**Software Design**

Strutturare il design del software è un procedimento che richiede tempo e attenzione ai dettagli: è necessaria una interazione tra i requisiti definiti e l’architettura del sistema, inoltre è saggio effettuare più tentativi, proprio perché non è semplice trovare una soluzione perfetta.

**ASTRAZIONE**

Siccome i dettagli non sono importanti a livello di design, è necessario effettuare un’operazione di *Astrazione*, in particolare *Astrazione dei Dati*: si mira a trovare una gerarchia nei dati a disposizione, per evitare di complicare la struttura di design con le varie specifiche.

**ACCOPPIAMENTO/COESIONE**

Al fine di ottenere un buon sistema software, bisogna fare in modo che il progetto abbia un elevato livello di *coesione* (misura del grado in cui gli elementi del modulo sono funzionalmente correlati, ovvero che mirano all’esecuzione di un unico aspetto) e un basso livello di *accoppiamento* (misura del grado di interdipendenza tra i moduli, ovvero che gli elementi non dipendono fortemente tra di loro); ciò permette una comunicazione più veloce e semplice, le modifiche non influenzano troppo altri moduli e la comprensibilità dell’intero sistema diventa più facile. Nel nostro progetto si parla di *coesione procedurale*, perché gli elementi devono seguire un determinato ordine per essere eseguiti (per esempio, il passaggio di un paziente dalla Lista Graduatoria alla Lista Operatoria), e *accoppiamento di dati* (i dati vengono passati tra i vari moduli).

In ogni fase del design, è lecito domandarsi quali componenti del sistema devono essere informati e quali no (*information hiding*): ciò permette un abbassamento del grado di accoppiamento (i componenti non sono sempre dipendenti gli uni dagli altri) e un aumento del grado di coesione (il segreto unisce le singole parti del modulo)

Si fa riferimento al paradigma MVC (Model-View-Controller).

Immagine che contiene diagramma, linea, testo, Piano

Descrizione generata automaticamente

***Design Pattern***

I pattern sono definiti come “soluzioni a problemi ricorrenti” e permettono di ottenere una definizione più semplice del design applicato; i pattern descrivono il contesto di applicazione, definiscono il problema da risolvere, forniscono soluzioni a tale problema e mostrano i risultati/vincoli ottenuti. Si possono definire anche gli *antipattern,* situazione che è preferibile evitare perché causano problemi nel funzionamento del sistema.

Per implementare il framework del progetto sono stati utilizzati vari tipi di design pattern:

* **General Hierarchy Pattern** (permette di trattare un gruppo di oggetti come se fossero l’istanza di un singolo oggetto, andando a definire una gerarchia tra le classi, come, per esempio, tra i medici e gli infermieri operanti nell’intervento);
* **Player Role Pattern** (sfruttando le classi astratte, si può far “giocare” ad un attore diversi ruoli, per esempio il medico che registra i pazienti, li inserisce nella Lista Operatoria e compila il verbale dell’intervento);
* **Facade Pattern** (pattern strutturale, il quale permette, attraverso un’interfaccia più semplice, l’accesso a sottoinsiemi che espongono interfacce complesse e diverse tra loro, come i pazienti e il Sistema di gestione dell’Ospedale per accedere alle informazioni richieste. La facciata permette di nascondere i meccanismi complessi dell’operazione dal lato *Client*);
* **Factory Pattern** (pattern creazionale utile per creare un framework che possa gestire diverse classi e oggetti, indipendentemente dall’applicazione: le sottoclassi sono libere di scegliere quale oggetto istanziare);
* **Delegation Pattern** (questo pattern permette di riciclare un metodo di una classe già usato in una nuova classe, come ad esempio medici e cabina di regia quando devono compilare le liste. Tale pattern garantisce una riduzione dei costi di sviluppo, grazie al riuso del codice);
* **Observer Pattern** (pattern comportamentale in cui viene definita una dipendenza uno a molti fra oggetti diversi: l’oggetto osservato (soggetto) può cambiare continuamente stato; pertanto, gli altri oggetti (osservatori) devono essere notificati di tali cambiamenti e sono chiamati a gestire gli eventi generati dal soggetto, nel nostro caso, le Liste Operatorie e Graduatorie possono continuamente cambiare, quindi i medici e la cabina di regia, coloro che operano direttamente sulle liste, devono essere informati del nuovo stato. Un discorso simile può essere fatto per il verbale dell’Intervento);

Nel nostro progetto sono stati sfruttati maggiormente il *Factory Pattern*, per la generazione e gestione dati, e il *Facade Pattern,* per creare un’interfaccia chiara e semplice rispetto al sistema complesso.