**Software Design**

Strutturare il design del software è un procedimento che richiede tempo e attenzione ai dettagli: è necessaria un’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*, più precisamente *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, oltre al fatto che le modifiche non influenzano esageratamente 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.

Nell’implementazione del framework è stato utilizzato un solo tipo di design pattern:

**Delegation Pattern**

Pattern utile nelle situazioni in cui, progettando un metodo di una nuova classe, si riconosce che un’altra classe già esistente esegue la stessa operazione. Per permettere la “delegazione” dalla classe originale alla nuova, viene istanziato un oggetto all’interno della classe da implementare: tale oggetto permette di rieseguire l’operazione desiderata, adattandola al contesto in cui ci si trova; tutto ciò permette di minimizzare i costi di sviluppo, attraverso il riciclo di codice.

Nel nostro progetto, tale pattern è stato utilizzato per fare in modo che le classi GUI (dove vengono specificati i metodi riguardanti login di Medico, modifica delle liste operatorie e compilazione verbale) trasferiscano i propri metodi al data Service (GestioneSale): per comporre le liste di più pazienti, è necessario tenere conto delle operazioni definite all’interno delle classi GUI; pertanto, le classi GUI delegano i propri metodi verso il Data Service perché possa costruire le liste/tabelle con tutte le informazioni ricercate.