

# Analisi e Progettazione del Software

UniShare

Davide Cozzi  
@dlcgold

Gabriele De Rosa  
@derogab

Federica Di Lauro  
@f\_dila

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Introduzione all’Ingegneria del software</b>	<b>3</b>
2.1	Modelli di Processo . . . . .	7
2.2	Modello UP . . . . .	15
2.2.1	Modello UP o RUP . . . . .	15
2.2.2	Processo Scrum . . . . .	20
<b>3</b>	<b>Analisi dei Requisiti e Casi d’uso</b>	<b>25</b>
3.1	Diagrammi dei casi d’uso . . . . .	35

# Capitolo 1

## Introduzione

Questi appunti sono presi durante le lezioni in aula. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: <https://github.com/dlcgold/Appunti>. Grazie mille e buono studio!

# Capitolo 2

## Introduzione all'Ingegneria del software

Durante il corso di Analisi e Progettazione del software si analizzano i modelli e i principi per lo sviluppo di un software mantenibile, andando anche ad analizzare, in maniera sommaria, anche tutti gli strumenti di ingegneria del software, necessari per lo sviluppo ottimale di un software.

In questo corso studieremo in dettaglio i seguenti argomenti:

- introduzione all'ingegneria del software
- progettazione sistemi orientati ad oggetti
- modellazione a dominio
- UML e analisi dei casi d'uso
- design pattern
- sviluppo test-driven(cenni)
- code smell e refactoring(cenni)

Il software, è l'insieme delle componenti modificabili e non fisiche di un calcolatore, viene diviso in due categorie:

**generici** , per un ampio range di clienti, come ad esempio gli elaboratori di testi

**custom** , per un singolo cliente, come ad esempio i gestionali specifici di un impresa

Questa differenza si sta sempre più assottigliando in quanto recentemente varie aziende stanno sviluppando un software generico, che viene poi adattato in base alle esigenze del singolo cliente, come ad esempio i software Oracle e Sas.

Nello sviluppo di software si utilizza spesso una base pre-esistente, infatti l'ingegneria del software si occupa di tutti gli aspetti per lo sviluppo del software, sfruttando di solito una base preesistente.

Di solito quando si sviluppa un software si presuppone che abbia una durata di alcuni anni, con la predisposizione al cambiamento e all'introduzione di nuove features infatti un software per essere utile deve essere continuamente cambiato.

Si hanno due tipologie di progetti:

1. **progetti di routine**, con soluzione di problemi e riuso di vecchio codice
2. **progetti innovativi**, con soluzioni nuovi

e solitamente l'ingegneria del software si occupa principalmente di progetti innovativi con specifiche del progetto variabili, con la presenza di cambiamenti continui e ovviamente non è uguale nè simile con l'ingegneria tradizionale.

Un programmatore normalmente lavora da solo, su un programma completo con specifiche note mentre un ingegnere del software lavora in gruppo, progetta componenti e l'architettura e identifica requisiti e specifiche. Il costo del software spesso supera quello hardware

Nello sviluppo di un software si hanno le seguenti fasi di sviluppo:

- **analisi dei requisiti**, che indica cosa deve fare il sistema
- **progettazione**, progetto del sistema d implementare
- **sviluppo**, produzione del sistema software
- **convalida**, verifica dei requisiti del cliente
- **evoluzione**, evoluzione al cambiare di requisiti del cliente

Esistono dei sistemi software per l'automazione delle attività svolte nel progetto software, i cosiddetti **CASE** (Computer-Aided Software Engineering) che si dividono in:

**CASE di alto livello** per il supporto alle prime attività di processo come la raccolta requisiti e la progettazione

**CASE di basso livello** per il supporto alle ultime attività di processo come la programmazione, il debugging, testing e reverse engineering

Nella figure X1 e X2 si hanno delle risposte alle comuni domande sull'ingegneria del software e le caratteristiche di un ottimo software, necessario per mantenerlo facilmente modificabile nel tempo.

Nello sviluppo di un software, qualsiasi tipologia esso sia, prevede i seguenti aspetti in comune:

**specifiche del software** vengono definite le specifiche e analizzato in dettaglio il comportamento e le funzionalità richieste da sviluppare

**sviluppo del software** viene effettivamente implementato e progettato il codice, attraverso i tools di sviluppo, rispettando le specifiche previste nella fase precedente.

**convalida del software** viene testato il software sviluppato nella fase precedente, al f

**evoluzione del software** viene mantenuto ed evoluto il software, al fine di riflettere i cambiamenti e delle funzionalità richieste dal cliente; è la fase più critica e costosa, dato che a volte la manutenzione richiede di sistemare e correggere gli errori e/o il cattivo design del progetto.

Il software deve essere accettato dagli utenti per i quali è stato sviluppato per cui deve essere comprensibile, usabile e compatibile con altri sistemi, oltre a dover essere mantenibile, affidabile ma soprattutto efficiente.

Esiste un codice etico, **ACM/IEEE**, con otto principi legati al comportamento degli ingegneri del software ed è fondamentale seguirlo dato che fornisce informazioni su come risolvere i problemi etici collegati a tutti i progetti software, come le informazioni da fornire all'esterno ed altri aspetti.

Un **sistema** è una collezione significativa di componenti interrelati che lavorano assieme per realizzare un obiettivo comune, quindi include software e parti meccaniche o elettriche; inoltre si ha che i vari componenti possono dipendere da altri componenti e che le proprietà e il comportamento dei vari componenti sono intrinsecamente correlati, quindi si hanno due macro categorie:

1. **sistemi tecnico-informatici**, in cui sono inclusi hardware e software ma non gli operatori e i processi operazionali, dove sono presenti le **proprietà emergenti**, dipendenti dalle sue componenti e le relazioni tra esse, misurabili soltanto sul sistema finale.  
Inoltre i sistemi tecnici sono non-deterministici, in quanto dato lo stesso

input non è detto che restituisca lo stesso output, in quanto il risultato è spesso dipendente dal comportamento degli operatori umani.

2. **sistemi socio-tecnici**, comprendente uno o più sistemi tecnici, assieme ai processi operazionali e agli operatori, sono fortemente condizionati da politiche aziendali e regole.

I sistemi Socio-tecnici sono sistemi pensati per raggiungere obiettivi aziendali o organizzativi e bisogna comprendere a fondo l'ambiente organizzativo nel quale un certo sistema è usato.

Vediamo qualche proprietà emergente:

- **volume:** il volume di un sistema varia a seconda di come sono disposti e collegati i componenti che lo formano.
- **affidabilità:** l'affidabilità del sistema dipende dall'affidabilità dei suoi componenti, ma interazioni impreviste possono produrre nuovi fallimenti e quindi influenzare l'affidabilità dell'intero sistema
- **protezione:** la protezione del sistema è una proprietà complessa che non può essere facilmente misurata, in quanto in futuro potrebbero essere inventati nuove modalità di accesso e/o attacco che rendono meno protetto il software.
- **riparabilità:** questa proprietà riflette la facilità con cui è possibile correggere un problema e ciò dipende dalla possibilità di diagnosticare il problema e soprattutto di accedere, modificare o sostituire i componenti difettosi.
- **usabilità:** questa proprietà mostra la facilità d'uso del sistema e dipende dai componenti del sistema tecnico, dai suoi operatori e dal suo ambiente operativo.

Oltre a questi sistemi software standard, sono presenti i **sistemi critici**, in cui fondamentalmente non sono ammessi errori e malfunzionamenti altrimenti causano dei gravi problemi se non morti, per questo le specifiche avvengono in linguaggio il più possibile formale possibile e usano un modello di sviluppo a cascata, inefficiente per gli altri sistemi software.

Abbiamo tre esempi di sistemi critici:

1. **sistemi safety-critical** dove i fallimenti comportano rischi ambientali o perdite di vite umane, come ad esempio un sistema di controllo per un impianto chimico.

2. **sistemi mission-critical** dove i fallimenti possono causare il fallimento di attività a obiettivi diretti, come un sistema di navigazione di un veicolo spaziale.
3. **sistemi business-critical** dove i fallimenti possono risultare in perdite di denaro sostenute, come ad esempio un sistema bancario.

I fallimenti di un software, sia che fossero piccoli che grandi, possono essere di diversi tipi:

- **fallimenti hardware**, errori di progetto, produzione o "consumo"
- **fallimenti software**, errori di specifica, progetto, implementazione
- **errori operativi**, errori commessi da operatori umani (forse una delle maggiori cause di fallimenti)

Nei sistemi critici, generalmente la fidatezza è la più importante proprietà del sistema in quanto si ha un alto costo dei fallimenti ed esso riflette il livello di confidenza che l'utente ha verso il sistema, cosa leggermente diversa dall'utilità, ossia la sicurezza di non avere problemi e/o intrusioni da parte di persone esterne.


Il costo dei fallimenti nei sistemi critici è così alto che i metodi di ingegneria del software non sono cost-effective per cui si utilizza un modello a cascata, in quanto si devono fare prima tutte le analisi, prima di poter implementare e testare il sistema e soprattutto si spende un sacco di soldi per il testing e le analisi per convalidare la fidatezza necessaria da raggiungere.

## 2.1 Modelli di Processo

Il modello di un processo software è una rappresentazione semplificata del processo, basata su un aspetto specifico:

- **modello a flusso di lavoro (Workflow)** per una sequenza di attività
- **modello a flusso di dati (Data-flow)** per il flusso delle informazioni
- **modello ruolo/azione (role/action)** per decidere i vari ruoli

si hanno poi tre modelli generici:

1. **modello a cascata (waterfall)**: modello in cui le diverse fasi, come si nota nella figura  Figura 2.1, avvengono in sequenza, dove i problemi e/o mancanze di interpretazioni si notano verso la fine dello sviluppo, nella



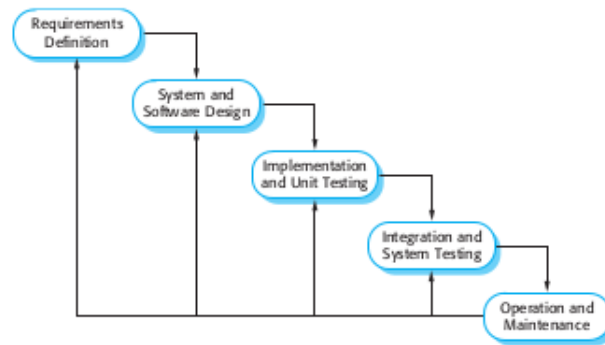


Figura 2.1: Modello a cascata fig:wf

fase di test e/o consegna, e ciò comporta una modifica di una o più fasi precedenti dello sviluppo, con notevoli aggravii di costi e tempestiche. È stato il primo modello di sviluppo, ma a partire dagli anni 90 si comprese che la maggioranza dei fallimenti dei software era da imputare al fatto di cercare di definire tutti i requisiti prima di sviluppare, cosa che è difficile da fare in maniera efficace in sistemi variabili nel tempo, come quasi tutti i sistemi software da sviluppare e/o già sviluppati. Vediamo una spiegazione più dettagliata:

- **Requirements analysis and definition:** si stabiliscono servizi, limiti e fini del software consultandosi col cliente
  - **System and software design:** si stabilisce un'architettura di sistema complessiva stabilendo l'hardware e il software necessario
  - **Implementation and unit testing:** si sviluppano le unità di test e si verifica che ogni parte del software risponda alle specifiche richieste presa singolarmente
  - **Integration and system testing:** si testa il software nella sua interezza, si verifica che il software rispetti ogni requisito e infine si consegna il prodotto al cliente
  - **Operation and maintenance:** una volta che il prodotto è stato consegnato si provvede a mantenerlo, risolvendo eventuali errori e aggiungendo eventuali funzionalità, soddisfacendo eventuali nuovi requisiti. Potrebbe non essere uno step presente ogni volta
2. **modello iterativo (Iterative development):** modello in cui le fasi di sviluppo avvengono nel corso del tempo, senza effettuare prima tutta la progettazione e poi l'implementazione, sviluppando versioni sempre più definite del software, portando subito in risalta i problemi e/o le

mancate interpretazioni del sistema da sviluppare.

Prevede diverse implementazioni di questo modello, come ad esempio l'UP, lo Scrum e la modellazione agile, in cui il concetto di fondo è quello di conoscere, sviluppare e progettare il sistema a passi successivi, al fine di avere ad ogni passo un sottoinsieme del sistema già funzionante e questo permette di poter reagire ai cambiamenti dei requisiti e permettere una manutenzione mantenibile nel tempo.

### **3. ingegneria del Software basata sui componenti (CBSE)**

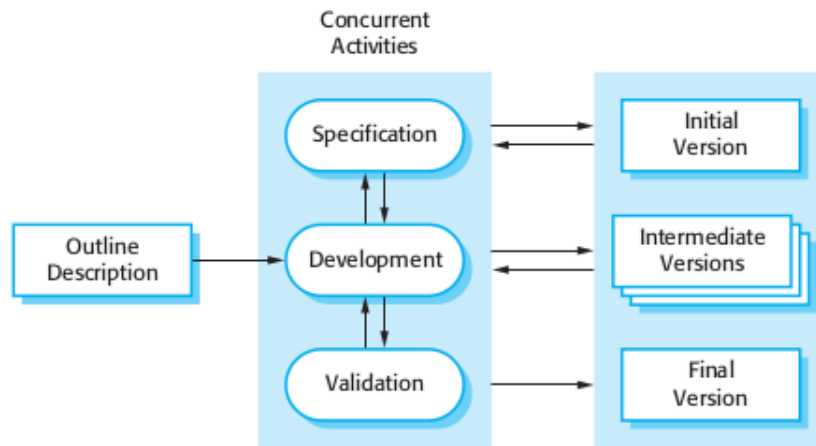


Figura 2.2: sviluppo incrementale fig:inc

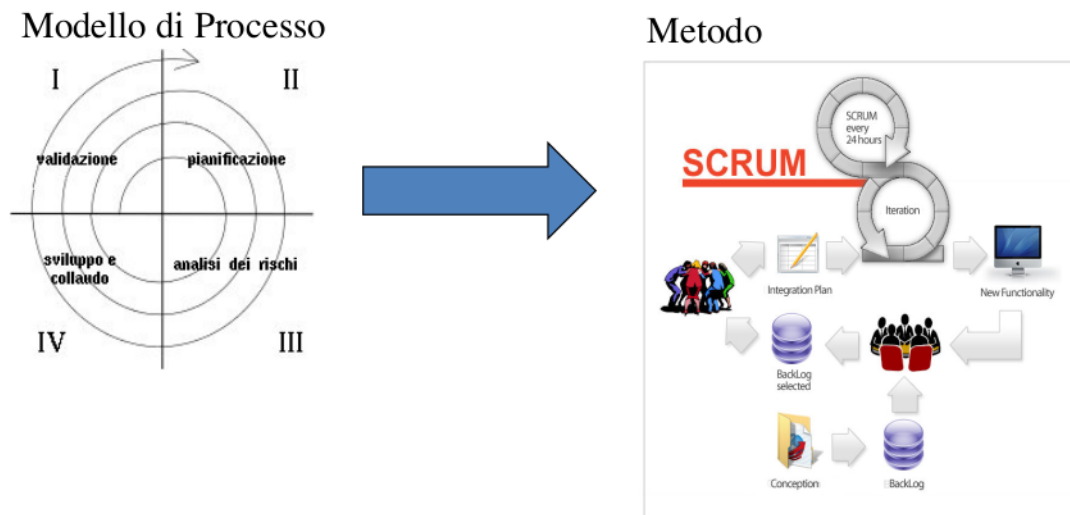
### Sviluppo incrementale

Lo sviluppo incrementale si basa sull'idea di sviluppare un'implementazione iniziale del software che viene revisionata dai futuri utenti dello stesso. Si procede nella stessa maniera con ulteriori versioni fino a giungere ad un'implementazione finale. Quindi sviluppo, indicato in figura fig:inc e consegna sono strutturati in una sequenza di incrementi, ognuno delle quali corrisponde a parte delle funzionalità richieste, con ovviamente un ordine di priorità dato dal cliente (le prime parti conterranno le funzionalità principali del software). Lo sviluppo incrementale ha 3 vantaggi principali rispetto al modello a cascata:

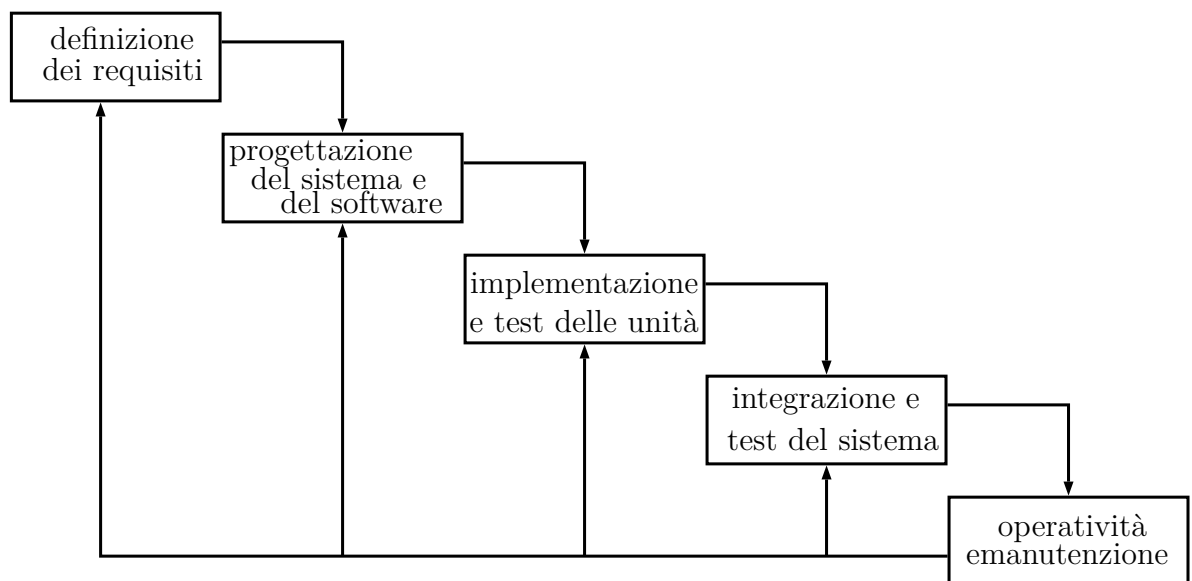
1. si riduce il costo relativo al cambio di specifiche del cliente, riducendo anche la quantità di analisi e documentazione che andrebbe rielaborata
2. si hanno feedback costanti sullo sviluppo grazie alla continua prova diretta delle varie implementazioni
3. si può fornire gradualmente al cliente un software funzionante anche se privo di alcune features, di modo che possa usare parte del software ben prima del suo completo sviluppo

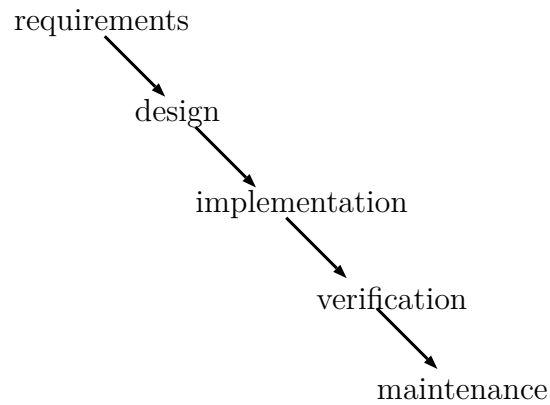
Inoltre i rischi di fallimento si abbassano e i servizi a priorità più alta sono testati più a fondo. Del resto questo modello può essere rallentato facilmente da problemi burocratici. Inoltre il processo di sviluppo è più difficile da controllare da parte di un manager e il continuo aggiungere features può danneggiare l'architettura generale del software **pagina 34**

I metodi di ingegneria del software sono l'implementazione dei seguenti modelli generici, e nella maggioranza si utilizza un modello iterativo, al fine di rendere incrementale lo sviluppo e ridurre il rischio di fallimento del progetto.



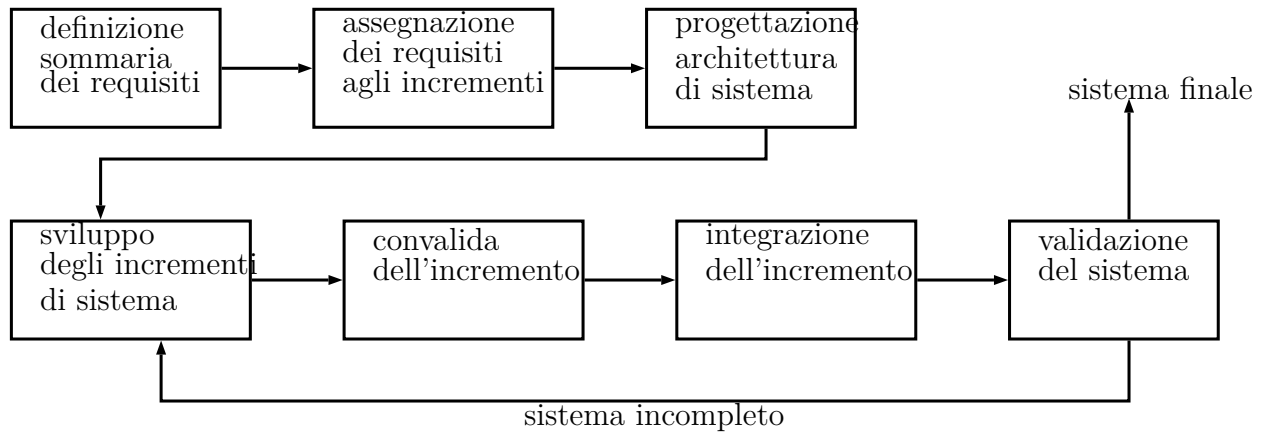
si ha il seguente feedback del modello a cascata:





Invece di rilasciare il sistema in una singola consegna, sviluppo e consegna sono strutturati in una sequenza di incrementi, ognuno dei quali corrispondenti a parte delle funzionalità richieste. Questa è la **consegna incrementale**. requisiti utente sono ordinati per priorità, i requisiti ad alta priorità sono inclusi nei primi incrementi.

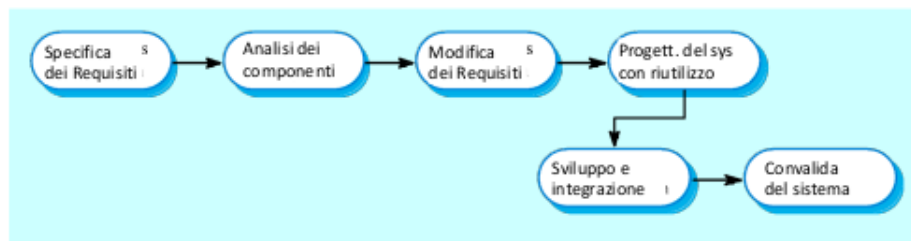
Una volta che lo sviluppo di un incremento inizia, i requisiti sono congelati; invece possono evolvere i requisiti per gli incrementi successivi:



Con la consegna incrementale si hanno i seguenti vantaggi:

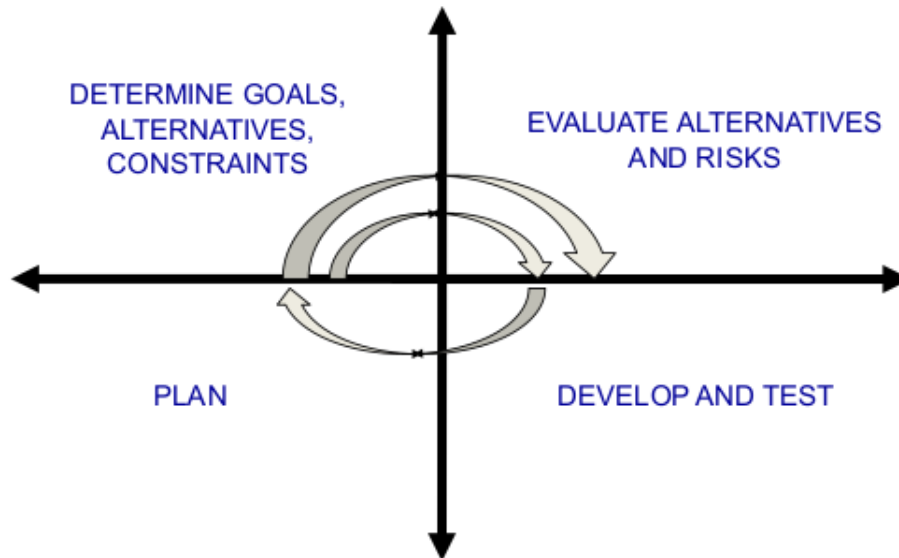
- funzioni utili per il cliente possono essere rilasciate ad ogni incremento, quindi alcune funzionalità di sistema sono disponibili sin dai primi incrementi
- i primi incrementi rappresentano dei prototipi che supportano la scoperta dei requisiti per i successivi incrementi
- i rischi di fallimento si abbassano
- i servizi a priorità più alta tendono ad essere collaudati più a fondo

Se si ha che i requisiti vengono scoperti attraverso lo sviluppo si ha lo **sviluppo evolutivo**. Si usano prototipi che poi non verranno utilizzati nel corso del progetto vero e proprio

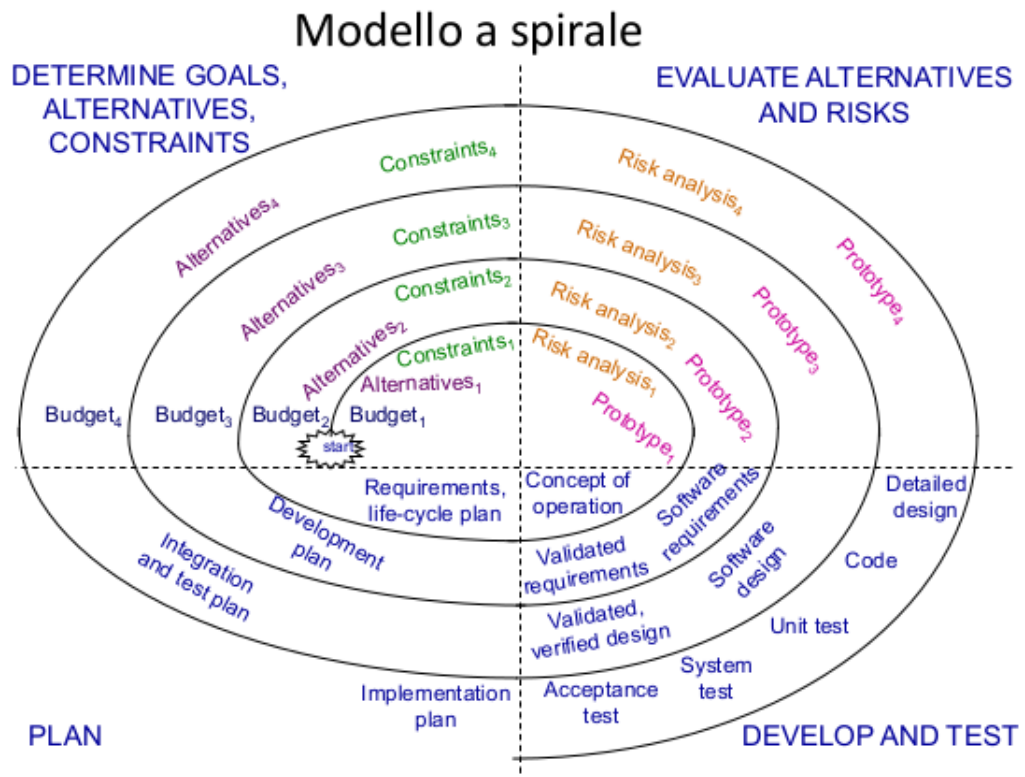


Si ha ingegneria del software basata su componenti se si hanno già librerie su cui lavorare.

Passiamo al primo modello iterativo, quello di B. Boehm.



si parte dal quadrante in alto a sinistra e si arriva al prodotto finale mediante una serie di iterazioni. Si sceglie ovviamente la soluzione più sicura. I vari passaggi sono qui rappresentati:



## 2.2 Modello UP

Il metodo di modellazione UP (Unified Process), conosciuto anche come RUP (Rational Unified Process), è un processo iterativo molto diffuso per lo sviluppo software, in cui sono presenti degli elementi tratti da altri metodi, come Extreme Programming, Scrum e così via.

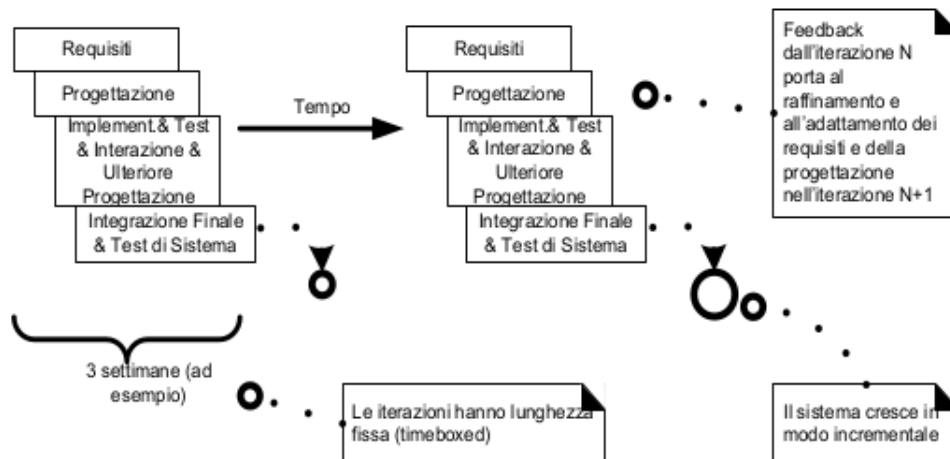
Lo sviluppo iterativo si ha nei modelli agili, che si basano sulla **consegna incrementale**. Si hanno iterazioni brevi e timeboxed, con un raffinamento evolutivo di piani, dei requisiti e del progetto. Questi metodi favoriscono la collaborazione nei gruppi e la riduzione dei costi.

### 2.2.1 Modello UP o RUP

Il metodo UP, *unified process*, conosciuto anche come RUP, *Rational Unified Process*, usa la notazione UML, *unified modeling language* ed è uno dei più importanti modelli agili. È un processo iterativo e incrementale. Si basa sulla suddivisione di un grande processo in iterazioni controllate. Si hanno



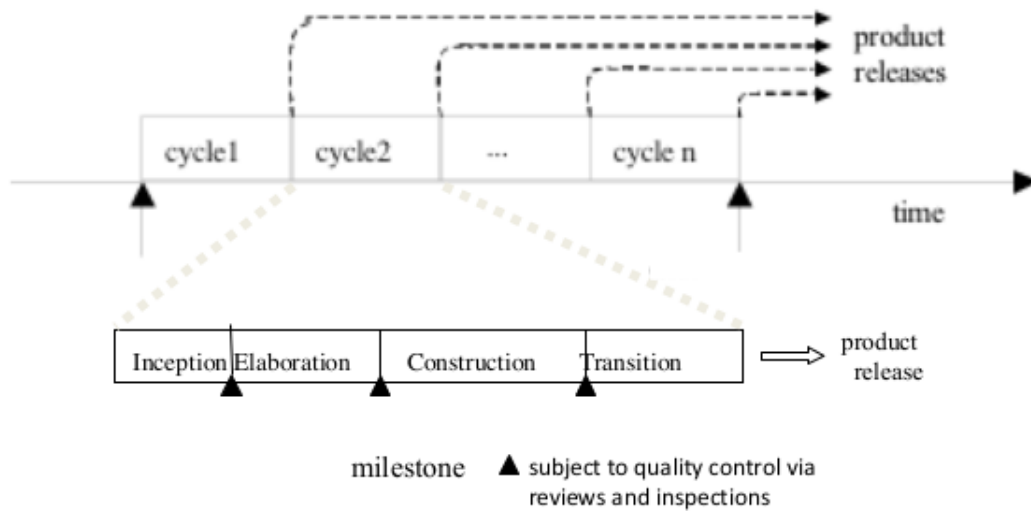
passi piccoli, timeboxed da 2 a 6 settimane, feedback rapido e adattamento, di requisiti, modelli, stime di sviluppo e costi e priorità.



Si hanno 4 fasi nel modello RUP:

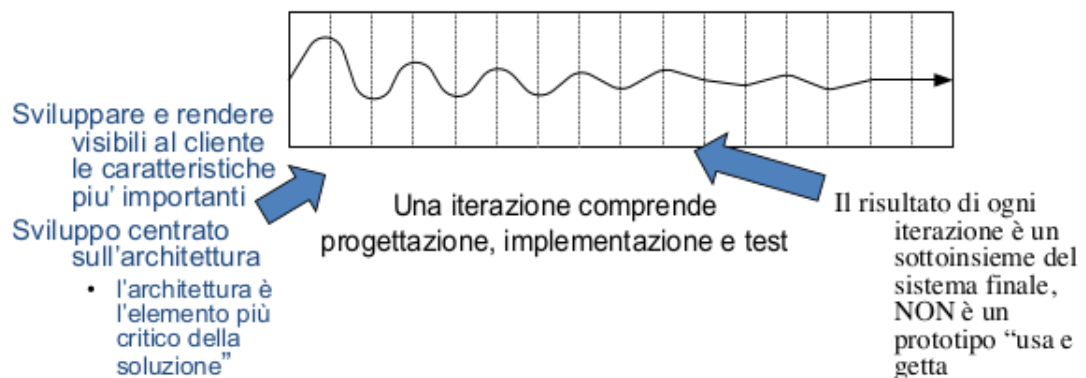
1. **avviamento:** dove viene stabilita la bussines rationale del progetto e stabiliti gli obiettivi, stime dei costi e dei tempi. Si ha uno studio di fattibilità, *Life-Cycle Objective Milestone*
2. **elaborazione:** dove si raccolgono i requisiti in modo più dettagliato, si procede con l'analisi ad alto livello per stabilire l'architettura di base e vengono analizzati i rischi principali. Si hanno stime più affidabili: *Life-Cycle Architecture Milestone*
3. **costruzione** che consiste di molte iterazioni, e ad ogni iterazione viene costruita una parte del sistema che soddisfa un sottoinsieme di requisiti. Viene effettuato il testing e l'integrazione. Si ha la preparazione al rilascio: *Initial Operational Capability Milestone*
4. **transizione:** dove vengono affrontati tutti gli aspetti legati al fine-tuning delle funzionalità, prestazioni, qualità, beta-testing, ottimizzazione, formazione degli utenti,... È la *Product Release Milestone*

Ogni ciclo è un incremento

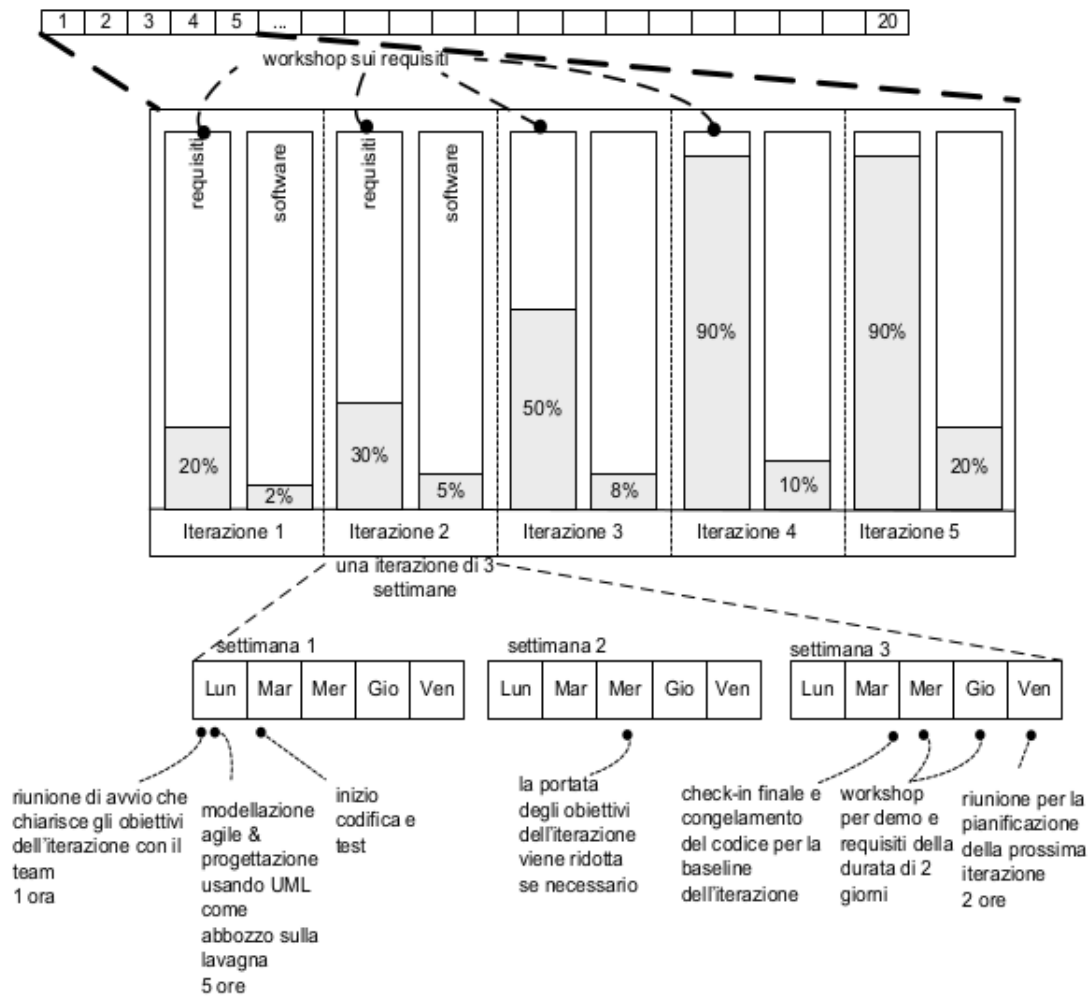


Si hanno i seguenti vantaggi dello sviluppo iterativo:

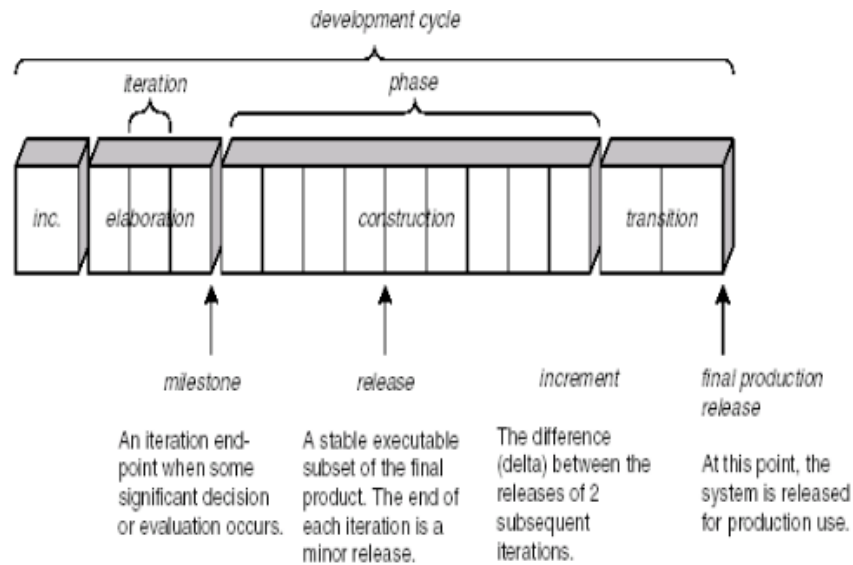
- minore chance di fallimento del progetto
- riduzione precoce dei rischi
- progresso visibile
- feedback precoce e coinvolgimento dell'utente
- "abbracciare il cambiamento"



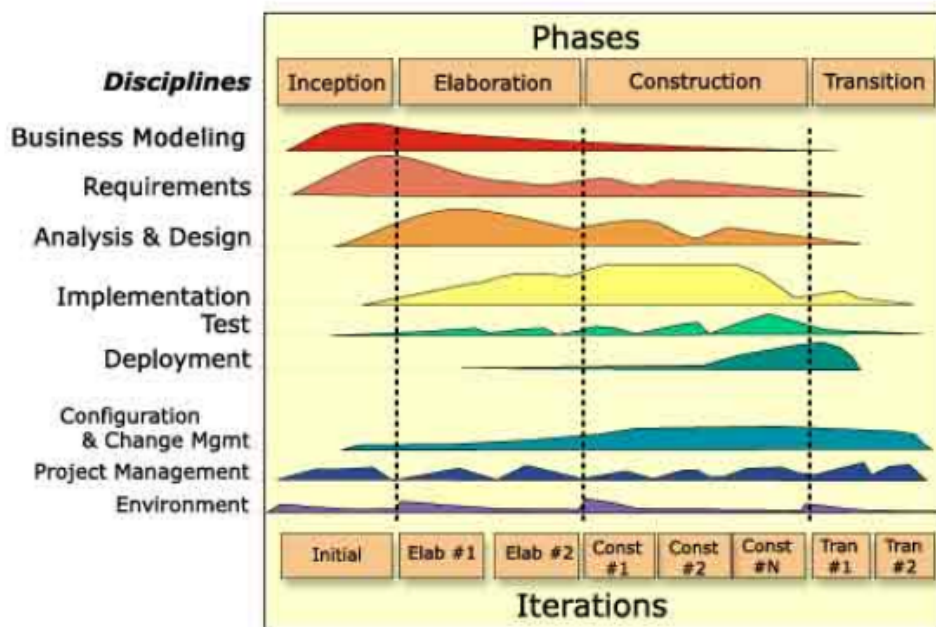
Vediamo un esempio con 5 iterazioni:



vediamo un'altra rappresentazione del ciclo di sviluppo:



vediamo anche un'immagine per rappresentare l'organizzazione del processo, con fasi, iterazioni e "discipline":

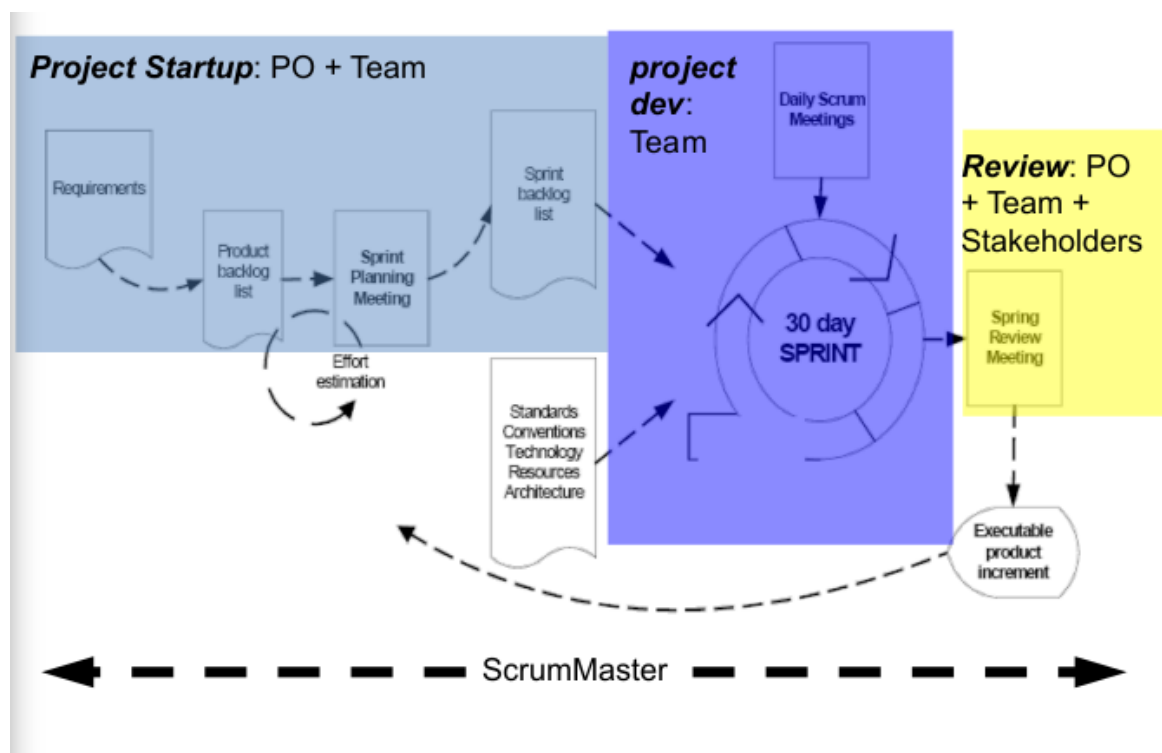


tra le discipline troviamo analisi e progettazione. Si hanno le seguenti caratteristiche principali per l'UP:

- è iterativo e incrementale
- si enfasi sul modello invece che sul linguaggio naturale
- è centrato sull'architettura

### 2.2.2 Processo Scrum

Iniziamo con un'immagine che rappresenta questo metodo agile:



È un processo iterativo basato sul controllo dello stato di avanzamento. Si hanno i seguenti principi fondamentali:

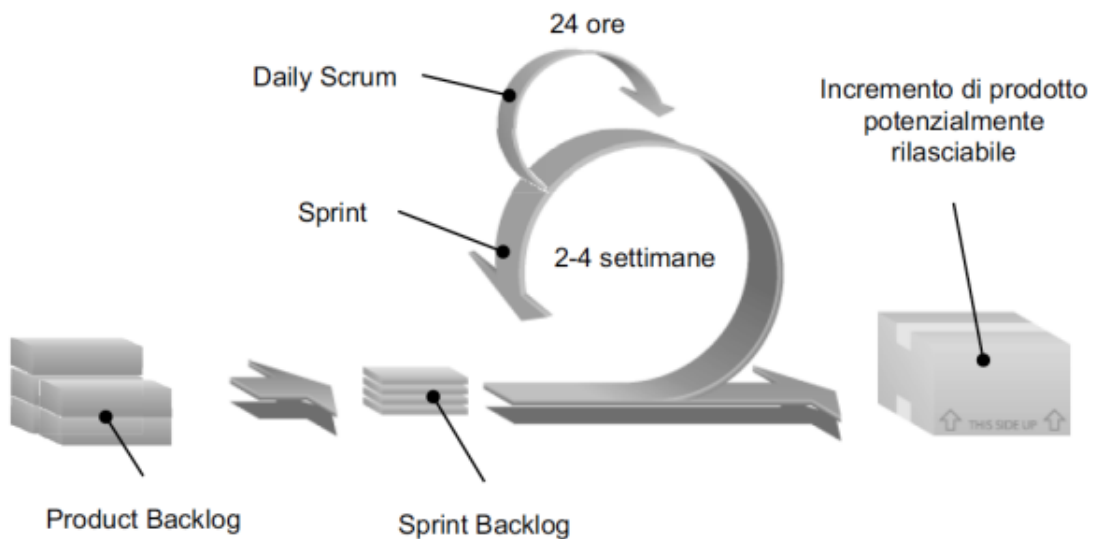
- **visibilità:** gli aspetti significativi del processo di sviluppo devono essere visibili a tutti e tutti gli aspetti devono essere chiari
- **ispezione:** poter ispezionare frequentemente il lavoro fatto per verificare che si sta procedendo verso gli obiettivi posti

- **adattamento**, se si sta deviando dagli obiettivi, occorre un adattamento per minimizzare altre deviazioni. Deve essere svolto nel minor tempo possibile in caso di necessità

Si hanno i seguenti ruoli:

- **product owner** che si occupa della definizione delle caratteristiche del progetto da sviluppare (1 sola persona, e.g. committente, o altri...); gestisce il Product Backlog. Lavora costantemente col team
- **team**: dedicato allo sviluppo e rilascio del prodotto attraverso incrementi successivi (da 3 a 7/8 membri)
- **scrum master**: responsabile che lo SCRUM venga applicato correttamente. Non è un project manager, fa da intermezzo tra team e product owner, collabora col team etc

Ecco un'immagine che rappresenta lo sviluppo con Scrum:



SI hanno quindi:

- releases brevi con sottoinsiemi consegnati velocemente
- in ogni istante si hanno test eseguibili, non si ha logica duplicata e si ha un numero minimo di features
- si ha *refactoring* con miglioramento continuo

- si un testo continuo e automatico, *testing first*
- si hanno due sviluppatori che lavorano insieme, *pair programming*
- il codice è proprietà del team e non del singolo dev, *collective owbership*
- si hanno check-in frequenti e integrazioni continue, *continuous integration*
- il cliente lavora col team

I **requisiti** sono una descrizione dei servizi del sistema e dei suoi vincoli operativi. Si hanno due tipi di requisiti:

- **requisiti utenti:** affermazioni in linguaggio naturale, corredate da tabelle e diagrammi, riguardanti i servizi che il sistema offre ed i vincoli operazionali. Sono scritti per i clienti e sono da loro comprensibili anche se non hanno conoscenze tecniche dettagliate.
- **requisiti di sistema:** un documento strutturato che definisce in modo dettagliato le funzioni del sistema, i servizi ed i vincoli operazionali. Definisce cosa deve essere implementato, quindi può essere parte del contratto tra acquirente e sviluppatore. È la base del progetto della soluzione e può essere illustrato utilizzando i **modelli di sistema**.

i requisiti possono avere 3 problemi:

- **ambiguità:** il sistema fornisce visualizzazioni appropriate per leggere i documenti
- **incompletezza:** i requisiti devono descrivere tutti i servizi forniti dal sistema (anche se in realtà tutto ciò è impossibile, per questo esistono i cambiamenti)
- **inconsistenza:** le descrizioni non devono contenere conflitti o contraddizioni (anche questa cosa è difficilissima da ottenere)

SI hanno anche i **requisiti funzionali** servizi che il sistema deve (o non deve) fornire. Si hanno anche i *requisiti non funzionali*, come vincoli sui servizi temporali, standard, sull'usabilità etc... I requisiti non funzionali possono essere più critici dei requisiti funzionali: **se non sono soddisfatti il sistema è spesso inutilizzabile**. Si hanno alcuni requisiti non funzionali:

- **prodotto:** specificano che il prodotto deve comportarsi in un modo particolare esempi: velocità di esecuzione, affidabilità.  
*L'interfaccia utente sarà implementata con HTML semplice senza frames e applets*
- **organizzativi:** conseguenza di politiche e procedure dell'organizzazione del cliente e dello sviluppatore. esempi: linguaggio di programmazione, metodo di sviluppo.  
*Il processo di sviluppo e la relativa documentazione sarà conforme alle norme XYZCo-SP-STAN-95*
- **esterni:** provengono da fattori esterni al sistema e al processo di sviluppo. esempi: interoperabilità, leggi e norme.  
*Il sistema non deve rilevare agli operatori nessuna informazione personale sui clienti oltre al nome e al numero di riferimento*



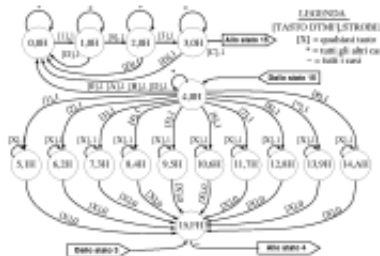


Tutto ciò perché il linguaggio naturale presenta mancanza di chiarezza, ambiguità, confusione etc... Esistono alternative:



Linguaggio  
naturale  
strutturato

Modello  
visuale  
informale



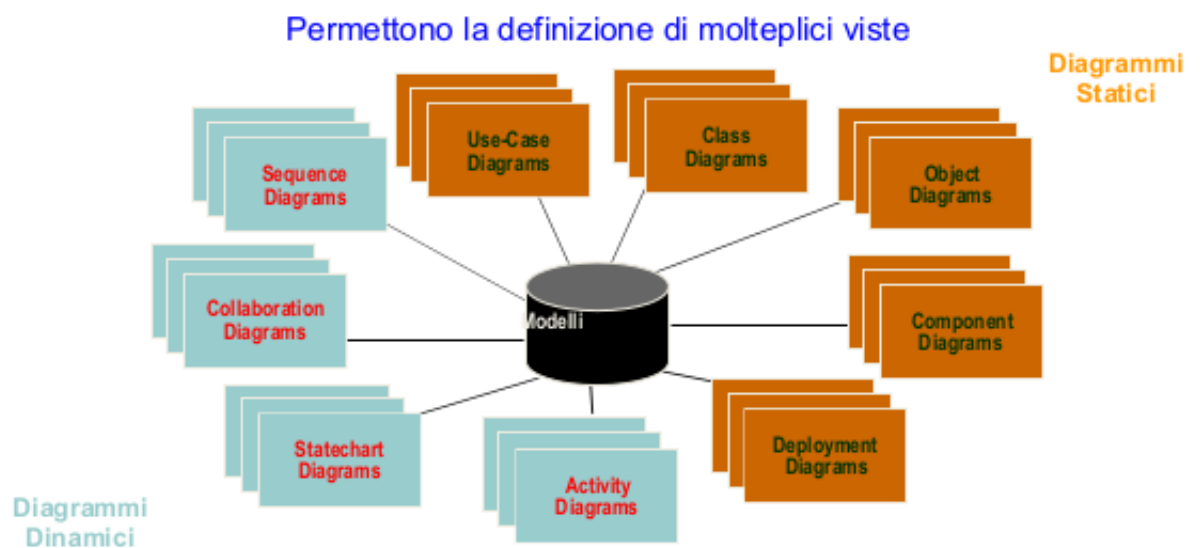
Modello  
visuale  
formale

Specifica  
testuale  
formale

BirthdayBook known : P NAME birthday : NAME $\Rightarrow$ DATE known $\Rightarrow$ data birthday
BirthdayBook $\Delta$ BirthdayBook name? : NAME date? : DATE  name? $\notin$ known birthday? = birthday $\cup$ {name? $\mapsto$ date?}

I requisiti hanno un formato standard, usano il linguaggio in modo consistente, evitano il gergo tecnico e evidenziano delle porzioni di testo per identificare le parti più importanti dei requisiti. Un esempio di standard è quello IEEE830.

I casi d'uso possono essere visualizzati con l'UML:



# Capitolo 3

## Casi d'uso

*I casi d'uso sono storie scritte, testuali, di qualche attore che usa un sistema per raggiungere degli obbiettivi.* I casi d'uso possono a loro volta influenzare molti altri elaborati dell'analisi, progettazione, implementazione, gestione del progetto e test. Passiamo a qualche definizione:

- un **attore** è qualcosa o qualcuno dotato di comportamento, come una persona (caratterizzata da un ruolo), un sistema informatico o un'organizzazione
- uno **scenario o istanza di caso d'uso** è una sequenza specifica di azioni e interazioni tra il sistema e alcuni attori. Uno scenario descrive una particolare storia nell'uso del sistema, o un percorso attraverso il caso d'uso

*Informalmente* un caso d'uso quindi può essere definito come una collezione di scenari correlati, di successo e fallimento, che descrivono un attore che usa un sistema per raggiungere un obiettivo.

**UP** definisce il modello di caso nell'ambito della disciplina dei Requisiti. Inoltre **i casi d'uso sono documenti di testo, non diagrammi, e la modellazione dei casi d'uso è innanzitutto un atto di scrittura di testi, non di disegno di diagrammi.** Il Modello dei Casi d'Uso può includere, opzionalmente, un diagramma UML dei casi d'uso che mostra i nomi dei casi d'uso e degli attori, nonché le relazioni tra di essi. Esso costituisce un buon diagramma di contesto di un sistema e del suo ambiente, oltre a costituire un indice di facile consultazione dei nomi dei casi d'uso. Si ricorda che **i casi d'uso non sono orientati ad oggetti.**

I casi d'uso sono un buon metodo per mantenere la semplicità e consentire agli esperti di dominio o ai fornitori di requisiti di scrivere essi stessi i casi d'uso, o perlomeno partecipare alla loro scrittura. Un altro valore dei casi

d'uso è che mettono in risalto gli obiettivi e il punto di vista dell'utente.

I casi d'uso sono requisiti, soprattutto requisiti funzionali o comportamentali, che indicano che cosa farà il sistema. In UP e in molti metodi moderni, i casi d'uso sono il meccanismo centrale che viene consigliato per la scoperta e la definizione dei requisiti.

Torniamo a parlare di attori. Un attore è qualcosa o qualcuno dotato di comportamento. Anche il sistema in discussione, *SuD*, *da system under discussion*, stesso è un attore, quando ricorre ai servizi di altri sistemi. Si hanno tre tipi di attori correlati al SuD:

1. **attore primario** che raggiunge degli obiettivi utente utilizzando i servizi del SuD. Identificare gli attori primari è utile per trovare gli obiettivi degli utenti, che guidano i casi d'uso
2. **attore di supporto** che offre un servizio (per esempio, informazioni) al SuD. Spesso è un sistema informatico, ma potrebbe essere un'organizzazione o una persona. Identificare gli attori di supporto è utile per chiarire le interfacce esterne e i loro protocolli
3. **attore fuori scena** che ha un interesse nel comportamento del caso d'uso, ma non è un attore primario o di supporto. Identificare gli attori fuori scena è utile per garantire che tutti gli interessi necessari vengano individuati e soddisfatti. Gli interessi degli attori fuori scena sono spesso sottili o facili da tralasciare, a meno che gli attori stessi non vengano esplicitamente nominati

Si hanno inoltre tre diversi livelli di **formalità** per i casi d'uso:

1. **formato breve** ovvero un riepilogo conciso di un solo paragrafo, normalmente relativo al solo scenario principale di successo. Si usa durante l'analisi iniziale dei requisiti, per capire rapidamente l'argomento e la portata. È possibile scriverlo in pochi minuti
2. **formato informale** ovvero più paragrafi, scritti in modo informale, relativi a vari scenari. Si usa come il formato breve ma qui si ha un livello di dettaglio maggiore
3. **formato dettagliato** dove tutti i passi e le variazioni sono scritti nel dettaglio; ci sono anche delle sezioni di supporto, come le pre-condizioni e le garanzie di successo. Va usato o perché molti casi d'uso sono stati identificati e scritti in formato breve, alcuni casi d'uso (circa il 10%) che hanno maggior valore e che sono più significativi dal punto di vista dell'architettura vengono scritti in formato dettagliato. Si ha quindi:

Sezione del caso d'Uso	Commento
<b>Nome del Caso d'Uso</b>	Inizia con un verbo
<b>Portata</b>	Il sistema che si sta progettando
<b>Livello</b>	"Obiettivo utente" o "sottofunzione"
<b>Attore Primario</b>	Nome dell'attore primario
<b>Parti Interessate e Interessi</b>	A chi interessa questo caso d'uso e che cosa desidera
<b>Pre-condizioni</b>	Che cosa deve essere vero all'inizio del caso d'uso
<b>Garanzia di successo</b>	Che cosa deve essere vero se il caso d'uso viene completato con successo
<b>Scenario Principale di Successo</b>	Uno scenario comune di attraversamento del caso d'uso, di successo e incondizionato
<b>Estensioni</b>	Scenari alternativi, di successo e di fallimento
<b>Requisiti speciali</b>	Requisiti non funzionali correlati
<b>Elenco delle variabili tecnologiche e dei dati</b>	Varianti nei metodi di I/O e nel formato dei dati
<b>Frequenza di ripetizione</b>	Frequenza prevista di esecuzione del caso d'uso
<b>Varie</b>	Altri aspetti, ad esempio i problemi aperti

Parliamo ora di scenari. Si hanno due tipologie:

1. **scenario principale (di successo)** che descrive un percorso di successo tipico che soddisfa gli interessi delle parti interessate. Si noti che spesso non comprende alcuna condizione o diramazione. Questo rende il caso d'uso più comprensibile e più estensibile. È formato da una sequenza di passi:

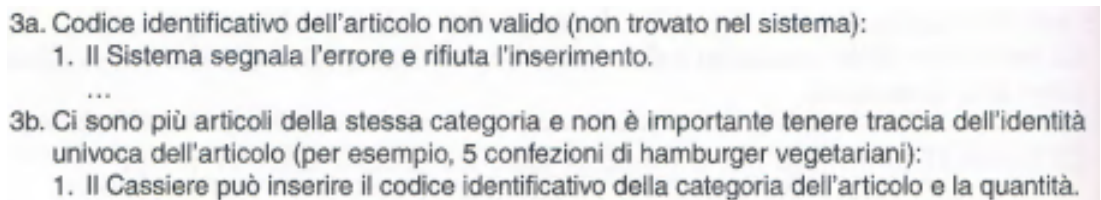
- (a) un'interazione tra attori
- (b) una validazione (solitamente effettuata dal sistema)
- (c) un cambiamento di stato da parte del sistema

Il primo passo di un caso d'uso non rientra sempre in questa classificazione, ma può indicare l'evento (trigger) che scatena l'esecuzione dello scenario. I nomi degli attori vengono di solito scritti con l'iniziale maiuscola, per facilitarne l'identificazione. Un esempio:

**Scenario principale di successo:**

1. Il Cliente arriva alla cassa POS con gli articoli da acquistare.
  2. Il Cassiere inizia una nuova vendita.
  3. Il Cassiere inserisce il codice identificativo dell'articolo.
  4. Il Sistema ...
- Il Cassiere ripete i passi 3-4 fino a che non indica che ha terminato.
5. ...

2. **scenari alternativi (o flussi alternativi o estensioni)** che descrivono tutte le diramazioni e gli altri scenari, sia di successo che di fallimento. La loro stesura è spesso lunga e complessa. Nella scrittura completa dei casi d'uso, la combinazione del flusso di base e degli altri scenari descritti dalle estensioni dovrebbe soddisfare "quasi" tutti gli interessi delle parti interessate. Tuttavia, alcuni interessi possono essere descritti come requisiti non funzionali, e pertanto riportati nelle Specifiche Supplementari piuttosto che nei casi d'uso. Gli scenari relativi alle estensioni sono diramazioni dello scenario principale di successo, e vengono indicati con riferimento ai suoi passi. Si contrassegnano con una lettera posta dopo il numero che identifica il passo (per il passo 3 si avrà 3a, 3b etc...) inoltra prima riporta la condizione, poi la reazione. Per esempio:



3a. Codice identificativo dell'articolo non valido (non trovato nel sistema):  
1. Il Sistema segnala l'errore e rifiuta l'inserimento.

...

3b. Ci sono più articoli della stessa categoria e non è importante tenere traccia dell'identità univoca dell'articolo (per esempio, 5 confezioni di hamburger vegetariani):  
1. Il Cassiere può inserire il codice identificativo della categoria dell'articolo e la quantità.

Un estensione è costituita da condizione e gestione. La gestione di un'estensione può essere riassunta in un unico passo, oppure può comprendere una sequenza di passi. Al termine della gestione di un'estensione, per default lo scenario si fonde di nuovo con lo scenario principale di successo, a meno che l'estensione non indichi diversamente (per esempio, l'arresto del sistema). In alcuni casi, i punti di estensione sono abbastanza complessi. Questo può essere un motivo per esprimere l'estensione come un caso d'uso separato.

Un caso d'uso può diramarsi in un altro caso d'uso e in effetti, i casi d'uso vengono spesso scritti come ipertesti, e l'esecuzione di un altro caso d'uso viene rappresentata come un collegamento ipertestuale. In questo caso, facendo clic sul nome del caso d'uso sottolineato, ne verrà visualizzato il testo.

Se un requisito non funzionale, un attributo di qualità o un vincolo si riferiscono in modo specifico a un caso d'uso, allora è bene registrarlo insieme al caso d'uso. Potrebbe trattarsi di qualità come prestazioni, affidabilità e usabilità, oppure di vincoli di progettazione, che siano stati imposti o considerati probabili. Si hanno quindi i **requisiti speciali**.

Vediamo ora un esempio completo di caso d'uso:

#### ElaboraVendita

Portata: Applicazione POS NextGen

Livello: Obiettivo utente

Attore primario: Cassiere

Parti Interessate e Interessi

- Cassiere: vuole un inserimento dei dati in modo preciso e rapido. Non vuole errore nei pagamenti, perché gli ammanchi di cassa vengono detratti dal suo stipendio
- Addetto alle vendite: Vuole che le commissioni sulle vendite siano aggiornate
- Cliente: vuole effettuare acquisti e fruire di un servizio rapido, nel modo più semplice possibile. Vuole una visualizzazione chiara degli articoli inseriti e dei loro prezzi. Vuole una prova d'acquisto per una eventuale restituzione o sostituzione
- Azienda: Vuole registrare accuratamente le transazioni effettuate e soddisfare gli interessi dei clienti. Vuole che vengano registrati i pagamenti da riscuotere tramite il Servizio di Autorizzazione di Pagamento. Vuole una certa tolleranza ai guasti per consentire di effettuare vendite anche se alcuni componenti del server (per esempio l'autorizzazione remota di pagamento con credito) non sono disponibili. Vuole un aggiornamento automatico e rapido della contabilità e dell'inventario.
- Direttore: Vuole essere in grado di eseguire rapidamente operazioni di sovrascrittura e risolvere in modo semplice i problemi del Cassiere.
- Enti governativi e Fiscali: Vogliono riscuotere le imposte su ciascuna vendita. Possono essere più enti: nazionale, regionale e provinciale.
- Servizio di Autorizzazione e di Pagamento: Vuole ricevere le richieste elettroniche di autorizzazione nel formato e nel protocollo corretto. Vuole una contabilità dettagliata dei suoi debiti verso il negozio.

Pre-condizioni: Il Cassiere è identificato e autenticato

Garanzia di successo (o Post-condizioni): La vendita viene salvata. Le imposte sono calcolate correttamente. La contabilità e l'inventario sono aggiornati. Le commissioni sono registrate. Viene generata una ricevuta. Le approvazioni alle autorizzazioni di pagamento sono registrate.

Scenario principale di successo

1. Il cliente arriva alla cassa POS con gli articoli e/o i servizi da acquistare
  2. Il cassiere inizia una nuova vendita
  3. Il cassiere inserisce il codice identificativo dell'articolo
  4. Il sistema registra la riga di vendita per l'articolo e mostra la descrizione dell'articolo, il suo prezzo, il totale parziale. Il prezzo è calcolato in base a un insieme di regole di prezzo.
- Il Cassiere ripete i passi 3-4 fino a che non indica che ha terminato
5. Il sistema mostra il totale con le imposte calcolate.
  6. Il Cassiere riferisce il totale al Cliente, e richiede il pagamento
  7. Il Cliente paga e il Sistema gestisce il pagamento
  8. Il Sistema registra la vendita completata e invia informazioni sulla vendita e sul pagamento ai sistemi esterni di Contabilità (per la contabilità e le commissioni) e di Inventario (per l'aggiornamento dell'Inventario)
  9. Il Sistema genera la ricevuta
  10. Il Cliente va via con la ricevuta e gli articoli acquistati

### Capitolo 3. Casi d'uso

---

#### Estensioni

- \*a In qualsiasi momento, il Direttore chiede di eseguire una operazione di sovrascrittura:
  - 1. Il sistema passa alla modalità autorizzata "Direttore"
  - 2. Il Direttore o il Cassiere esegue un'operazione nella modalità "Direttore"; per esempio riprendere una vendita sospesa su un altro registratore di cassa, annullare una vendita, cambiare la chiusura di cassa, e così via.
  - 3. Il sistema torna alla modalità autorizzata "Cassiere"
- \*b In qualsiasi momento, il sistema fallisce: per consentire il ripristino e una gestione corretta della contabilità, bisogna garantire che tutto lo stato transazionale significativo possa essere ripristinato, a partire da qualsiasi passo dello scenario
  - 1. Il Cassiere riavvia il Sistema, si autentica, e richiede il ripristino dello stato precedente
  - 2. Il Sistema ricostruisce lo stato precedente
    - 2a Il Sistema rileva delle anomalie che impediscono il ripristino:
      - 1. Il Sistema segnala un errore al Cassiere, registra l'errore e passa a uno stato pulito
      - 2. Il Cassiere inizia una nuova vendita
- 1a Il Cliente o il Direttore chiedono di riprendere una vendita sospesa
  - 1. Il Cassiere esegue l'operazione di ripresa e inserisce il codice identificativo della vendita da riprendere
  - 2. Il Sistema visualizza lo stato della vendita ripresa, con il totale parziale
    - 2a Vendita non trovata:
      - 1. Il Sistema segnala l'errore al Cassiere
      - 2. Il Cassiere probabilmente inizia una nuova vendita e reinserte tutti gli articoli
  - 3. Il Cassiere continua con vendita (probabilmente inserendo altri articoli o gestendo il pagamento)
- 2-4a Il cliente dice al Cassiere di godere di un'esenzione dalle imposte (per esempio perché è anziano)
  - 1. Il Cassiere verifica, quindi inserisce il codice per lo stato di esenzione dalle imposte
  - 2. Il Sistema registra lo stato (che utilizzerà durante il calcolo delle imposte)
- 3a Codice identificativo dell'articolo non valido (non trovato nel sistema):
  - 1. Il Sistema segnala l'errore e rifiuta l'inserimento
  - 2. Il Cassiere risponde all'errore:
    - 2a C'è un codice identificativo dell'articolo leggibile (per esempio un codice UPC numerico):
      - 1. Il Cassiere inserisce il codice dell'articolo manualmente
      - 2. Il sistema visualizza descrizione e prezzo
    - 2a Codice identificativo dell'articolo non valido: Il Sistema segnala l'errore. Il cassiere prova in un altro modo.
  - 2b Non c'è un codice identificativo dell'articolo, ma sul cartellino è presente un prezzo:
    - 1. Il Cassiere chiede al direttore di eseguire un'operazione di sovrascrittura
    - 2. Il Direttore esegue la sovrascrittura
    - 3. Il Cassiere richiede l'inserimento manuale del prezzo, inserisce il prezzo e richiede la tassazione standard per questo importo (poiché non vi sono informazioni sul prodotto, il calcolatore delle imposte non saprebbe altrimenti come effettuare la tassazione)
      - 2c Il Cassiere esegue Aiuto Ricerca Prodotto per ottenere il vero codice identificativo dell'articolo e il suo prezzo.
      - 2d Altrimenti, il Cassiere chiede a un dipendente il codice identificativo effettivo dell'articolo o il suo prezzo, e inserisce il codice o il prezzo manualmente (vedi sopra)
- 3b ...



Requisiti speciali:

- Interfaccia utente di tipo touch screen su un monitor piatto grande. Il testo deve essere visibile da una distanza di un metro
- Risposta all'autorizzazione di credito entro 30 secondi il 90% delle volte
- In qualche modo si desidera un ripristino robusto quando non riesce l'accesso ai servizi remoti, come per esempio il sistema di inventario
- Internazionalizzazione della lingua sul testo visualizzato
- Regole di business inseribili nei passi da 3 a 7
- ...

Elenco delle varianti tecnologiche e dei dati:

- \*a Richiesta di sovrascrittura da parte del Direttore effettuata passando una scheda apposita attraverso un lettore di schede, oppure inserendo un codice di autorizzazione con la tastiera
- 3a Codice identificativo dell'articolo inserito tramite lettore laser di codici a barre (se il codice a barre è presente) oppure tramite tastiera
- 3b Il codice identificativo dell'articolo può essere basato su uno tra gli schemi di codifica UPC, EAN, JAN o SKU.
- 7a Le informazioni sulla carta di credito sono inserite tramite lettore di schede o tramite tastiera
- 7b Firma per il pagamento con carta di credito ottenuta su ricevuta cartacea. Ma si prevede che, entro due anni, molti vorranno la cattura digitale della firma.

Frequenza di Ripetizione: Potrebbe essere quasi ininterrotta

Problemi aperti:

- Come variano le leggi fiscali?
- Esaminare la questione del ripristino dei servizi remoti
- Quale personalizzazione è necessaria per le diverse aziende?
- Il cassiere deve estrarre e portare via il cassetto della cassa quando effettua il logout?
- Il cliente può usare direttamente il lettore di schede o le deve fare il Cassiere?

Si hanno delle linee guida per la stesura dei casi d'uso. Si cerca uno stile essenziale, ignorando l'interfaccia utente e concentrandosi sullo scopo dell'attore, ignorando come un sistema fa qualcosa ma concentrandosi su cosa deve fare. Per facilitare la lettura dei requisiti, è perciò opportuno scrivere i casi d'uso in modo conciso, e allo stesso tempo in modo completo (con una chiara indicazione di soggetto verbo e di eventuali frasi subordinate).

I **casi d'uso a scatola nera** sono il tipo più comune e consigliato; non descrivono il funzionamento interno del sistema, i suoi componenti o aspetti relativi al suo progetto. Piuttosto, il sistema è descritto come dotato di responsabilità. Questa è una metafora comune e unificante nel pensare orientato agli oggetti; gli elementi software hanno responsabilità e collaborano con altri elementi dotati di responsabilità. Usando i casi d'uso a scatola nera per definire le responsabilità del sistema, è possibile specificare che cosa deve fare il sistema (comportamento o requisiti funzionali) senza decidere come lo farà (progettazione). In effetti, la definizione dell'"analisi" rispetto alla "progettazione" è talvolta riassunta come "che cosa" rispetto a "come". Questo è un tema importante nello sviluppo del software: durante l'analisi dei requisiti bisogna specificare il comportamento esterno del sistema, considerato a scatola nera, evitando di prendere decisioni sul "come". Successivamente, durante la progettazione, andrà creata una soluzione che soddisfa le specifiche.



Per esempio:

Stile a scatola nera	Non a scatola nera
Il Sistema registra la vendita.	Il Sistema memorizza la vendita in una base di dati. ...o (ancora peggio): Il Sistema esegue un'istruzione SQL INSERT per la vendita...

È anche importante trovare i casi d'uso e si ha una procedura di base per farlo:

1. scegliere i confini del sistema, ovvero definire il sistema al meglio. Per chiarire la definizione dei confini del sistema in corso di progettazione, è utile sapere che gli attori primari e gli attori di supporto sono considerati esterni al sistema. Una volta identificati gli attori esterni, i confini diventano più chiari.
2. identificare gli attori primari. Alcune volte sono gli obiettivi a svelare gli attori, o viceversa. Si possono usare una serie di domande per facilitare l'operazione:

Chi avvia e arresta il sistema?	Chi si occupa dell'amministrazione del sistema?
Chi si occupa della gestione degli utenti e della sicurezza?	Il "tempo" è un attore, nel senso che il sistema esegue operazioni in risposta ad alcuni eventi temporali?
Esiste un processo di monitoraggio che riavvia il sistema se si verifica un errore?	Chi valuta le attività e le prestazioni del sistema?
Come vengono gestiti gli aggiornamenti software?	Chi valuta i log?
Sono aggiornamenti automatici o a richiesta?	Vi si accede in remoto?
Oltre agli attori primari <i>umani</i> , ci sono sistemi software o robotizzati esterni che utilizzano i servizi del sistema?	Chi viene avvisato quando si verificano errori o guasti?

3. identificare gli obiettivi di ciascun attore primario. Si crea quindi un elenco attori-obiettivi, tipo questo:

Attore	Obiettivo	Attore	Obiettivo
Cassiere	elaborare le vendite elaborare i noleggi gestire le restituzioni cash in cash out ...	Amministratore del Sistema	aggiungere utenti modificare utenti eliminare utenti gestire sicurezza gestire tabelle di sistema ...
Direttore	avviare il sistema arrestare il sistema ...	Sales Activity System	analizzare dati sulla vendita ...

inoltre risulta più produttiva, nella realtà, chiedere all'attore stesso quale sia il suo obiettivo.

Un altro approccio per trovare attori, obiettivi e casi d'uso è basato sull'identificazione di eventi esterni. Spesso un gruppo di eventi appartiene allo stesso caso d'uso. Per esempio:

Evento esterno	Dall'attore	Obiettivo/Caso d'uso
inserire la riga di vendita per un articolo	Cassiere	elaborare una vendita
inserire il pagamento	Cassiere o Cliente	elaborare una vendita
...		

- definire i casi d'uso che soddisfano gli obiettivi degli utenti; il loro nome va scelto in base all'obiettivo. Di solito, i casi d'uso a livello di obiettivo utente saranno in corrispondenza biunivoca con gli obiettivi degli utenti. **Il nome dei casi d'uso deve iniziare con un verbo e, in generale va definito un caso d'uso per ciascun obiettivo utente.** È opportuno assegnare al caso d'uso un nome simile all'obiettivo dell'utente.

*un'eccezione comune alla scelta di un caso d'uso per obiettivo è quella che riunisce gli obiettivi separati CRUD (acronimo di create, retrieve, update, delete, ovvero creare, leggere, aggiornare, eliminare) in un unico caso d'uso CRUD, chiamato Gestisci <X>. Per esempio, gli obiettivi "modifica utente", "elimina utente" e così via sono tutti soddisfatti dal caso d'uso Gestisci Utenti.*

**Naturalmente, nello sviluppo iterativo ed evolutivo, non tutti gli obiettivi o i casi d'uso saranno identificati completamente o correttamente fin dall'inizio. Anche la scoperta dei requisiti è evolutiva.** Bisogna poi verificare la validità dei casi d'uso. Si hanno quindi tre test:

- test del capo:** *il capo vi chiede: "Cosa avete fatto tutto il giorno?" e voi rispondete: "Il login!". Il vostro capo sarà felice?.* Se non lo è il caso d'uso non supera il test del capo, il che significa che non è fortemente mirato a ottenere risultati il cui valore sia misurabile. Potrebbe essere un caso d'uso a un livello più basso, ma non al livello a cui è desiderabile concentrarsi durante l'analisi dei requisiti. Ciò non significa che bisogna sempre ignorare i casi d'uso che non superano il test del capo
- test EBP (*Elementary Business Process*),** che deriva dall'ingegneria dei processi di business: *Un processo di business elementare è un'attività svolta da una persona in un determinato tempo e luogo, in*

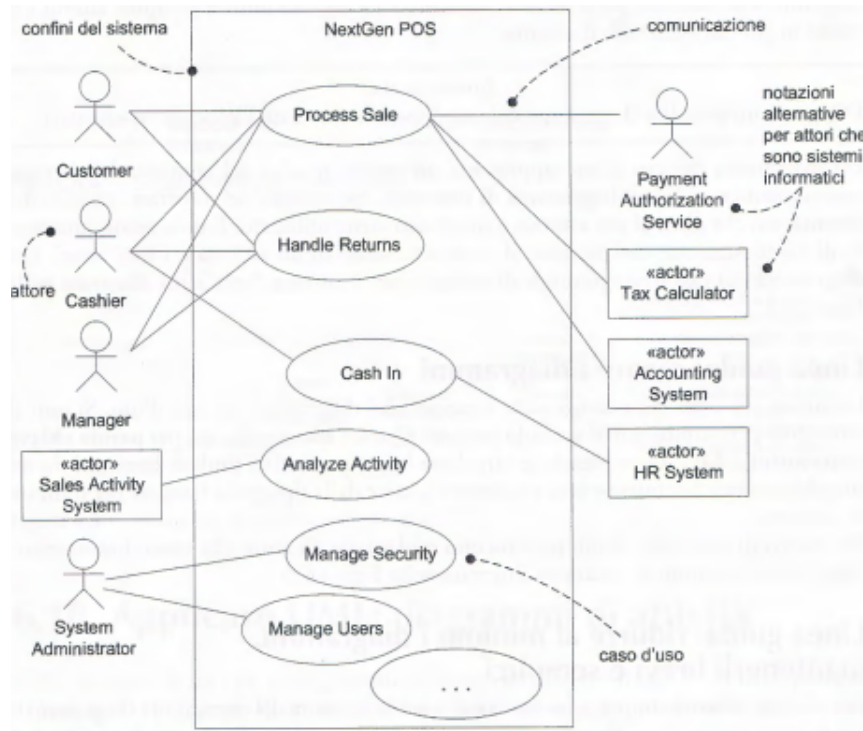
risposta a un evento di business, che aggiunge un valore di business misurabile e lascia i dati in uno stato coerente; per esempio, "Approva un credito" o "Stabilisci il prezzo per un ordine". Come nella definizione di UP, enfatizza l'aggiunta di un valore di business osservabile o misurabile, e giunge a una situazione in cui il sistema e i dati sono in uno stato stabile e coerente. **Il test EBP è simile al test del capo, soprattutto in termini di qualifica in termini di valore aziendale misurabile.**

- **test della dimensione**, che si basa sul fatto che un caso d'uso è formato da diversi passi (e nel complesso si hanno da 3 a 10 pagine di testo). Un errore comune nella modellazione dei casi d'uso è definire un caso d'uso a sé formato da un singolo passo.

Ci potrebbero essere delle eccezioni a questi test.

### 3.1 Diagrammi dei casi d'uso

UML fornisce una notazione dei diagrammi dei casi d'uso che consente di illustrare i nomi e gli attori dei casi d'uso, nonché le relazioni tra gli stessi, per esempio:



**I diagrammi dei casi d'uso e le relazioni tra casi d'uso sono secondari nel lavoro che riguarda i casi d'uso. I casi d'uso sono documenti di testo. Lavorare sui casi d'uso significa scrivere testo.** Un diagramma dei casi d'uso rappresenta un ottimo quadro del contesto del sistema; esso costituisce un buon diagramma di contesto, che consiste nel mostrare i confini del sistema, ciò che giace al suo esterno e come esso viene utilizzato. È utile come strumento di comunicazione che riassume il comportamento di un sistema e i suoi attori.

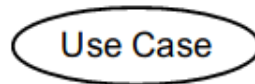
*Per ribadire, il lavoro importante dei casi d'uso è la scrittura del testo, non i diagrammi o le relazioni tra casi d'uso. Se un'organizzazione spreca molte ore (o peggio ancora, giorni) a lavorare su un diagramma dei casi d'uso e a discutere le relazioni tra casi d'uso anziché concentrarsi sulla scrittura del testo, vuol dire che il tempo è stato impiegato male.*

A livello di notazione si ha:

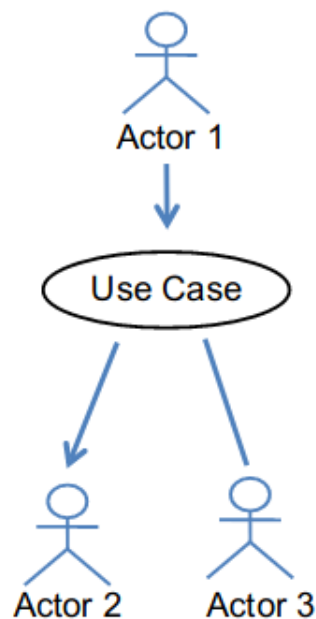
- **gli attori** sono rappresentati da un omino, col nome scritto sotto:



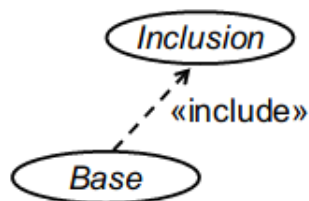
- **i casi d'uso** sono rappresentati con un ovale contenente la dicitura del caso d'uso:

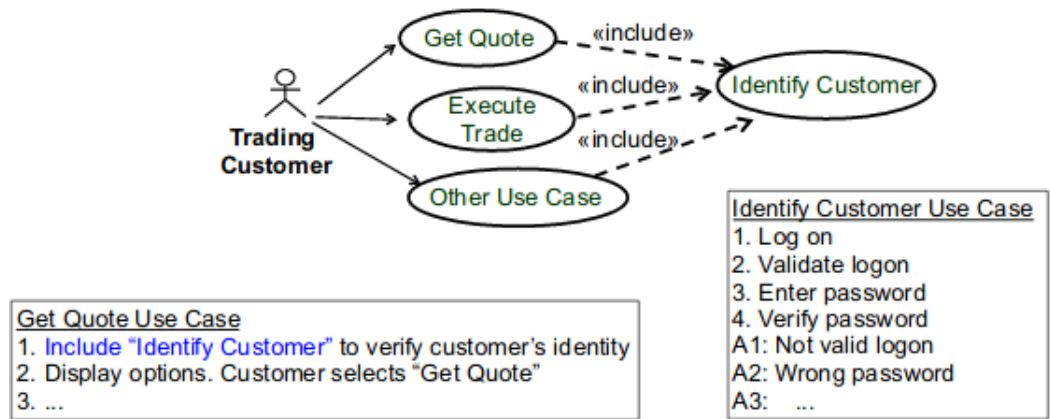


- le **associazioni** sono canali di comunicazione tra attore e caso d'uso. Si hanno due rappresentazioni:
  1. **una linea continua direzionata**, per specificare chi da inizio all'interazione
  2. **una linea continua non direzionata**, per indicare che entrambe le parti possono dare inizio ad un'interazione



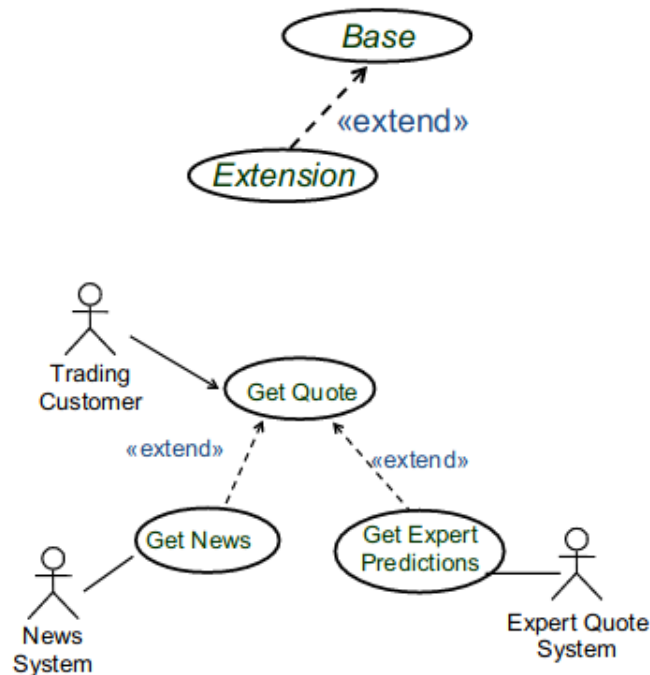
- l'**include (inclusione)** si rappresenta con una linea tratteggiata direzionata con l'indicazione «include» e indica la relazione tra un caso d'uso base ed un caso d'uso incluso nel caso d'uso base:

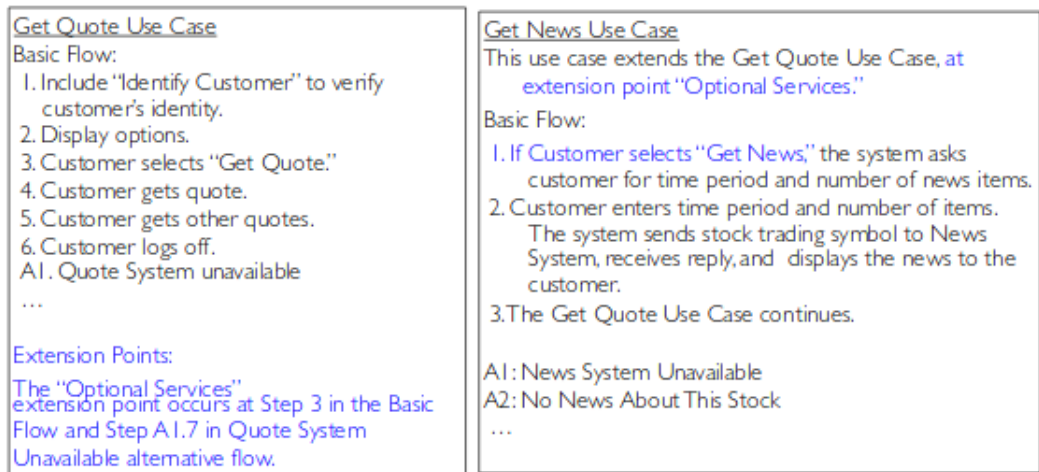




il caso d'uso è eseguito completamente quando viene raggiunto il punto di inclusione

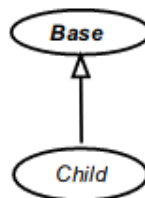
- **l'extend (estensione)**, si rappresenta come l'include (specificando «*extend*») e connette un caso d'uso esteso ad un caso d'uso base. Aggiunge varianti ad un caso d'uso base e viene inserito solo se la condizione di estensione è vera. Si hanno estensioni inserite solo in corrispondenza degli extension point:





*Il caso d'uso esteso è eseguito quando il punto di estensione è raggiunto e la condizione di estensione è vera*

- **la generalization (generalizzazione)** si ha quando un caso d'uso base è una generalizzazione di un caso d'uso child. Viene rappresentata con una linea continua direzionata con la freccia non riempita:



*Il caso d'uso generalizzato viene eseguito se la condizione di generalizzazione è vera inoltre la generalizzazione è puramente concettuale, non ci sono regole da seguire, come ad esempio l'utilizzo di punti di estensione. Si ha:*

