Presentazione Sparta

Durante lo sviluppo di un software vengono presi in considerazione gli elementi conosciuti, mentre degli elementi che risultano sconosciuti se ne occupa la gestione del rischio; un rischio è una variabile che può assumere valori tali da compromettere il progetto.

Nel documento di risk list sono stati elencati i maggiori rischi per il sistema Virtual Mechanic e per ognuno di essi sono state pianificate delle misure di risoluzione o di contenimento, che ora verranno illustrate;

* Nel primo caso, se un componente abbandona il progetto, si potranno verificare dei rallentamenti nello sviluppo. Per ridurre il rischio i task verranno svolti da più persone in parallelo, in modo da evitare ritardi in caso di abbandono; **Contingency Plan:** Tutti i componenti del team hanno le conoscenze basilari necessarie al completamento del progetto.
* Nel secondo caso, se avviene un guasto hardware o una perdita dati possono essere perse varie componenti del software, che dovranno essere riscritte causando ritardi più o meno significativi. Per ridurre il rischio, oltre ad essere salvati sulle macchine in locale, i dati vengono salvati su repository online; in caso di emergenza, basta recuperare le copie di backup dal repository o dalle altre macchine funzionanti
* Nel terzo caso, un cambiamento di specifiche in corso d’opera può comportare ritardi in quanto sono necessarie correzioni al codice. Per ridurre i rischi di ritardi, si richiederà al committente di approvare il documento di Change request; in caso di emergenza, verranno effettuate le correzioni riassegnando le risorse
* Nel quarto caso, il ricorso a tecnologie innovative o poco note può portare a rallentamenti, dovuti al naturale apprendimento delle nuove tecnologie. Per ridurre il rischio di ritardi, si cerca utilizzare tecnologie conosciute dove possibile; in caso di emergenza, si cerca di assumere le conoscenze necessarie in tempi ragionevoli, senza destinare tutte le risorse umane a tale scopo.

Il documento di riuso prende in considerazione le scelte effettuate per utilizzare codice già scritto, in modo da ridurre i tempi di progettazione; Gli approcci utilizzati sono:

* Design pattern, che sono soluzioni progettuali generali a problemi ricorrenti [ Si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software ]
* Generatori (getter, setter, 9patch), software che sono in grado di generare al loro volta software parametrizzati in base alle specifiche fornite dall’utente.
* Riuso di classi e codice già presente in altri programmi
* Framework, un'architettura logica di supporto su cui un software può essere progettato e realizzato

Gli effetti del riuso nel progetto Virtual mechanic, sono completamente cannati!

Passo ora la parola a Federico

I diagrammi delle sequenze descrivono le interazioni tra gli oggetti organizzate in sequenza temporale.

I seguenti diagrammi sono stati realizzati con un approccio ad alto livello.

I primi due diagrammi descrivono l’interazione tra l’utente e l’applicazione Titanic Assistance:

* nel primo caso l’utente, selezionando il problema riscontrato, trova una soluzione soddisfacente fornita dall’applicazione.
* nel secondo caso l’utente, selezionato il problema riscontrato, non trova soluzioni soddisfacenti e quindi viene visualizzato un codice e la possibilità di chiamare l’assistenza tramite l’app ; l’assistenza fornirà all’utente una soluzione adeguata

I seguenti diagrammi descrivono l’interazione tra l’operatore e l’applicazione Neptune Rescue:

* nel primo caso l’operatore, inserendo dei dati da elaborare, riceve dall’aplicazione i dati elaborati
* nel secondo caso, l’operatore riceve una telefonata da parte di un utente e fornirà una risposta adeguata al caso; durante tale fase, potrà comunque svolgere le operazioni descritte dal primo diagramma

Passo ora la parola a Federico

Prima di affrontare i principi SOLID implementati, verrà fornita una breve spiegazione di tali principi

* **Singola responsabilità:** Una classe dovrebbe avere una sola ragione per cambiare
* **Apertura/chiusura:** Le entita dovrebbe essere aperte per le estensioni, ma chiuse alle modificazioni.
* **Principio di Liskov:** I sottotipi dovrebbero essere sostituibili per i supertipi; le classi figlie non devono mai rompere la definizione delle classi genitrici
* **Segregazione delle Interfacce:** I Client non dovrebbero essere costretti ad usare interfacce che non possono usare
* **Inversione delle dipendenze:** Un modulo ad alto livello non dovrebbe dipendere dai moduli a basso livello; entrambi dovrebbero dipendere dalle astrazioni.

Sia nell’applicazione Neptune Rescue che in Titanic Assistance, sono stati implementati la singola responsabilità, l’apertura/chiusura e il principio di liskov, e non è stato violato il principio di inversione delle dipendenze

Per lo svolgimento dei test, è stato programmato l’utilizzo del White Box testing, composto dalle fasi

* **Unit test**, per verificare che le singole unità di sviluppo assolvano le loro funzioni seguendo i requisiti.
* **Integration test**, per scoprire problemi che possono nascere dall’interazione dei vari componenti durante la costruzione del sistema
* **System test**, per risolvere tutti i possibili casi di ingressi non desiderati

Durante l’esecuzione dei test, in tutte e tre le fasi sono stati riscontrati alcuni problemi di scarsa rilevanza, che sono stati prontamente risolti; alcune sottofasi del System test non sono state applicate in quanto non era possibile applicare il test

Passo ora la parola a qualcuno