Ingeria del Software

luca carabini

September 2023

Contents

Chapter 1

Il ciclo di vita

Le attività svolte durante il ciclo di vita di un sistema informatico sono :

- **Definizione strategica** : Vengono prese deisioni sull'area aziendale che deve essere oggetto di automazione
- **Pianificazione** : Vengono definiti gli obiettivi, evidenziati i fabbisogni e viene condotto uno studio di fattibilità per individuare possibili strategie di attuazione e avere una prima idea dei costi, dei benefici e dei tempi.
- **Controllo qualità** : Viene predisposto un piano di controllo di qualitàper il progetto, per garantire il rispetto delle specifiche e di controllolare che il sistema realizzato si comporti come previsto
- **Analisi dei requisiti** : Formalizza i requisiti usando tecniche di modellazione della realtà e produce macrospecifiche per la fase di progettazione(1.1
- **Progettazione del sistema** : Interpreta i requisiti in una soluzione architetturale di massima. Produce specifiche indipendenti dai particolari strumenti che saranno usati per la costruzione del sistema
- **Progettazione Esecutiva** : Vengono descritti struttura e comportamento dei componenti dell'architettura, producendo specifiche che possano dar luogo, attraverso il ricorso a strumenti di sviluppo opportuni, a un prodotto funzionante
- **Realizzazione e collaudo in fabbrica** : Il sistema viene implementato sulla piattaforma prescelta e viene testato internamente (α-test) sulla base dei casi prova definiti durante la fase di analisi
- **Certificazione** : L'attività di certificazione del software ha lo scopo di verificare che esso sia stato sviluppato secondo i criteri previsti dal metodo tecnico di progetto, in conformità alle specifiche di sistema e a tutta la documentazione di progetto
- **Installazione** : Il sistema viene installato e configurato, e vengono recuperati gli eventuali dati pregressi
- **Collaudo del sistema installato** : Gli utenti testano in vitro il prodotto installato (β-test). Si possono evidenziare errori bloccanti (malfunzionamenti che pregiudicano l'attività di collaudo), errori non bloccanti (malfunzionamenti che non pregiudicano l'attività di collaudo), problemi di operatività (una funzionalità richiesta non viene attuata adeguatamente) e funzionali (una funzionalità richiesta non è implementata)
- **Esercizio** : Quando il collaudo dà esito positivo il sistema viene avviato (messo in produzione), inizialmente affiancando e poi sostituendo gradualmente l'eventuale sistema preesistente
- **Diagnosi** : Durante l'esercizio gli utenti rilevano eventuali errori

- **Manutenzione** : Gli errori che si manifestano durante il funzionamento vengono segnalati e corretti (manutenzione correttiva). Può inoltre essere necessario intervenire sul software per adattarlo ai cambiamenti del dominio applicativo (manutenzione adattativa)
- **Evoluzione** : Si valutano le possibilità di far evolvere il sistema incorporando nuove funzionalità o migliorandone l'operatività (manutenzione evolutiva o perfettiva)

1.1 Analisi dei requisiti

Lo scopo dell'analisi dei requisiti è produrre un documento di **specifica dei requisiti** che diventi l'input per fasi di progettazione e realizzazione L' oggetto dell'analisi è l'organizzazione nel suo complesso (sottoinsiemi aziendali, risorse, processi, flussi informativi, ecc..)

1.1.1 Specifica dei requisiti

La specifica dei requisiti è un accordo tra il produttore di un servizio e il suo consumatore.
In questa fase, attraverso la specifica dei requisiti l'utente finale e il progettista si accordano sulle funzionalità messe a disposizione dal software. La difficoltà per questo tipo di specifica è data dalla diversità dei linguaggi usata dalle due parti.

Qualità per la specifica dei requisiti

Le qualità che deve avere la specifica dei requisiti sono:

- **Chiarezza** : ogni specifica deve indicare quanto più chiaramente possibile le operazioni e i soggetti del processo che descrive
- **Non ambiguità** : il processo descritto dalla specifica deve essere definito in modo completo e dettagliato
- **Consistenza** : le specifiche non devono contenere punti contraddittori

Perchè è i requisiti sono importanti?

Perchè più tardi viene scoperto un errore nel ciclo di sviluppo, maggiore sarà il costo di riparazione (si può arrivare a 20 volte in più, se scoperto nella fase di manutenzione)

1.1.2 Alcune definizioni di Analisi

- **De Marco**: l'analisi è lo studio di un problema, prima di intraprendere qualche azione
- **Coad**: l'analisi è lo studio del dominio di un problema, che porta a una specifica del comportamento **esternamente osservabile**; una descrizione **completa**,**coerente** e **fattibile** di ciò che occorre realizzare; una trattazione quantitativa delle caratteristiche operazionali(cioè **affidabilità**, **disponibilità**, **prestazioni**)
- **Davis**: l'analisi del problema è il momento in cui viene definito lo spazio del prodotto; la descrizione del prodotto comporta la scelta di una soluzione e l'esplicitazione del comportamento esterno del prodotto dimostrando che esso soddisfa i requisiti

1.1.3 Cosa e come modellare

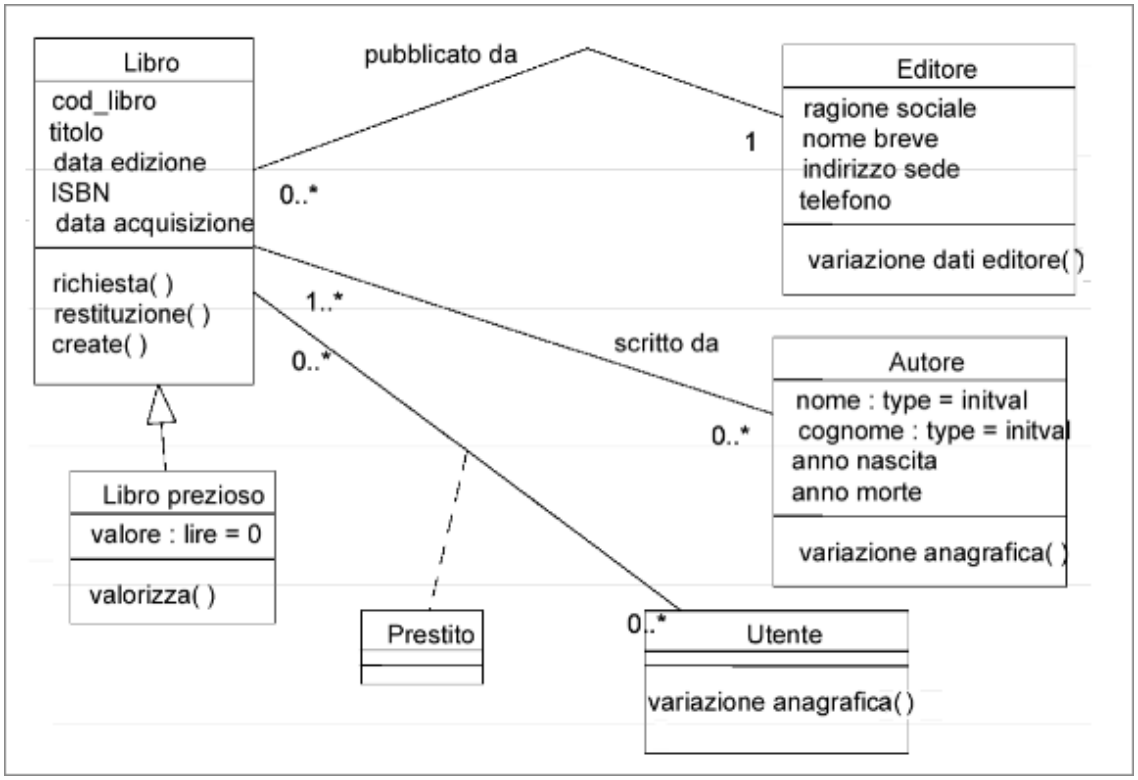
Il processo di analisi è **incrementale** e porta per passi successivi alla stesura di un insieme di documenti in grado di rappresentare un modello dell'organizzazione e comunicare, in modo non ambiguo, una descrizione esauriente, coerente e realizzabile dei vari aspetti **statici**, **dinamici** e **funzionali** di un sistema informatico.

1.1.4 Metodi di analisi

Differenti problemi richiedono differenti approcci e differenti strumenti di analisi:

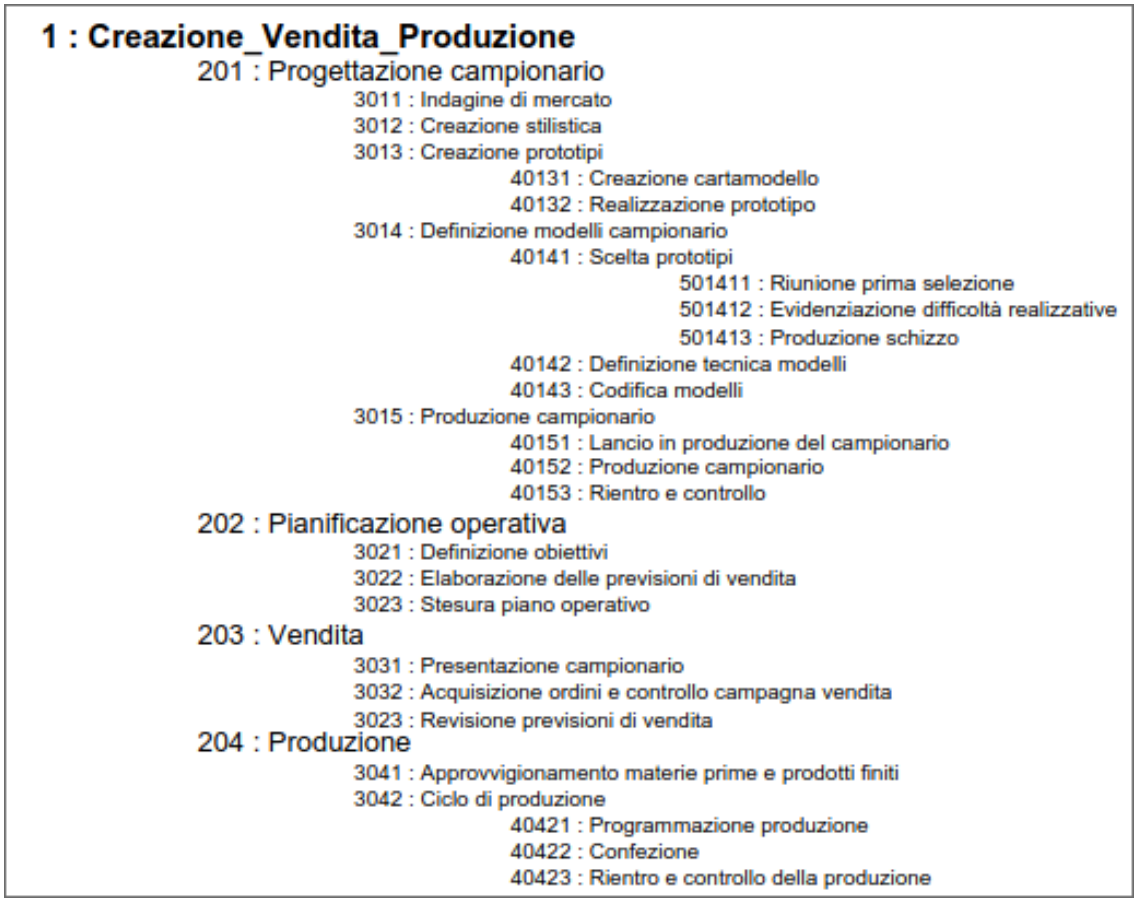
Analisi orientata agli oggetti

L' analisi orientata agli oggetti fa riferimento ad aspetti statici, l'enfasi è posta sull'**identificazione degli oggetti** e sulle **interrelazioni tra oggetti**
Nel tempo le proprietà strutturali degli oggetti osservati restano abbastanza stabili, mentre l'uso che degli oggetti si fa può mutare in modo sensibile.



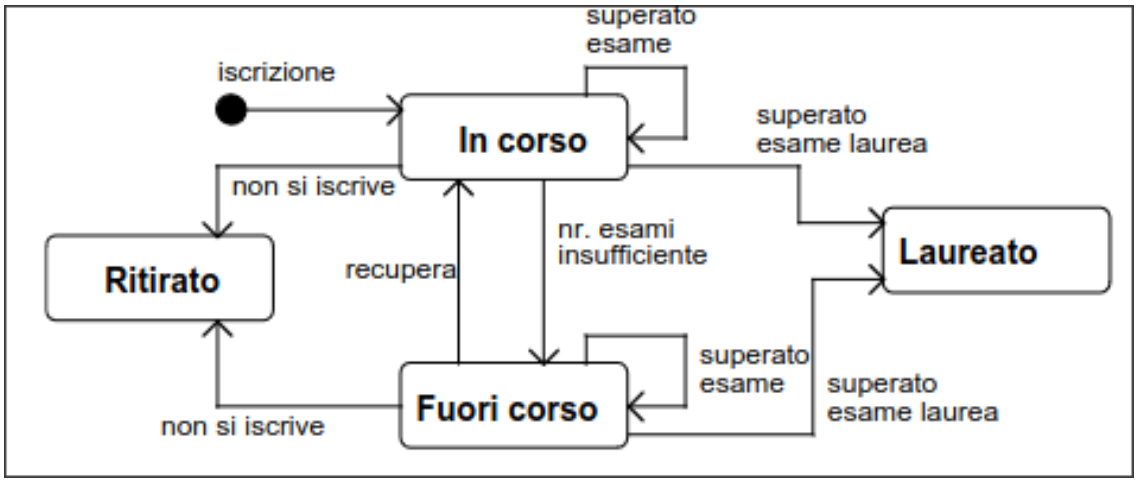
Analisi orientata alle funzioni

L'analisi orientata alle funzioni fa riferimento ad aspetti funzionali, e ha l'obiettivo di rappresentare un sistema come **un insieme di flussi informativi** e come **una rete di processi che trasformano flussi informativi**.
Ciò corrisponde alla progressiva costruzione di una gerarchia funzionale



Analisi orientata agli stati

L'analisi orientata agli stati fa riferimento ad aspetti dinamici.
Per alcune categorie di applicazioni può essere utile pensare fin dall'inizio in termini di stati operativi, in cui si può trovare il sistema allo studio, e transizioni di stato



1.1.5 Uso dei metodi d'analisi

La tendenza attuale è integrare metodi dei tre tipi, tenendo però conto della tipologia di applicazione:

- **Applicazioni orientate agli oggetti**: l'aspetto più significativo è costituito dalle informazioni, le funzioni svolte sono relativamente semplici
- **Applicazioni orientate alle funzioni**: la complessità risiede nel tipo di trasformazione input-output operata
- **Applicazioni orientate al controllo**: l'aspetto più significativo da modellare è la sincronizzazione fra diverse attività cooperanti nel sistema

1.1.6 Astrazione

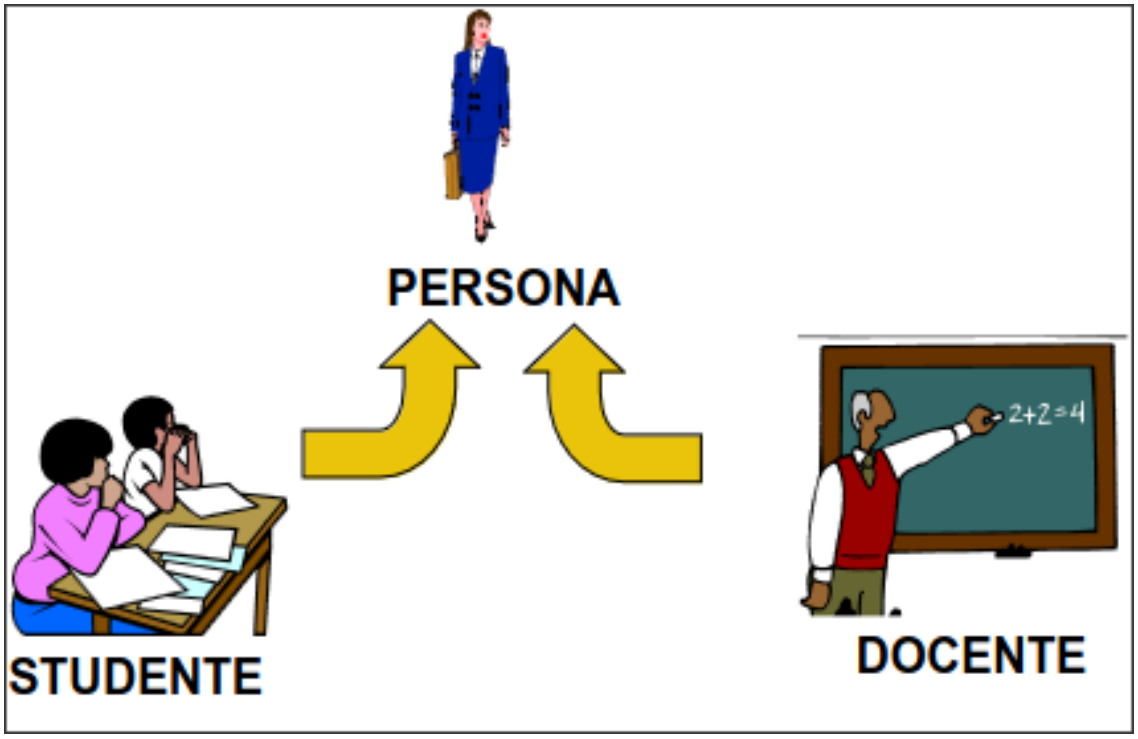
Gli **oggetti** possono essere descritti a partire da termini molto generici (edificio, strada) fino ad arrivare a livello di dettaglio specifici (la torre degli Asinelli)
Le **funzioni** possono essere espresse in modo vago (controllare il livello di gas nocivi nell'aria) e successivamente precisate (la programmazione del livello di soglia per l'allarme della centralina viene attivata premendo il pulsante P)
Gli **stati** possono essere descritti a un elevato livello di astrazione(la centralina è in stato di errore) o specificati in maggior dettaglio (è acceso il segnalatore di errore nel sensore S)
Ci sono 4 tipi di meccanismi di astrazione e sono:

Classificazione

La classificazione consente di raggruppare in classi oggetti, funzioni, o stati in base alle loro proprietà. (esempio classe dei computer)

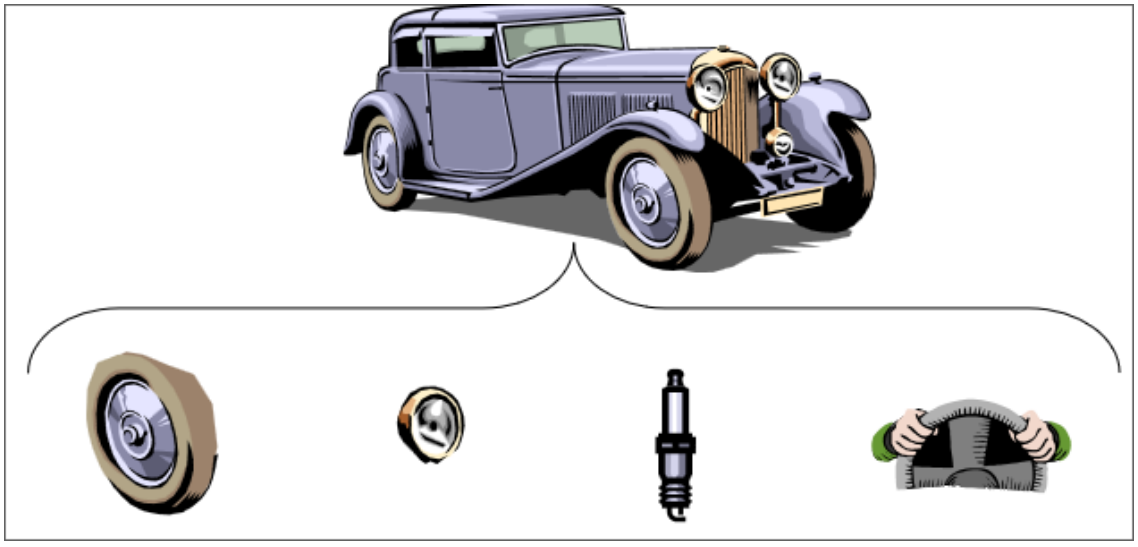
Generalizzazione

La generalizzazione cattura le relazioni è-un(is-a) ovvero permette di astrarre le caratteristiche comuni fra più classi definendo superclassi.



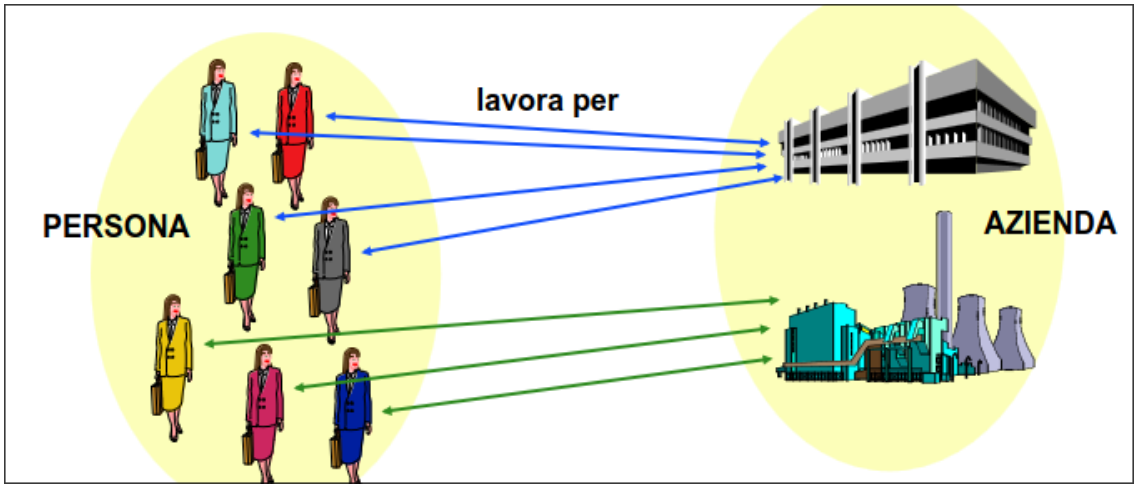
Aggregazione

L'aggregazione esprime le relazioni parte-di che sussistono tra oggetti, tra funzioni, tra stati



Associazioni

Oltre ai meccanismi citati è importante modellare le associazioni che sussistono fra le varie classi



1.1.7 Linguaggi per la specifica dei requisiti

Linguaggi informali

Il linguaggio naturale, alla base della comunicazione durante le interviste tra analista e utente, non può essere adottato come unico mezzo per produrre documenti di specifica per le innumerevoli ambiguità di significato

Linguaggi semiformali

notazione grafica, che presenta una semantica sfumata, accoppiata con descrizioni in linguaggio naturale (esempi : E/R, DFD)

Linguaggi formali

linguaggi di specifica basati sulla logica dei predicati, algebrici o per basi di dati; considerati però troppo complicati. La diversità degli obiettivi posti dalla specifica dei requisiti implica l'utilizzo di notazioni diverse per la rappresentazione delle informazioni:

Formalismi operazionali

Definiscono il sistema descrivendone il comportamento(normalmente mediante un modello).
Fornisce inoltre una rappresentazione più semplice poichè più simile al modo di ragionare della mente umana: **Facilità di acquisizione**, **Facilità di verifica della correttezza**, **Facilità di comprensione da parte dei programmatori**
Es
E è il percorso che si ottiene muovendosi in modo che la somma delle distanze tra due punti fissi p1 e p2 rimanga invariata

Formalismi dichiarativi

Definiscono il sistema dichiarando le proprietà che esso deve avere
Fornisce una rappresentazione che non si presta ad ambiguità
Es.

$$ax^2 + by^2 + c = 0$$

1.2 Progettazione

La progettazione riguarda tutte quelle attività che permettono di passare dalla raccolta ed elaborazione dei requisiti di sistema software alla sua effettiva realizzazione (fa da ponte tra la fase di specifica e la fase di codifica), passa quindi da **che cosa** deve essere realizzato a **come** deve avere luogo la realizzazione.
Il sistema viene suddiviso in sottoinsieme per ridurre la complessità(in quanto la complessità delle singole parti è minore di quella totale) e aiuta ad assegnare le part ai gruppi di lavoro in modo da renderli il più possibile indipendenti.

1.2.1 Il progetto astratto o dettagliato?

Il progetto deve essere sufficientemente astratto per poter essere agevolmente confrontato con le specifiche da cui viene derivato ma anche sufficientemente dettagliato in modo tale che la codifica possa avvenire senza ulteriori necessità di chiarire le operazioni che devono essere realizzate Non esiste un metodo generale per la progettazione del software

1.2.2 Obiettivi della progettazione

Gli obiettivi sono iminuzione dei costi e tempi di produzione e nell'aumento della qualità del software. I costi maggiori si hanno nella manuntezione del software quindi avere un codice manutenibile è fondamenntale.

Chapter 2

Paradigma a oggetti

2.1 Oggetti

Sono gli elementi di base del paradigma, e corrispondono a entità (non necessariamente "fisiche") del dominio applicativo.
Un oggetto è un individuo sostanziale che possiede:

- **identità**(OID, Object IDentifier): che gli viene assegnato alla creazione, non può essere modificata ed è indiendente dallo stato corrente dell'oggetto
- **stato**: definito come l'insieme dei valori assunti a certo istante da un insieme di attributi
- **comportamento** : definito da un insieme di operazioni

Quando un oggetto include riferimenti ad altri oggetti, si dice oggetto complesso.

2.1.1 Operazioni

Ogni operazione dichiarata da un oggetto specifica il **nome** dell'operazione, gli oggetti che prende come **parametri** e il **valore restituito** (**signature**)

2.1.2 Interfaccia

L'insieme di tutte le signature delle operazioni di un oggetto sono dette **interfaccia** dell'oggetto

2.2 Astrazione

E' una rappresentazione di un insieme di oggetti "simili", caratterizzato da una struttura per i dati e da un'interfaccia che definisce quali sono le operazioni associate agli oggetti, ovvero l'insieme dei servizi implementati.
Un tipo è sottotipo di un supertipo se la sua interfaccia contiene quella del supertipo(Un sottotipo eredita l'interfaccia del suertipo, l'interfaccia non vincola l'implementazione del servizio offerto, quindi oggetti con la stessa interfaccia possono avere implementazioni diverse)

2.3 Classe

Fornisce una realizzazione di un tipo di dati astratto, specifica cioè un'implementazione per i metodi a esso associati.
Un oggetto è sempre istanza di esattamente una classe, e tutti gli oggetti di una classe hanno gli stessi attributi e metodi

2.4 Incapsulamento

Protegge l'oggetto nascondendo lo stato dei dati e l'implementazione delle sue operazioni.
l principio di incapsulamento sancisce che gli attributi di un oggetto possono essere letti e manipolati solo attraverso l'interfaccia che l'oggetto stesso mette a disposizione.

2.4.1 Vantaggi

- Per l'utilizzo di una classe è sufficiente conoscerne l'interfaccia pubblica; i dettagli implementativi sono nascosti all'interno
- La modifica dell'implementazione di una classe non si ripercuote sull'applicazione, a patto che non ne venga variata l'interfaccia
- viene fortemente ridotta la possibilità di commettere errori nella gestione dello stato degli oggetti
- Il debugging delle applicazioni è velocizzato, poiché l'incapsulamento rende più semplice identificare la sorgente di un errore

2.4.2 Metodi

Un metodo cattura l'implementazione di una operazione. Un metodo può essere: **pubblico**, **privato** o **protetto**
I metodi possono essere classificati come:

- **costruttori**: per costruire oggetti a partire da parametri di ingresso restituendo l'OID dell'oggetto costruito
- **distruttori**: per cancellare gli oggetti ed eventuali altri oggetti ad essi collegati
- **accessori**: per restituire informazioni sul contenuto degli oggetti (proprietà derivate)
- **trasformatori**: per modificare lo stato degli oggetti e di eventuali altri oggetti ad essi collegati

2.5 Ereditarietà

Il meccanismo di ereditarietà permette di basare la definizione e implementazione di una classe su quelle di altre classi.
È possibile definire relazioni di specializzazione/ generalizzazione tra classi: la classe generalizzante viene detta superclasse, la classe specializzante viene detta sottoclasse o classe derivata.
Ciascuna sottoclasse eredita dalla sua superclasse la struttura ed i comportamenti, ovvero gli attributi, i metodi e l'interfaccia; può però specializzare le caratteristiche ereditate e aggiungere caratteristiche specifiche non presenti nella superclasse.

2.5.1 Ereditarietà multipla

Si parla di ereditarietà multipla quando una sottoclasse può essere derivata contemporaneamente da più superclass

Chapter 3

Uml

La struttura di UML è composta da:

- costituenti fondamentali(gli elementi di base):
  - entità
  - relazioni
  - diagrammi
- meccanismi comuni(tecniche comuni per raggiongere specifici obiettivi):
  - specifiche
  - ornamenti
  - distinzioni comuni
  - meccanismi di estensibilità
- arcgitettura(l'espressione dellarchitettura del sistema

3.1 Entità

3.1.1 Caso d'uso

è una funzionalità

3.1.2 Componente

è un modulo

3.1.3 Componente

libreria API ecc..

3.1.4 Modulo

pezzo di hardware(server, client ecc..)

3.1.5 stato

già visto prima(stundente in corso o no ecc...)

3.1.6 Interazione

Quando un oggetto interagisce con un altro oggetto

3.1.7 Package

Cassetto dentro cui mettiamo classi che vogliamo tenere insieme

3.1.8 Annotazione

post-it per dire vincoli inespressi

3.2 relazioni

3.2.1 Associazione

Uguale a quella dell'er collega istanze di classificatori tra di loro(persona titolare di una carta)

3.2.2 Aggergazione

part-of dalla parte del (forma più debole di part-of le parti esistono indipendemente da tutto)

3.2.3 Composizione

part-of, forma forte (es aula è composta da muri, soffitti e pareti)

3.2.4 Generalizzazione

semantica is-a per creare gerarchie di classi

3.2.5 Realizzazione

quando una classe realizza una implementazione di una interfaccia oppure raffinamento specifica una classe astratta(es una classe formata nell'analisi viene raffinata della fase di pregettazione)

3.2.6 Dipendenze

Quando una modifica di A modifica il comportamento di B

3.2.7 Contenimento

Specifico per i package.

3.3 Casi d'uso

Rappresenta il ruolo di utilizzzo del sistema da parte di uno o più utilizzatori(**attori**):

- esseri umani(dipendenti, clienti)
- organizzazioni, enti, istituzioni
- altre applicazioni o sistemi(hardware, software), sottoinsiemi

Descrivino l'interazionr tra attori e sistema, non la logica interna della funzione ne la struttura del sistema. Possono essere definiti a livelli diversi, ma sempre dal punto di vista dell'utente

3.3.1 Attore vs Cso d'uso

Un **attore** identifica il ruolo che un entità esterna assume quando interagisce direttamente con il sistema
Un **caso d'uso** è la specifica di una sequenza di azioni che un sistema, un sottosistema o una classe può eseguire interagendo con attori esterni.

3.3.2 Ruolo dei casi d'uso

Nelle fasi iniziali della progettazione servono per chiarire cosa dovrà fare il sistema, è utile nella comunizazione con persone non esperte nella progettazione perchè è semplice
Inoltre costituisce un punto di partenza per la progetazione del sistema.

3.3.3 Scenari

Ogni specifica esecuzione(istanza) di un caso d'uso è detta **scenario**. Esistono scenari di successo e scenario ddi fallimento.
La prasi più diffusa per la descrizione degli scenari di un caso d'uso, è quella di definire uno **scenario base**, cioè lo scenario più semplice possibile che porta al successo del caso d'uso. Allo scenario base vengono agganciati le **varianti**, che lo rendono più complesso e possono portare al successo o al fallimento del caso d'uso.

3.3.4 Specifiche del caso d'uso

La specifica di un caso d'uso ha un ruolo centrale nella comunicazione tra diversi soggetti coinvolti nello sviluppo del sistema(dal committente agli utilizatori, dai progettisti agli specialisti dei test), si utilizza un diagramma di attività o di sequenza per descrivere il caso d'uso.

3.3.5 Realizzare i casi d'uso

La ralizzazione dei casi d'uso può essere espressa con una **collaborazione** costituita da classi che interagendo tra loro svolgono i passi specificati nel caso d'uso.
Può essere descritta:

- a livello **statico** mediante un diagramma delle classi o gli oggetti coinvolti nella collaborazione.
- a livello dianmico mediante un diagramma di interazione che evidenzi i messaggi che gli oggetti si scambiano nell'ambito della collaborazione

3.4 Diagrammi delle classi

Sono il nucleo fondante di UML.
Descrivono la struttura statica del sistema in termini di classi e loro relazioni reciproche:

- Una **classe** descrive un gruppo di oggetti con proprietà, comportamento e relazioni comuni
- Un **attributo** è un valore che caratterizza gli oggetti di una classe
- Un'**operazione** è una trasformazione che può essere applicata (o invocata da) gli oggetti di una classe. Ogni operazione ha come argomento imlicito l'oggetto destinazione

3.4.1 Notazioni

Per gli attributi della classe : **visibilità** nome **molteplicità** : **tipo** = **valore di default**

- Visibilità
  - publica +
  - privata -
  - protetta #
  - package *sim*
- Molteplicità : per esempio [5], [2..*]
- Tipo: Integer, String ecc..
- Ambito: istanza o classe

Per le **operazioni** della classo :
visibilità

$$\underbrace{\textit{nome}\,(\textbf{parametro,...}):\textbf{tipoRestituito}}_{\textit{signature}}$$

- Parametri : **direzione nomeParametro: tipoParametro=valoreDefault nomeParametro**
- Direzione
  - in
  - out
  - inout
  - return (si usa quando l'operazione restituisce più valori)
  - Ambito: istanza, classe

3.4.2 Associazione

È una connessione tra classi, tipicamente bidirezionale.
La molteplicità può essere:

- Esattamente 1 (1)
- Opzionale 1 (0..1)
- Da x a y inclusi (x..y)
- Solo valori a,b,c (a,b,c)
- 1 o più (1..*)
- 0 o più (0..*) o (*)

È possibile indicare il **verso di lettura**, definire associazioni monodirezionali e specificare i ruoi di una associazione
Si possono specificare anche vincoli e classi associtive

Associazioni qualificate

Le associazioni qualificate riducono un'associazione molti-a-molti a una del tipo uno-a-uno, specificando un attributo che permette di selezionare un unico oggetto destinazione svolgendo il ruolo di identificatore o chiave di ricerca

Associazioni n-arie

È possibile definire associazioni n-arie(cioè tra n classi).
Ogni istanza dell'associazione è una tupla formata da n oggetti delle rispettive classi.
Le modalità di un ruolo rappresenta il numero di istanze dell'associazione quando sono stati fissati n-1 oggetti
I numeri di istanze dell'associazione quando è fissato un solo oggetto sono implicitamente assunti essere tutti a "molti"

3.4.3 Elemento derivato

Un **elemento derivato** può essere calcolato a partire da un altro ma viene mostrato, per motivi di chiarezza o per scelte di progettazione, nonostante non aggiunga alcuna ulteriore informazione semntica.
Nello schema viene posizionato uno slash prima del suo nome e i dettagli su come calcolarlo possono essere inseriti in una nota.

3.4.4 Aggregazione

È un caso speciale di associazione con semantica part-of.

3.4.5 Composizione

È un'aggregazione in cui il tutto "possiede" le sue parti

3.4.6 Generalizzazione

Tutti gli attributi, le operazioni e le relazioni della superclasse vengono ereditati dalle sottoclassi.
Supporta l'ereditarietà multipla e possono essere indicati insiemi di generalizzazione e vincoli (**overlapping, disjoint, complete, incomplete**)

3.4.7 Classi astratte

Sono clasi che non possono essere istanziate da oggetti, sono utili come radici di gerarchie di specializzazione

3.4.8 Powertyping

Un **powertype** è una (meta)classe le cui istanze sono classi che specializzano un'altra classe

3.4.9 Dipendenza

In generale, A dipende da B quando una variazione in B può comportare una variazione in A.
Nel caso delle classi, una dipedenza indica che una classe cliente dipende da alcuni servizi di una classe fornitore, ma non ha una struttura interna che dipende da quest'ultima.
Lo stereotipo più usato è ¡¡use¿¿ Oppure si può rappresentare il fatto che un'operazione della classe cliente ha argomenti che appartengono al tipo di un'altra classe.