**Manutenzione**

Creare un progetto software perfettamente funzionante non è impresa facile e può capitare di riscontrare errori nella struttura/funzionamento del codice solo a consegna effettuata; a questo proposito, si sfrutta il processo definito **Manutenzione,** il cui scopo principale è correggere gli errori postumi alla consegna e adattare il software alle mutevoli esigenze (degli utenti e dell’ambiente).

Affinché l’attività di manutenzione sia completa e risponda ad ogni possibile aspetto, occorre considerare tutti i tipi di manutenzione disponibili:

* *Correttiva*: per correggere gli errori del progetto;
* *Adattiva*: il progetto deve adattarsi ai cambiamenti dell’ambiente, sia hardware che software;
* *Perfettiva*: il progetto deve adattarsi alle esigenze degli utenti (possono mutare in base alla singola persona);
* *Preventiva*: per aumentare il grado di manutenibilità del sistema (aggiorna la documentazione, aggiunge commenti…);

Qui sotto vi è riportato un grafico a torta che mostra l’importanza data ad ognuno dei tipi di manutenzione rispetto al nostro progetto.

Immagine che contiene schermata, diagramma, Policromia, Elementi grafici

Descrizione generata automaticamente

Come si può notare, la manutenzione *correttiva* occupa la maggior parte dello spazio dedicato, proprio per risolvere errori in seguito alla consegna, tipicamente fault nel codice; la manutenzione *preventiva* è importante, affinché il grado di manutenibilità aumenti sempre di più e ciò porti ad un miglioramento di qualità (la documentazione deve essere in grado di spiegare nel dettaglio il funzionamento del progetto). Siccome il nostro progetto non è ancora pronto per la commercializzazione, durante la manutenzione non si tiene molto conto dei cambiamenti dell’ambiente di sviluppo o delle richieste degli utenti, ma si possono fare delle ipotesi circa l’utilità complessiva.

L’attività di manutenzione è iniziata subito dopo la consegna dell’elaborato e ogni membro del gruppo ha lavorato in maniera attiva sulle singole componenti, in particolare sull’analisi del codice e correzione di errori; le principali ricerche rispetto alla manutenzione riguardano:

* Conoscenza insufficiente del Sistema e del Dominio;
* Utilizzo improprio di gerarchie di classe;
* Utilizzo di codice “eccessivo” (non importante per la realizzazione delle richieste);
* Assenza di commenti e Documentazione “povera” (spiegazioni non complete);
* Utilizzo improprio di Tools o attività di Refactoring;

L’obiettivo finale è raggiungere un sistema non troppo complesso (a livello di struttura), che sia dettagliato a livello di documentazione, che funzioni correttamente e fare in modo che non si verifichino fallimenti frequenti (ciò è possibile grazie ad attività di monitoraggio e aggiornamento continuo, unita all’attività di testing).

***Refactoring***

Il refactoring è un processo attraverso il quale il codice viene ristrutturato, senza aggiungere o modificare funzionalità, ma migliorandone la qualità interna ed eliminando i cosiddetti “bad smells”, come per esempio:

* Codice duplicato;
* Uso di variabili globali;
* Utilizzo di metodi troppo lunghi;
* Classi pigre (impiegano più tempo del dovuto a completare una funzione);
* Invidia tra le classi (porterebbe a scontri tra più classi, compromettendo la qualità del sistema);

Il procedimento permette di diminuire il codice utilizzato, garantendo un aumento della qualità finale del prodotto e una riduzione di necessità di manutenzione; di conseguenza, si è più preparati ad anticipare nuovi cambiamenti e si evita di usare determinati frammenti di codice, che altrimenti sarebbero “inutili” o eccessivi. Il refactoring è composto da una serie di piccole azioni, che, se eseguite in sequenza, producono una ristrutturazione significativa (l’essere composto da procedimenti corti comporta una diminuzione di possibilità che l’intero processo vada storto).

I principali metodi di refactoring utilizzati sono:

* Extract Method;
* Extract Local Variables;
* Push Down Method;
* Push Down Field;
* Remove Method;
* Remove Field;

L’utilizzo di questi metodi permette di estrarre metodi/variabili locali da una superclasse, importarli in una sottoclasse e rimuovere campi/metodi non strettamente necessari per il conseguimento delle funzioni in atto; insieme, questi metodi evitano la riscrittura di codice duplicato e minimizzano i costi di progettazione.