



DOCUMENTAZIONE

A CURA DI GIORGIO CHIRICO E RAFFAELE GIACOMO GIOVANNI DI MAIO



LEGENDA

Capitolo 1: Software Engineering Management

Capitolo 2: Software Life Cycle

Capitolo 3: Configuration Management

Capitolo 4: People Management and Team Organization

Capitolo 5: Software Quality

Capitolo 6: Requirement Engineering

A pair of black-rimmed glasses rests on an open book. A red bookmark is visible on the left page. The background is a soft, out-of-focus blue-grey.

LEGENDA

Capitolo 7: Modelling

Capitolo 8: Software Architecture

Capitolo 9: Software Design

Capitolo 10: Software Testing

Capitolo 11: Software Maintenance

CAPITOLO 1: Software Engineering Management

- Project Plan

E' possibile prendere visione del nostro project plan all'interno della repository su GitHub al seguente link, andandolo poi a scaricare.

<https://github.com/giorgio-hash/progettoSE/blob/master/SoftwareEngineeringProject/documentazione/ProjectPlan.docx>

CAPITOLO 2: Software Life Cycle

- Metodi Agile

Per la realizzazione del progetto è stata utilizzata una filosofia di tipo Agile, mirata a seguire tutti i punti presenti nel manifesto.

In particolare troviamo:

- Concentrarsi su persone e interazioni invece che su processi e strumenti;
- Concentrarsi sul funzionamento del software invece che sulla comprensività della documentazione;
- Concentrarsi sulla collaborazione con il cliente invece che sulla negoziazione dei contratti;
- Concentrarsi sulle risposte al cambiamento del software invece che sul seguire un piano.

CAPITOLO 2: Software Life Cycle

- XP: Xtreme Programming

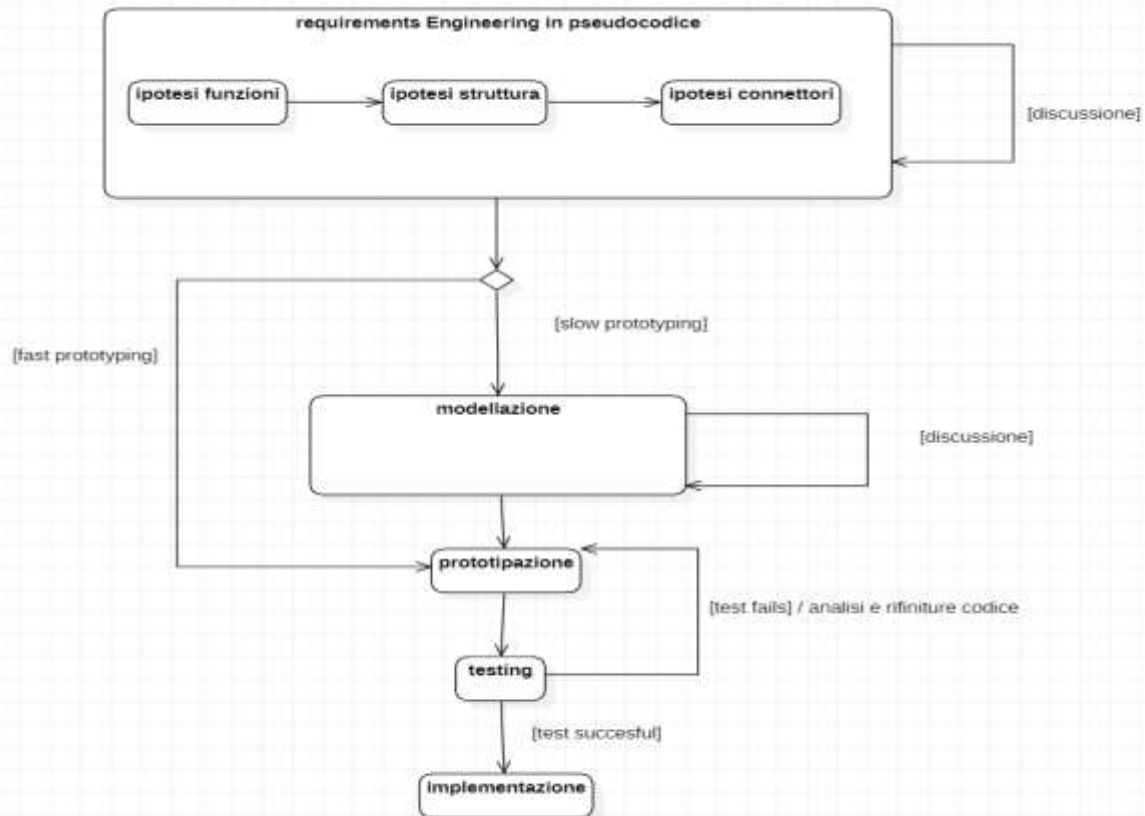
All'interno del gruppo di metodi Agile abbiamo scelto di utilizzare l'Xtreme Programming.

I punti chiave di questo processo di sviluppo sono:

- Il codice viene scritto a piccoli passi;
- il sistema deve SEMPRE compilare e deve essere SEMPRE eseguibile;
- il cliente deve essere coinvolto nella realizzazione del progetto;
- tutti gli sviluppatori hanno la stessa responsabilità per quanto riguarda il software e la metodologia utilizzata.

CAPITOLO 2: Software Life Cycle

MDA:



CAPITOLO 3: Configuration Management

Il configuration management si basa sulla possibilità di gestire qualsiasi tipo di «strumento» durante lo sviluppo del software.

Esso è cruciale per quanto riguarda progetti di larga scala.

- GitHub

Come sistema per il configuration management del nostro progetto abbiamo scelto di utilizzare GitHub che ci ha permesso di condividere le varie modifiche ed implementazioni successive del nostro progetto sulla baseline.

CAPITOLO 3: Configuration Management (Branches)

giorgio-hash / progettoSE Private

Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Security Insights Settings

Overview Yours Active Stale All branches Search branches...

Default branch

master Updated 12 days ago by giorgio-hash Default

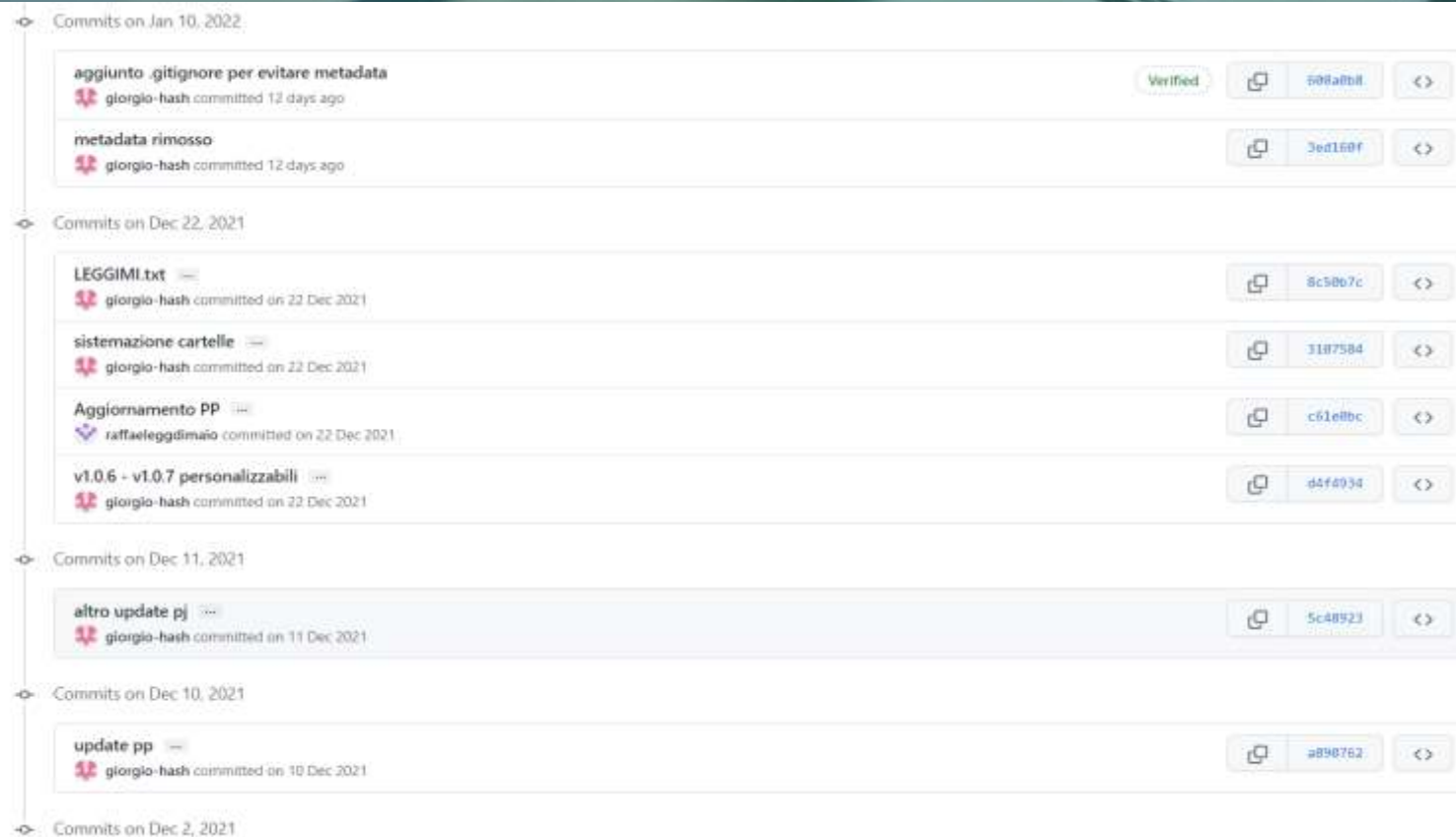
Active branches

PowerPoint Updated last month by giorgio-hash 2 | 0 New pull request

© 2022 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

<https://github.com/giorgio-hash/progettoSE/branches/yours>

CAPITOLO 3: Configuration Management (Commits)



The screenshot displays a Git commit history interface. It is organized into sections by date, with expandable/collapsible arrows on the left. The top section, 'Commits on Jan 10, 2022', contains two commits by 'giorgio-hash' from 12 days ago. The first commit, 'aggiunto .gitignore per evitare metadata', is marked as 'Verified' and has a commit hash of '508a8b1f'. The second commit, 'metadata rimosso', has a commit hash of '3ed160f'. The next section, 'Commits on Dec 22, 2021', lists four commits. The first three are by 'giorgio-hash': 'LEGGIMI.txt' (hash '8c50b7c'), 'sistemazione cartelle' (hash '318758d'), and 'Aggiornamento PP' (hash 'c61e0bc'). The fourth commit, 'v1.0.6 - v1.0.7 personalizzabili', is by 'raffaeleggidimaio'. The 'Commits on Dec 11, 2021' section shows one commit by 'giorgio-hash' titled 'altro update pj' with hash '5c48923'. The 'Commits on Dec 10, 2021' section shows one commit by 'giorgio-hash' titled 'update pp' with hash 'a890762'. The bottom section, 'Commits on Dec 2, 2021', is currently empty.

Commits on Jan 10, 2022

- aggiunto .gitignore per evitare metadata
giorgio-hash committed 12 days ago
Verified 508a8b1f
- metadata rimosso
giorgio-hash committed 12 days ago
3ed160f

Commits on Dec 22, 2021

- LEGGIMI.txt
giorgio-hash committed on 22 Dec 2021
8c50b7c
- sistemazione cartelle
giorgio-hash committed on 22 Dec 2021
318758d
- Aggiornamento PP
raffaeleggidimaio committed on 22 Dec 2021
c61e0bc
- v1.0.6 - v1.0.7 personalizzabili
giorgio-hash committed on 22 Dec 2021
d4f4034

Commits on Dec 11, 2021

- altro update pj
giorgio-hash committed on 11 Dec 2021
5c48923

Commits on Dec 10, 2021

- update pp
giorgio-hash committed on 10 Dec 2021
a890762

Commits on Dec 2, 2021

CAPITOLO 3: Configuration Management (Issues)

giorgio-hash / progettoSE Private

Unwatch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Security Insights Settings

Filters is:issue is:closed Labels 9 Milestones 0 New issue

Clear current search query, filters, and sorts

<input type="checkbox"/>	0 Open ✓ 2 Closed	Author	Label	Projects	Milestones	Assignee	Sort
<input type="checkbox"/>	richiesta di convalida Project Plan #2 by giorgio-hash was closed yesterday						1
<input type="checkbox"/>	.metadata non va sul repo #1 by garganti was closed 12 days ago						4

ProTip! Add `no:assignee` to see everything that's not assigned.

© 2022 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

CAPITOLO 4: People Management and Team Organization

Le persone coinvolte nel progetto sono stati due studenti dell'Università degli studi di Bergamo:

- Giorgio Chirico;
- Raffaele Giacomo Giovanni Di Maio.

Il team è stato sempre composto da due sviluppatori che hanno ricoperto simultaneamente diversi ruoli, talvolta analoghi, utilizzando inoltre anche il paradigma del pair-programming.

CAPITOLO 4: People Management and Team Organization

Per questa fase sono stati utilizzati 2 stili di organizzazione di team e persone:

- Relation style

Questa metodologia si basa sul lavoro innovativo, complesso e specializzato con decision-making basato su riunioni con il team.

- Integration style

Questa metodologia viene utilizzata quando il lavoro è di natura esplorativa e le varie task del progetto sono molto interdipendenti tra di loro.

CAPITOLO 4: People Management and Team Organization

Il Relation style è stato utilizzato per creare l'idea del «Parcheggio a pagamento» e, attraverso una serie di riunioni settimanali, prendere decisioni su come il lavoro sarebbe poi stato portato avanti.

L'integration style invece è stato utilizzato nell'ambito della suddivisione del lavoro per la creazione di parti di codice che sarebbero poi state integrate tra di loro.

Questo per fare in modo che i pezzi di codice implementati non fossero slegati completamente rispetto al resto del programma.

CAPITOLO 5: Software Quality

Nell'implementazione del nostro progetto ci siamo basati sul seguire la tabella dei Quality Factors.

In particolare troviamo:

- Correttezza

Ci siamo impegnati a rispettare quello che era stato deciso in sede di project plan centrando quindi tutti gli obiettivi prefissati volti alla realizzazione del nostro progetto.

- Affidabilità

Abbiamo fatto in modo che il programma risultasse affidabile eseguendo una serie di test utilizzando i vari prototipi da noi creati durante la realizzazione del progetto.

CAPITOLO 5: Software Quality

- Efficienza

La realizzazione del nostro progetto ha portato alla scrittura di molte linee di codice che, nel corso dei vari refactoring effettuati, sono state snellite e quindi rese più efficienti.

- Integrità

Per l'accesso al database contenente il registro dei ticket erogati è presente l'accesso tramite autenticazione.

- Usabilità

L'applicazione è stata realizzata per risultare più user friendly possibile.

CAPITOLO 5: Software Quality

- Manutenibilità

Per la manutenibilità del nostro progetto è richiesta una conoscenza di base della realtà implementata. Presa coscienza della realtà in esame, esso risulta facilmente manutenibile.

- Testabilità

E' stata creata una struttura modulare in modo da rendere più semplice eseguire test mirati alle singole zone di interesse.

- Flessibilità

E' stato fatto in modo che il codice fosse abbastanza flessibile per poter, in futuro, implementare il software per la gestione di un parcheggio a pagamento «barrier-less».

CAPITOLO 5: Software Quality

- Portabilità

Il nostro software è portabile in quanto non dipende da nessun hardware (è richiesto l'utilizzo un qualsiasi computer general-purpose) nonostante sia comunque necessario utilizzare un ambiente di sviluppo Java.

- Riusabilità

Il codice risulta essere riusabile in quanto parti del nostro codice completo derivano, a loro volta, da altre parti di codice esistente.

- Interoperabilità

E' richiesta stabilità di connessione tra i nodi del sistema realizzato, tra server-database e tra server-circuito bancario.

CAPITOLO 6: Requirement Engineering

L'ingegneria dei requisiti è descritta utilizzando lo standard IEEE 830.

1. Introduction

1.1 Purpose: l'obiettivo del nostro progetto è quello di realizzare un software che sia in grado di gestire i flussi di entrate e uscite da un parcheggio a pagamento introducendo un sistema di interfacce utente per i terminali e la possibilità di pagare con metodi diversi.

1.2 Scope: il software del parcheggio a pagamento contiene interfacce utente per i terminali di facile comprensione così da facilitare il pagamento, ritiro e consegna del ticket.

Inoltre è fatto in modo che esso sia fault tolerance grazie all'introduzione di un database che possa contenere i dati dei ticket in caso di blackout o malfunzionamenti generali.

CAPITOLO 6: Requirement Engineering

2. Overall description

2.1 Product perspective: l'obiettivo del software è quello di permettere ai vari clienti di accedere al parcheggio attraverso il ritiro del biglietto presso il terminale di entrata e di uscire dallo stesso attraverso il terminale di uscita consegnando il biglietto e pagando la somma richiesta per la sosta al suo interno. Tutto questo meccanismo è regolato da un semaforo e da due sbarre posizionate rispettivamente presso il terminale di entrata e di uscita. Inoltre per garantire la fault tolerance è stato aggiunto un database che contiene al suo interno tutti i ticket degli autisti ancora presenti all'interno del parcheggio in modo da non perdere i dati.

CAPITOLO 6: Requirement Engineering

2.2 Product function: il sistema prevede 4 funzioni principali:

- Erogazione del ticket per cliente in entrata;
- Ritiro e pagamento del ticket per cliente in uscita;
- Salvataggio del ticket erogato in entrata all'interno di un database;
- Controllo del semaforo e delle sbarre per la gestione corretta dei flussi di veicoli in entrata ed in uscita dal parcheggio.

2.3 User characteristics: gli utenti che si rivolgeranno a questo tipo di sistema sono coloro che guidano veicoli di ogni tipo e dimensione che necessita di un parcheggio dove sostare per qualsiasi motivo, da andare a fare una passeggiata fino ad andare a lavorare passando magari per fare delle compere.

CAPITOLO 6: Requirement Engineering

2.4 Constrains: gli utenti sono autorizzati soltanto a ritirare, pagare e consegnare il ticket. Di conseguenza si limitano semplicemente ad usufruire del parcheggio per svolgere le loro attività senza mettere mano a questioni interne al software.

CAPITOLO 6: Requirement Engineering

3. Specific requirements

3.1 External interface requirements

3.1.1 User Interfaces: le due user interfaces create svolgono due funzioni diverse. In particolare quella che si trova sul terminale d'entrata si occupa di erogare il ticket al guidatore nel caso in cui quest'ultimo preme il bottone per l'erogazione (e ci sia almeno un posto libero nel parcheggio), mentre quella sul terminale di uscita si occupa di ritirare il biglietto e di far pagare la somma prestabilita.

3.1.2 Hardware interfaces: l'interfaccia utente è screen-oriented.

CAPITOLO 6: Requirement Engineering

3.1 External interface requirements

3.1.3 External Software Interfaces: a livello di interfacce software esterne al progetto sono state utilizzate:

- la libreria prog4ed per la simulazione dell'input ticket e pagamento;
- la libreria Connectors di Java per la comunicazione con il database su MySQL.

3.1.4 Communication interfaces: l'interfacciamento database è stato creato utilizzando MySQL con la libreria Connectors di Java. La comunicazione, su MySQL, è di tipo socket TCP con autenticazione. Inoltre il terminale d'entrata/uscita comunica con il GestoreParcheggio attraverso porte socket.

CAPITOLO 6: Requirement Engineering

3.1 Functional requirements

3.1.1 Interfaccia d'entrata: uso di libreria Swing con implementazione CardLayout;

3.1.2 Interfaccia d'uscita: uso di libreria Swing con implementazione CardLayout;

CAPITOLO 6: Requirement Engineering

3.1 Functional requirements

3.1.3 Erogazione ticket: comunicazione socket con server per controllare disponibilità posti così da erogare il ticket in caso di riscontro positivo.

3.1.4 Ritiro e pagamento ticket: comunicazione socket per validazione ticket e pagamento di quest'ultimo.

CAPITOLO 6: Requirement Engineering

3.1 Functional requirements

3.1.5 Semafori: lo sviluppo di un'applicazione real time ha portato all'uso dei semafori per la gestione dei flussi di entrata/uscita dal parcheggio.



CAPITOLO 6: Requirement Engineering

3.3 Performance requirements: il sistema dovrà supportare un massimo di due flussi simultanei.

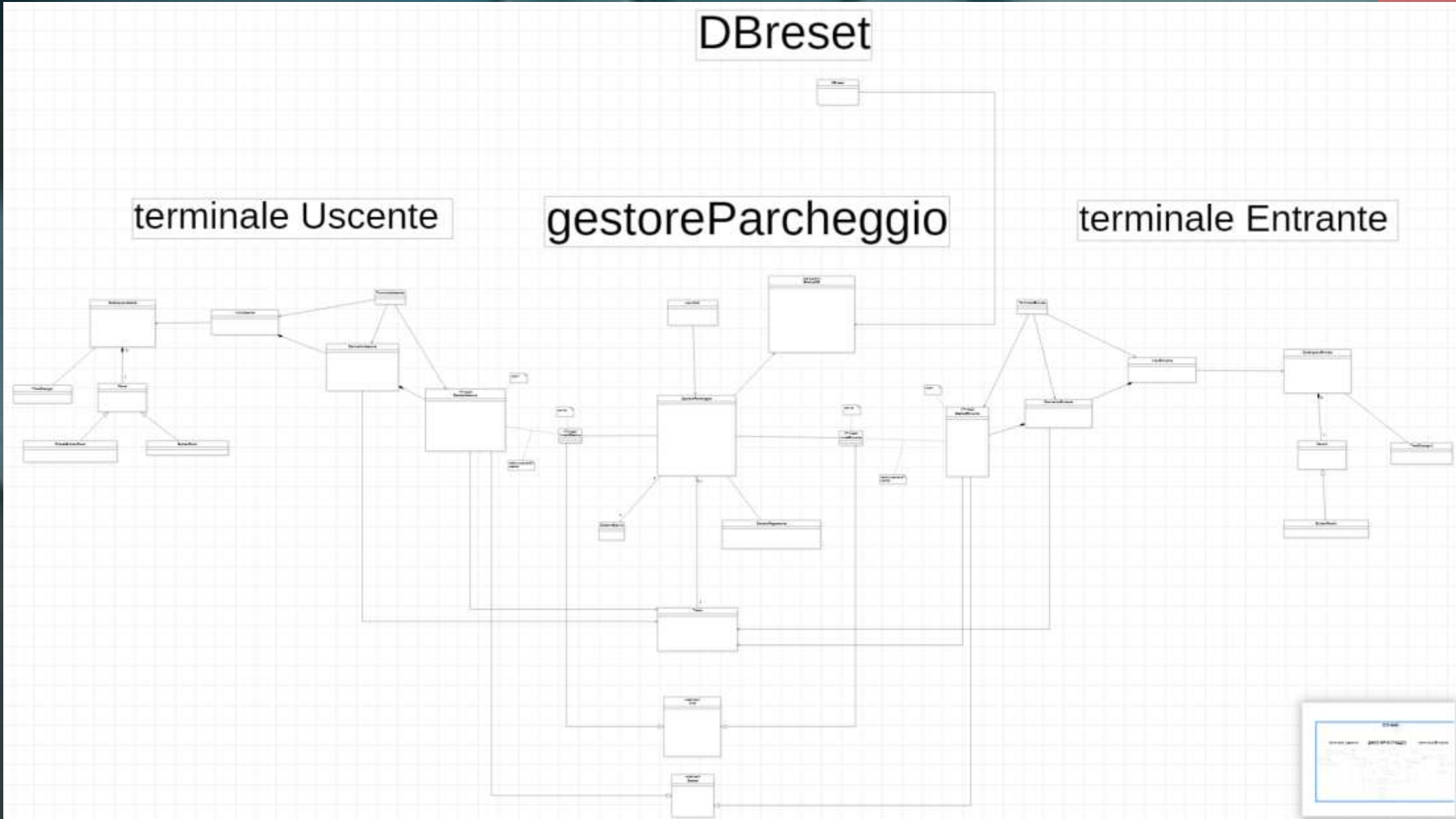
E' necessaria una connessione TCP stabile tra i vari componenti interconnessi tra loro.

Si vuole evitare una possibile starvation per i due terminali d'entrata e d'uscita.

CAPITOLO 6: Requirement Engineering

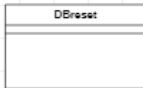
3.5 Software system attributes: tutte le caratteristiche di correttezza, affidabilità, efficienza, ecc legate al nostro software sono disponibili al Capitolo 5: Software Quality.

CAPITOLO 7: Modelling(Completo)

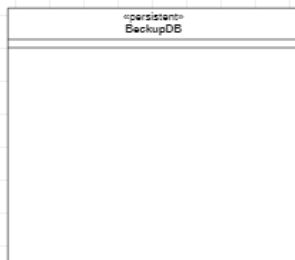


CAPITOLO 7: Modelling(DBReset)

DBreset



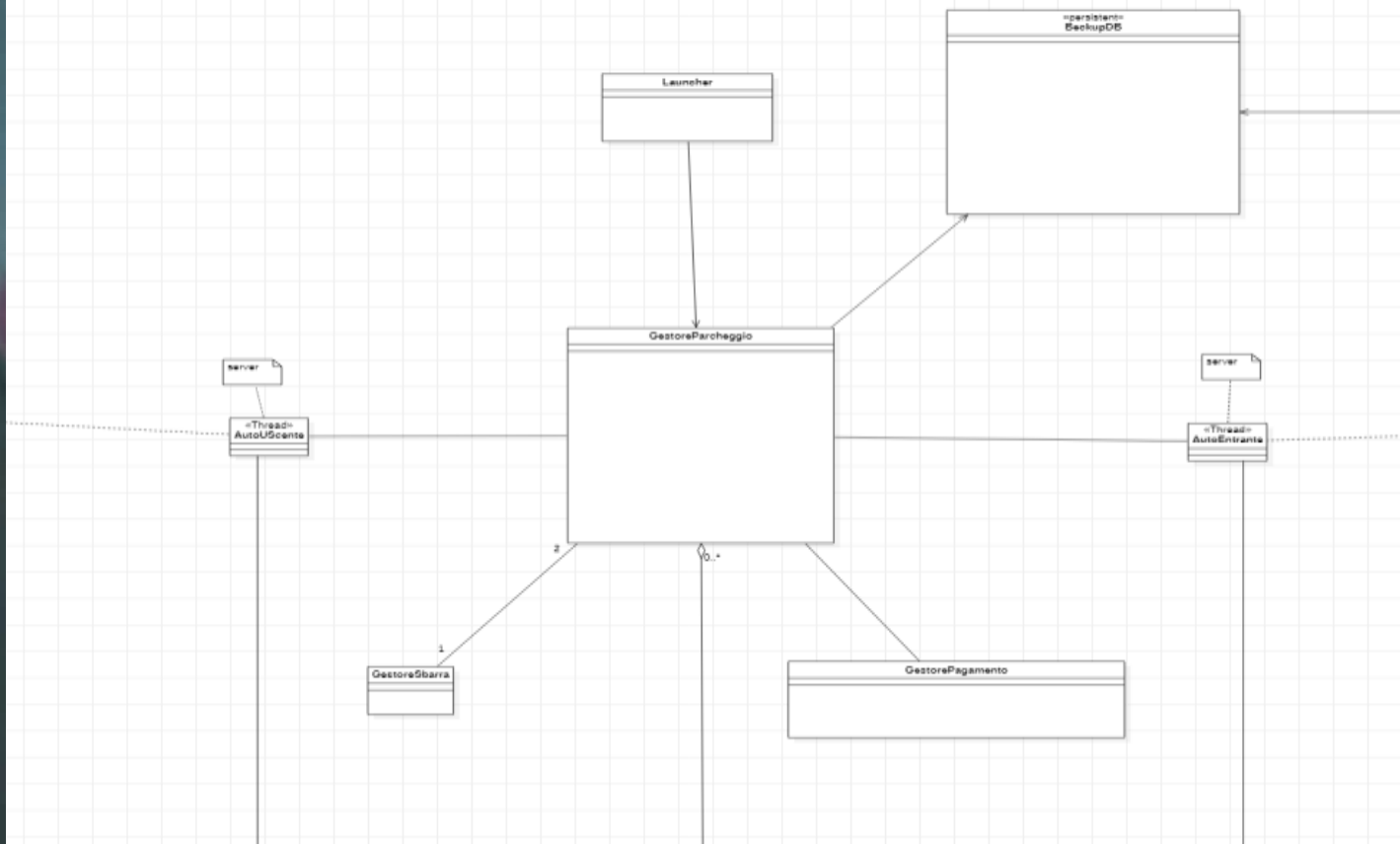
Parcheggio



CAPITOLO 7:

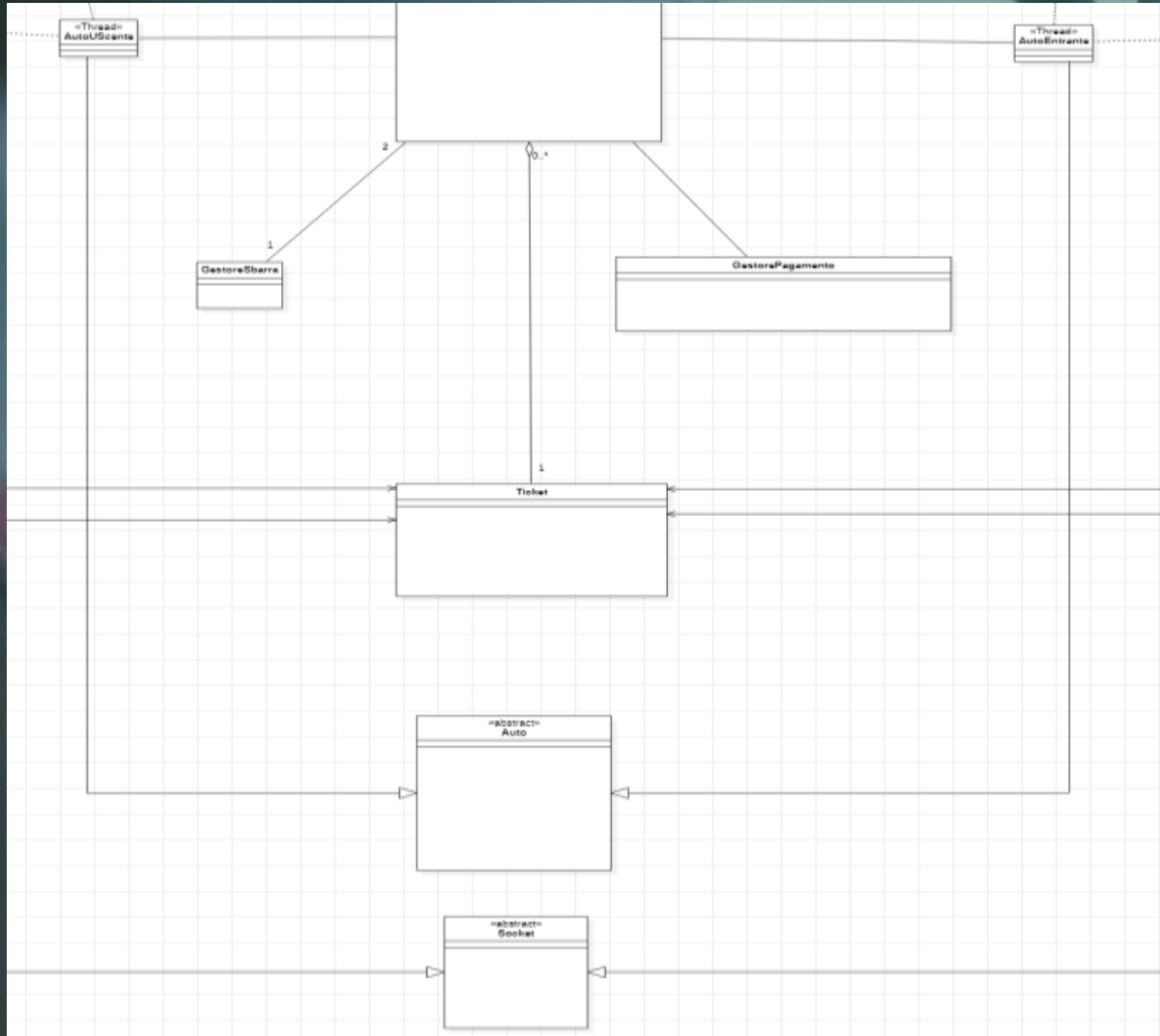
Modelling (GestoreParcheggio 1/2)

gestoreParcheggio



CAPITOLO 7:

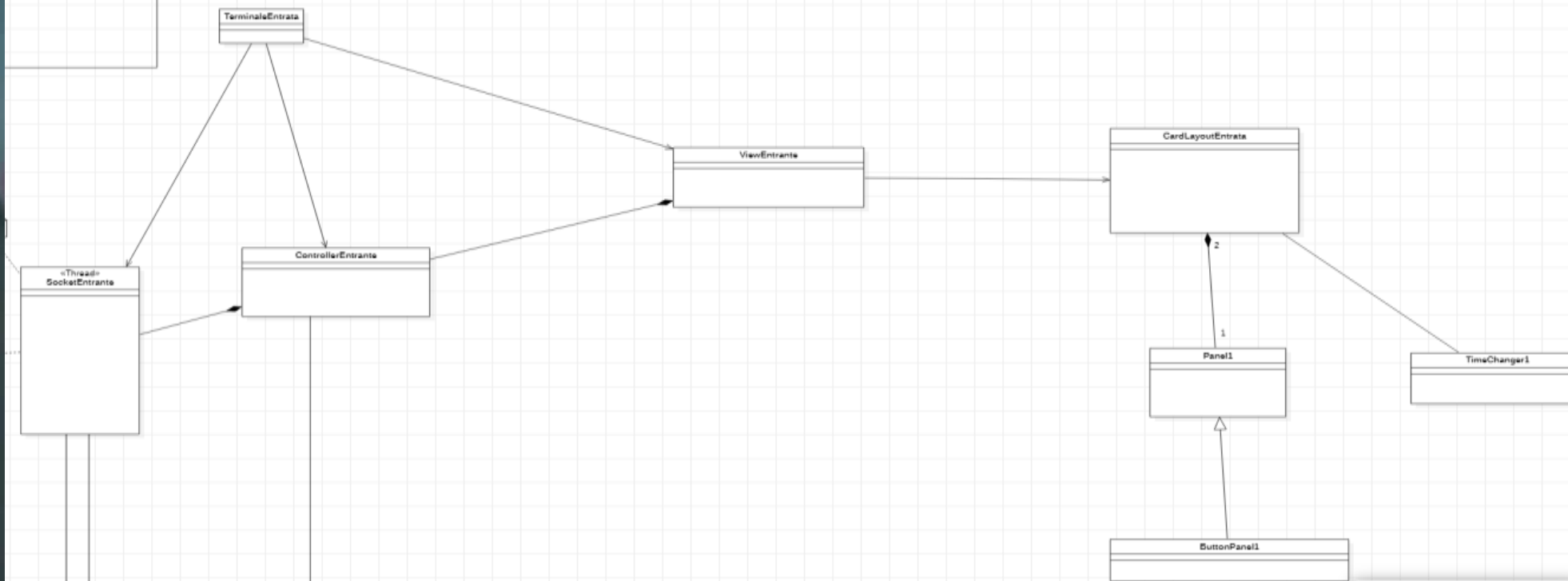
Modelling (GestoreParcheggio2/2)



CAPITOLO 7:

Modelling(TerminaleEntrante)

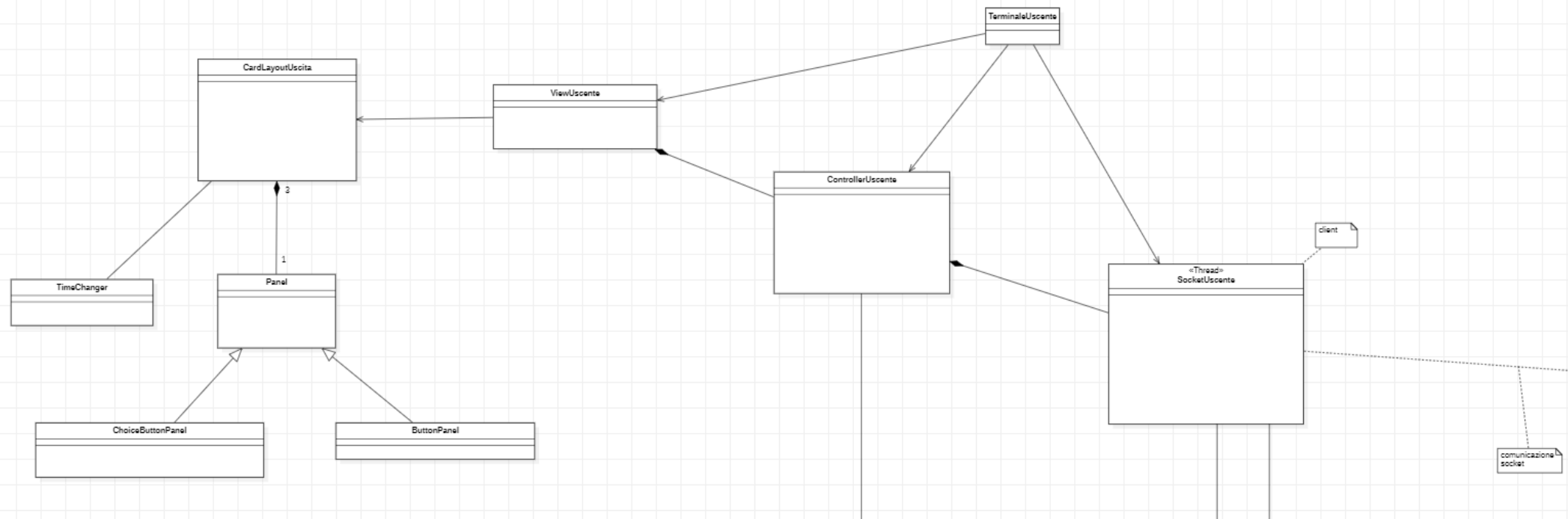
terminale Entrante



CAPITOLO 7:

Modelling(TerminaleUscente)

terminale Uscente



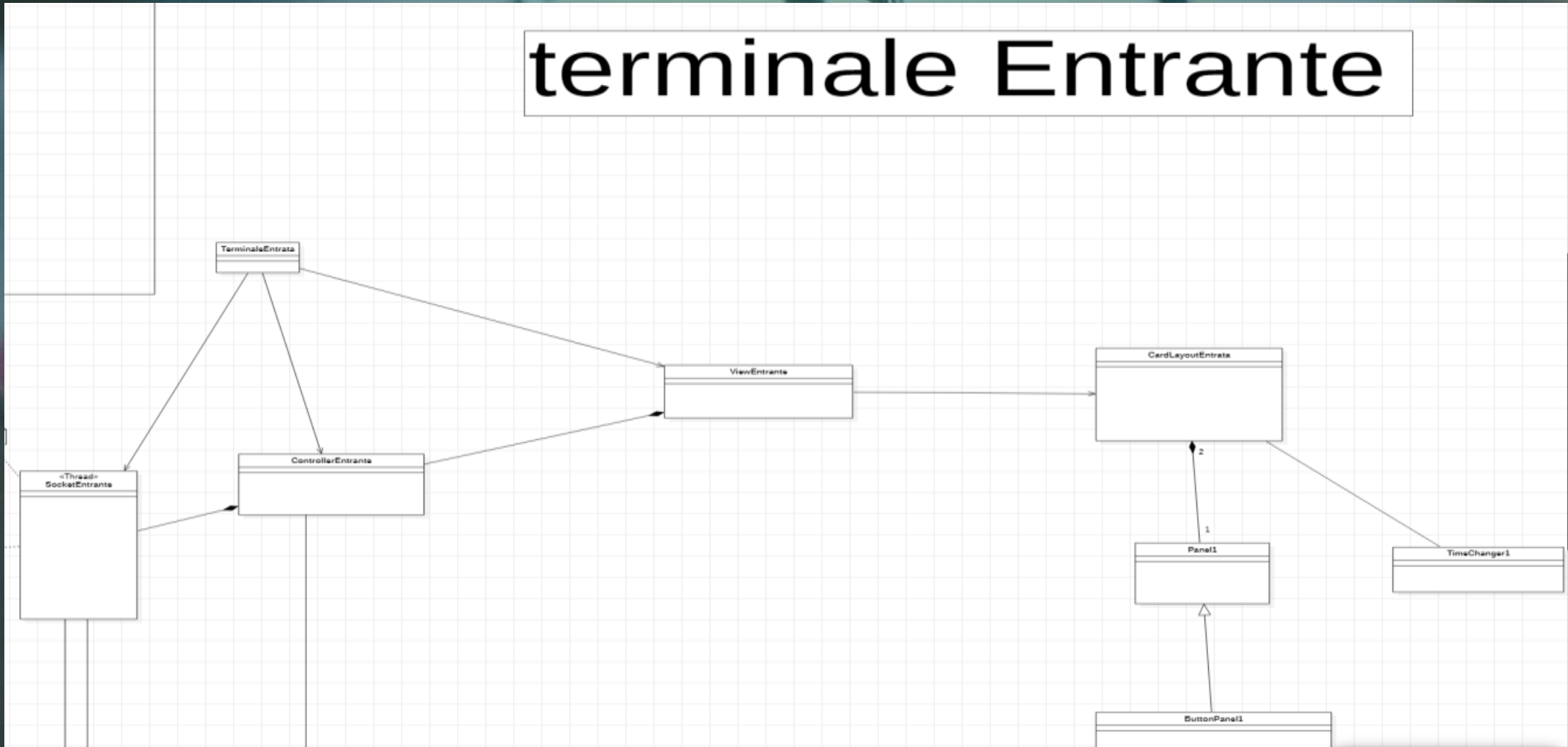
CAPITOLO 8: Software Architecture

Come modello architetturale per la realizzazione del nostro progetto abbiamo scelto il Model-View-Controller. MVC prevede un'architettura composta da tre parti diverse: i dati (Model), la visualizzazione dei dati (View) e la gestione degli input (Controller). Questi tre componenti sono interconnessi: lo stato del sistema (Model) viene mostrato tramite la View all'utente, il quale produce gli input con cui il Controller aggiorna il Model. Nel nostro caso, la View è la parte di interfaccia utente che si trova sui due terminali di entrata e uscita; il Controller è la parte di gestione del parcheggio che si occupa dello scambio di messaggi tra i terminali ed il GestoreParcheggio per regolare i flussi di entrata e di uscita; il Model è la parte contenente tutti i ticket che sono stati erogati e che sono ancora presenti all'interno del database e quindi rappresenta tutti quei clienti non ancora usciti dal parcheggio.

CAPITOLO 8: Software Architecture

Architectural View Terminale Entrante

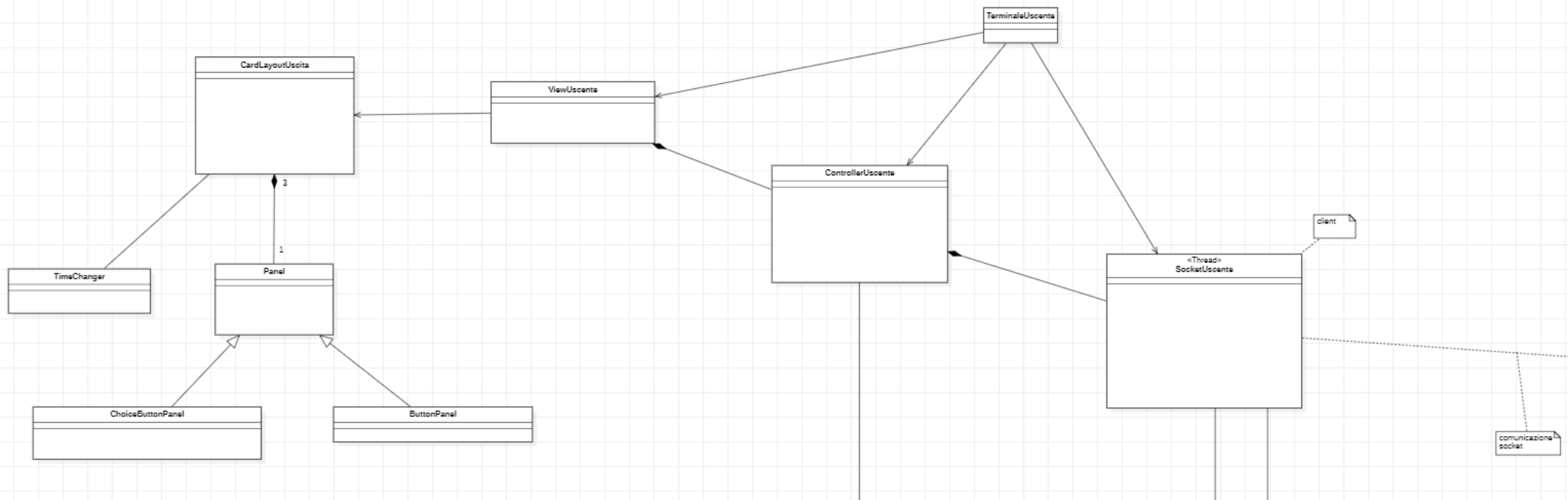
terminale Entrante



CAPITOLO 8: Software Architecture

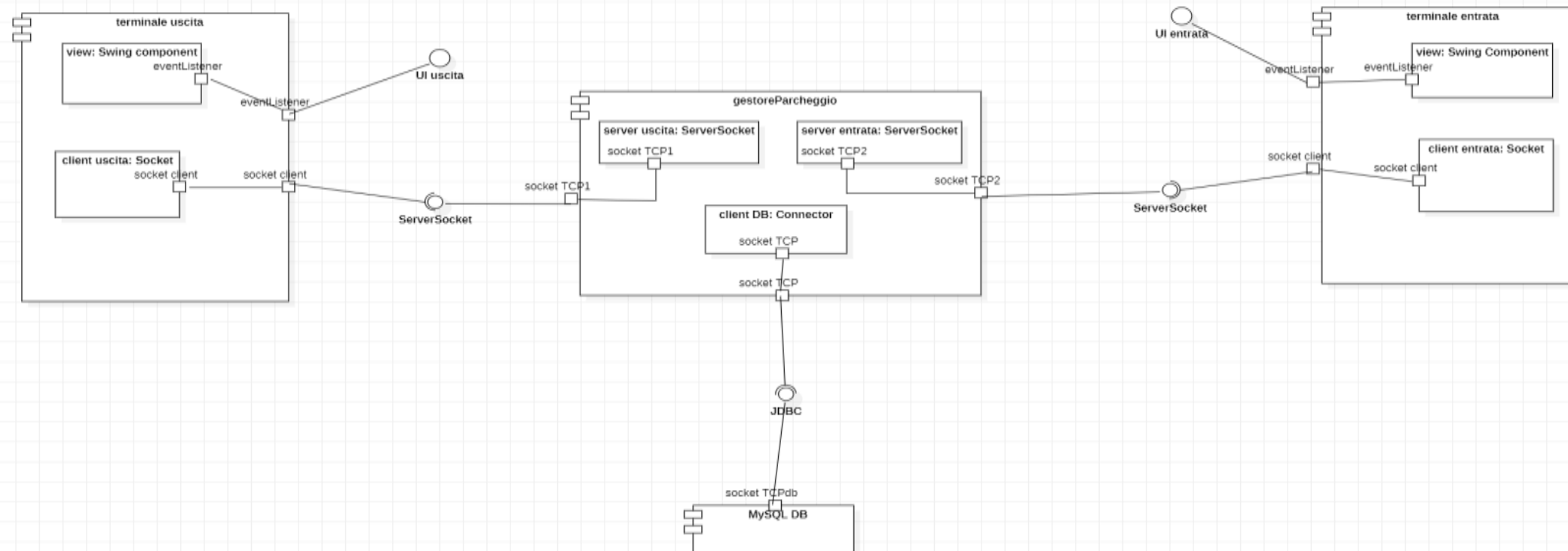
Architectural View Terminale Uscente

terminale Uscente



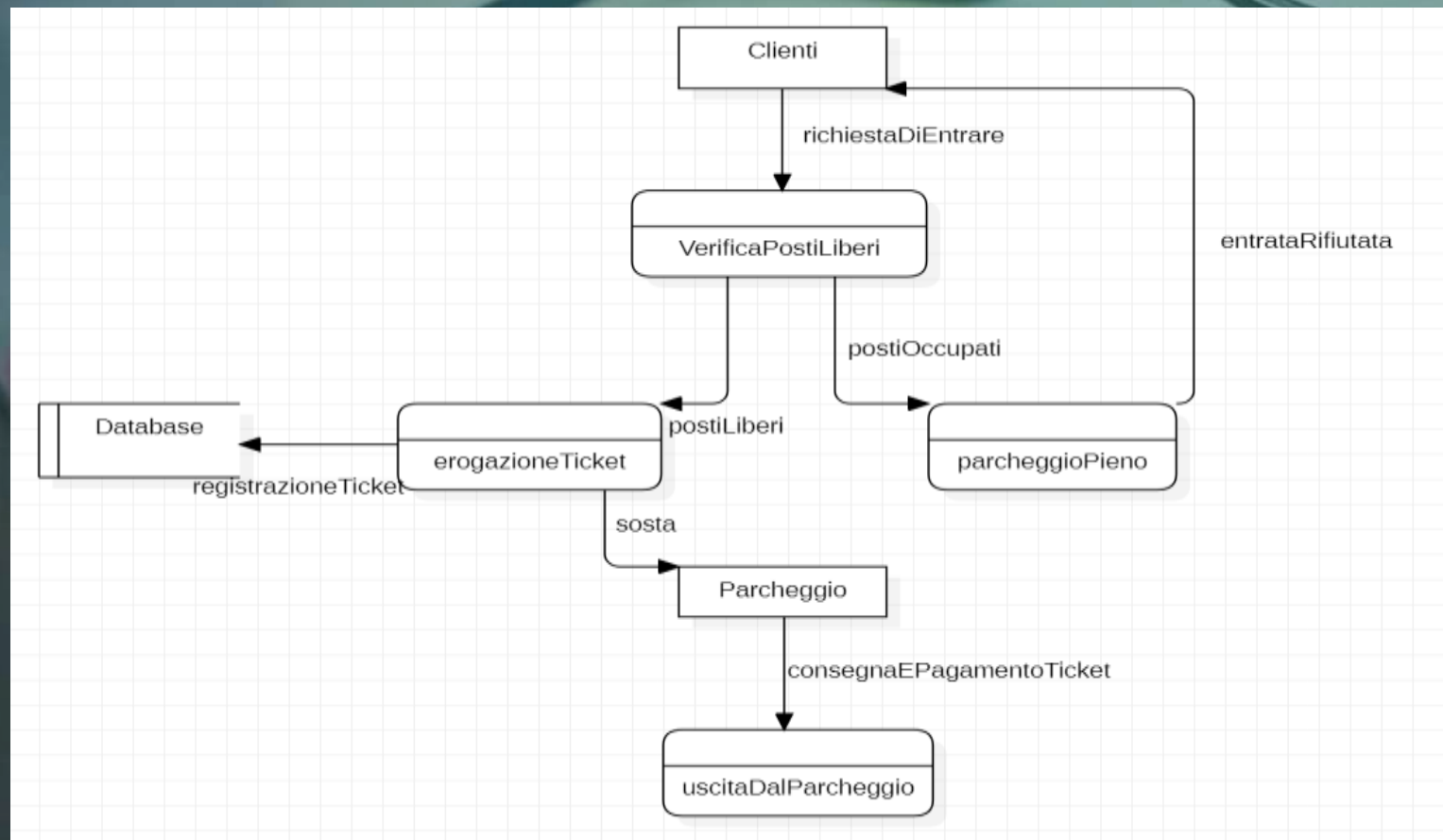
CAPITOLO 8: Software Architecture

Vista connettori e componenti



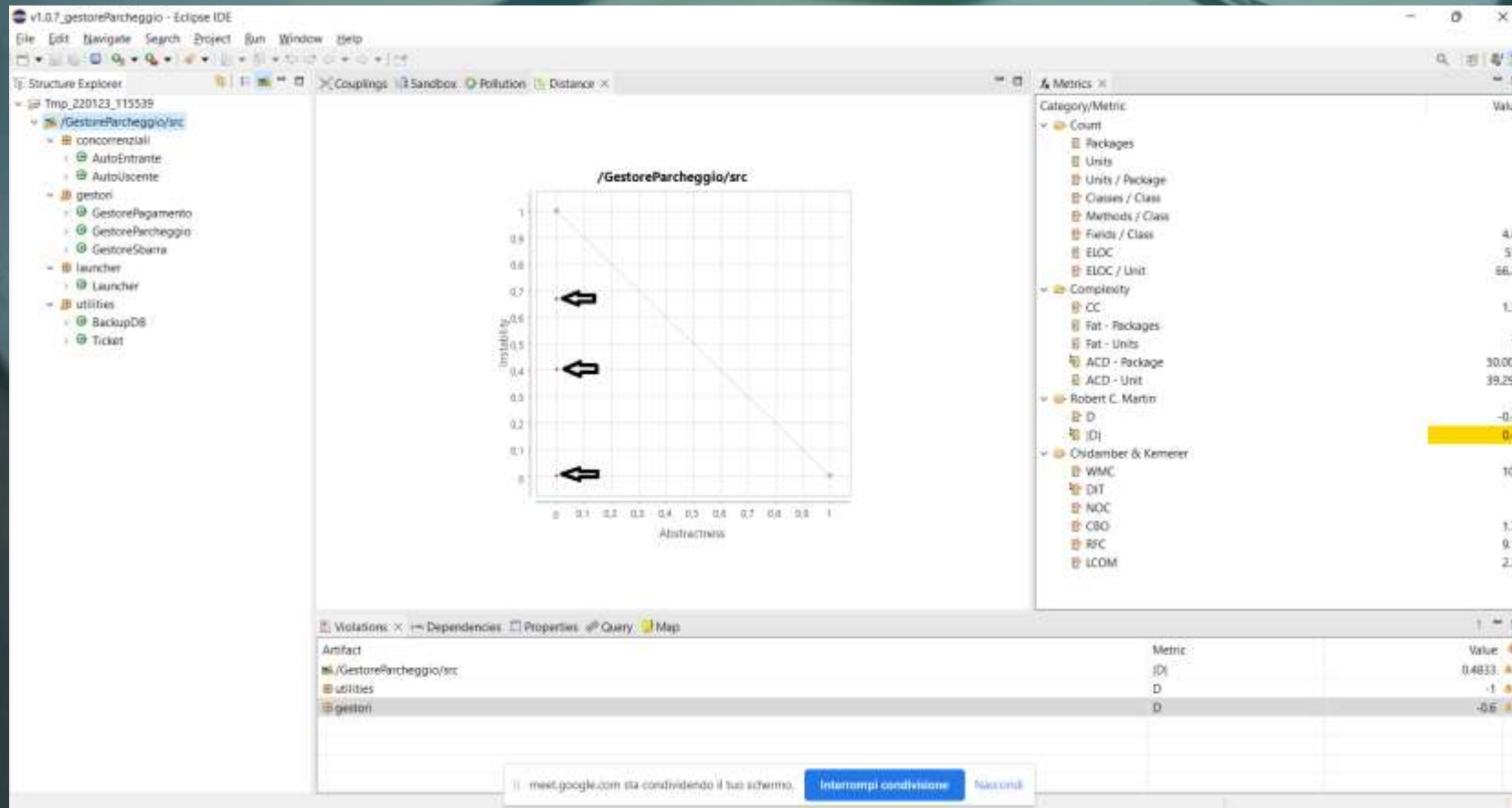
CAPITOLO 9: Software Design

Design Architecture:



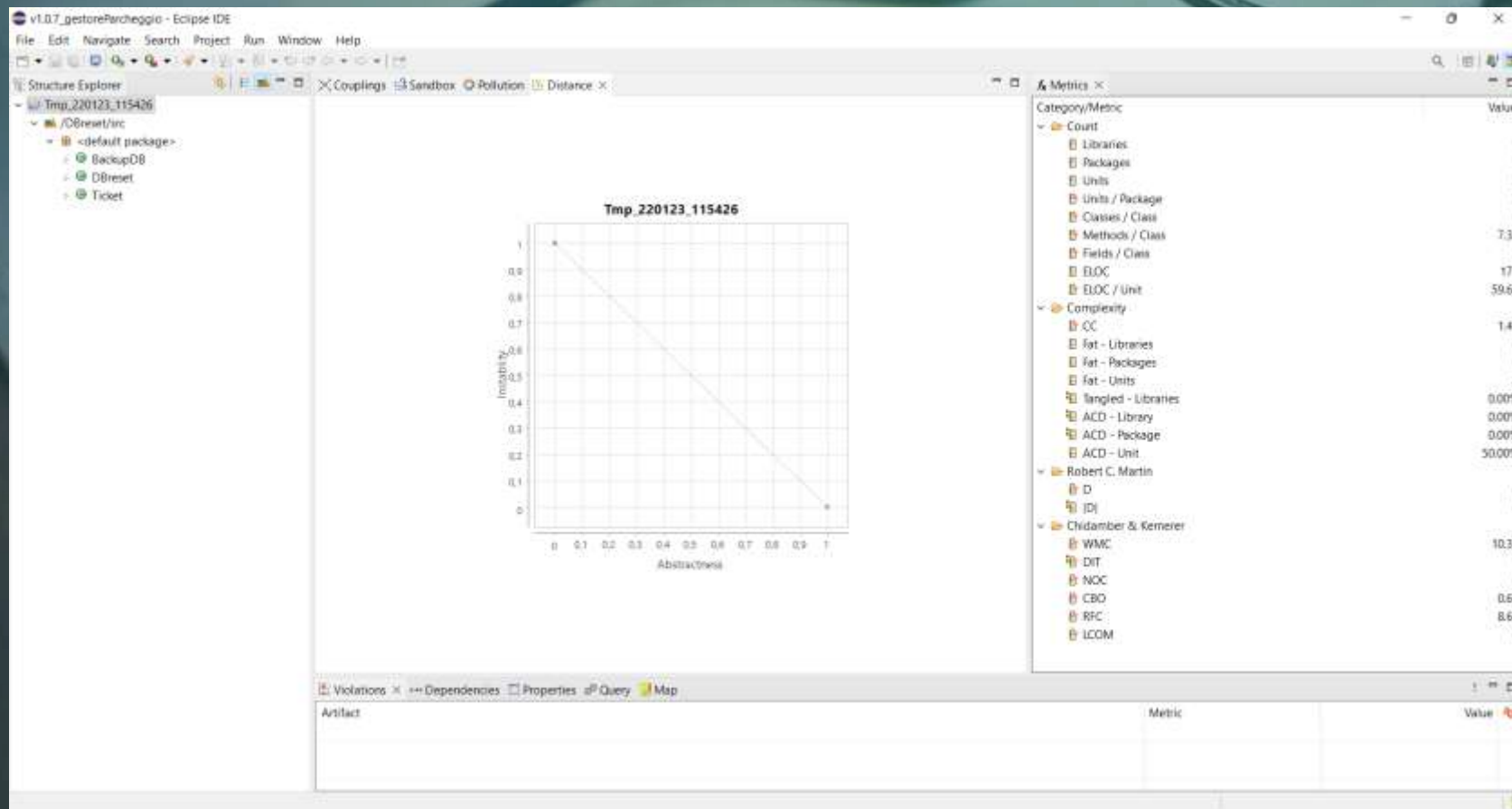
CAPITOLO 9: Software Design

Analisi STAN IDE parcheggio:



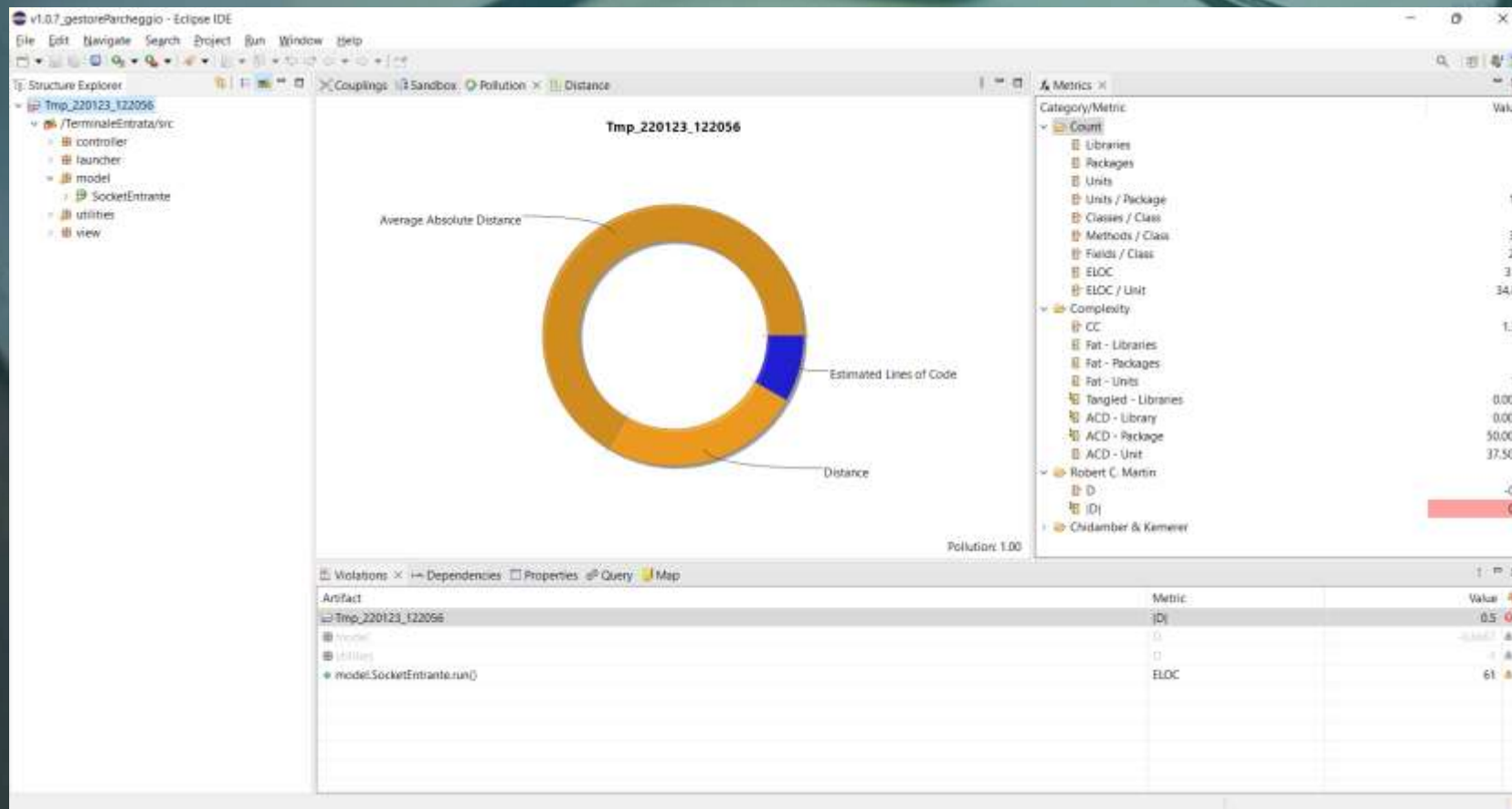
CAPITOLO 9: Software Design

Analisi STAN IDE DBreset:



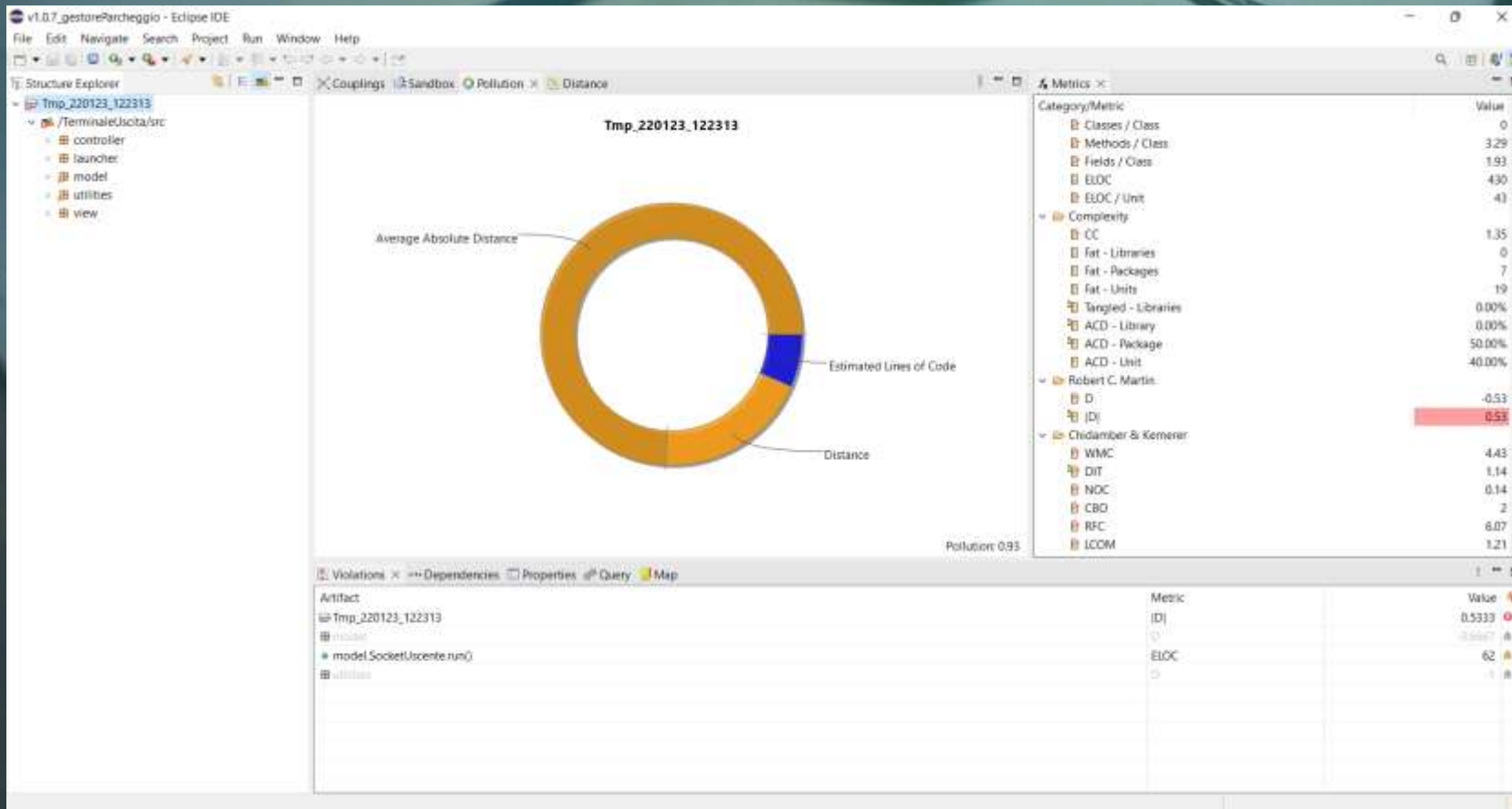
CAPITOLO 9: Software Design

Analisi STAN IDE terminale entrante:



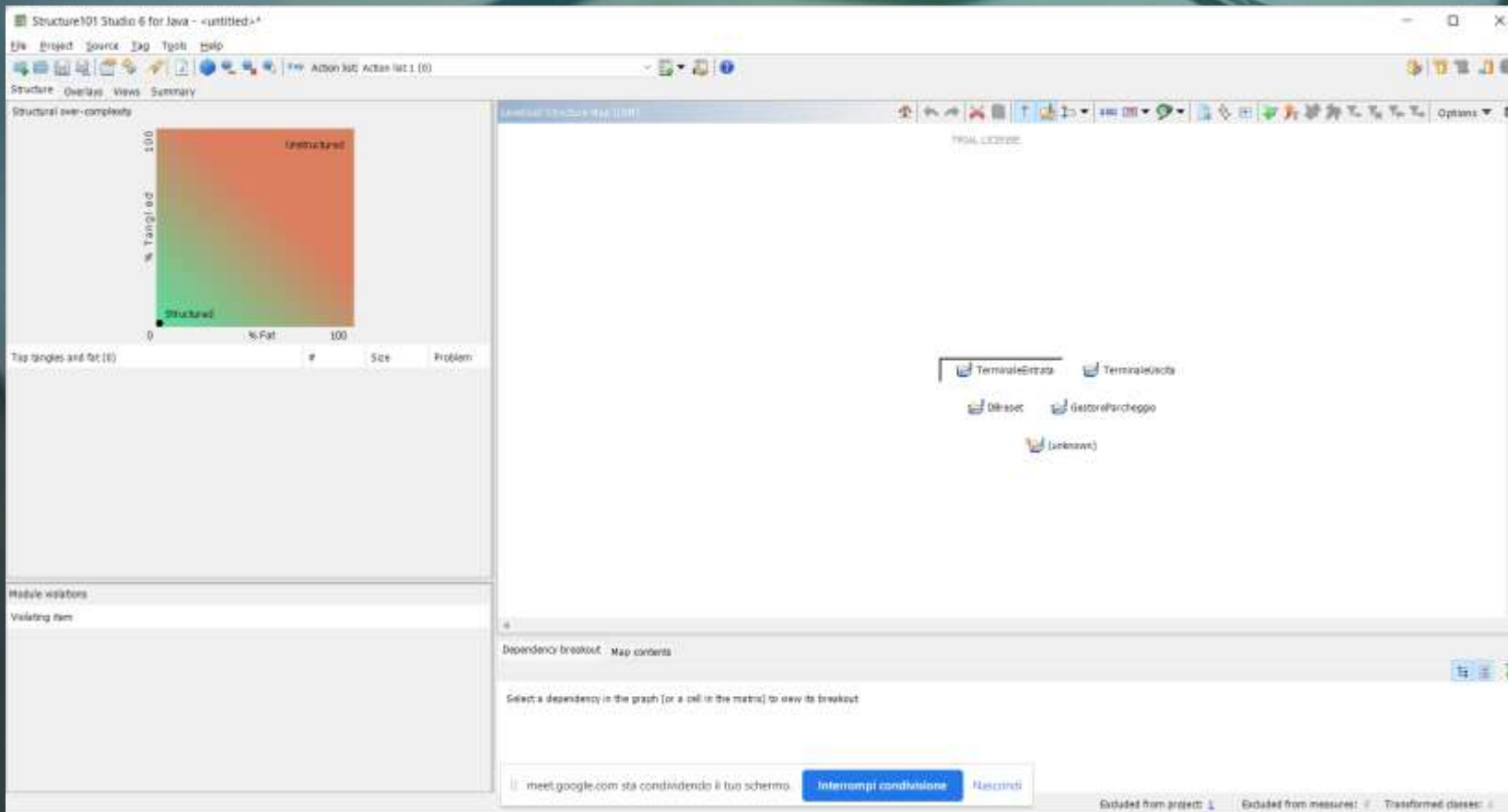
CAPITOLO 9: Software Design

Analisi STAN IDE terminale uscente:



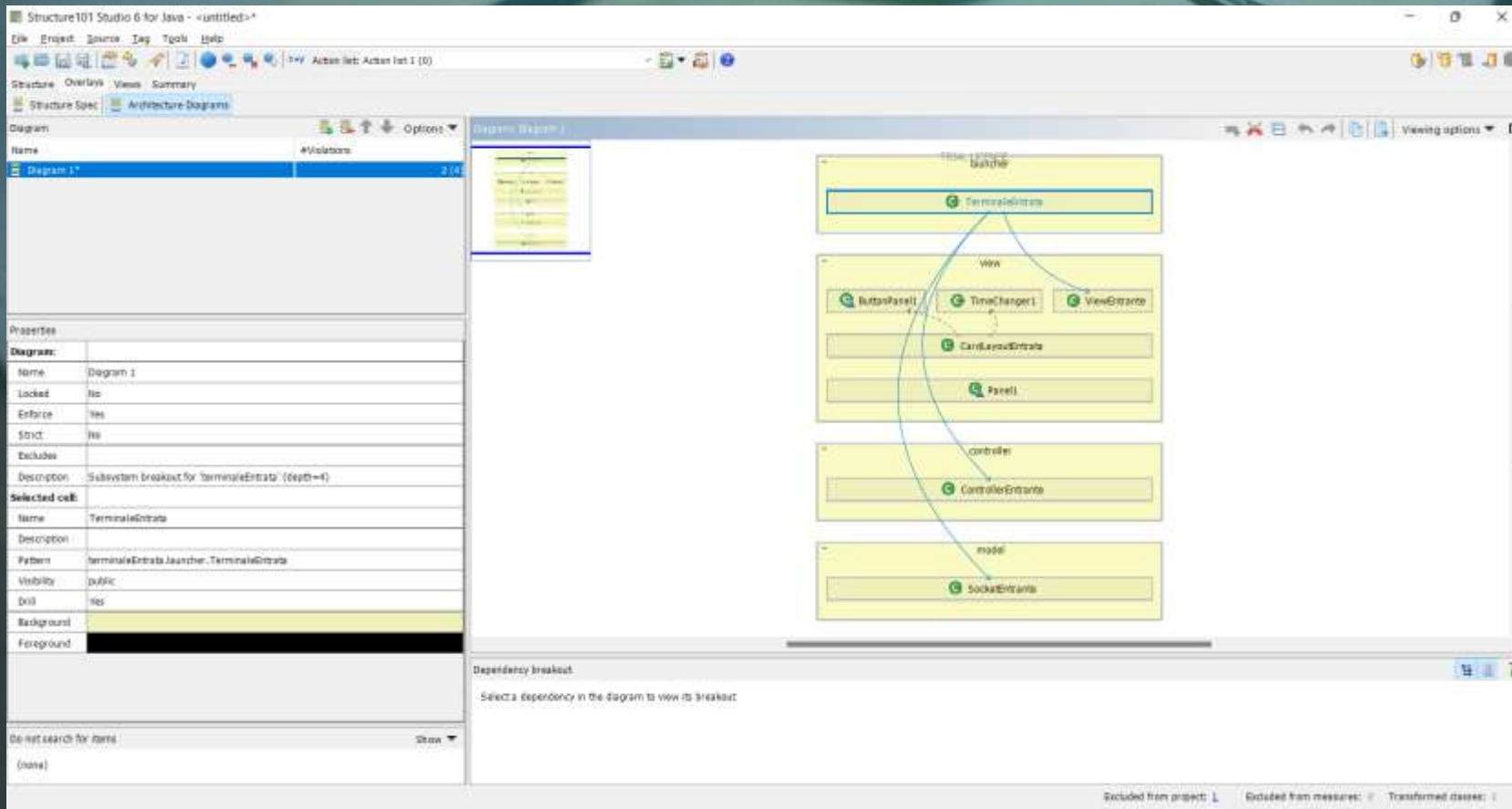
CAPITOLO 9: Software Design

Analisi structure101 generica:



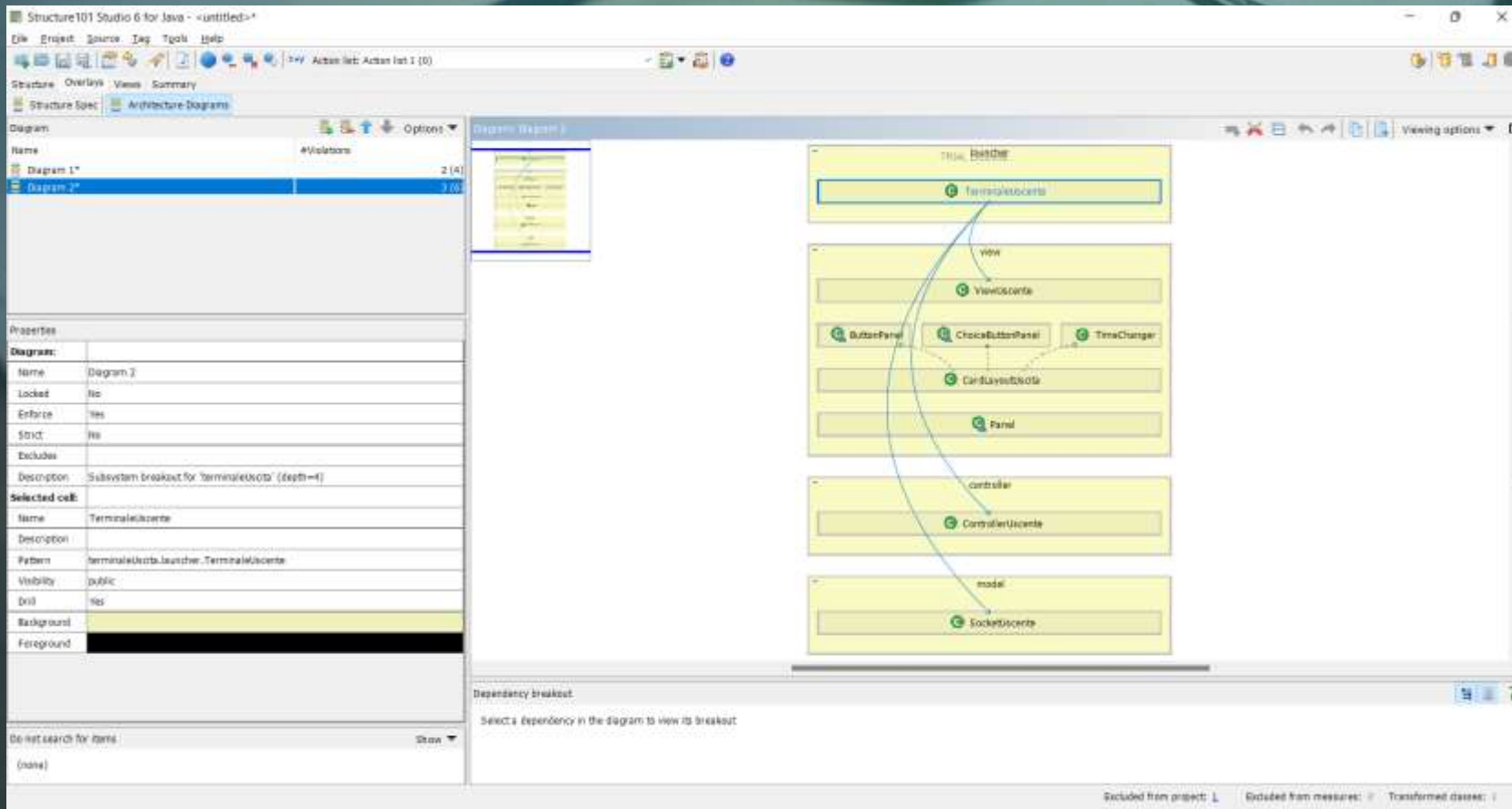
CAPITOLO 9: Software Design

Analisi structure101 terminale entrata:



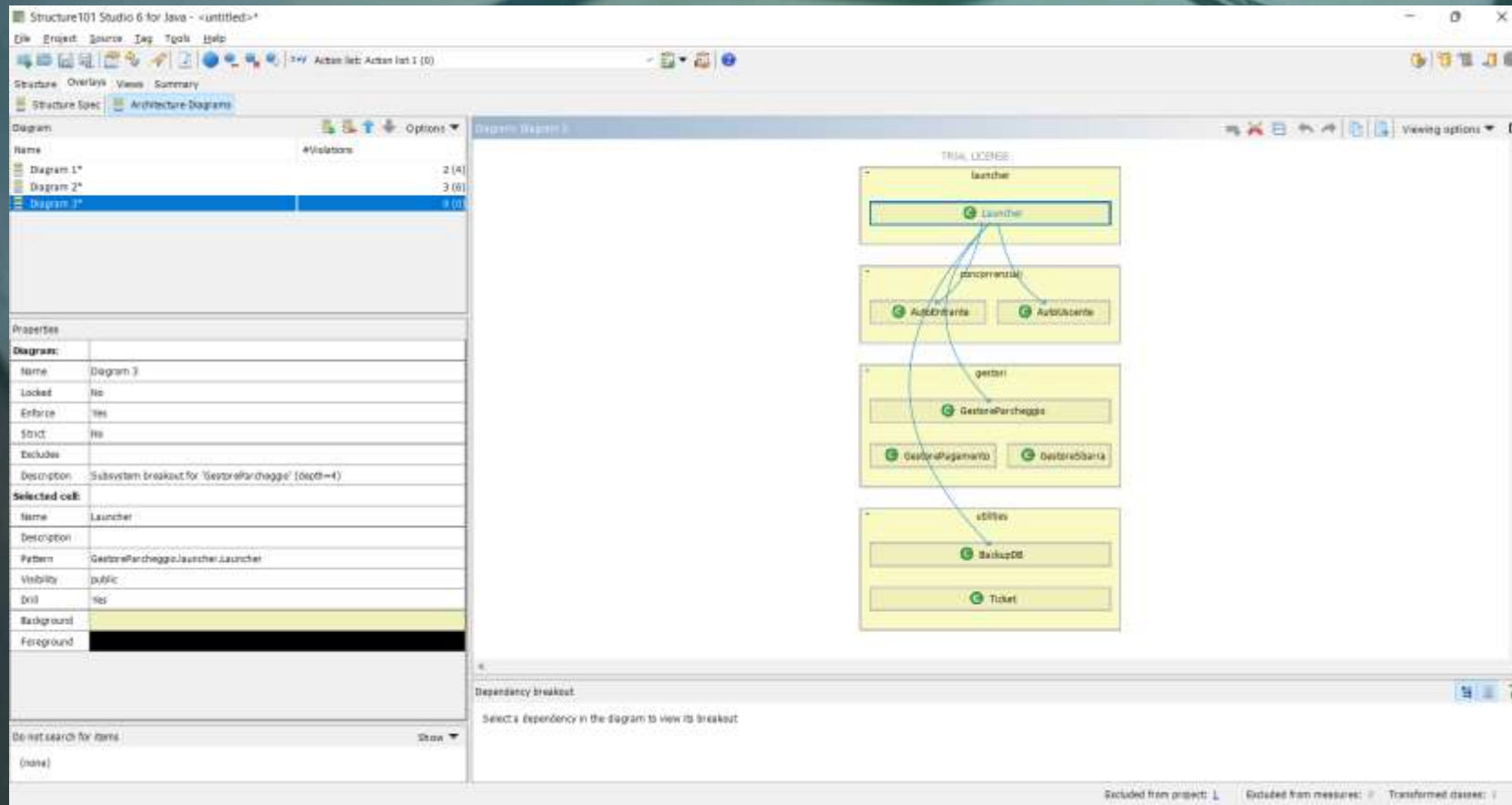
CAPITOLO 9: Software Design

Analisi structure101 terminale uscente:



CAPITOLO 9: Software Design

Analisi structure101 server:



CAPITOLO 9: Software Design

Analisi structure101 totale:

The screenshot displays the Structure101 Studio 6 for Java interface. The main window shows a software design diagram with three columns representing different components: **terminaInizio**, **GestoreParcheggi**, and **terminaEntrata**. Each column contains a hierarchy of elements: **launcher**, **view**, **controller**, and **model**. Below these, there are **DBreset** and **BackupDB** elements. At the bottom, there are **com**, **prog**, and **Ticket** elements. The diagram is connected by numerous dashed lines representing dependencies.

On the left side, the **Diagram** pane lists several diagrams, with **Diagram 5** selected. Below this, the **Properties** pane shows the details for **Diagram 5**, including its name, locked status, enforce status, strict status, excludes, and description (Top-level breakout). The **Selected cell** pane shows the details for the selected cell, including its name, description, pattern, visibility, drill status, background, and foreground.

At the bottom, the **Dependency breakout** pane shows a list of dependencies, with a note to "Select a dependency in the diagram to view its breakout".

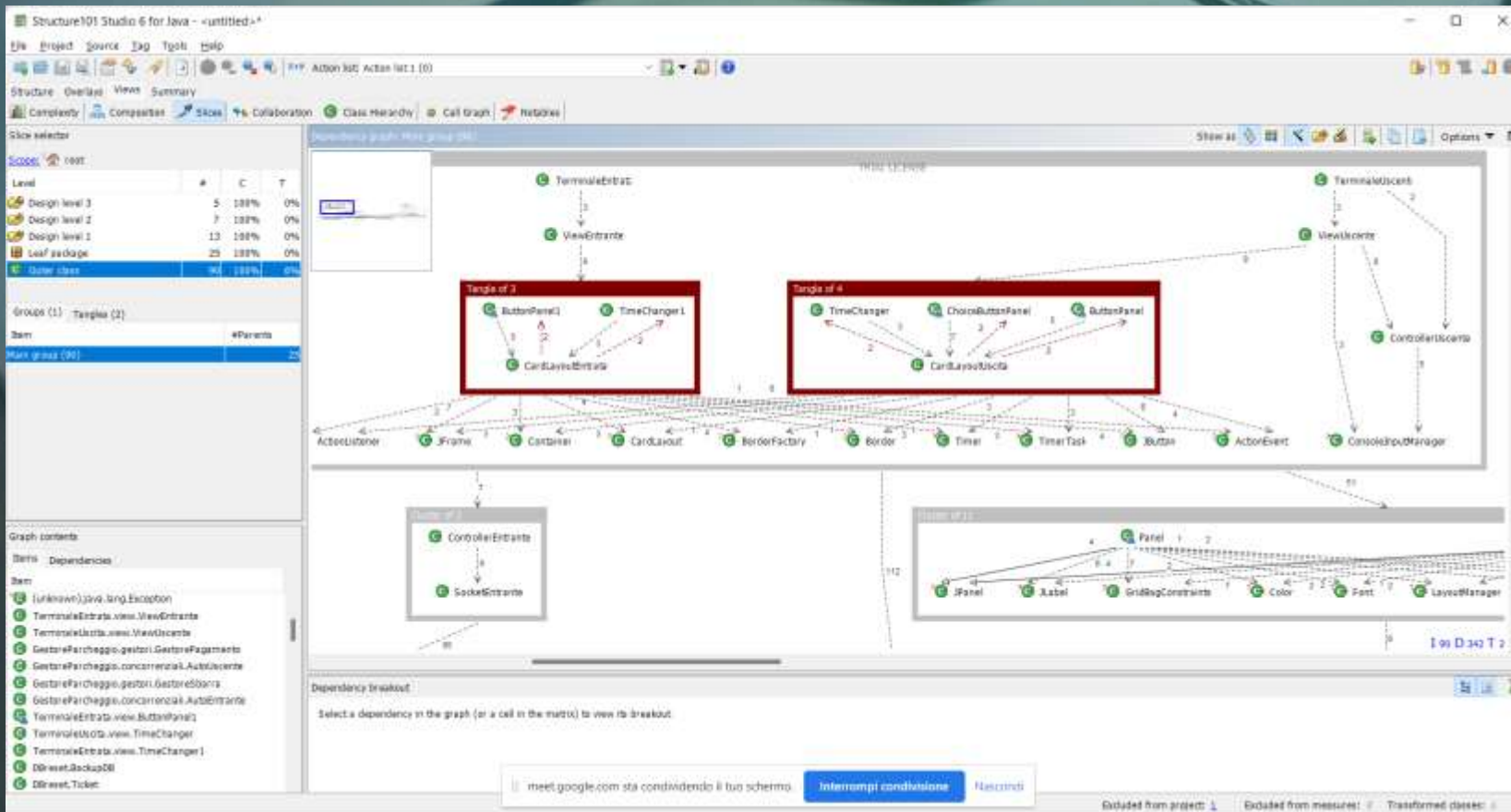
The **Diagram** pane also includes a table showing the number of violations for each diagram:

Diagram	#Violations
Diagram 1*	2 (4)
Diagram 2*	3 (6)
Diagram 3*	0 (0)
Diagram 4*	0 (0)
Diagram 5*	3 (11)

A tooltip indicates that the number in parentheses represents the number of weighted violations (number of unique violations).

CAPITOLO 9: Software Design

Analisi structure101 cicli controllati CardLayout:



CAPITOLO 9: Software Design(Pattern applicati)

Delegation Pattern: l'obiettivo è quello di minimizzare il costo di sviluppo del software andando a riutilizzare dei metodi già creati. L'uso di questo pattern è presente nella creazione delle due interfacce utente presenti sui terminali di entrata ed uscita.

Observer Pattern: l'obiettivo è quello di massimizzare la flessibilità del sistema permettendo a tutti gli oggetti, dipendenti da un altro oggetto, di essere notificati e quindi cambiare il loro stato se il «super-oggetto» lo cambia.

L'uso di questo pattern è presente nella creazione dei terminali di entrata ed uscita.

CAPITOLO 10: Software Testing

Nel corso dello sviluppo del software sono state effettuate una serie di attività di testing.

In particolare è stato adottato il Manual Testing cioè una procedura per cui i test sul software sono eseguiti manualmente dai programmatori senza utilizzo di strumenti automatizzati.

Il testing viene utilizzato per identificare eventuali bug, problemi e difetti nell'applicativo software.

Il Manual testing ha come caratteristica quella di aiutare particolarmente a trovare bug critici.

Lo svantaggio è che necessita di un maggiore sforzo per valutare la fattibilità dell'automazione.

L'obiettivo principale del Manual Testing è che l'applicazione sia priva di errori e che il software soddisfi i functional requirements.

CAPITOLO 10: Software Testing

La maggior parte dei testing effettuati si sono basati sui paradigmi dell'Integration testing e del White Box testing. Integration testing → Un livello di test che si concentra sulle interazioni tra componenti o sistemi.

L'Integration Testing è un'attività che serve a verificare:

- l'integrazione tra le varie unità/moduli sviluppati;
- l'interazione dei moduli con il sistema;
- che i requisiti software sia a basso livello che ad alto livello siano soddisfatti.

CAPITOLO 10: Software Testing

Gli obiettivi dell'attività di Integration Testing sono i seguenti:

- testare l'integrazione tra componenti appena possibile in modo da ridurre il rischio;
- verificare che i comportamenti funzionali e non funzionali dei moduli siano progettati come da specifiche;
- trovare eventuali difetti nelle integrazioni tra moduli, interfacce, API, micro-servizi, database, strumenti di terze parti, sistema, sotto-sistemi;
- trovare eventuali difetti quando vengono apportate modifiche a moduli, componenti, interfacce, ecc...;
- cercare di evitare di far passare i difetti ai livelli successivi di test che ne aumenterebbero la complessità.

CAPITOLO 10: Software Testing

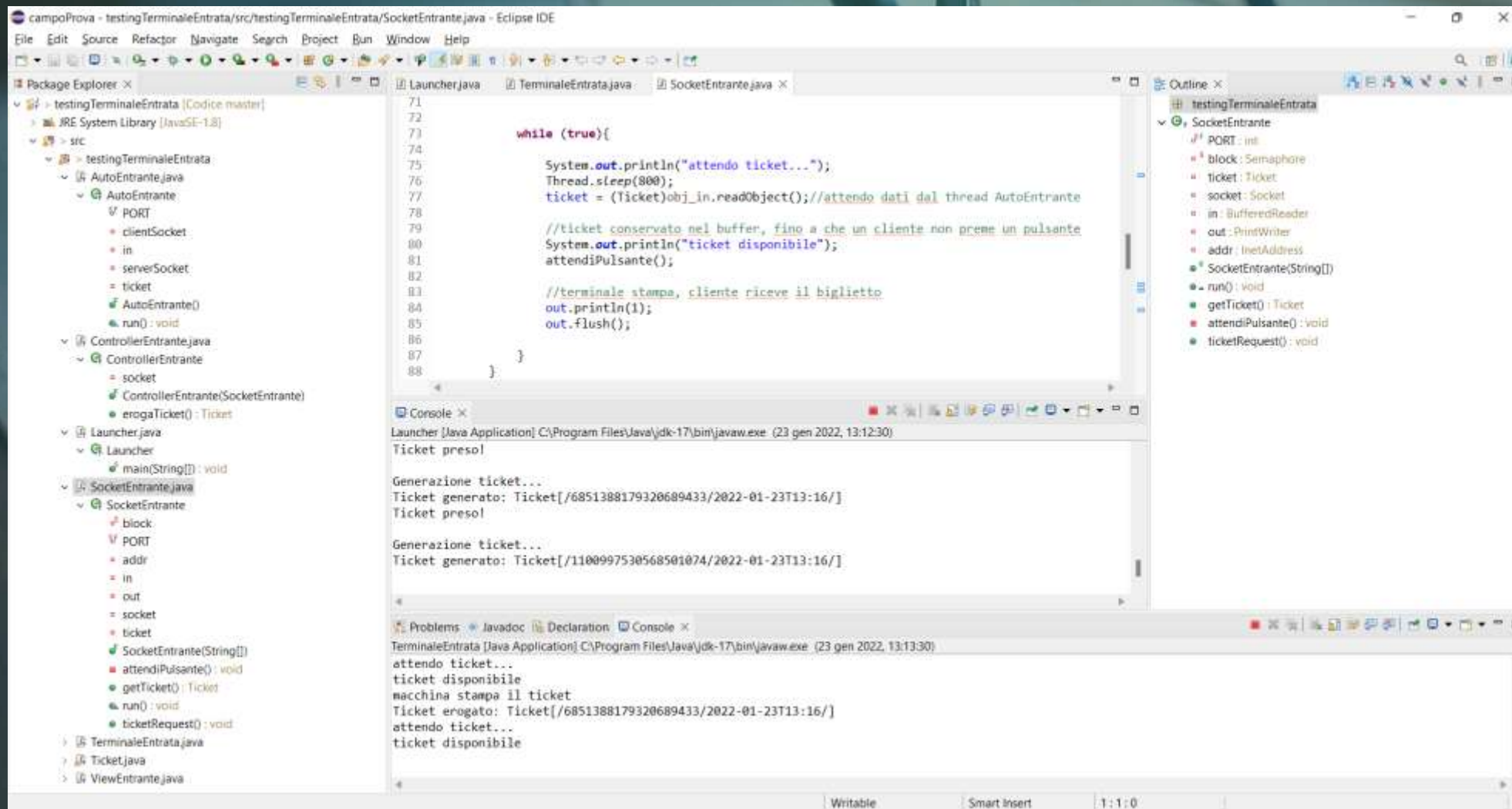
White Box testing → Un livello di test in cui le strutture interne, il design ed il codice del software sono testati per verificare i flussi di input-output e per migliorare la sicurezza e l'usabilità.

In questo tipo di testing il codice è visibile a coloro che lo testano.

Il White Box testing è un'attività che serve a verificare:

- eventuali problemi di sicurezza interni al software;
- il flusso degli input;
- i valori attesi in output;
- il testare individualmente qualsiasi componente del codice software;
- verificare che il codice sia scritto bene;
- alcune funzionalità all'interno di possibili cicli.

CAPITOLO 10: Software Testing



CAPITOLO 10: Software Testing

Nell'immagine precedente è stato testato il prototipo `v0.4.0_testingTerminaleEntrata_v2.0`.

In questo caso di testing abbiamo verificato il corretto funzionamento dello scambio di messaggi tra terminale d'entrata e server, ovvero della corretta generazione e passaggio dei ticket.

L'esecuzione del test è automatizzata.

CAPITOLO 10: Software Testing

campoProva - testDB/src/testingTerminaleUscita/Launcher.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer: X

- testDB [Codice master]
- JRE System Library [JavaSE-1.8]
- src
 - testingTerminaleUscita
 - AutoEntrante.java
 - AutoUsciente.java
 - BackupDB.java
 - ControllerUsciente.java
 - DBreset.java
 - GestorePagamento.java
 - GestoreParcheggio.java
 - GestoreSbarra.java
 - Launcher.java
 - Parcheggio.java
 - ParcheggioSimulato.java
 - Semaforo.java
 - SleepEntrante.java
 - SleepUtilities.java
 - SocketUsciente.java
 - TerminaleEntrante.java
 - TerminaleUsciente.java
 - Ticket.java
 - ViewUsciente.java
- Referenced Libraries

TerminaleEntrante.java TerminaleUsciente.java Launcher.java X

```
6 Launcher
7
8 static void main(String args[]) {
9     chggio server;
10    kupDB db;
11    {
12        db = new BackupDB("jdbc:mysql://localhost:3300","root","admin");
13        db.setupDB();
14        db.describeTableDB();
15    }
16    server = new GestoreParcheggio(db);
17
18    //ho "cortocircuitato" il thread per le macchine d'entrata per testare se funziona il terminale
19    // (contesto non concorrenziale)
20
21    //dentro ho "cortocircuitato" anche il parcheggio simulato
22
23    //ho anche cortocircuitato i semafori relativi alla simulazione
24
25    //Thread entranteThread = new Thread(new AutoEntrante(server));
26    Thread uscenteThread = new Thread(new AutoUsciente(server));
27
28    uscenteThread.start();
29    //entranteThread.start();
30
31
32    db.closeDB();
33
34
35    ch (IOException e) {
36        // TODO Auto-generated catch block
```

Outline X

- testingTerminaleUscita
 - Launcher
 - main(String[]):void

Console X

Launcher (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe

Struttura tabella di backup:
barcode anno mese giorno ora minuti

Stato del backup....

EchoServer D'USCITA: started
Server Socket D'USCITA: ServerSocket[addr=0.0.0.0/0
(SERVER_USCITA) Connection accepted: Socket[addr=0.0.0.0/0

Problems X Javadoc X Declaration X Console X

TerminaleUsciente [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (23 gen 2022, 13:23:30)

EchoClient: started
Client Socket: Socket[addr=localhost/127.0.0.1,port=1050,localport=63261]
codice biglietto:

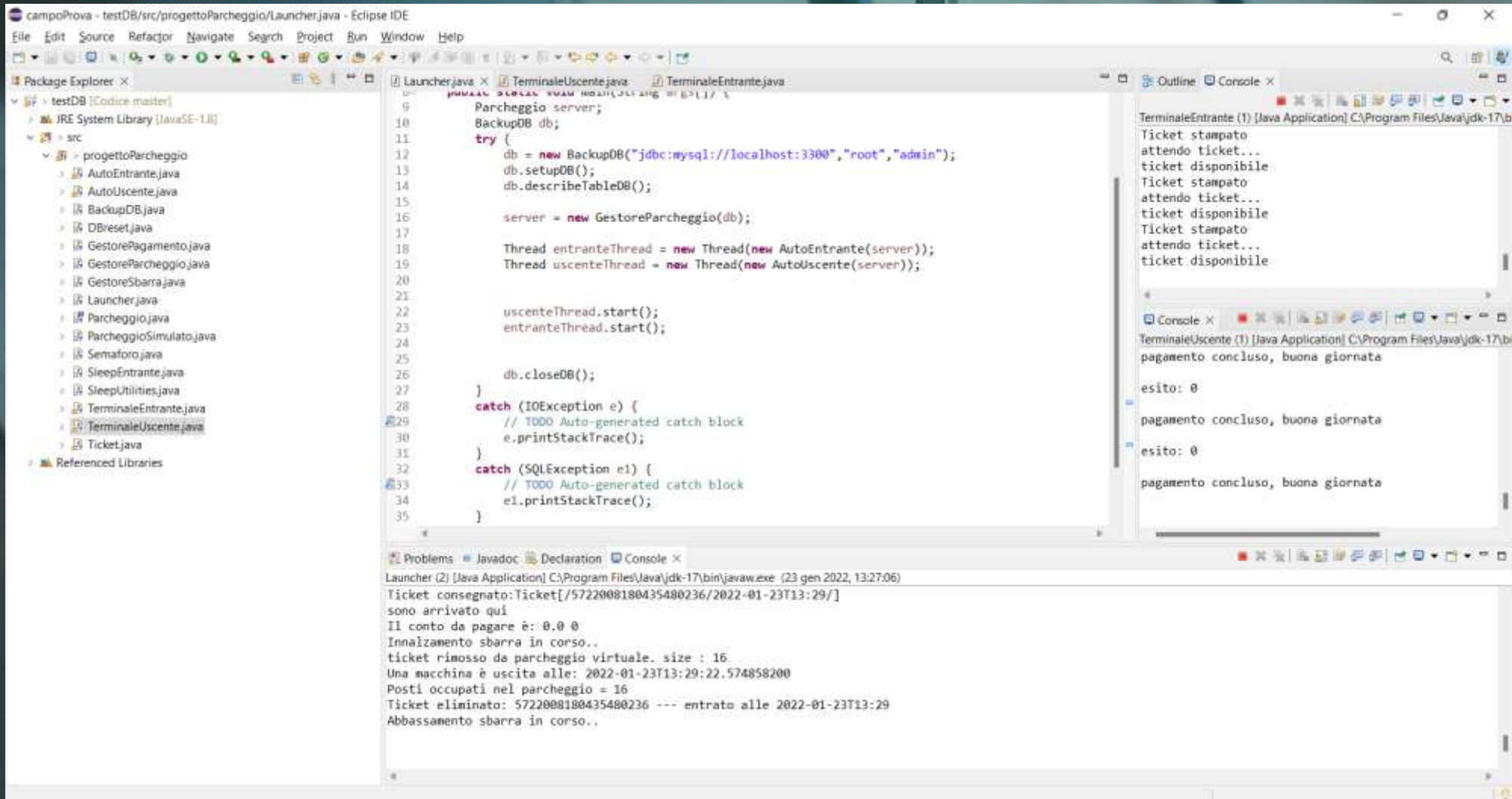
CAPITOLO 10: Software Testing

Nell'immagine precedente è stato testato il prototipo `v0.4.0_testingTerminaleUscita_v2.0_reteScarica`.

In questo caso di testing abbiamo verificato il corretto funzionamento dello scambio di messaggi tra server e terminale d'uscita, cortocircuitando il terminale d'entrata.

Lo scopo del testing era di verificare il riconoscimento dei ticket e la corretta cancellazione di quest'ultimi dal database in caso di utente che esce dal parcheggio.

CAPITOLO 10: Software Testing



The screenshot displays the Eclipse IDE interface for a Java project named 'campoProva'. The package explorer on the left shows the project structure, including a 'src' directory with a 'progettoParcheggio' package. The main editor shows the 'Launcher.java' file, which contains the following code:

```
9 Parcheggio server;  
10 BackupDB db;  
11 try {  
12     db = new BackupDB("jdbc:mysql://localhost:3300","root","admin");  
13     db.setupDB();  
14     db.describeTableDB();  
15  
16     server = new GestoreParcheggio(db);  
17  
18     Thread entranteThread = new Thread(new AutoEntrante(server));  
19     Thread uscenteThread = new Thread(new AutoUscente(server));  
20  
21  
22     uscenteThread.start();  
23     entranteThread.start();  
24  
25  
26     db.closeDB();  
27 }  
28 catch (IOException e) {  
29     // TODO Auto-generated catch block  
30     e.printStackTrace();  
31 }  
32 catch (SQLException e1) {  
33     // TODO Auto-generated catch block  
34     e1.printStackTrace();  
35 }
```

The console output shows the application's execution, including ticket status and payment confirmation:

```
TerminaleEntrante (1) [Java Application] C:\Program Files\Java\jdk-17\bin\java.exe (23 gen 2022, 13:27:06)  
Ticket stampato  
attendo ticket...  
ticket disponibile  
Ticket stampato  
attendo ticket...  
ticket disponibile  
Ticket stampato  
attendo ticket...  
ticket disponibile  
  
TerminaleUscente (1) [Java Application] C:\Program Files\Java\jdk-17\bin\java.exe (23 gen 2022, 13:27:06)  
pagamento concluso, buona giornata  
  
esito: 0  
pagamento concluso, buona giornata  
  
esito: 0  
pagamento concluso, buona giornata
```

The bottom of the console shows the application's output, including ticket status and payment confirmation:

```
Launcher (2) [Java Application] C:\Program Files\Java\jdk-17\bin\java.exe (23 gen 2022, 13:27:06)  
Ticket consegnato: Ticket[/5722008180435480236/2022-01-23T13:29/]  
sono arrivato qui  
Il conto da pagare è: 0.0 0  
Innalzamento sbarra in corso..  
ticket rimosso da parcheggio virtuale. size : 16  
Una macchina è uscita alle: 2022-01-23T13:29:22.574858200  
Posti occupati nel parcheggio = 16  
Ticket eliminato: 5722008180435480236 --- entrato alle 2022-01-23T13:29  
Abbassamento sbarra in corso..
```


CAPITOLO 10: Software Testing

Nell'immagine precedente è stato testato il prototipo v0.3.1_prototipoDB_v2.0.

In questo caso di testing abbiamo verificato il corretto funzionamento del database in un contesto concorrentiale e inter-procedurale.

La concorrenzialità è risultata centrale in quanto si voleva valutare eventuali starvation e deadlock all'interno dell'esecuzione del software.



CAPITOLO 11: Software Maintenance

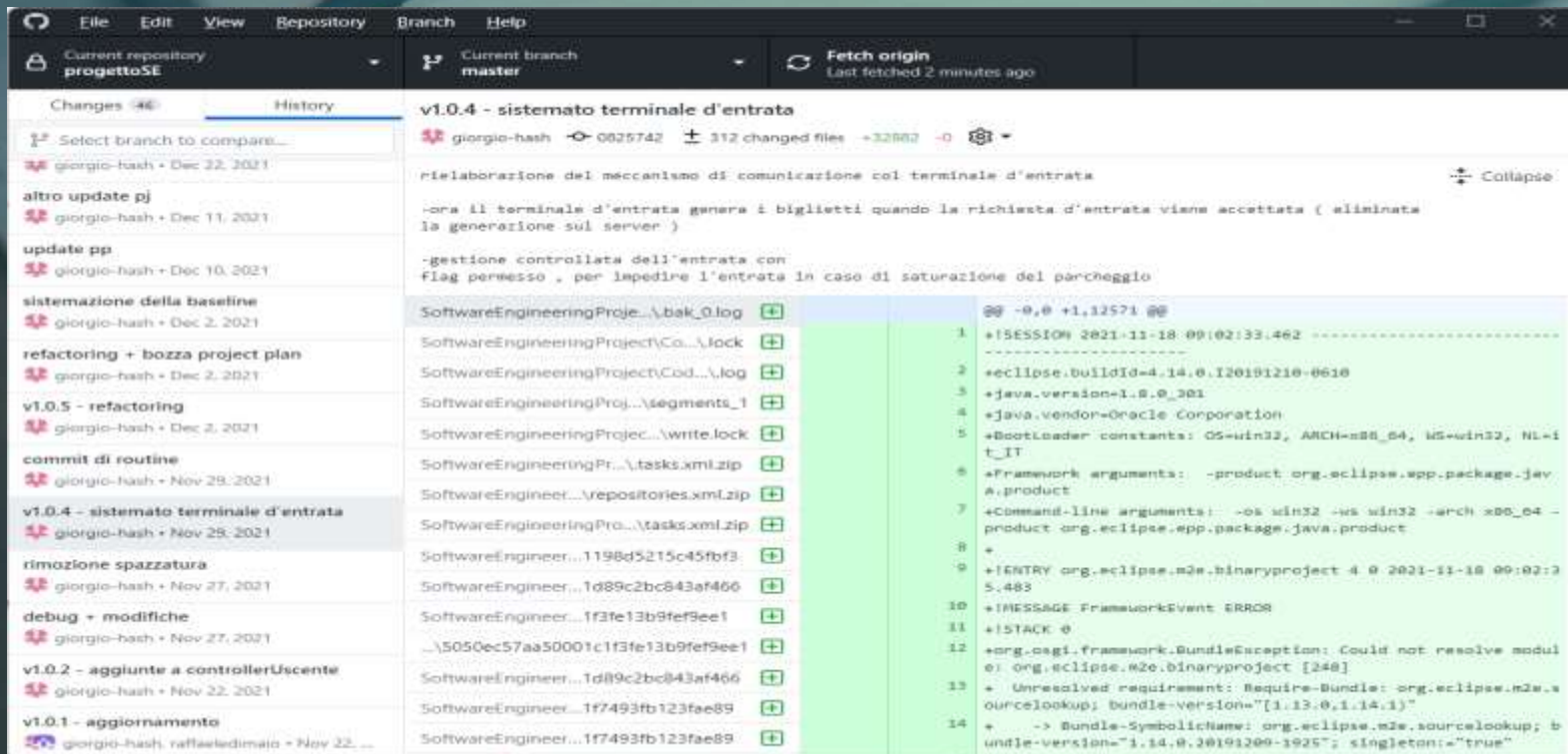
Nel corso dello sviluppo del software sono state effettuate una serie di attività di refactoring.

In particolare i due metodi utilizzati sono stati:

- Manutenzione correttiva;
- Manutenzione preventiva.

CAPITOLO 11: Software Maintenance

Manutenzione correttiva:



The screenshot displays the Eclipse IDE interface with a dark theme. The top menu bar includes File, Edit, View, Repository, Branch, and Help. Below the menu, the 'Current repository' is set to 'progettoSE' and the 'Current branch' is 'master'. A 'Fetch origin' button indicates the last fetch was 2 minutes ago.

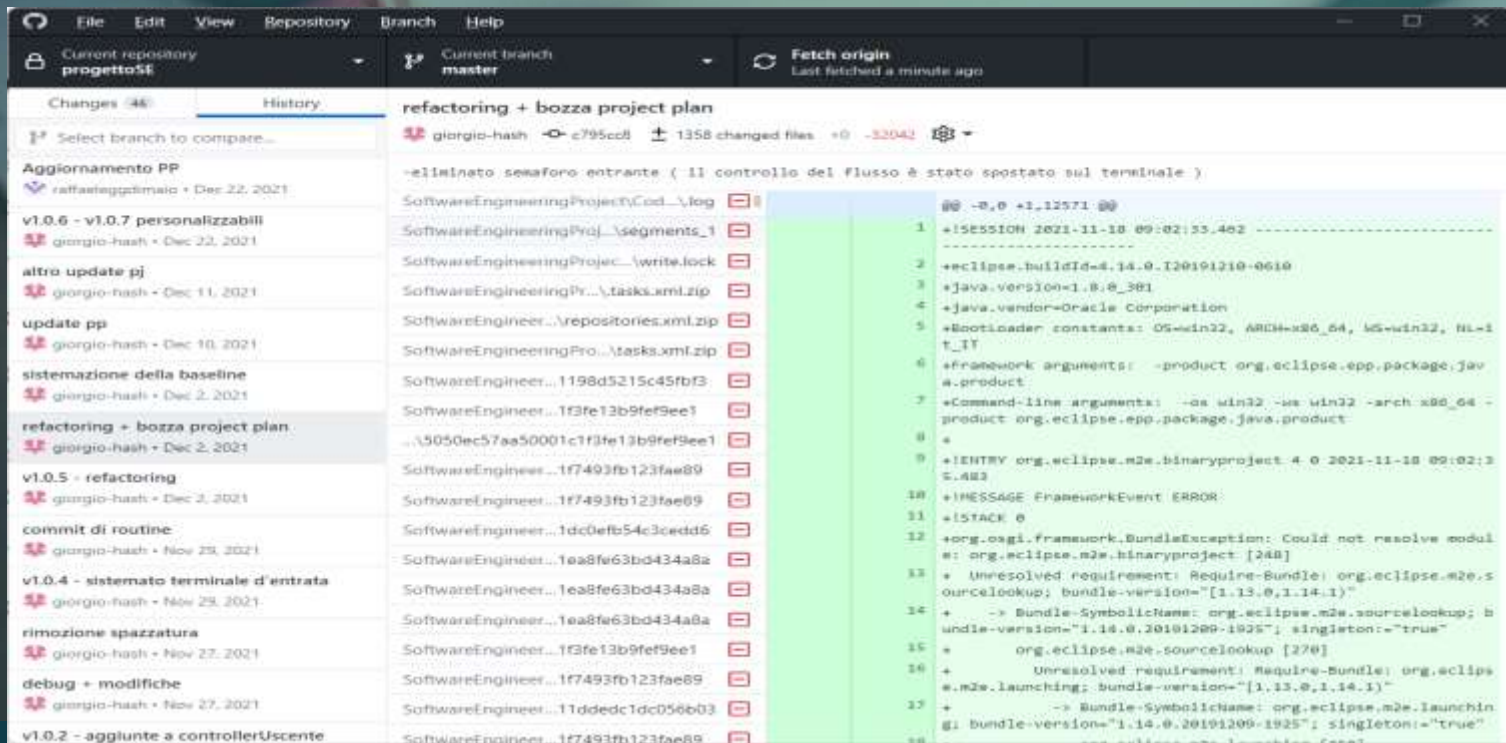
The left sidebar shows the 'Changes' view with a list of commits. The commit 'v1.0.4 - sistemato terminale d'entrata' by 'giorgio-hash' on Nov 28, 2021, is selected. The main area shows the diff for this commit, highlighting changes in the 'SoftwareEngineeringProject\...\bak_0.log' file. The diff shows a series of log entries, including session information, build details, and a stack trace indicating a 'BundleException: Could not resolve module org.eclipse.m2e.binaryproject [248]'.

The commit history on the left includes the following entries:

- Select branch to compare...
- giorgio-hash • Dec 22, 2021
- altro update pj
- giorgio-hash • Dec 11, 2021
- update pp
- giorgio-hash • Dec 10, 2021
- sistemazione della baseline
- giorgio-hash • Dec 2, 2021
- refactoring + bozza project plan
- giorgio-hash • Dec 2, 2021
- v1.0.5 - refactoring
- giorgio-hash • Dec 2, 2021
- commit di routine
- giorgio-hash • Nov 29, 2021
- v1.0.4 - sistemato terminale d'entrata
- giorgio-hash • Nov 28, 2021
- rimozione spazzatura
- giorgio-hash • Nov 27, 2021
- debug + modifiche
- giorgio-hash • Nov 27, 2021
- v1.0.2 - aggiunte a controllerUscente
- giorgio-hash • Nov 22, 2021
- v1.0.1 - aggiornamento
- giorgio-hash, raffaeledimato • Nov 22, 2021

CAPITOLO 11: Software Maintenance

Manutenzione preventiva:



The screenshot displays the Eclipse IDE interface. The top menu bar includes File, Edit, View, Repository, Branch, and Help. Below the menu, the 'Current repository' is set to 'progetto54', and the 'Current branch' is 'master'. The 'Fetch origin' button indicates the last fetch was a minute ago. The main workspace is divided into three panes. The left pane shows the 'Changes' view with a list of commits, including 'refactoring + bozza project plan' by 'giorgio-hash' on Dec 2, 2021. The middle pane shows the 'History' view for the selected commit, listing files such as 'SoftwareEngineeringProjectCod...log' and 'SoftwareEngineeringProj...segments_1'. The right pane shows the diff view for the selected commit, displaying changes to the 'SoftwareEngineeringProj...write.lock' file. The diff shows a series of changes, including the removal of a semaphore and the addition of a new project plan.

```
refactoring + bozza project plan
giorgio-hash c795cc8 1358 changed files +0 -32042

-eliminato semaforo entrante ( il controllo del flusso è stato spostato sul terminale )
SoftwareEngineeringProjectCod...log
SoftwareEngineeringProj...segments_1
SoftwareEngineeringProjec...write.lock
SoftwareEngineeringPr...tasks.xml.zip
SoftwareEngineer...repositories.xml.zip
SoftwareEngineeringPro...tasks.xml.zip
SoftwareEngineer...1198d5215c45fbf3
SoftwareEngineer...1f3fe13b9fef9ee1
...5050ec57aa50001c1f3fe13b9fef9ee1
SoftwareEngineer...1f7493fb123fae89
SoftwareEngineer...1f7493fb123fae89
SoftwareEngineer...1dc0efb54c3cedd5
SoftwareEngineer...1ea8fe63bd434a8a
SoftwareEngineer...1ea8fe63bd434a8a
SoftwareEngineer...1ea8fe63bd434a8a
SoftwareEngineer...1f3fe13b9fef9ee1
SoftwareEngineer...1f7493fb123fae89
SoftwareEngineer...11ddcd1dc056b03
SoftwareEngineer...1f7493fb123fae89
```