PROGETTAZIONE DEL SOFTWARE

1. **WATERFALL MODEL**

È un modello tradizionale basato su **fasi** ed **attività.** Utilizza la **progressione lineare** per passare da una fase a quella successiva. Questo modello è facilmente praticabile e controllabile.

È caratterizzato da due gruppi di fasi:

- **FASI INIZIALI**
    - **STUDIO DELLA FATTIBILITA':** produce il **documento di fattibilità** il quale è basato sull'idea che ci si fa del problema e che contiene:
        - **Descrizione e analisi preliminare del problema**
        - **Possibili modi per risolvere il problema**
        - **Costi**
    - **ANALISI E SPECIFICA DEI REQUISITI:** viene effettuata un'**analisi del dominio** e vengono valutati con l'utente i **requisiti specifici**. Questa fase produce il **RASD** che risponde alle cinque WH-Question, contiene i **requisiti funzionali**, i **requisiti non funzionali** e i **vincoli dati dall'utente**, e deve avere le seguenti proprietà:
        - **Precisione**
        - **Completezza**
        - **Congruenza**
        - **Comprensibilità**
        - **Modificabilità**

      Durante questa fase è possibile allegare un **manuale preliminare per l'utente** e una **piano di test di sistema.**

    - **DESIGN:** durante questa fase vengono definite l'**architettura del software** e lo **scopo del progetto** e viene prodotto il **documento di progetto.**
- **CODIFICA**
- **FASI FINALI**
    - **INTEGRATION TEST**
    - **MESSA IN OPERA**
    - **MANUTENZIONE:** possono essere svolte tre tipi di attività:
        - **Attività correttiva**
        - **Attività adattativa**
        - **Attività perfettiva**

2. **STANDARD ISO9126**

È uno standard internazionale che si occupa della **qualità del software** che viene definita in base a 6 caratteristiche indipendenti tra loro ciascuna delle quali è a sua volta definita da sotto-caratteristiche.

1. **FUNZIONALITA':** insieme degli attributi che riguardano l'esistenza di un complesso e le specifiche proprietà.
2. **AFFIDABILITA':** insieme delle caratteristiche degli attributi che riguardano la capacità del progetto software di mantenere il livello di prestazioni con condizioni valutate rispetto ad un insieme di utenti specifici e limiti di tempo fissato.
    a. **Maturità:** insieme di attributi che riguardano la frequenza di fallimenti dovuti ad errori presenti nel software.
3. **USABILITA':** insieme di attributi che riguardano lo sforzo necessario per utilizzare il prodotto e per dare una valutazione individuale di tale utilizzo.
4. **EFFICIENZA:** relazione esistente tra prestazioni e risorse, cioè indica il numero di transazioni eseguite.
5. **MANUTENIBILITA':** insieme degli attributi che riguardano lo sforzo necessario ad eseguire modifiche.
6. **PORTABILITA':** insieme degli attributi che riguardano la capacità del prodotto software di essere trasferito da un ambiente ad un altro.

3. CLASSI ED OGGETTI

CLASSE: fornisce dei **servizi ai clienti** e definisce un **tipo**, cioè l'insieme dei valori, le operazioni che possono essere effettuate sui dati e la dimensione dei dati.
OGGETTO: è una **variabile** il cui tipo è una classe, cioè è un'istanza di una classe.

Le **classi** e gli **oggetti** contengono come elementi i **dati membro** e come metodi i **metodi membro.**

TIPOLOGIE DI ELEMENTI

- **ELEMENTI DI ISTANZA:** esiste una copia indipendente per ogni oggetto dell'istanza della classe.
- **STATIC: ELEMENTI DI CLASSE:** esiste una copia unica comune a tutti gli oggetti della classe.
- **PUBLIC: ELEMENTI PUBBLICI:** sono elementi visibili a tutte le classi e a tutti gli oggetti.
- **PROTECTED: ELEMENTI PROTETTI:** elementi a cui possono accedere tutte le classi, le sottoclassi, i metodi e gli oggetti appartenenti allo stesso package.
- **ELEMENTI CON VISIBILITA' LIMITATA AL PACKAGE:** favorita dall'incapsulamento.
- **PRIVATE: ELEMENTI PRIVATI:** elementi non visibili all'esterno della classe o dell'oggetto. (**Consigliato per metodi e dati membro**).

TIPOLOGIE DI METODI MEMBRO

- **SET:** metodo membro non privato che definisce il valore dei dati membro.
- **GET:** metodo membro non privato che accede al valore dei dati membro.
- **UTILITY:** metodo di servizio privato.
- **COSTRUTTORI:** sono metodi chiamati automaticamente durante l'esecuzione all'atto della **creazione di un oggetto** per **inizializzare i dati membro dell'oggetto.** Sono metodi **void** che possiedono lo **stesso nome della classe** e vengono chiamati una volta sola per oggetto e se non vengono dichiarati viene chiamato quello della superclasse. I costruttori possono essere **PARAMETRICI** oppure **DI DEFAULT.**
- **FINALIZE: FINALIZZATORE:** è un metodo membro che viene chiamato dal **garbage collector** all'atto della **distruzione** di un oggetto che avviene quando l'oggetto **perde significato** oppure quando viene **eliminata la possibilità di accedere all'oggetto.**

THIS: indica un riferimento tramite il quale è possibile raggiungere un oggetto e viene utilizzato al termine di un metodo per restituire il riferimento all'oggetto che si è manipolato oppure all'interno di un metodo per permettere di accedere a quei dati membro di una classe o di un oggetto che sono nascosti dai dati locali dei metodi che hanno lo stesso nome.

EREDITARIETA': gli oggetti di una classe possono anche essere oggetti di una **sottoclasse**. Nelle sottoclassi si possono aggiungere nuovi dati e metodi e si possono ridefinire gli oggetti protetti e pubblici. Per riferirsi alla superclasse si utilizza la parola **SUPER** e all'inizio dell'esecuzione di un costruttore della sottoclasse viene sempre chiamato un costruttore della superclasse, mentre bisogna chiamare esplicitamente il finalizzatore della superclasse.

- **METODI E CLASSI FINAL:** non possono essere ridefinite in sottoclassi.
- **METODI E CLASSI ABSTRACT:** devono essere ridefiniti in sottoclassi.

INTERFACE: INTERFACCIA: designano scheletri di classi in cui i **metodi** sono tutti **PUBLIC ABSTRACT** e i **dati** sono obbligatoriamente **PUBLIC FINAL STATIC.** Le interfacce servono per descrivere classi astratte che descrivono tipi che possono essere usati dalle classi, non possono essere istanziate, sono legate tra loro da relazioni di ereditarietà e possono essere estese o implementate da più interfacce.

- **Conversione di tipo e riferimenti agli oggetti:** il riferimento ad un oggetto di una **sottoclasse** viene convertito **implicitamente** in un riferimento ad un oggetto di una delle sue **superclassi**. Il riferimento all'oggetto della superclasse non può usare i dati e i metodi aggiunti nella sottoclasse.

Polimorfismo: permette il riuso del software perché permette di utilizzare il metodo o il dato membro corretto per ogni oggetto tramite il **binding dinamico** che permette di eseguire il metodo con la stessa **segnatura,** che viene definita dal **tipo ritornato, nome del metodo, ordine** e **tipo dei parametri in ingresso,** del metodo che è stato dichiarato nella classe o nella sua superclasse.

4. INTRODUZIONE AL DESIGN DEL SOFTWARE

L'attività di design del software produce l'architettura del software che definisce il sistema in termini di componenti computazionali.

DUE LIVELLI DI ASTRAZIONE

1. **MECCANISMI:** definiscono di cosa sono fatte le componenti, cioè descrivono come un'architettura è stata costruita.
    a. **MODULI:** sono la parte di sistema che si occupa di fornire servizi computazionali ad altri moduli e l'insieme dei servizi forniti viene chiamato interfaccia.
        i. **TRE TIPI DI RELAZIONI:**
            1. **USES:** Permette l'utilizzo da parte dei moduli di servizi esportati da altri moduli.
            2. **IS_COMPONENT_OF:** descrive tramite come sono aggregate tra loro i moduli.
            3. **INHERITS_FROM:** descrive la gerarchia tra i moduli.
2. **STILI:** sono degli schemi di programmazione che permettono di riconoscere la modalità di risoluzione adottata per organizzare il software, cioè indicano ciò che caratterizza un'architettura.

PRINCIPI BASE DELLA PROGETTAZIONE DEL SOFTWARE

- **COME SELEZIONARE I MODULI:** i moduli devono essere coesi, devono minimizzare l'accoppiamento e le dipendenze tra di loro.
- **COME DEFINIRE LE INTERFACCE IN BASE AI MODULI:** bisogna rendere pubblico solo lo stretto necessario.
- **COME DEFINIRE LE RELAZIONI USES**

Concetti e principi chiave della progettazione del software

Decomposizione, astrazione, information hiding, moduli, estendibilità, architettura della macchina virtuale, gerarchia, famiglie e sotto gruppi di programmi.

Obiettivi principali

Gli obiettivi del design sono dominare la complessità, cioè prendere un problema di grandi dimensione e dividerlo in sotto-problemi unendo poi le soluzioni, e l'evoluzione, cioè prevedere i cambiamenti negli algoritmi, nelle strutture dati, nella macchina astratta, nell'ambiente e i cambiamenti già previsti.

5. DESIGN

Design object-oriented: utilizza l'ereditarietà per trovare le parti che accomunano le componenti.

DESIGN DA CONTRATTO

I moduli e le classi vengono definite tramite un **contratto** che è un **accordo** preso tra **cliente** e **contractor** che serve per definire degli obblighi al fine di ottenere dei benefici.

- **PRECONDIZIONI:** possono essere **deboli** o **forti**, anche se è consigliabile una via di mezzo e il cliente deve garantire che le proprietà vengano soddisfatte.
- **POSTCONDIZIONI:** il contractor deve garantire che le proprietà promesse vengano implementate nel metodo.

6. ARCHITETTURA

Gli **stili di progetto** permettono di avere un insieme di forme di progetto di facile riconoscimento che sono:

Componenti: clienti, server, filtri, strati, database.

Connettori: chiamata di procedure, trasmissione di eventi, protocolli, pipes.

- **ARCHITETTURA FUNZIONALE:** il sistema viene scomposto in operazioni astratte che possono chiamarsi a vicenda e che sono degli schemi statici che esprimono chi sta chiamando chi, ma non l'ordine in cui avviene la chiamata. I connettori utilizzato sono operazioni di chiamata o di return e vengono aggiunti tramite dati condivisi. L'architettura funzionale può essere sviluppata in maniera **tradizionale,** in cui le funzioni sono subordinate a programmi monolitici e i dati utilizzati sono comuni durante l'esecuzione della routine, oppure in maniera **object oriented** in cui le funzioni sono dei metodi che appartengono alle classi e i dati utilizzati sono dati delle classi.
- **STILE A STRATI:** il sistema è organizzato in livelli astratti formando una gerarchia data da relazioni uses di macchine astratte.
- **PIPES E FILTRI:** è un'architettura computazionale tra funzioni pure. Quest'architettura è di tipo compositivo e le modifiche sono abbastanza semplici in quanto si possono sostituire facilmente i filtri, tuttavia essa non presenta persistenza nel tempo.
- **SISTEMI BASATI SU EVENTI:** Sono sistemi che hanno un manager di eventi e delle componenti target che si occupano degli eventi stessi e che possono essere aggiunte o tolte in qualsiasi momento senza richiedere l'intervento del manager. Questo sistema è basato sulla reattività di una certa azione ma gli eventi non sono coordinati.
- **SISTEMI A REPOSITORY:** sono sistemi in cui le varie componenti gestiscono un ammasso di dati e comunicano tra loro attraverso un repository. Le componenti sono la parte attiva mentre il repository è la parte passiva. In questo sistema è presente un gestore delle transazioni che legge in input le transazioni e richiama le funzioni aggregate.
- **BLACKBOARD:** le componenti leggono e scrivono nel blackboard il cui stato cambia in base all'attivazione delle varie componenti.

7. UML: UNIFIED MODELING LANGUAGE

MODELLI: descrivono in maniera visuale il sistema da costruire e sono un veicolo per la comunicazione.

La **modellizzazione visuale** è basata sul **business process** che è formato da tre componenti: **Richiesta, Ordine** e **Consegna** e permette l'**astrazione** cioè la descrizione di un sistema che ne riporta solo le caratteristiche rilevanti.

MODULARITA': è un modo per affrontare la complessità di un sistema tramite il divide et impera ed è un sistema è modulare se è diviso in parti che hanno una sostanziale autonomia individuale e una ridotta interazione delle parti.

- **CRITERI**
  - **Scomponibilità modulare**
  - **Componibilità modulare**
  - **Comprensione modulare**
  - **Continuità modulare**
  - **Protezione modulare**

METODI: rappresentano le unità linguistiche dei moduli e un modulo deve comunicare con il minor numero di moduli e le interfacce devono essere piccole ed esplicite.

DESCRIZIONI

Sono prodotte in svariate fasi del processo

- **Descrizione:** insieme di affermazioni riguardo una qualche realtà di interesse.
- **Definizione del problema:** descrizione del problema da risolvere.
- **Progetto del sistema:** descrizione del programma da realizzare.

LINGUAGGIO DI DESCRIZIONE: è un linguaggio adatto a scrivere descrizioni e che ha le seguenti caratteristiche:

- **Completezza:** strumenti per descrivere tutti gli aspetti di interesse.
- **Accuratezza:** descrizione precisa.
- **Consistenza:** evitare contraddizioni.
- **Comprensibilità:** facilmente comprensibile.
- **Formalità:** rigore con cui sono definite la sintassi e la semantica del linguaggio.
    - **Informale:** linguaggio naturale.
    - **Semi-formale:** ha sintassi ma poca semantica.
    - **Formale:** ha una sintassi e una semantica rigorosa.
- **Stile:** aspetto del sistema più facile da descrivere utilizzando il linguaggio.
    - **Descrittivo:** definisce le proprietà desiderate.
    - **Operazionale:** definisce il comportamento desiderato.

UML: è un linguaggio per definire, progettare, realizzare e documentare i sistemi software ad oggetti. È un insieme di linguaggi che consentono di descrivere e modellare tutti o quasi gli aspetti rilevanti di un sistema. Esso utilizza una notazione grafica ed è un linguaggio semi- formale.

- **VISTE DI UN SISTEMA**
    - **Casi d'uso:** descrive i casi d'uso che forniscono valore agli utenti e i casi d'uso essenziali vengono usati come proof of concept per l'architettura di implementazione.
    - **Strutturale:** rappresenta gli elementi strutturali per implementare la soluzione.
    - **Comportamentale:** rappresenta l'interazione dinamica tra i componenti del sistema.
    - **Implementativa:** descrive i documenti di implementazione dei sotto-sistemi logici definiti nella vista strutturale.
    - **Ambientale:** rappresenta la tipologia hardware del sistema e definisce come vengono assegnati i componenti software ai nodi hardware.

CLASS DIAGRAM: descrive le classi, o meglio le relazioni tra di esse ed è un diagramma stativo. Il processo di design è iterativo e mano a mano viene personalizzato con l'aggiunta di attributi che sono caratteristiche della classe. Inoltre la classe è formata da nome e metodi che ne descrivono il comportamento. Ci sono anche delle operazioni che sono le funzioni o trasformazioni applicabili ad un'istanza di una classe e sono i metodi get e set.

- **RELAZIONI FRA CLASSI**
    - **ASSOCIAZIONI:** è un canale di associazione bidirezionale che connette delle classi che hanno qualcosa in comune e sono caratterizzare da **molteplicità,** che definisce il numero di istanze che prendono parte alla relazione, **navigabilità,** che indica la terminazione dell'associazione, **ruolo** e **qualificatore.**
    - **AGGREGAZIONI:** sono un caso particolare di associazione che riguarda oggetti. Una relazione di **composizione** è un'aggregazione forte.
    - **GENERALIZZAZIONE:** riguarda le classi ed è una sorta di ereditarietà.

INTERFACCIA: è una dichiarazione di un seti di caratteristiche ed obblighi pubblici. Essa specifica un contratto che deve essere rispettato da tutte le istanze e può essere **fornita** o **richiesta.**

CLASSI TEMPLATE: rappresentano una classe con uno o più tipi parametrici e possono essere **classi utility,** cioè classi di servizio, **enumerazioni,** cioè classi che definiscono dei valori oppure **stereotipi**, che rappresentano classi che si specializzano nella semantica di elementi predefiniti in UML come ad esempio **entity,** che è una classe passiva, **control,** che è una classe le cui istanze controllano le interazioni, oppure **boundary** dove la classe svolge un ruolo da intermediario tra attori e altre parti.

OBJECT DIAGRAM: questi diagrammi descrivono singole istanze di classi e associazioni rappresentate in un particolare class diagram. Un'associazione sono dette **link** cioè una connessione fisica o concettuale tra due istanze.

PACKAGE: sono contenitori di elementi UML e definiscono un **namespace.**

- **Relazioni tra package**
  - Generalizzazione
  - Dipendenza: relazione d'uso
  - Aggregazione
  - Merge: indica la fusione di due package da usare quando sono stati definiti in due modi diversi.
  - Access: garantisce solo la possibilità di creare references.
  - Import: il sistema controlla che non ci siano conflitti.

USE CASE DIAGRAM: sono diagrammi che definiscono le funzionalità, i processi principali e le reazioni di un sistema. Essi sono un'unità funzionale del sistema che fornisce un valore ad un utente.

Il **documento dello use case** è caratterizzato da **attori, titolo, autori, descrizione, requisiti, vincoli** e **scenari.**

- **Relazioni tra use case**
  - **Generalizzazione:** ereditarietà.
  - **Inclusione:** il comportamento dello use case target viene incluso nella sequenza di azioni svolte dalle istanze dello use case di base.
  - **Estensione:** le funzionalità vengono estese ad un altro use case.