEAN;
- KANBAN;
- FDD.

Costo dei difetti in funzione del tempo di identificazione:



Sviluppo agile

Sono più importanti auto-organizzazione, collaborazione, comunicazione tra membri del team e adattabilità del prodotto rispetto ad ordine e coerenza delle attività del progetto

- Privilegiare
  - Individui rispetto a processi e strumenti
  - Disponibilità di software funzionante rispetto alla documentazione
  - Collaborazione con il cliente rispetto alla negoziazione dei contratti
  - Pronta risposta ai cambiamenti rispetto all'esecuzione di un piano
- Agilità
  - Considerare positivamente le richieste di cambiamento anche in fase avanzata di sviluppo
  - Fornire release del sistema software funzionante frequentemente
  - Costruire sistemi software con gruppi di persone motivate
  - Continua attenzione all'eccellenza tecnica

Extreme Programming (XP) [Beck 2000]

- Approccio basato sullo sviluppo e la consegna di piccoli incrementi di funzionalità
  - Solo 2 settimane per lo sviluppo degli incrementi
  - Piccoli gruppi di sviluppatori (da 2 a 12 persone)
  - Costante miglioramento del codice
  - Poca documentazione: uso di Story Card e CRC (Class Responsibility Collabor)
  - Enfasi su comunicazione diretta tra persone
  - Iterazioni corte e di durata costante
  - Coinvolgimento di sviluppatori, clienti e manager
  - Testabilità dei prodotti e prodotti testati sin dall'inizio
- Adatto per progetti in cui
  - I requisiti non sono stabili, XP è fortemente adattativo
  - I rischi sono grandi, es. tempi di consegna brevi, software innovativo per gli sviluppatori

Extreme Programming (XP)

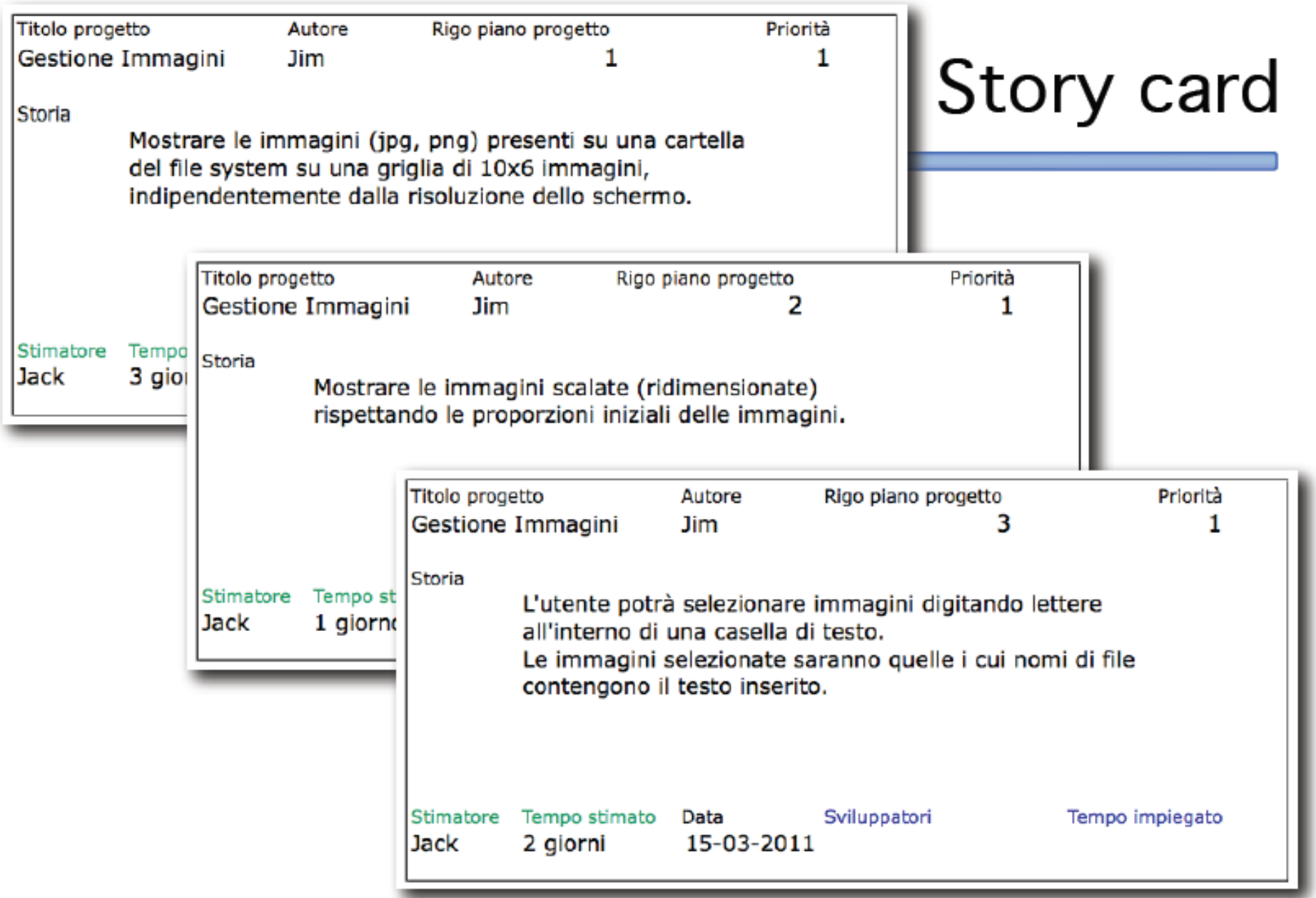
- Princìpi di XP
  - Avere feedback rapidamente
  - Assumere la semplicità
  - Cambiamenti incrementali
  - Supportare i cambiamenti
  - Produrre lavoro di qualità
- Libro Consigliato
  - Bec k. Extreme Programming Explained. Addison-Wesley
- Siti web
  - www.xprogramming.com
  - www.extremeprogramming.org

12 Pratiche di XP [Beck]

- Gioco di pianificazione
- Piccole release
- Metafora
- Testing
- Refactoring
- Pair Programming (programmazione a coppie)
- Cliente in sede
- Design semplice
- Possesso del codice collettivo
- Integrazione continua
- Settimana di 40 ore
- Usare gli standard per il codice

Story Card

- Storie utente (story card) = casi d'uso leggeri
- Dimensioni card 5" x 3" circa 12x7 cm
- Descrizione storie: 2-3 frasi su una card che
  - Sono importanti per il cliente e sono scritte dal cliente
  - Possono essere testate
  - Permettono di ricavare una stima del loro tempo di sviluppo
  - Possono essere associate a priorità
- Template per story card (ovvero campi di una story card)
  - Data, Numero, Priorità, Tempo stimato, Riferimenti
  - Descrizione requisito
  - Lista di task per ciascun requisito, ovvero ciò che lo sviluppatore dovrà fare
  - Note


Titolo progetto
Gestione Immagini
Autore
Jim
Rigo piano progetto
1
Priorità
1
Storia
Mostrare le immagini (jpg, png) presenti su una cartella del file system su una griglia di 10x6 immagini, indipendentemente dalla risoluzione dello schermo.
Titolo progetto
Gestione Immagini
Autore
Jim
Rigo piano progetto
2
Priorità
1
Storia
Mostrare le immagini scalate (ridimensionate) rispettando le proporzioni iniziali delle immagini.
Titolo progetto
Gestione Immagini
Autore
Jim
Rigo piano progetto
3
Priorità
1
Storia
L'utente potrà selezionare immagini digitando lettere all'interno di una casella di testo.
Le immagini selezionate saranno quelle i cui nomi di file contengono il testo inserito.
Stimatore
Jack
Tempo stimato
2 giorni
Data
15-03-2011
Sviluppatori
Tempo impiegato
Story card

Story Board

Prima settimana

To do

Done

Seconda settimana

To do

Terza settimana

To do

XP: Gioco di pianificazione

- Gli utenti (clienti) scrivono le storie (sono i requisiti)
- Gli sviluppatori stimano il tempo per lo sviluppo di ciascuna storia
  - Se le storie sono troppo complesse da stimare, ritornare dal cliente e far dividere le storie
- Gli utenti dividono, fondono e assegnano priorità alle storie
  - Riempiono 3 settimane scegliendo le storie
  - Non preoccuparsi delle dipendenze
- Gli addetti al business prendono decisioni su
  - Date per le release, contesto, priorità dei task
- Pianificare l'intera release (grossolanamente) e la nuova iterazione
  - Non pianificare troppo in avanti
- Per l'attuale release, gli sviluppatori:
  - Dividono ciascuna storia in task, stimano i task, ciascuno si impegna per realizzare un task
  - Vengono svolti prima i task più rischiosi

XP: Piccole release

- Rendere ogni release il più piccola possibile
  - Tempo di sviluppo della release 2 settimane
- Effettuare un design semplice e sufficiente per la release corrente
- Piccole release forniscono agli sviluppatori
  - Feedback rapidamente
  - Un senso di: “ho ottenuto qualcosa di valido”
  - Rischio ridotto
  - La fiducia del cliente
  - Possibilità di fare aggiustamenti per requisiti che cambiano

XP: Metafora

- Guidare il progetto con una singola metafora
  - Es.: “La UI è un desktop”
  - Deve rappresentare l’architettura
    - Rende le discussioni più semplici
  - Il cliente deve essere a suo agio con essa

XP: Design Semplice

- Il giusto design per il software si ha quando
  - Passa i test
  - Non ha parti duplicate
  - Esprime ciascuna intenzione importante per i programmatori
  - Ha il numero più piccolo di classi e metodi
- Non preoccuparsi di dover apportare cambiamenti dopo
  - Fare la cosa più semplice che può funzionare
  - Paga quanto usi
- Usare le CRC card (Class Responsibility Collaboration) per documentare il design
  - Permettono di ragionare meglio in termini di oggetti
  - Contribuiscono a fornire una visione complessiva del sistema

CRC card

Class: ImgBrowser	
Responsibility	Collaboration
Mostra un frame con due pannelli: uno con le immagini e l'altro con i controlli	ImgPanel CtrlPanel JFrame (superclasse)

Class: ImgPanel	
Responsibility	Collaboration
Visualizza le immagini Risiede sulla parte superiore della finestra Calcola le proprie dimensioni	Cartella Img JPanel (superclasse)

Class: Cartella	
Responsibility	Collaboration
Legge i file immagine da una cartella Filtra i file in base ad un criterio di selezione Tiene il numero di immagini presenti Tiene il numero di immagini selezionate	Img File (da java.io)

XP: Testing

- Si testa tutto ciò che potenzialmente può andar male, per tutto il tempo
  - Si eseguono i test più volte al giorno, non appena è stato prodotto del nuovo codice
- I test sono la specifica dei requisiti!
  - Una specifica in formato eseguibile!
- Due tipi di test
  - Test funzionali
  - Unit test

XP: Test

- Test funzionali
  - Scritti dall'utente (punto di vista dell'utente)
  - Effettuati da: utenti, sviluppatori e team di testing
  - Automatizzati
  - Eseguiti almeno giornalmente
  - Sono una parte della specifica dei requisiti, quindi documentano i requisiti
- Unit test
  - Scritti dagli sviluppatori (punto di vista del programmatore)
  - Scritti prima della codifica (TDD, Test Driven Development) ed anche dopo la codifica
  - Supportano design, codifica, refactoring e qualità

XP: Pair Programming

- Programmatori esperti e motivati
- Ruolo di uno dei partner
  - Usa il mouse e la tastiera
  - Pensa al miglior modo di implementare il metodo
- Ruolo dell'altro
  - L'approccio funzionerà?
  - Pensa ai test
  - Potrebbe essere fatto più semplicemente?
- Scambio dei partner
- Pair programming aiuta la disciplina, sparge la conoscenza sul sistema

XP: Possesso del codice collettivo

- Chiunque può aggiungere qualunque codice su qualunque parte del sistema
- Unit test proteggono le funzionalità del sistema
- Chiunque trova un problema lo risolve
- Ciascuno è responsabile per l'intero sistema

XP: Integrazione continua

- Integrazione del codice testato ogni poche ore (max un giorno)
- Tutti gli unit test devono essere superati
- Se un test fallisce la coppia che ha prodotto il codice deve ripararlo
- Se non può ripararlo, buttare il codice e ricominciare

XP: 40 ore a settimana

- Se per te non è possibile fare il lavoro in 40 ore, allora hai troppo lavoro
- 40 ore a settimana ti lasciano "fresco" per risolvere i problemi
- Previene l'inserimento di errori difficili da trovare
- Pianificazioni frequenti evitano a ciascuno di avere troppo lavoro
- Ore extra di lavoro è sintomo di un problema serio

XP: Cliente sul sito

- Scrive i test funzionali
- Stabilisce priorità e fornisce il contesto per le decisioni dei programmatori
- Risponde alle domande
- Porta avanti il suo proprio lavoro
- Se non puoi avere il cliente sul sito, forse il progetto non è così importante?

XP: Standard di codifica

- Costruzioni complicate (per il design) non sono permesse
  - Mantenere le cose semplici
- Il codice appare uniforme
  - Più facile da leggere
- Usare tutti la stessa convenzione, così non si ha necessità di riformattare il codice, sapere come usare
  - Spazi e Tab per indentazione
  - Posizione parentesi graffe
  - Scelta di nomi classi, metodi, attributi
  - Posizione commenti

Vedere convenzioni per linguaggio Java!
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

XP: Refactoring

- Refactoring significa migliorare la struttura del codicesenza influenzarne il comportamento
- Fatto in piccoli passi
- Supportato dagli unit test, design semplice e pair programming
- Puntare a codice senza ripetizioni
- Refactoring fatto in coppia dà più coraggio

XP: In breve

- XP focalizza sul codice
  - Fare solo le cose che sveltiscono la produzione del codice
  - Codifica e test
- XP si orienta sulla gente
  - La conoscenza del sistema è trasferita attraverso la comunicazione tra la gente
- XP è leggero
  - Rimuovere i costi aggiuntivi
  - Creare prodotti di qualità tramite test rigorosi
- I princìpi di XP non sono nuovi

Scrum

- La prima guida su Scrum è stata prodotta nel 2010 da Schwaber e Sutherland
- Scrum è una struttura di base per aiutare le persone a generare soluzioni per problemi complessi
- Scrum promuove un ambiente in cui
  - Un **Product Owner** (proprietario) elenca il lavoro da svolgere su un problema complesso tramite un **Product Backlog** (lavoro arretrato)
  - Lo **Scrum Team** (sviluppatori) trasforma una parte del lavoro in un incremento durante uno **Sprint**(iterazione)
  - Lo **Scrum Team** e chi ha interesse nel prodotto (stakeholder) valutano i risultati e fanno aggiustamenti per il prossimo **Sprint**
  - Si ripete quanto sopra

Principi di Scrum

- Scrum è semplice, e incompleto. Vari processi e tecniche possono essere usate sulla struttura di Scrum
- Scrum è basato sull'empirismo e sul concetto di lean
  - **Empirismo**: la conoscenza viene dall'esperienza e le decisioni sono prese in accordo a quel che è osservato
  - **Lean** (snello, agile): ridurre lo spreco e concentrarsi sull'essenziale
- Scrum impiega un approccio iterativo e incrementale per ottenere prevedibilità e controllo del rischio

Pilastri di Scrum

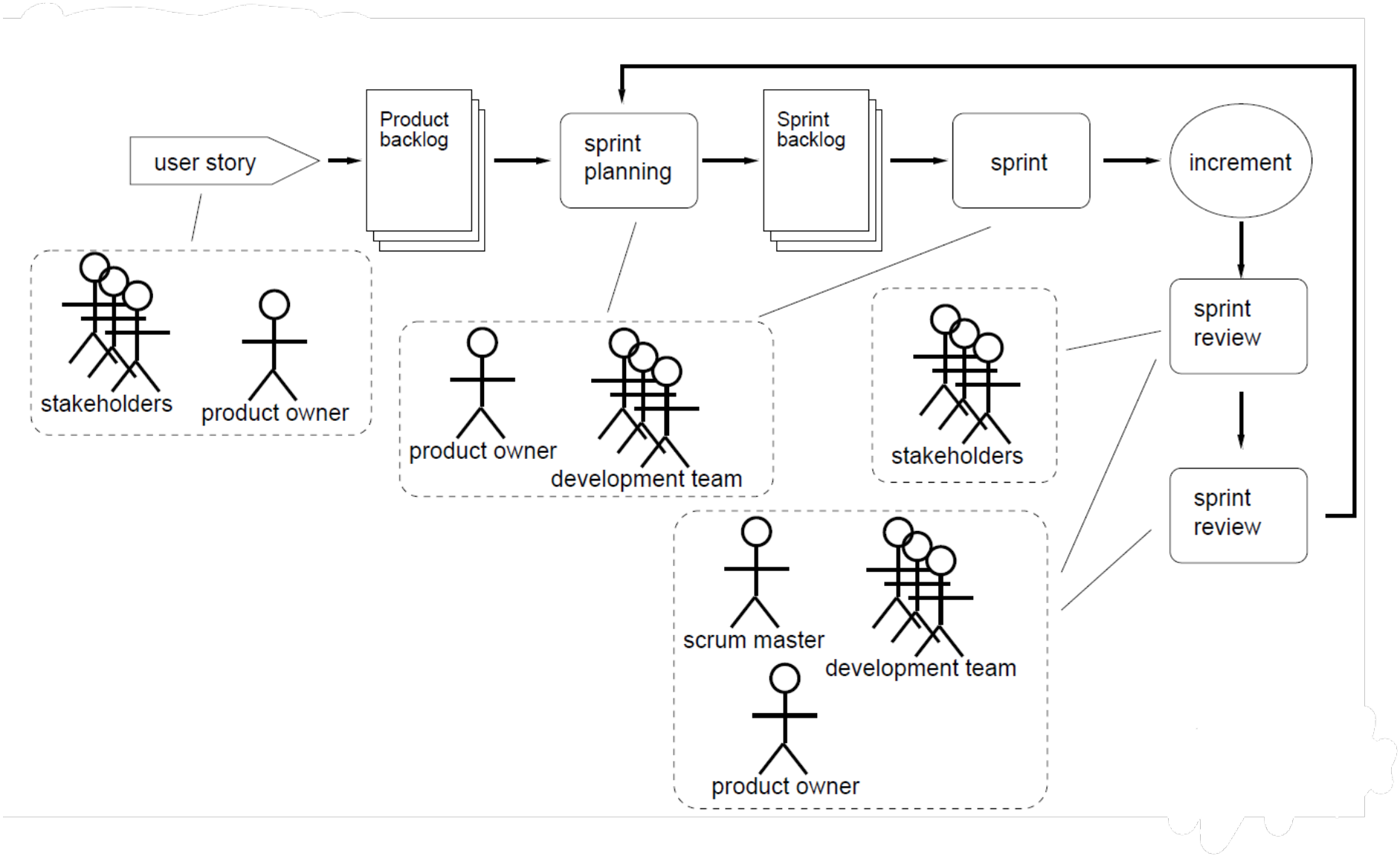
- Scrum si sostiene su tre pilastri: trasparenza, ispezione e adattamento

- **Trasparenza**: il processo e il lavoro devono essere visibili a chi svolge il lavoro e a chi riceve i risultati. Le decisioni importanti sono basate sui tre **artefatti** (Product backlog, Sprint backlog, Incremento). La trasparenza abilita l'ispezione
- **Ispezione**: gli artefatti di Scrum e il progresso verso l'obiettivo devono essere ispezionati frequentemente e in modo diligente per scoprire potenziali problemi. Scrum fornisce la cadenza delle ispezioni tramite cinque **eventi**. L'ispezione permette l'adattamento
- **Adattamento**: se qualche aspetto del processo o se il risultato diventano inaccettabili allora il processo o ciò che è prodotto deve essere aggiustato. L'aggiustamento si attua subito per evitare ulteriori divergenze

Eventi di Scrum

- Gli **Sprint** (iterazioni) hanno lunghezza fissata, tipicamente un mese
- Lo **Sprint Planning** produce la pianificazione del lavoro da svolgere nello Sprint
- **Daily Scrum** serve a ispezionare il progresso e adattare lo **Sprint Backlog**. E' un evento di 15 minuti, svolto ogni giorno nello stesso posto. Migliora la comunicazione, identifica ostacoli, promuove il processo decisionale
- **Sprint Review** ispeziona i risultati dello sprint e determina gli adattamenti. E' il penultimo evento della Sprint. Dura al massimo 4 ore (per una Sprint di un mese)
- **Sprint Retrospective** ha lo scopo di migliorare qualità e efficacia. Il team ispeziona come è andata l'ultima Sprint riguardo gli individui, le interazioni, i processi, i tool, e la Definition of Done. La **Definition of Done** è una descrizione dello stato dell'incremento quando esso soddisferà le misure di qualità richieste per il prodotto

Il processo I'm Agicle



Il processo I'm Agicle

Il processo illustrato si basa sulle metodologie Scrum ed XP.

Ogni ciclo (sprint) produce un incremento.
La durata (massima?) di ogni ciclo è due settimane

- **sprint planning**: scelta degli obiettivi (dal backlog) da realizzare con l'incremento;
- **sprint review**: presentazione dell'incremento agli stakeholder;
- **sprint (development)**: realizzazione dell'incremento;
- **sprint retrospective**: discussione dello sprint concluso per preparare lo sprint successivo.

I partecipanti:

- **stakeholders**: rappresentanti del committente;
- **product owner**: rappresentante del committente che partecipa a tutte le attività insieme agli sviluppatori;
- **scrum master**: facilitatore delle comunicazioni nel team e fra team e stakeholder, partecipa alle attività di valutazione dello sprint.

Il processo I'm Agicle

Gli artefatti:

- **user story**: raccolta di documenti simili a requisiti o casi d'uso, ma diversi;
  - Independent (per quanto possibile)
  - Negotiable (Non sono contratti, possono essere aggiunte, rimosse modificate, unite, divise)
  - Valuable (Devono avere valore per l'utente, l'utente in genere non è il committente)
  - Estimable (Per gli sviluppatori è importante poter stabilire la dimensione della storia)
  - Small (Sufficientemente piccola da poter rientrare in un'iterazione del team da 1 a 10 gg)
  - Testable (Devono essere forniti i criteri di accettazione della storia)
- **product backlog**: lista di user story da realizzare o bug da eliminare;
- **sprint backlog**: lista di elementi del product backlog scelti come obiettivi dello sprint corrente.