**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.

L’analisi della qualità del design è stata effettuata mediante *Structure101*. I risultati sono riportati di seguito. È importante, però, specificare che nell’analisi non è stato tenuto in conto il modulo contenente il codice generato da Jooq.

**RELAZIONE TRA USI**

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

I due grafi mostrano le relazioni tra i moduli (a sinistra) e tra le classi (a destra) del software. Da questi si può sicuramente evincere il fatto che il programma è ben strutturato, in quanto non sono presenti cicli e, anzi, ci sia una gerarchia rigida tra i moduli e tra le classi.

La buona struttura del codice è confermata anche dal seguente grafico, che rappresenta l’overcomplessità strutturale del software.

Immagine che contiene testo, schermata, Carattere, Sistema operativo

Descrizione generata automaticamente

Il programma risulta infatti essere molto strutturato, con l’unico piccolo difetto sull’asse orizzontale, che rappresenta l’eccessiva dimensione all’interno delle classi. In questo caso, l’unica classe ad avere questo difetto è la classe *DataService*.

**ACCOPPIAMENTO**

Al fine di ottenere un buon sistema software, bisogna fare in modo che il progetto abbia un elevato livello di *coesione* e un basso livello di *accoppiamento*; 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, e *accoppiamento di dati* (i dati vengono passati tra i vari moduli).

In particolare, il seguente grafo mostra le chiamate da un modulo all’altro e permette di analizzare l’accoppiamento del software.

Immagine che contiene diagramma, linea, mappa, Piano

Descrizione generata automaticamente

Dal grafo traspare come ci sia basso accoppiamento tra i moduli riguardanti l’interfaccia grafica (view) e tra model e db\_sqlite. C’è invece un alto accoppiamento tra model e i moduli dell’interfaccia. Questo fatto però, è inevitabile, poiché le interfacce devono notificare in molte occasioni la classe DataService, contenuta in model, poiché fa da controller.

***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, ad esempio, 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.