

Peso's takeaway

Progetto di ingegneria del software

27 gennaio 2021

Autori: Brembilla Luca, Colombo Alessandro, Pesenti Luca

Peso's takeaway

Progetto di ingegneria del software

Prima di iniziare lo sviluppo di un progetto software, dobbiamo pianificare le fasi di sviluppo con molta attenzione. Tuttavia, questa non è un'attività svolta in una singola volta, ma si andrà ad aggiornare con il tempo, mentre il progetto continua a definirsi meglio. Per questo ci serviamo di un project plan, senza che questo sia troppo rigoroso e che quindi ci impedisca di far aggiornamenti che ci piacciono man mano che il codice progredisce.

Project Plan

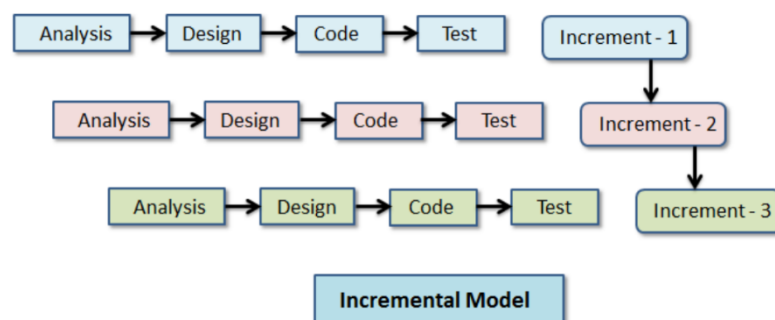
Introduzione:

Per il corso di Ingegneria del software dobbiamo portare un progetto in cui dimostriamo di sapere applicare ad un caso di studio abbastanza corposo i concetti, le metodologie e le tecniche studiate. Il progetto consiste in un'applicazione java per gestire le vendite di un negozio di panini e bibite. Le persone che svilupperanno questo progetto sono 3 (che è il numero massimo di partecipanti): Brembilla Luca, matricola n. 1067730; Colombo Alessandro, matricola n. 1066001; Pesenti Luca, matricola n. 1067681.

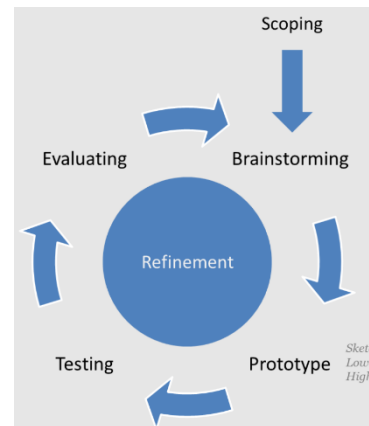
Process model:

Abbiamo deciso di utilizzare un modello agile dato che il progetto non è troppo grosso, siamo poche persone che oltre a essere colleghi sono anche buoni amici. Inoltre, non vogliamo essere limitati dal documento iniziale senza avere una specifica dei requisiti troppo estesa, ma vorremmo che questa cambiasse con le considerazioni che andremo a fare con il costruirsi del progetto e senza escludere miglioramenti che potremmo trovare lungo il percorso.

Precisamente, dati i requisiti del progetto che ci siamo già definiti in anticipo ma che ci riserviamo di modificare piano piano, abbiamo scelto il modello di sviluppo incrementale. Le funzionalità del sistema sono prodotte e consegnate al cliente in piccoli incrementi e miglioramenti. Partendo dalla situazione iniziale procediamo in piccoli passi. In ciascuno di questi passi applichiamo un approccio a fasi (simile a quello del waterfall model). In questo modo, viene messa maggior attenzione sulle feature essenziali: non avendo mai svolto un progetto in java di questo genere (collegando il progetto a un database amministrato da noi e facendo una GUI) preferiamo innanzitutto capire come fare le feature principale richieste dal cliente (noi, in questo caso), e quindi il login dei clienti e la possibilità di acquisto, e poi feature secondarie, che potrebbero anche venirci in mente con lo sviluppo incrementale progetto.



In teoria questo modello è simile al prototyping: potremmo infatti fornire un prototipo al cliente ogni qual volta un incremento ne consenta la creazione. Questo prototipo può essere utilizzato come punto di partenza per esplicitare le richieste del cliente mentre il progetto progredisce (questo punto è fondamentale per tutti i modelli agili, in cui collaborare con il cliente è meglio che negoziare all'inizio il contratto), per verificare di star rispettando i requisiti di questa fase e non aver commesso errori.



Prendiamo anche in considerazione il modello RAD: Rapid Application Development. Infatti, vogliamo che i miglioramenti siano costanti nel progetto e verificarli insieme ogni slot di tempo (1 o 2 giorni). Le attività devono essere svolte (in linea di massima, non ci sono sanzioni) entro questo tempo. Nonostante sia una clausola un po' vessatoria riteniamo sia necessaria per il buon progredire del progetto dato che ognuno ha i propri impegni e nessuno vuole rimanere indietro con la deadline del progetto.

Dall'eXtreme Programming mutuiamo il fatto che il codice sia di tutti, il fatto dei piccoli release (infatti usiamo principalmente un modello incrementale) e che cerchiamo il più possibile di uniformare per questo il nostro codice, con anche refactoring, ossia eliminando periodicamente le parti inutili e semplificando la struttura. Tuttavia, data la facilità di accesso al codice, ci piacerebbe che ognuno sia libero di lavorare quando vuole al proprio compito, quindi se uno vuole fare lavoro straordinario questo è ben accetto. Pensiamo anche che lo sviluppo debba essere guidato dai test (test driven), per cui l'insieme di test per un modulo è rieseguito ad ogni modifica del modulo. Il pair programming potrebbe esserci utile (ma non solo in coppie, anche in 3 dato che siamo solo 3) durante le sessioni settimanali o in caso di richiesta anche durante le brevi sessioni ogni 1 o 2 giorni.

Organization of the project:

Il progetto non ha un vero e proprio cliente. Noi siamo gli unici che rispondiamo del lavoro, cercando di utilizzare al massimo ciò che abbiamo appreso durante questo corso e cercando di interpretare le richieste che un possibile cliente possa fare. Per esempio, possiamo mettere una feature in cui il cliente (ossia il manager del negozio) possa gestire non solo gli ordini ma anche i suoi dipendenti, i prodotti e i suoi clienti.

Per quanto riguarda la divisione del lavoro nello sviluppo del software, non c'è un vero e proprio project manager: insieme decideremo come dividerci il lavoro, cercando di prediligere la volontà e le capacità di ognuno di noi. Utilizzeremo un modello di team agile: è favorita la creatività, non c'è una gerarchia (nemmeno quando si fa pair programming e c'è un pilota e un co-pilota). La collaborazione e la comunicazione saranno per lo più spontanee, non ci sono schemi rigidi. Data la natura del progetto non solo è permesso, ma è incentivata la sperimentazione per capire come fare le cose: sbagliando si impara. Tutto questo è possibile sia grazie al numero esiguo di sviluppatori (3), alla conoscenza reciproca di questi e al fatto di rispondere non a un cliente o a un'organizzazione ma a noi stessi, come una forma di auto-organizzazione.

La divisione in task non è decisa da nessuno in particolare, saremo noi ad auto-organizzarci e a dividerci il lavoro in base a come ci sentiamo e a come pensiamo sia più funzionale alla buona riuscita del progetto. Questa divisione sarà decisa in comune, cercando di assegnare ad ognuno cose che si hanno già svolto o che si è più interessati a fare. Per questo non si ha una divisione netta dei ruoli: una persona può

una volta fare testing e una volta fare sviluppo (capiterà spesso di fare entrambe le cose, ma anche la prima da sola).

Standards, guidelines, procedures:

Il lavoro diviso verrà eseguito singolarmente. Per aggiornarci vicendevolmente possiamo fare una riunione di 10 minuti o più ogni 1 o 2 giorni in base agli impegni di ognuno e fare anche pair programming se richiesto. Ogni 7 giorni invece si può fare una riunione più lunga, anche di diverse ore, per discutere sia degli sviluppi conseguiti e di problemi riscontrati che di sviluppo futuro e divisione del lavoro.

Le riunioni avverranno principalmente a distanza, con l'utilizzo del software Discord.

Per quanto riguarda lo scambio del codice e di documenti questo avviene mediante GitHub. La maggior parte del codice ivi caricato dovrà essere approvato da qualcun altro, per cui bisogna evitare di forzare le proprie pull request, tranne nei casi in cui si modifichi di poco il codice.

Dato che la consegna suggeriva di progettare in un linguaggio orientato agli oggetti, abbiamo deciso di utilizzare Java.

Management activities:

Le attività manageriali sono guidate da obiettivi e dalle priorità date alle varie implementazioni del progetto. Per esempio, nelle discussioni settimanali si farà molta attività manageriale tutti insieme. Per quanto riguarda le priorità possiamo suddividerle in base a quali ci aspettiamo sarebbero le richieste del cliente: così avrà maggior priorità il fatto che un cliente faccia acquisti e in secondo piano ci sarà il fatto che il cliente possa gestire i propri dipendenti. Questo aspetto è meglio definito nella specifica dei requisiti. Di volta in volta andremo poi a dividerci i task da svolgere, come detto anche precedentemente. Non ci saranno report formali (generalmente, ma non è detto che non ci servano del tutto) dell'avanzamento del progetto ma ciò sarà discusso oralmente.

Risks:

Nella realizzazione di questo sistema c'è la possibilità di entrare in contatto con alcuni rischi che potrebbero creare dei problemi durante l'utilizzo del software. Questi rischi possono essere causati da diversi fattori, da quelli amministrativi a quelli tecnologici. Il rischio principale in cui potremo incorrere riguarda la specifica dei requisiti; il fatto di non avere un cliente che ha commissionato il lavoro ci ha obbligato a pensare ai vari requisiti da poter scrivere, l'unico problema è che questi potrebbero non soddisfare la richiesta dell'utente. Anche durante la fase di design o implementazione la mancanza di un utente potrebbe causare lo sviluppo di alcuni rischi: uno tra tanti potrebbe essere la mancata approvazione del design o di una funzionalità, realizzati per il software. Per quanto riguarda la tecnologia i rischi sono rappresentati dai malfunzionamenti della parte software o della parte hardware. I rischi generati dai software possono riguardare la mancata funzionalità di XAMPP e quindi del database, oppure direttamente delle classi java, sia per la parte riguardante la comunicazione tra le classi che per la user interface.

Staffing:

Non ci saranno cambiamenti di personale. Saremo sempre 3 persone a sviluppare il software. Ciò che potremmo fare è aumentare il tempo lavorato quando ci sarà più lavoro, per sopperire all'impossibilità di assumere altre persone.

Methods and techniques:

Descriviamo qui le tecniche e i metodi che utilizzeremo per la realizzazione di questo software.

La prima tecnica fa riferimento alla specifica dei requisiti. Non avendo un cliente vero e proprio siamo costretti ad immedesimarci, in modo tale da ricavare i vari requisiti base su cui costruire il progetto.

Per quanto riguarda la parte di implementazione abbiamo deciso di utilizzare vari tools e software:

- 1) Eclipse: IDE per il linguaggio Java
- 2) StarUML: utilizzato per fare i diagrammi UML. Il diagramma delle classi sarà utilizzato come scheletro del nostro sistema.
- 3) XAMPP: è un'applicazione per hostare localmente un web server. In questo caso noi la utilizziamo per testare la nostra applicazione mediante un database MySQL.
- 4) GitHub: per la gestione e la condivisione di codice;
- 5) Google Drive: per lo scambio di file/immagini utili ai fini del progetto;
- 6) Discord: utilizzata per gli incontri online.

Per quanto riguarda la parte di testing la tecnica che utilizzeremo inizialmente sarà quella di testare pagina per pagina. Una volta fatto ciò passeremo alla fase due del testing: eseguiremo un test sotto sforzo dell'intero programma, in modo tale da notare tutti i punti in cui il sistema è più sensibile. Infine, verrà eseguito un ultimo test generale del software, eseguito da una persona casuale, per vedere come gli utenti medi potrebbero approcciarsi.

Quality assurance:

Il nostro software si basa essenzialmente su tre qualità fondamentali: portabilità, efficienza e fruibilità.

Queste verranno analizzate passo dopo passo, ogni qual volta che il design, i requisiti o

l'implementazione verranno modificati. L'obiettivo finale è quello di implementare un software in grado di essere utilizzato su diverse piattaforme, da diversi utenti, sempre al massimo delle prestazioni.

Abbiamo deciso di improntare il nostro lavoro su queste tre caratteristiche per il semplice fatto che il cliente deve essere in grado di utilizzarlo al massimo ogni volta che è disponibile.

Work packages:

Durante ogni riunione in cui sono presenti tutti i membri del gruppo vengono prima mostrati i progressi ottenuti da ognuno e successivamente si rivalutano le priorità del progetto, vengono suddivisi e assegnati nuovi incarichi. Anche se il risultato di ogni iterazione non è totalmente completo, si riconsidera comunque lo sviluppo nella sua interezza. Tutti, quindi, saranno regolarmente informati sullo stato di sviluppo.

Inoltre, ciò ci permette di adattarci in tempi brevi a cambiamenti di richieste o a nuove proposte, senza lavorare troppo a lungo su parti che potrebbero dover essere radicalmente cambiate in futuro.

Alcuni incarichi vengono distribuiti in base alle preferenze e all'esperienza di ognuno dei membri della squadra, dando comunque precedenza alle priorità del progetto.

Altri invece vengono fatti da tutti in concomitanza con altre attività, come l'implementazione e il testing di parti precedenti. In questi casi è possibile avvisare chiunque nella squadra di eventuali errori da sistemare o cambiamenti secondari necessari, così che sia possibile implementarli al più presto senza dover aspettare la riunione successiva.

Come è facile intuire, non c'è una vera e propria gerarchia all'interno del gruppo, ognuno ha la libertà di lavorare quando è più comodo per lui, a patto di rispettare le consegne assegnate di comune accordo.

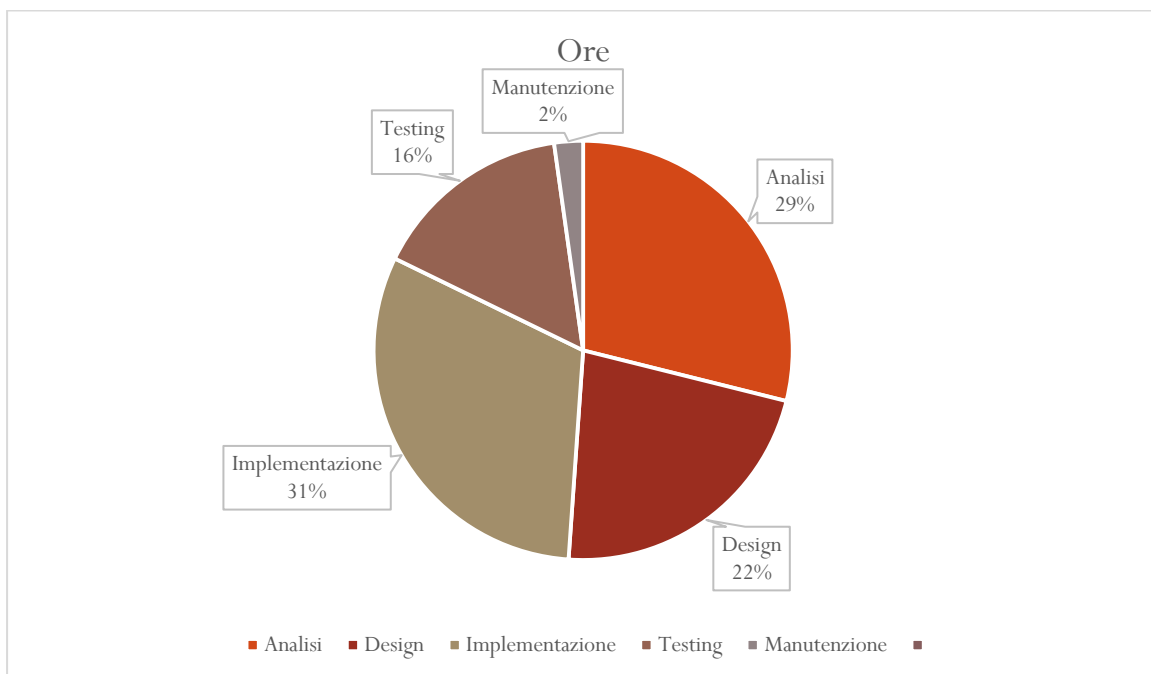
Resources:

Per realizzare questo Progetto avremo a disposizione tre computer (uno per ognuno) e diversi software trovati on-line. Tra questi troviamo: discord per i meeting, gitHub per caricare il codice e i vari upgrade, starUml per la realizzazione dei diagrammi uml, eclipse per la scrittura del codice in java ed infine xampp per la creazione di un database.

Budget and schedule:

Il budget a nostra disposizione è nullo, però saremo in grado di valorizzare tutte le varie risorse a nostra disposizione. Risorse che in questo caso divideremo in base alle esigenze che incontreremo durante lo sviluppo. Lo schedule non sarà lineare: siccome per questo progetto preferiamo un process model di tipo agile non si avranno mai dei requisiti troppo specifici sin dall'inizio, ma questi saranno modificati con il progredire del progetto. Inoltre, ogni fase non è in un punto specifico del tempo, ma comunicherà con le altre senza che una avvenga necessariamente prima o dopo.

Abbiamo cercato di schematizzare le ore utilizzate per il progetto (all'incirca 45 ore a testa), cercando il più possibile di dividerle in base alle attività svolte nonostante queste spesso si sovrappongano:



Changes:

Durante lo sviluppo del software, ma anche a seguito, potrà essere necessario effettuare delle modifiche al codice sorgente o alla struttura generale del progetto. Inizialmente queste modifiche saranno correttive nel caso di errori nell'esecuzione o adattive a nuove richieste nate internamente o esternamente al gruppo, come l'aggiunta di nuove funzionalità secondarie.

In futuro si prevedono anche cambiamenti perfettivi in caso di nuove richieste da parte del cliente: poiché il nostro codice sarà ben organizzato in diverse classi e suddiviso in due pacchetti principali, l'aggiunta di nuove caratteristiche, ad esempio nuovi pulsanti o tabelle nel database, non comporteranno problemi.

L'organizzazione del codice è infatti stata pensata in maniera preventiva di eventuali cambiamenti, dove ogni classe rappresenta una delle tante schermate che si presentano al cliente. Qualora si voglia

apportare una modifica basterà lavorare su quella specifica parte senza preoccuparsi troppo del resto del codice.

Delivery:

Volta per volta consegniamo un prototipo funzionante e testato del progetto (come richiede il metodo prototyping), grazie a cui il cliente può sia verificare il progresso del sistema sia richiedere feature o modifiche dell'attuale prototipo. Una volta concluso e testato il software, in seguito all'approvazione da parte del cliente, verrà consegnato insieme a tutta la documentazione.

SPECIFICA DEI REQUISITI

Introduzione:

Peso's Takeway è un'applicazione per PC che funziona da sistema di gestione di un fast food, dove il cliente può scegliere i prodotti che preferisce tra una lista di panini e bibite, modificabili dai dipendenti.

Il sistema è sviluppato in Java tramite l'IDE Eclipse, che tramite l'importazione della jar "mysql-connector" permette la connessione ad un database. Questo sarà creato tramite Xampp, applicazione esterna che tramite MySQL e Apache permette di creare un database locale ove salvare i dati necessari al corretto utilizzo, anche su più iterazioni, del nostro software.

Descrizione Generale:

Possiamo vedere la nostra applicazione da due punti di vista: quello del cliente e quello del dipendente.

Questa distinzione è ben visibile nel Diagramma dei Casi d'Uso, disegnato come analisi preliminare per capire quali avrebbero dovuto essere le principali funzionalità da implementare.

Il CLIENTE, una volta registrato e loggato, avrà la possibilità di scegliere quali articoli ordinare tramite due liste separate di panini e bibite. Di conseguenza, gli sarà possibile vedere l'elenco dei propri ordini precedenti, ordinati tramite un id acquisto e la data. Infine, può vedere i dati del proprio profilo, e cambiare la password qualora lo ritenesse necessario.

Il DIPENDENTE, similmente, avrà accesso all'elenco dei prodotti, suddivisi nello stesso modo. Diversamente dal cliente, il dipendente può modificare le caratteristiche di ogni articolo, aggiungerli, eliminarli o vederli ad un determinato cliente.

Può anche vedere l'elenco di tutti gli ordini precedenti, ordinandoli secondo diverse logiche, oltre ad un elenco di tutti i clienti. Ovviamente anche lui potrà vedere e modificare il proprio profilo e password.

Qualora il dipendente avesse il ruolo di MANAGER, avrebbe la possibilità di gestire i suoi dipendenti e aggiungerne altri.

Vincoli e Ipotesi:

Il sistema di acquisto di un prodotto è limitato al semplice click di un pulsante, che andrà a scalare la quantità ordinata da quella disponibile. Non è previsto un sistema di pagamento integrato nell'applicazione, ma si presuppone che questo avvenga al momento della consegna.

Inoltre, non è disponibile un vero e proprio database condiviso tra le varie applicazioni, ma ognuno ha il proprio database locale. Ovviamente questo è a semplice scopo dimostrativo.

Requisiti Specifici:

L'interfaccia utente è gestita tramite una serie di JPanel di Java, che si alternano a seconda dell'azione svolta dall'utente. Sarà necessario quindi poter avviare l'applicazione tramite un ambiente di sviluppo per il linguaggio Java. Inoltre, come spiegato in precedenza, a fini dimostrativo è necessario che su ogni postazione sia presente Xampp per l'avvio e la gestione del database.

Requisiti Relativi allo Sviluppo in Team:

Per lo sviluppo è stato necessario l'utilizzo di tre sistemi aggiuntivi:

- Discord è stato utilizzato per la comunicazione durante le riunioni del team.
- GitHub e Google Drive sono stati utilizzati per lo scambio di parti del software, come codice e immagini.

Descrizione del Codice:

Il codice è stato diviso in due parti principali, oltre alla classe "start" che serve solo ad avviare il programma.

Una cartella "Attributi" presenta tutte le classi principali degli oggetti sviluppati (prodotti, utenti...) oltre che quelli di sistema come il tema e la connessione al database.

La cartella "Activity" invece è composta da tutte le attività che possono essere svolte dalle classi sopra citate. Quasi ognuna di questa aprirà uno specifico pannello e sarà connesso a pannelli precedenti e successivi, con diversi pulsanti per muoversi tra essi.

ARCHITETTURA

Una volta conclusa la specifica dei requisiti siamo passati all'analisi dell'architettura e del design.

Abbiamo deciso di realizzare un software basato sul tipo di architettura sviluppata da Kruchten, la 4+1 view model. A supporto di questa architettura entrano in gioco i diagrammi UML che abbiamo realizzato: Di classe, d'uso, di stato, di sequenza e di attività. Abbiamo scelto questo tipo di modello per poter descrivere il nostro software dal punto di vista di tutte le parti interessate, ossia utenti, sviluppatori, ingegneri di sistema e project managers .

Questo modello, come possiamo notare dal suo nominativo è composto da 4 viste (logica, sviluppo, processo e vista fisica) e da uno o più scenari o casi d'uso (Rappresenta il cosiddetto "+1")

Logical View:

Tramite la logical view siamo riusciti a definire lo sviluppo completo dei requisiti. Abbiamo analizzato tutti le possibili interazioni che l'utente o l'impiegato potrebbero incontrare durante l'utilizzo del software; da queste analisi siamo riusciti a ricavare due diagrammi UML che ci hanno permesso di descrivere gli stati, le relazioni e gli oggetti in grado di soddisfare tutti i requisiti ricavati dalla specifica (stiamo parlando del Class Diagram e dello State Machine Diagram)*. L'analisi che abbiamo intrapreso ci ha portato a suddividere il sistema in numerosi componenti, in modo tale da poter esaminare ogni qualsiasi tipo di relazione dal punto di vista dell'utente; così facendo le funzionalità richieste dai requisiti sono state rappresentate in maniera corretta.

Process View:

In questa sezione abbiamo analizzato e studiato tutti i possibili processi presenti nel nostro software. Nello specifico la maggior parte dei processi riguardano la comunicazione tra l'applicazione e il database,

infatti per ogni qualsiasi operazione, dal login all'acquisto, tutti i dati passano dall'applicazione all'archivio. Da questo studio siamo riusciti a sviluppare un sistema in grado di raggiungere performance di un certo livello con una tolleranza agli errori abbastanza ridotta. Tutto ciò ci ha spinto a realizzare altri due diagrammi UML (Sequence Diagram e Activity Diagram)* che descrivono tutti le possibili comunicazioni dinamiche tra i vari processi che compongono l'applicazione.

Development view:

Per quanto riguarda la struttura base del software abbiamo deciso di applicare una suddivisione netta e precisa tra i vari componenti. Il database, infatti, è gestito dal tool esterno (XAMPP), mentre il sistema vero proprio è stato realizzato tramite un'ulteriore applicazione (Eclipse). L'altra divisione principale la possiamo trovare dentro al sistema di base, ovvero quella tra attività, oggetti e tema. Queste suddivisioni ci hanno permesso di progredire migliorando la struttura del codice senza andare ad intaccare la funzionalità e l'efficienza del software. Fondamentale in questa fase il jdbc Connector che, come anticipato precedentemente, ci ha permesso di gestire il database da un tool esterno.

Physical view:

La caratteristica fondamentale della nostra applicazione è sicuramente la sua portabilità, ovvero la capacità di adattarsi a qualsiasi tipo di ambiente fisico e virtuale. Per quanto riguarda la parte fisica, infatti, non sono necessari specifici computer dotati di particolari CPU, basta un semplice portatile in cui ci sia presente l'ambiente java.

Scenario:

Anche i casi d'uso sono fondamentali per la realizzazione di questo tipo di applicazione. Il nostro rappresenta una possibile attività nell'ambito della ristorazione e della vendita al dettaglio. Lo abbiamo rappresentato tramite lo Use Case diagram* su UML. Come previsto la suddivisione tra i vari prodotti o le varie attività eseguibili da impiegati e clienti ci ha permesso di strutturare tutta l'architettura e il design che abbiamo deciso di apportare al nostro sistema, dalla divisione delle varie componenti virtuali ai singoli scambi di messaggi eseguiti da più parti.

*I diagrammi UML si possono visionare nel file da noi caricato.

DOCUMENTO DI TEST

Test Plan:

Vogliamo testare il funzionamento del nostro sistema. Dato l'utilizzo di un metodo agile nello sviluppo del software, abbiamo già testato di volta in volta le funzionalità del nuovo codice, facendo in modo di consegnare sempre un'applicazione funzionante, senza errori. Questo permette di utilizzare meno tempo nella manutenzione correttiva e concentrarsi di più sulle altre fasi del progetto.

Dopo aver effettuato test con le tecniche fault-based ed error-based, descriviamo qui la tecnica coverage-based, finalizzata a testare la maggior parte delle funzionalità del sistema. Questa parte riguarda la verifica del sistema. I test saranno effettuati facendo partire l'applicazione e facendo ciò che ci aspettiamo faccia un utente reale, cliccando gli appositi bottoni e facendo le azioni di un normale utente.

Per quanto concerne invece la validazione del sistema, dobbiamo verificare che il sistema sviluppato soddisfi i requisiti che ci eravamo posti.

Test log e test case:

Di seguito elenchiamo tutti i casi di test effettuati e se questi son passati o portano a malfunzionamenti del sistema. I test verranno effettuati utilizzando l'applicazione sviluppata e confrontando le azioni effettuate anche con i cambiamenti avvenuti sul database.

Test riferiti al login o alla registrazione:

- Registrazione di un nuovo cliente: OK
- Login di un utente: OK
- Login di un commesso: OK

Test riferiti alle azioni del cliente:

- Vedi cibo: OK
- Ricerca mediante ID: OK
- Ricerca mediante nome: OK

- Vedi bibite: OK
- Ricerca mediante ID: OK
- Ricerca mediante nome: OK

- Vedi acquisti precedenti: OK

| FoodID | Name | Price | Quantity | Description |
|--------|----------|-------|----------|-------------------------|
| 101 | KingKong | 3.5 | 7 | Banana-bread alla ru... |

| PurchaseID | ProductID | ProductName | Amount | Co |
|------------|-----------|-------------|--------|-----|
| 6 | 00203 | Fanfa | 1 | 2.0 |
| 2 | 00101 | KingKong | 2 | 7.0 |

- Visualizzazione profilo: OK
- Modifica profilo:
 - o Modifica nome: OK
 - o Modifica cellulare: OK
 - o Modifica Indirizzo: OK

ID Utente: c003

Nome: Luca

Cellulare: +39 3314757734

Indirizzo: Via Roma 6, Dalmine

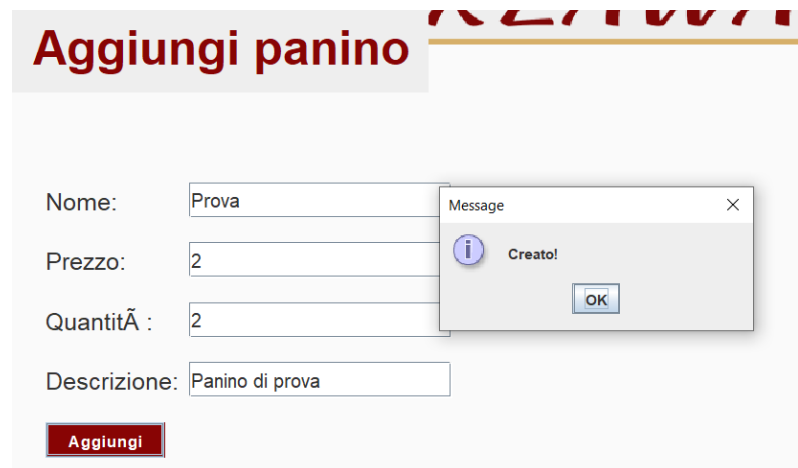
Invia

Message: Profilo aggiornato! OK

- Cancellazione profilo: OK
- Cambio password: OK
- Acquisto di 2 PolloPazzo: OK (verifico anche l'inserimento nella lista degli acquisti precedenti)
- Acquisto di 2 Bebsi: OK (verifico anche l'inserimento nella lista degli acquisti precedenti)

Test riferiti alle azioni di un impiegato:

- Vedi cibo: OK
- Aggiungi cibo (Prova, 2, 2, Panino di prova): OK
- Ricerca panino: OK
- Modifica panino: OK
- Vendi panino: OK
- Elimina panino: OK
- Stessa cosa per le bibite: OK
- Vista cliente: OK
- Ricerca cliente: OK
- Modifica cliente: OK
- Cancella cliente: OK
- Vedi tutti gli ordini: OK
- Vedi profilo: OK
- Modifica profilo: OK
- Cambia password: OK



The screenshot shows a web form titled "Aggiungi panino" in red. The form contains the following fields and values:

- Nome: Prova
- Prezzo: 2
- Quantità: 2
- Descrizione: Panino di prova

At the bottom of the form is a red button labeled "Aggiungi". Overlaid on the right side of the form is a "Message" dialog box with a blue information icon, the text "Creato!", and an "OK" button.

Test riferiti alle azioni del titolare (oltre alle azioni di un normale impiegato):

- Vista impiegati: OK
- Ricerca impiegati: OK
- Aggiungi impiegato: OK
- Modifica impiegato: OK
- Cancella impiegato: OK