Documento Design

• Progettare il sistema
• Scrivere un documento in formato libero che includa :
• Diagrammi delle classi
• Diagrammi di sequenza per le interazioni più significative
• Eventuali altri diagrammi se necessari
• Commenti ai diagrammi, discutendo le scelte effettuate in termini di coesione, accoppiamento e principi di buona progettazione

La coesione e l'accoppiamento sono concetti fondamentali nel design del software e nell'ingegneria dei requisiti, essenziali per creare sistemi robusti e manutenibili. Ecco una panoramica:

Coesione

La coesione si riferisce al grado in cui gli elementi all'interno di un modulo (classe, metodo, ecc.) sono correlati tra loro. Una coesione elevata indica che i componenti di un modulo sono strettamente correlati e hanno un singolo scopo ben definito. Ecco alcune caratteristiche della coesione:

- **Alta coesione**: Ogni modulo svolge un compito specifico e ben definito, facilitando la comprensione, la manutenzione e il riutilizzo del codice. Ad esempio, una classe `Ordine` dovrebbe gestire solo le funzionalità relative agli ordini, senza includere logiche non correlate.
- **Bassa coesione**: Un modulo svolge compiti multipli e non correlati, rendendo il codice complesso e difficile da mantenere. Ad esempio, una classe che gestisce ordini, clienti e pagamenti violerebbe il principio della coesione.

Accoppiamento

L'accoppiamento si riferisce al grado di interdipendenza tra i moduli di un sistema. Un accoppiamento basso è desiderabile poiché i moduli possono funzionare in modo indipendente l'uno dall'altro. Ecco alcune caratteristiche dell'accoppiamento:

- **Accoppiamento basso (Loose coupling)**: I moduli interagiscono tra loro tramite interfacce ben definite e limitano le dipendenze esterne. Questo rende il sistema più flessibile e facile da mantenere. Ad esempio, utilizzare interfacce o astrazioni per interagire con i moduli riduce l'accoppiamento.
- **Accoppiamento alto (Tight coupling)**: I moduli sono strettamente dipendenti l'uno dall'altro, rendendo difficile apportare modifiche a un modulo senza influenzare gli altri. Ad esempio, se una classe dipende direttamente da

un'altra classe concreta, le modifiche a una delle due classi possono avere effetti a catena.

Principi di Buona Progettazione

Per garantire un design di qualità, ci sono diversi principi di buona progettazione che possono essere seguiti:

1. **Single Responsibility Principle (SRP)**: Una classe dovrebbe avere una sola responsabilità o ragione per cambiare.
2. **Open/Closed Principle (OCP)**: Le entità software (classi, moduli, funzioni) dovrebbero essere aperte all'estensione ma chiuse alla modifica.
3. **Liskov Substitution Principle (LSP)**: Le classi derivate dovrebbero essere sostituibili con le loro classi base senza alterare il corretto funzionamento del programma.
4. **Interface Segregation Principle (ISP)**: I clienti non dovrebbero essere costretti a dipendere da interfacce che non utilizzano. È meglio avere interfacce più piccole e specifiche.
5. **Dependency Inversion Principle (DIP)**: Le entità di alto livello non dovrebbero dipendere da quelle di basso livello. Entrambe dovrebbero dipendere da astrazioni.

Descrizione Diagramma delle Classi

Commento:

Descrizione Diagramma di Sequenza

Commento:

Storico delle versioni

Versione	Data	Descrizione	Autori

1	27/11/2024	Creazione del documento; Primi schizzi del diagramma delle classi;	De Rosa, Zito
2	29/11/2024	Realizzazione diagramma delle classi; Abbozzo del mockup;	De Rosa, Loffredo, Zito
3	02/12/2024	Aggiornamento diagramma delle classi; Creazione Doxyfile;	De Rosa, Loffredo, Zito
4	03/12/2024	Realizzazione del sequence diagram;	De Rosa, Loffredo, Zito
5	04/12/2024	Revisione delle classi nel codice; Correzione diagramma di Sequenza;	De Rosa, Loffredo, Zito