



**UNIVERSITÀ
DEGLI STUDI
DI BERGAMO**

Department
of Management, Information
and Production Engineering

Progetto di ingegneria del software

Versione 1.1

Partecipanti: Andrea Trapletti 1066781

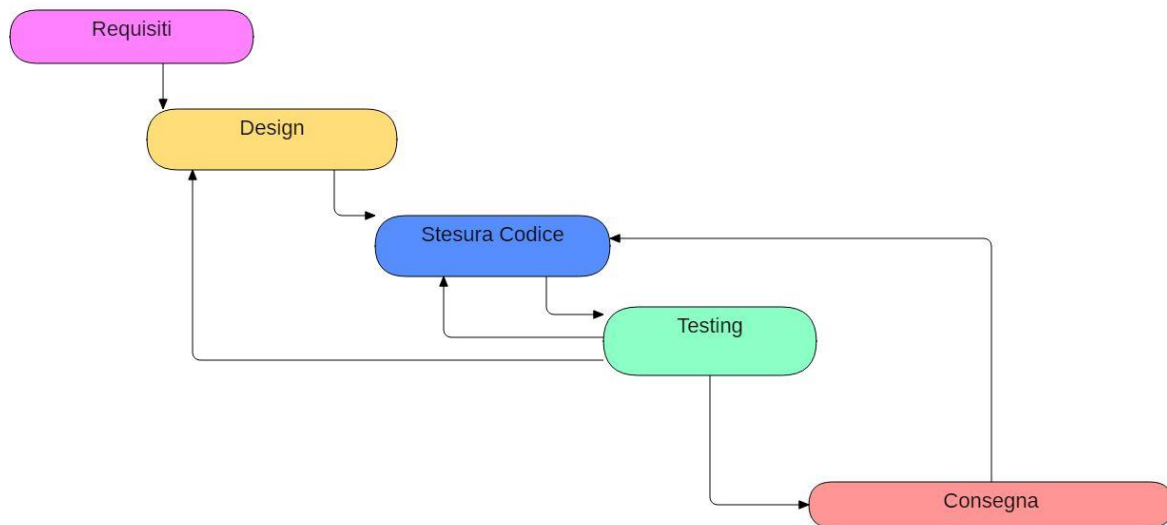
Mattia Giudici 1065452

Michele Verdi 1067606

Software lifecycle	2
Software configuration managment	3
People Management and Team Organization	7
Software Quality	7
Requirement Engenieering	8
Modeling	8
Software Architecture	9
Software Testing	9
Software maintenance	10
Software Design	10
Modifiche Design pattern	11

Software lifecycle

Date le circostanze e le caratteristiche dei componenti del gruppo è stato deciso di utilizzare un modello Agile in particolare seguendo uno schema Scrum. Abbiamo di conseguenza avuto continue interazioni con il product owner per fare in modo di consegnare un prodotto il più vicino possibile a quello desiderato, abbiamo inoltre consegnato diverse versioni funzionanti del software da poter testare all'interno dell'attività giornaliera. Questo ci ha permesso di risolvere problemi che non erano stati pensati durante la stesura del project plan. Attraverso questo processo siamo stati in grado di vedere in prima persona le problematiche del nostro software, così da implementare velocemente nuove funzionalità.

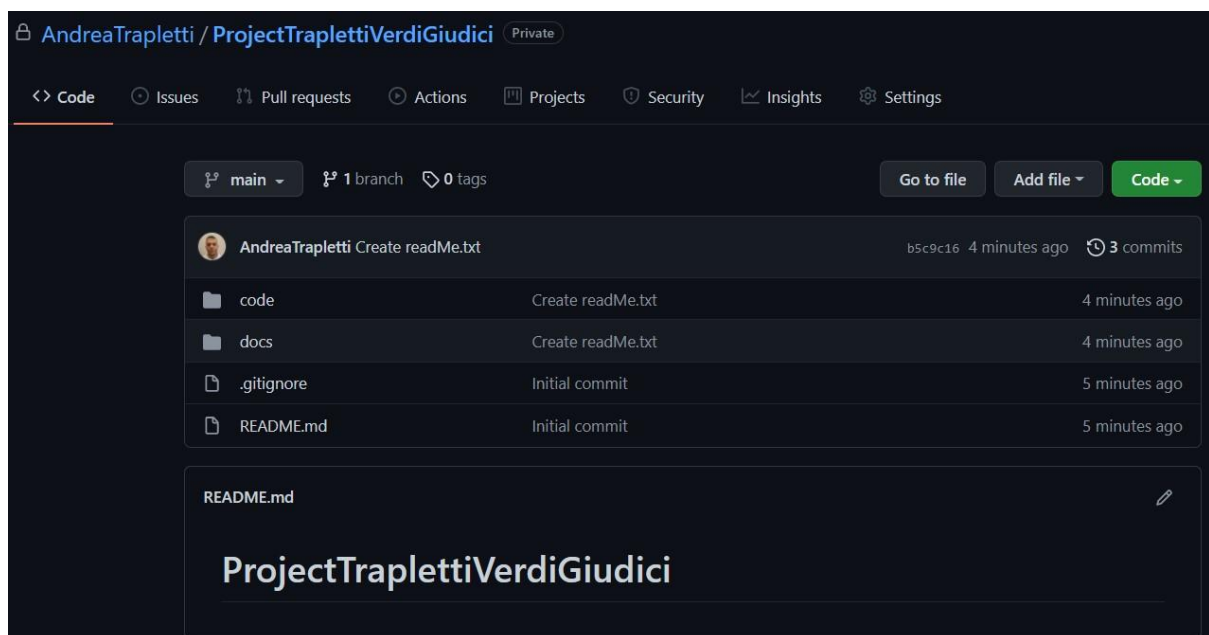


Software configuration management

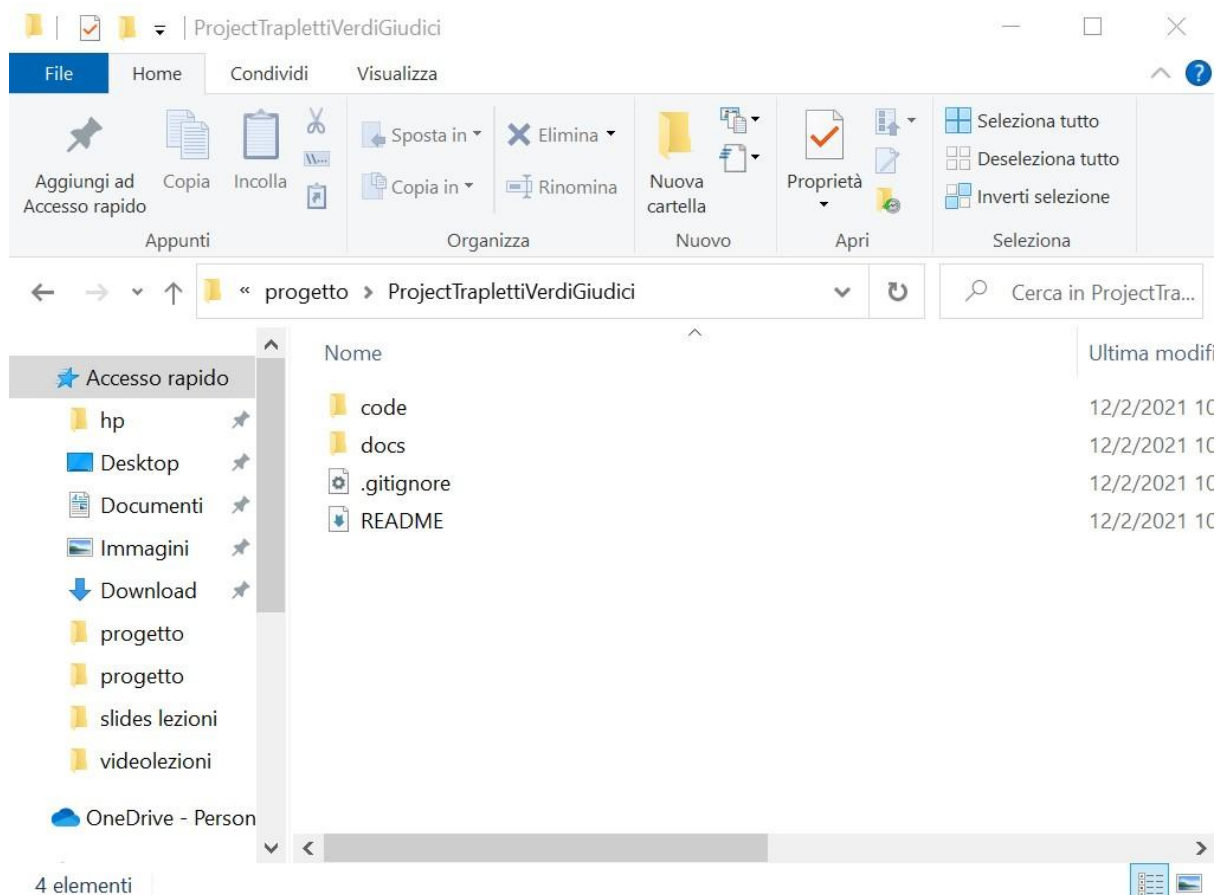
Abbiamo deciso che il miglior modo per condividere il codice del nostro software in tempo reale tenendolo sempre aggiornato fosse quello di usare GitHub, software perfetto per la comunicazione anche a distanza tra i membri dello staff per eventuali nuove versioni del software.

Risulta fondamentale che ognuno di noi installi sul proprio PC GitHub desktop. Quest'app ci permette di modificare in locale (direttamente nella nostra cartella contenente il source code), il codice eseguendo semplici comandi di Push (caricare il codice nella repository condivisa) e Pull (per aggiornare il proprio codice in locale con l'ultima versione recuperata attraverso il comando di Fetch).

Essendo già tutti in possesso di un account GitHub, uno dei membri del nostro staff (Andrea Trapletti) ha creato una repository e l'ha condivisa con gli altri membri del suo staff. A questo punto GitHub ci permette di scegliere il linguaggio che useremo nel nostro progetto, ovvero Java.



A questo punto possiamo lavorare direttamente attraverso GitHub Desktop. Cloniamo la repository del progetto in locale scegliendo un percorso.



Un membro dello Staff a questo punto crea (utilizzando Eclipse) un nuovo progetto. Seleziona una repository all'interno della cartella "code" in locale dentro cui creare il proprio progetto Java.

Questo PC > Desktop > ingegneria del software > progetto > ProjectTraplettiVerdiGiudici > code > Ristorante

Nome	Ultima modifica	Tipo	Dimensione
src	12/2/2021 10:21 AM	Cartella di file	
.classpath	12/2/2021 10:21 AM	File CLASSPATH	1 KB
.gitignore	12/2/2021 10:21 AM	File di origine Git Ign...	1 KB
.project	12/2/2021 10:21 AM	File PROJECT	1 KB

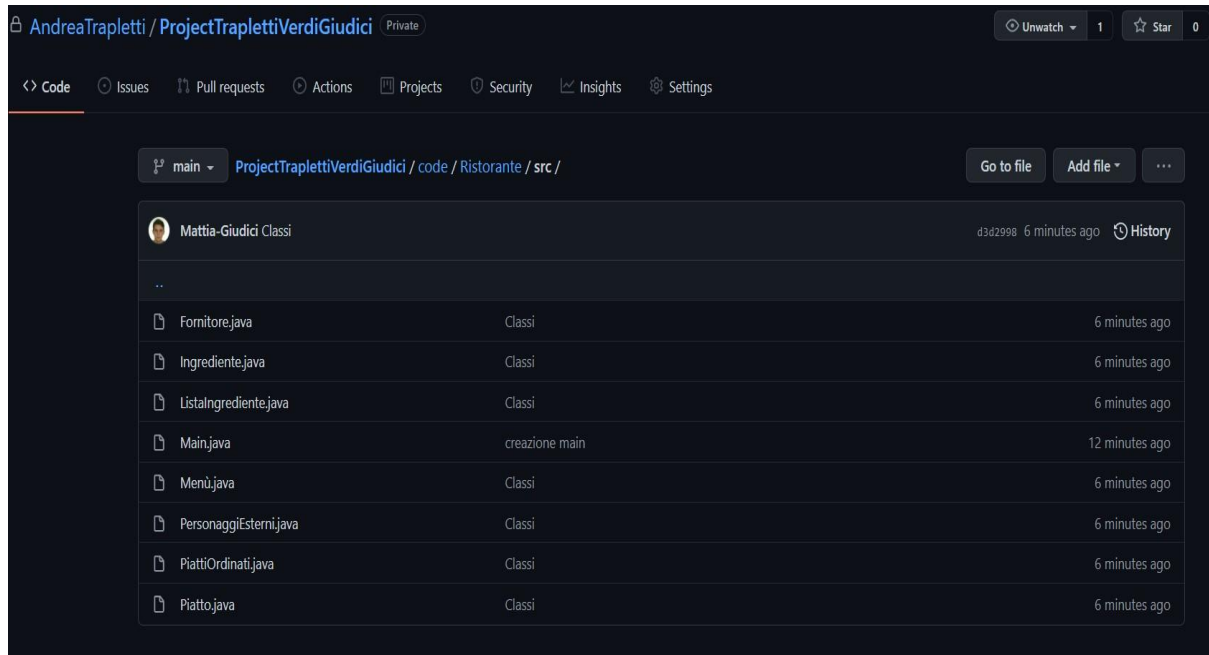
A questo punto posso fare un Push per mandare il progetto anche agli altri partecipanti: da GitHub desktop, effettuo il "commit to main".

Gli altri partecipanti a questo punto possono eseguire il pull del progetto. Sempre da GitHub desktop, cliccano su "Fetch Origin" e successivamente su "pull origin".

università > ingegneria del software > progetto 2021 > ProjectTraplettiVerdiGiudici > code > Ristorante > src

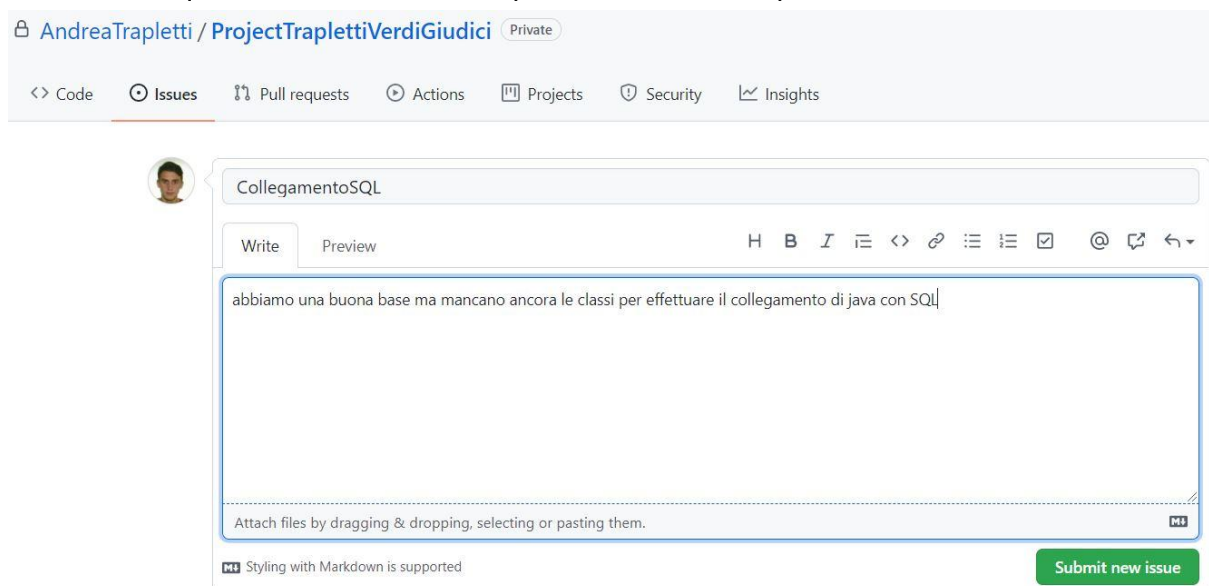
Nome	Ultima modifica	Tipo	Dimensione
Fornitore	02/12/2021 10:26	File JAVA	1 KB
Ingrediente	02/12/2021 10:25	File JAVA	1 KB
ListaIngrediente	02/12/2021 10:26	File JAVA	1 KB
Main	02/12/2021 10:17	File JAVA	1 KB
Menù	02/12/2021 10:25	File JAVA	1 KB
PersonaggiEsterni	02/12/2021 10:26	File JAVA	1 KB
PiattiOrdinati	02/12/2021 10:24	File JAVA	1 KB
Piatto	02/12/2021 10:25	File JAVA	1 KB

Possiamo notare che le classi che abbiamo creato vengono direttamente anche caricate in remoto su GitHub.



GitHub è fondamentale se dobbiamo estendere il nostro progetto o abbiamo da risolvere dei problemi.

Abbiamo dunque dovuto creare issues per risolvere i nostri problemi.




A questo punto creo un branch, per crearlo vado su GitHub Desktop: clicco in alto dove c'è Current branch e aggiungo un branch a mio nome, successivamente effettuo un publish. Il membro dello staff a cui è assegnato il compito di risolvere il problema, crea tutti gli elementi necessari per completare il suo compito, in seguito continua con un Push da GitHub desktop e possiamo verificare che su GitHub, nella sezione branch a questo punto avremo 2 branch, uno con il codice iniziale, l'altro che contiene il codice modificato.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: MattiaGiudici ✓ Able to merge. These branches can be automatically merged.


 collegamento SQL

Write

Preview

H B I ≡ <> 🔗 ☰ ≡ ☑ @ ↻ ↶

Modifiche complete

Attach files by dragging & dropping, selecting or pasting them. 

Create pull request

Mattia-Giudici requested your review on this pull request. [Add your review](#)

collegamento SQL #2

[Open](#) Mattia-Giudici wants to merge 1 commit into `main` from `MattiaGiudici`


Conversation 0

Commits 1

Checks 0

Files changed 1


+5 -0


 **Mattia-Giudici** commented 2 minutes ago


Modifiche complete


collegamento SQL

d786e8a

 **Mattia-Giudici** requested review from **AndreaTrapletti** and **MicheleVerdi** 2 minutes ago

 **Mattia-Giudici** assigned **Mattia-Giudici** and unassigned **Mattia-Giudici** 2 minutes ago

 **Mattia-Giudici** added the **help wanted** label 1 minute ago

 **AndreaTrapletti** added the **good first issue** label 33 seconds ago

Review requested

Review has been requested on this pull request. It is not required to merge. [Learn more](#).

Show all reviewers

2 pending reviewers

Continuous integration has not been set up

[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.


This branch has no conflicts with the base branch


Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Reviewers

 **AndreaTrapletti**

 **MicheleVerdi**

Assignees

No one—assign yourself

Labels

good first issue **help wanted**

Projects

None yet

Milestone

No milestone

Linked issues

Successfully merging this pull request may close these issues.



None yet

Notifications

[Unsubscribe](#)

You're receiving notifications because you're watching this repository.

2 participants

Lock conversation

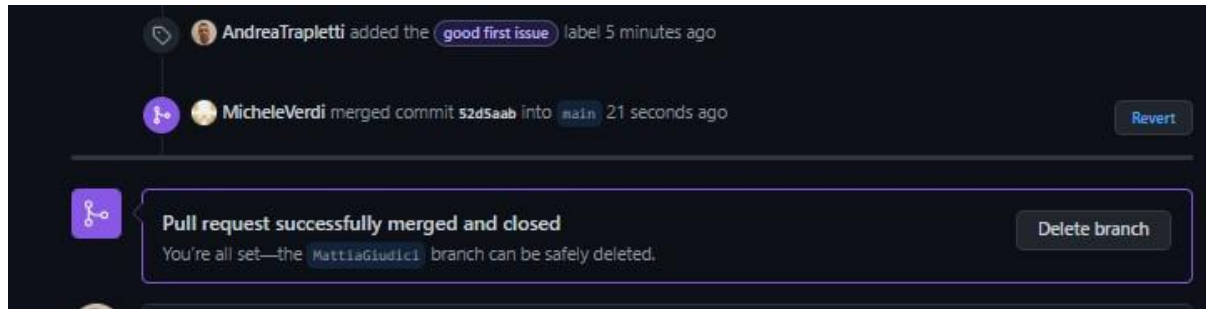
Write

Preview

H B I ≡ <> 🔗 ☰ ≡ ☑ @ ↻ ↶

Leave a comment

Una volta effettuata la revisione da parte di un altro membro dello staff (Verdi in questo caso), se non ci sono problemi si procede con il merge.



Quando le modifiche sono completate, si prosegue con un “pull request” (direttamente da GitHub) e il codice che si trovava nel branch secondario viene messo nel branch main.

People Management and Team Organization

Dato che il nostro team ha scelto di intraprendere la strada di un processo Agile Scrum, il nostro team è organizzato come un tipico Scrum Team:

- Product Owner: “Ristorante La Monasterola”
- Team Master: Mattia Giudici
- Sviluppatori: Verdi Michele, Trapletti Andrea.

Essendo un team piccolo è difficile definire in modo preciso dei ruoli distinti, di conseguenza tutti e 3 i partecipanti collaborano nella stesura del progetto, trovandosi spesso sia in presenza che attraverso riunioni a distanza.

Abbiamo assunto inoltre una persona esterna che si è occupata di collegare il database del nostro ristorante al nostro software.

Software Quality

Riteniamo che il nostro software sia in grado di rispettare varie qualità, tra cui la correttezza e l’affidabilità nella sua capacità di gestire le ordinazioni del ristorante nel modo più efficiente e veloce possibile.

Riteniamo inoltre che la sicurezza del software sia garantita dalla sua non necessità di connettersi a una rete, e la facilità con la quale si possono eseguire servizi di manutenzione.

Siccome il software ha bisogno di essere caricato su più palmari e su un computer centrale, ha anche un’elevata portabilità.

Requirement Engineering

Il ristorante “La Monasterola” di Monasterolo Del Castello (BG), ci ha richiesto un incontro durante il quale abbiamo discusso in maniera molto informale e diretta su alcuni problemi di gestione della loro attività e come si potevano risolvere. Durante l’incontro abbiamo stilato un documento che comprendeva le problematiche del ristorante, successivamente ci siamo trovati in sede privata per comprendere come si potessero risolvere al meglio i problemi riguardanti la loro attività.

L’idea si basa sulla necessità del ristorante di tenere traccia dell’utilizzo delle materie prime presenti in magazzino.

Il software verrebbe installato sui dispositivi in possesso dei camerieri per gestire le comande, e su un computer centrale; i camerieri prendono gli ordini di ogni tavolo inviandoli poi in cucina, allo stesso tempo il sistema registra gli ingredienti necessari per la realizzazione di ogni piatto, andando poi a sottrarli da quelli presenti in magazzino.

Nel caso un ingrediente stia per finire, il sistema informa il gestore che quindi ha la possibilità di bloccare ulteriori ordini. Il software inoltre preparerà una lista di ingredienti provvisoria, che il gestore può modificare o confermare, andando poi ad inviarla al fornitore. Ogni giorno il software verifica la permanenza di un ingrediente in magazzino confrontandolo con la data di scadenza inserita al momento della consegna da parte del fornitore, quando un ingrediente si avvicina a quella data il gestore viene informato, così che possa provvedere all’utilizzo degli ingredienti in scadenza, quando l’ingrediente supera quella data viene segnalato e rimosso dall’elenco.

Il sw in automatico attraverso i piatti prenotati e i prezzi inseriti nel menù calcola lo scontrino da far pagare ai clienti per ogni tavolo.

Durante lo sviluppo del software il team master ha incontrato più volte il product owner per mostrargli e consegnargli i pezzi di software completi come volevano gli accordi precedentemente stipulati. Durante tutti questi meeting il product owner verificava che i requisiti fossero raggiunti per poi validare il nostro software.

Al termine dello sviluppo, tutto il team ha incontrato il product owner, il quale ha validato l’intero progetto.

Modeling

I diagrammi sono stati svolti in UML attraverso starUML e sono caricati nella stessa cartella di questo file. i modelli sono :

- Uno use case diagram che rappresenta l’utilizzo del software in generale
- Un class diagram che rappresenta tutte le classi e i metodi poi usati nel software.
- Un Sequence diagram che rappresenta tutte le azioni del sistema dalla sua creazione, in particolare a tutte le azioni che avvengono in modo periodico, e alla chiamata di sistema per una nuova ordinazione e un successivo pagamento.
- Un activity diagram che descrive le azioni del sistema dal momento in cui un cliente richiede un piatto e il cameriere ne conferma l’ordinazione, mostra inoltre la richiesta di pagamento e come rispondere ad eventuali errori
- Uno State machine diagram che descrive tutti gli stati dell’oggetto piatto e dei suoi ingredienti. descrivendo come passa da attivo a disattivo.

Software Architecture

Abbiamo deciso di utilizzare dei diagrammi uml per rappresentare alcune parti architetture del nostro software. in particolare il class diagram e lo state machine diagram rappresentano la logical view; il primo diagramma raffigurando tutte le classi presenti nel nostro software, mentre lo state machine raffigura la gestione dei piatti ordinati e la loro presenza nel menù.

Il sequence diagram e l'activity diagram rappresentano la process view, andando a rappresentare il collegamento sequenziale dell'intero sistema concentrandosi in particolare sulla gestione dei controlli giornalieri o settimanali. mentre l'activity diagram si occupa di descrivere dal momento dell'ordinazione fino al pagamento del conto.

Software Testing

Nello sviluppo del nostro software non abbiamo avuto una vera e propria fase di testing, ma l'abbiamo svolto per ogni componente aggiunta. il testing si è svolto in 3 modi diversi:

- static testing : svolto in automatico dal compilatore, in questo caso eclipse.
- inspection : dopo ogni seduta di testing il team si è riunito e 2 persone controllavano e comprendevano il codice scritto dalla terza. Già in questa fase siamo riusciti a trovare e correggere molti errori, prima che questi potessero influire negativamente sulla prossima fase.
- Per poter ottenere un software funzionante dopo l'implementazione di ogni parte di esso, abbiamo dovuto testare queste parti in continuazione per essere sicuri che queste non avessero errori e non influissero negativamente con quelle già esistenti. Un errore rilevato durante questa fase viene se possibile immediatamente corretto, in caso il problema fosse troppo grande per essere corretto nella fase in esame esso viene segnalato e risolto nei giorni successivi.

Per fare questo abbiamo sviluppato 3 classi di test utilizzando JUnit, contenute nel package test, che ci permettessero in automatico di tenere monitorate le 3 classi più problematiche.

Con questo abbiamo constatato durante la stesura del codice l'effettiva correttezza di quest'ultimo.

Un esempio di errore è stato nel testing della funzione che permette di inserire un piatto nella lista dei piatti ordinati da un tavolo. Durante il testing ci siamo accorti che i piatti venivano inseriti nonostante l'assenza degli ingredienti necessari. Questo errore era dovuto all'assenza di un controllo nella funzione; riteniamo questo un errore già dalla fase di planning, abbiamo quindi modificato questa parte nei nostri diagrammi e nel nostro project plan per risolvere il problema.

Software maintenance

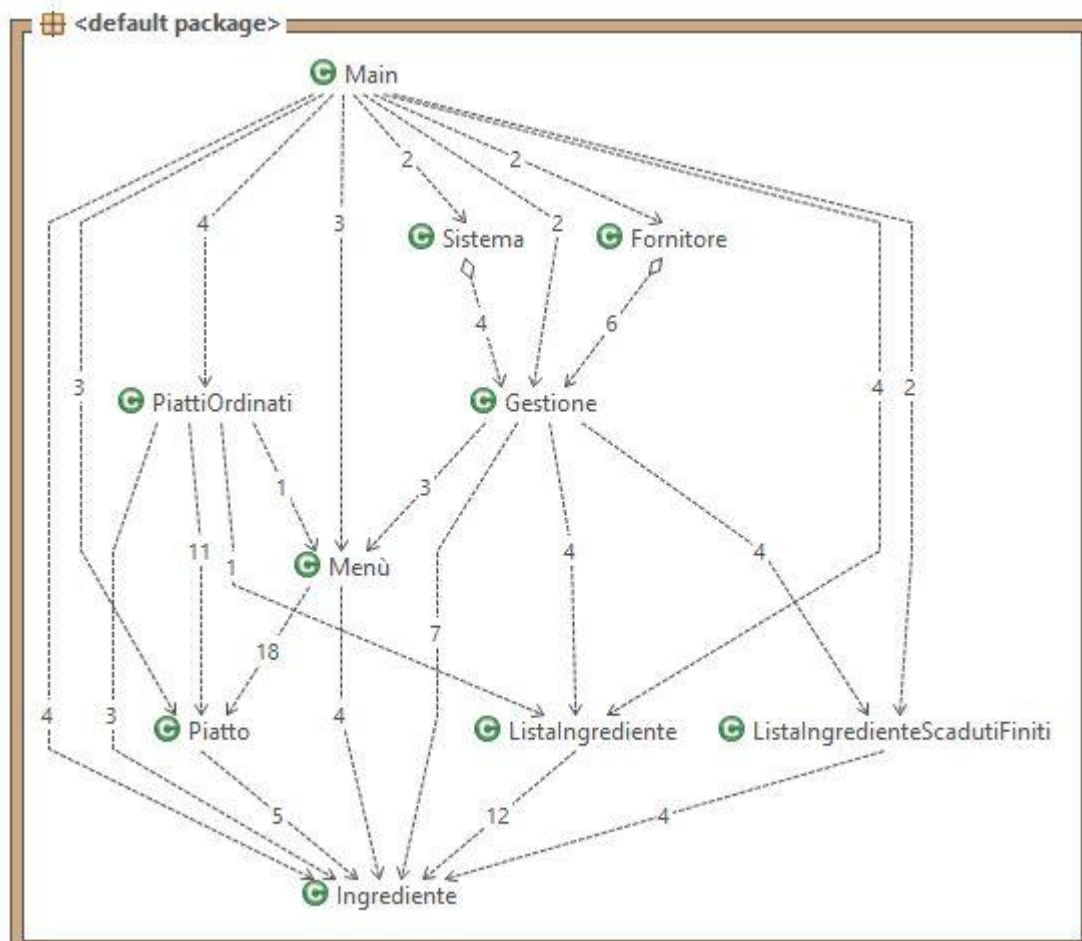
Avendo scritto manualmente la totalità del codice non è stato necessario effettuare nessuna attività di reverse engineering. abbiamo però dovuto effettuare dei refactoring, in particolare ci siamo accorti che alcune classi risultavano troppo lunghe e intrecciate tra di loro, andando a complicare notevolmente la comprensione del testo (caso di bad smell : Large Class). Abbiamo quindi effettuato dei refactoring andando a dividere queste classi in due oppure andando ad aggiungerne una direttamente, come nel caso della classe Gestione. Tutti questi cambiamenti ci hanno portato ad effettuare delle modifiche anche nel project plan e nella modellazione del nostro software. Durante la revisione del codice ci siamo inoltre accorti che vi erano state delle duplicazioni di esso, problema probabilmente causate dal lavorare in team e dalla suddivisione dei compiti, Abbiamo quindi rimosso il codice duplicato senza però andare a modificare la funzionalità dal codice.

Per quanto riguarda la manutenzione correttiva e perfettiva, abbiamo un accordo per effettuare aggiustamenti/miglioramenti su base mensile per adattare il software nel modo più preciso possibile alle esigenze del cliente.

Software Design

Il software viene abbondantemente descritto attraverso i diagrammi UML consegnati al Product owner durante i primi incontri.

Calcolo di complessità del codice, eseguito attraverso stanj4 :



Category/Metric	Value
Count	
Units	10
Classes / Class	0
Methods / Class	5.2
Fields / Class	2.3
ELOC	509
ELOC / Unit	50.9
Complexity	
CC	2.1
Fat	24
ACD - Unit	38.89%
Robert C. Martin	
D	0
A	0
I	1
Ca	0
Ce	0
Chidamber & Kemerer	
WMC	10.9
DIT	1
NOC	0
CBO	4
RFC	7.9
LCOM	0.6

Modifiche Design pattern

Proseguendo con il nostro progetto, ci siamo accorti che la classe menù deve essere vista come un'istanza singola che il nostro programma crea una e una sola volta; quindi abbiamo utilizzato un pattern creazionale: il Singleton Pattern, per far ciò abbiamo creato una classe di supporto chiamata Singleton.

Questa nuova soluzione ci ha portato a una nuova versione del nostro progetto: versione 2.0. Abbiamo creato, infatti, un nuovo class diagram: "class_diagram_2.0".

Sempre in questa nuova versione 2.0, notiamo che il nostro codice è scritto in modo tale che il fornitore creato può essere solo uno. Abbiamo deciso quindi di utilizzare un pattern creazionale: il Factory Pattern.

Abbiamo dunque reso astratta la classe Fornitore (classe che continua comunque ad estendere i thread), per poi creare una nuova classe: MattiaGiudici. La seguente classe estende Fornitore; in tal modo ogni volta che verrà registrato un nuovo fornitore del nostro ristorante, basterà creare una nuova classe identica a MattiaGiudici che estende Fornitore. Questa implementazione la si può notare nel "class_diagram_2.0".