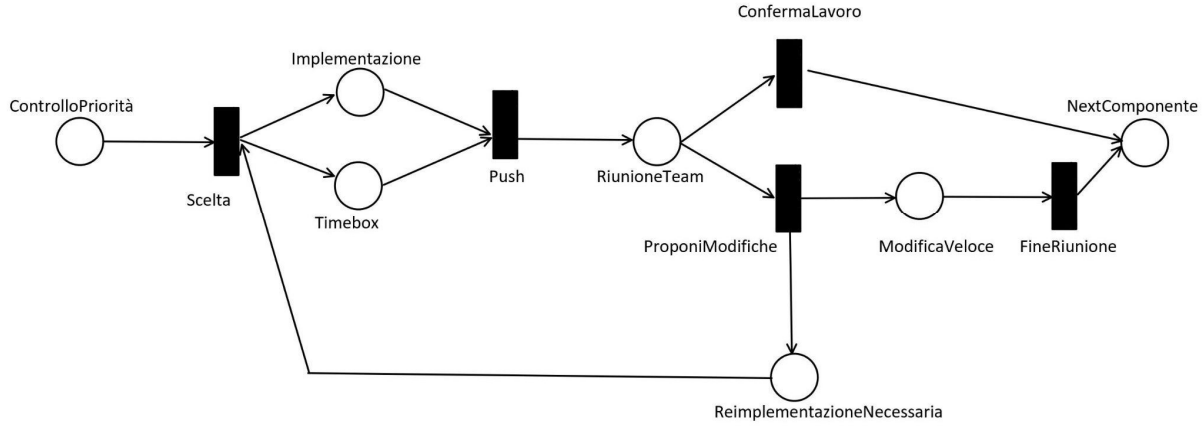
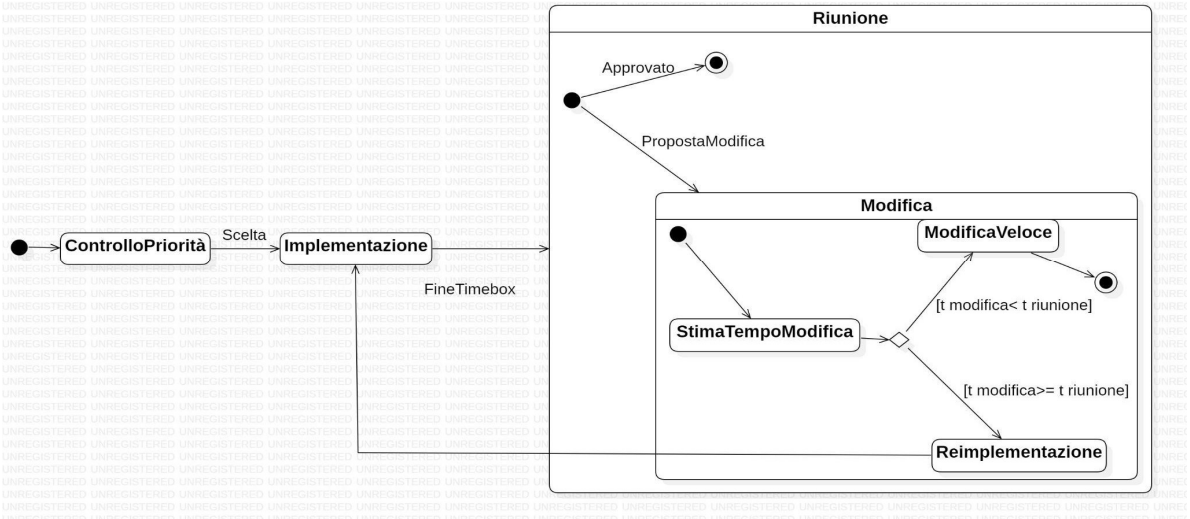
Documentazione

CAP 3

Per questo progetto abbiamo scelto di seguire una modalità di sviluppo del software di tipo RAD (Rapid Application Development) perché adatto a progetti seguiti da poche persone in un periodo di tempo limitato. Tramite la tecnica triage assegniamo le priorità ai requisiti e stimiamo degli slot temporali (time box) di sei/sette giorni in cui lavorare individualmente su determinati componenti. Al termine della time box si tiene una riunione in cui tutti i membri discutono quanto ottenuto. Le time boxes sono state successivamente ridotte a tre giorni poiché con l'avvicinarsi della data di consegna ci siamo resi conto che non saremmo riusciti a completare per tempo il progetto senza questa modifica.

Nella rete di Petri e nel diagramma degli stati UML di seguito, descriviamo il procedimento per confermare l'implementazione di un nuovo componente.
Scegliamo inizialmente il componente controllando la lista delle priorità. Iniziamo la fase di implementazione la cui durata è di una time box prefissata. Al termine della time box il lavoro svolto viene condiviso col resto del gruppo che durante una riunione sceglie se confermarlo, apportare delle modifiche veloci (da intendersi come risolvibili durante la riunione stessa) o se è necessario re-implementare in un secondo momento il componente in questione.





È stata costruita un'architettura guidata dal modello, in cui l'attenzione non è più solo sul codice, ma principalmente sullo sviluppo di modelli. Grazie a questo software abbiamo potuto tradurre automaticamente in codice il modello creato (generiamo il codice a partire dal class diagram). Procedendo in questo modo viene resa più leggera la fase di scrittura del codice e più sostenibile un mantenimento futuro dell'applicazione, svolto quindi a livello del modello.

CAP 4

Poiché lo sviluppo è diviso tra i membri del team durante le time boxes, è importante avere un modo per restare sempre aggiornati sullo stato del progetto e comunicare problemi e cambiamenti. A tal fine utilizziamo come Software Control Manager: GitHub.

GitHub è utile nelle fasi in cui, per ottimizzare i tempi, si vuole portare avanti parti diverse del progetto contemporaneamente. Ad esempio, durante la fase di modellizzazione con StarUML, ognuno dei membri del team ha creato un diagramma differente; oppure, durante l'implementazione del codice di classi diverse.
Al completamento di ogni parte svolta autonomamente, sempre rispettando le time boxes prefissate, viene effettuata una richiesta di merge con il main branch. I cambiamenti che si vogliono apportare vengono controllati durante una riunione e una volta chiariti tutti i dubbi l'operazione di merge può essere effettuata.
Oltre ai branch sono stati assegnati issue a vicenda tra i membri del gruppo per tenere traccia delle modifiche da apportare o dei progressi del progetto. L'assegnee dell'issue può quindi concentrarsi sul problema da risolvere e aggiornare il team sul raggiungimento di tale obbiettivo, lavorando su un branch dedicato.

CAP 5

Per lo sviluppo del nostro software adottiamo il modello di organizzazione del team: SWAT (Skilled With Advanced Tools).
Questo modello è adatto a piccoli gruppi come il nostro in modo che la comunicazione sia diretta e non rallentata da formalismi. Inoltre si adatta bene a modelli di sviluppo iterativi o evoluzionari come il RAD perchè avendo tre membri indipendenti e allo stesso livello (dal punto di vista delle competenze) ognuno può occuparsi in autonomia dello sviluppo di un componente durante lo slot temporale assegnatoli.
In questo modello ci si aggiorna costantemente lavorando a stretto contatto, occupando la stessa stanza (anche virtuale). Usiamo strumenti di groupware come Microsoft Teams per riunioni e sessioni di brainstorming e GitHub per la condivisione degli elaborati.
Non c'è nessun leader e nessuna organizzazione gerarchica in quanto il team di dimensioni ridotte, affiatato e con competenze omogenee.
Un'organizzazione di tipo SWAT richiede molta motivazione nel raggiungimento dell'obiettivo finale, ed estrema coesione. A tal proposito il nostro gruppo, che addotta il nome OSLO, è nato circa due anni fa per collaborare ai progetti di gruppo universitari.

CAP 6

La qualità del software è una proprietà intrinseca del nostro progetto, pensata sin dalle prime fasi. Procediamo in modo che il software abbia:

- Qualità del prodotto:
    - Durante il funzionamento del software vogliamo garantire correttezza, prevenire un uso improprio del software stesso, affidabilità e un utilizzo intuitivo.
    - Diamo importanza alla manutenibilità del nostro sistema: è mantenibile, flessibile e testabile.
    - Il software è portabile su tutti i computer che hanno una JVM.
    - Interfaccia: deve essere di facile utilizzo per i clienti dell'aeroporto, ma al tempo stesso permettere ai dipendenti tutte le operazioni per la gestione del sistema.
    - Privacy: dato che il software immagazzina i dati sensibili dei clienti, bisogna permettere solo a chi è autorizzato di accedervi.
    - Velocità: il software deve essere abbastanza reattivo in modo da fornire un'esperienza piacevole ai clienti.
- Qualità del processo:
    - Documentazione: le fasi di sviluppo del progetto sono documentate prima e durante lo svolgimento dello stesso seguendo lo standard IEEE 9001

CAP 9

La specifica dei requisiti si divide in quattro fasi:

- *Elicitation*, l'estrazione dei requisiti avviene usando la tecnica basata sullo scenario, pensando quindi a come l'utente userà il software. Una volta estrapolati i requisiti saranno organizzati con il MoSCoW.

Must haves	- assegnazione gate aereo, - gestione carburante, - assegnazione piloti a ogni volo, - gestire prenotazioni per tutti i voli, - protezione dati clienti.
Should haves	- permettere ricerca voli.
Could haves	- interfaccia per ricerca voli intuitiva e facile da utilizzare.
Won't haves	- interfaccia grafica, - DB per storico dei voli.

- *Specification*

Secondo lo standard IEEE 830:

1. Introduzione:
   In questo documento si specificano i requisiti e le funzionalità richieste per un software per la gestione di aeroporti. Il software vuole sviluppare le funzioni descritte nel Project Plan che costituisce un punto di partenza per la fase di design.

2. Descrizione generale:
   Non sono presenti né un database né un'interfaccia grafica dedicata, non attualmente richiesti.
   Il sistema presenta due funzionalità distinte, una per l'utente e una per i dipendenti dell'aeroporto. Le funzionalità sono descritte nel Project Plan e nel terzo paragrafo di questo documento.
3. Specifica dei requisiti*:*
   a) Requisiti funzionali: inizialmente l'utente dovrà specificare se è cliente dell'aeroporto o un dipendente, nel secondo caso è richiesto un codice d'accesso (vedi Capitolo 13). Di seguito i vari tipi di accesso:

      Accesso cliente: verrà mostrata un'interfaccia per la ricerca e prenotazione di voli, e la gestione delle proprie prenotazioni.

      Acceso dipendente manutenzione: l'interfaccia presenta la funzionalità per aggiungere carburante agli aerei mostrando in una lista quali necessitano di fare rifornimento.

      Accesso dipendente amministrazione: i dipendenti che hanno accesso a questa sezione possono gestire i voli.

      Accesso dipendente torre di controllo: segnala atterraggio di nuovi aerei assegnando loro un gate nel caso ce ne sia uno disponibile, altrimenti impedisce l'atterraggio. Permette il decollo di un aereo con conseguente eliminazione del volo, delle prenotazioni associate, e liberazione di un nuovo gate.

      In futuro si potrebbe implementare un'interfaccia grafica e un database per il salvataggio più intuitivo delle liste di voli e prenotazioni.

   b) Requisiti non funzionali: il sistema deve gestire fino a sette aerei contemporaneamente in transito nell'aeroporto. Proteggiamo i dati sensibili degli utenti tramite l'accesso con password. Il software può rimanere attivo anche per lunghi periodi di tempo.

- *Validation*

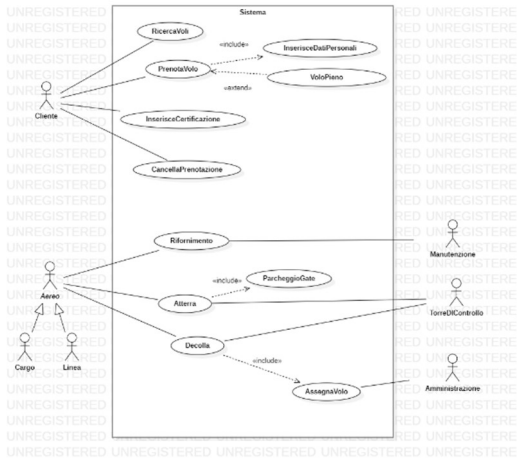
Le fasi di verifica e validazione sono svolte in ogni fase del processo di sviluppo del sistema.
Leggiamo più volte i requisiti controllando che rispettino le varie proprietà di correttezza, completezza, consistenza, accuratezza, leggibilità e quindi siano adatti allo sviluppo del progetto.

- *Negotiation*

Il proprietario dell'aeroporto che ha richiesto il software accetta il fatto che in questo progetto non saranno implementati né l'interfaccia grafica né un database contenente lo storico dei voli, i quali potranno essere aggiunti in un progetto futuro. Vengono quindi accettati i requisiti specificati al punto precedente.

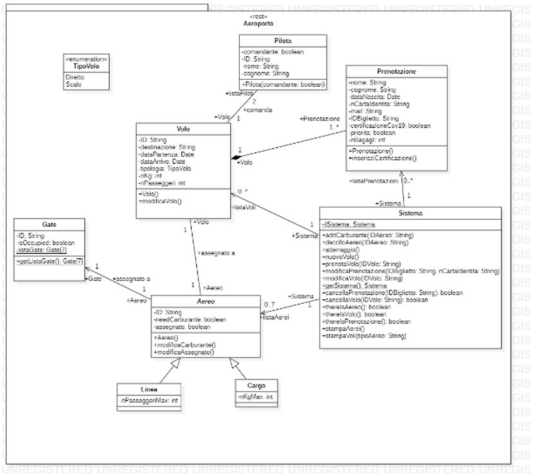
CAP 10

Use Case Diagram:



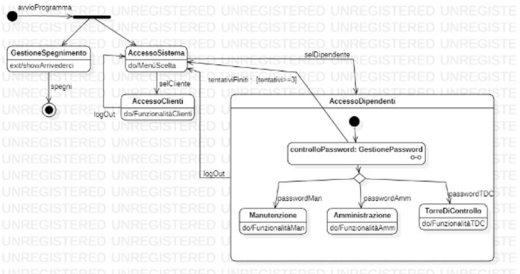
Viene illustrato come i vari attori interagiscono con il sistema.
Nel nostro caso il Cliente può accedere a una serie di funzionalità: cercare e prenotare voli, inserire una certificazione Cov-19 o cancellare una prenotazione esistente.
I tre tipi di dipendenti interagiscono con l'aereo, in particolare l'addetto alla manutenzione può fare rifornimento, l'addetto all'amministrazione assegna l'aereo ad un volo, e l'operatore della torre di controllo può far atterrare e decollare aerei.
Più specificatamente le fasi di atterraggio e di decollo incorporano al loro interno, tramite la freccia 'include', il comportamento rispettivamente di parcheggio in un gate e di assegnazione volo. Al contrario durante la fase di prenotazione del volo del cliente, quest'ultima può utilizzare il comportamento del volo pieno, nel caso non ci siano più posti disponibili. Questa relazione è espressa tramite la freccia 'extend'.

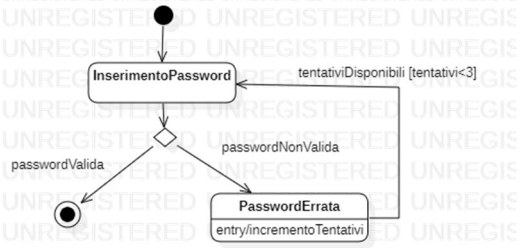
Class Diagram:



È un diagramma di struttura che modella delle classi e le loro interazioni per mostrare il codice e il suo funzionamento.
La classe Prenotazione è collegata con una composizione forte alla classe Volo, questo significa che non esistono prenotazioni senza un volo di appartenenza, e se il volo viene cancellato, anche le prenotazioni dello stesso verranno eliminate.
La classe Sistema raggruppa le funzionalità delle altre classi, e gestisce il salvataggio dei dati inseriti tramite delle liste (listaPrenotazioni, listaVoli, listaAerei).
La classe Aereo è una classe astratta che generalizza le sottoclassi Linea e Cargo; infatti, queste due differiscono soltanto per gli attributi nPasseggeriMax e nKgMax.

State Machine Diagram:

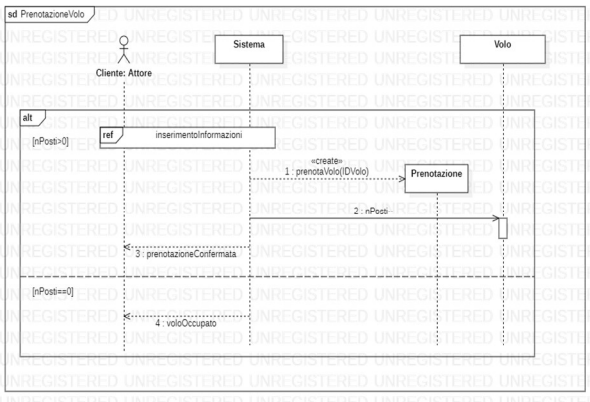


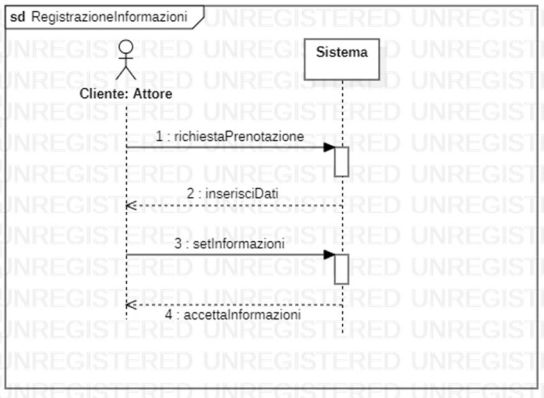


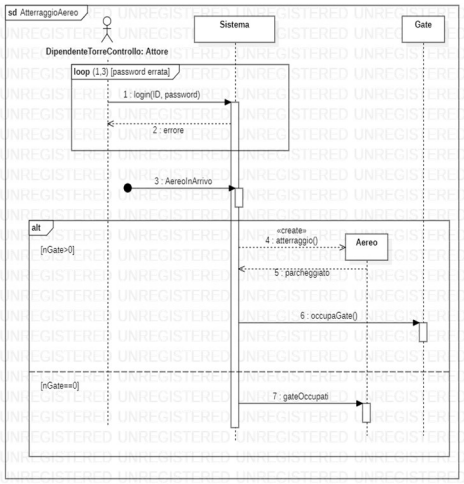
Modella il funzionamento del software tramite macchina a stati finiti.
Vengono eseguiti contemporaneamente la gestione dell'utilizzo del software (AccessoSistema) e il controllo spegnimento, in modo che sia possibile spegnere il programma in un qualsiasi momento durante l'esecuzione.
Dallo stato AccessoSistema si può decidere se accedere come cliente o dipendente, in entrambi i casi si entra nello stato che fornisce le funzionalità richieste.
Nel caso si voglia accedere come dipendente sarà richiesta una password. Il controllo della password è schematizzato in un secondo State Machine Diagram, dove si mostra come sia possibile inserire password errate massimo tre volte prima di essere rimandati allo stato AccessoSistema.

Sequence Diagram:
Questo tipo di diagramma modella le interazioni tra gli oggetti nel sistema.
In particolare abbiamo deciso di modellare tre casi d'uso: la prenotazione del volo, l'atterraggio e il decollo di un aereo.

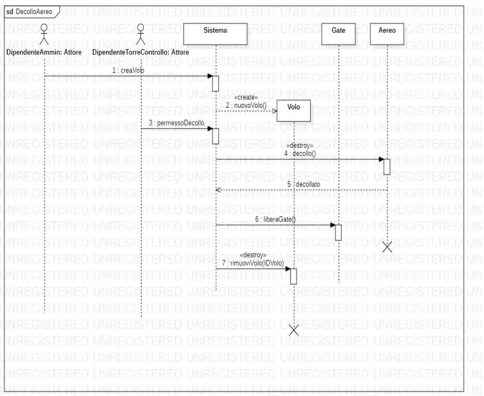
La prenotazione del volo è stata implementata sfruttando un ulteriore Sequence Diagram denominato RegistraInformazioni per rappresentare l'inserimento delle informazioni personali da parte del cliente nel sistema. Questo Sequence Diagram supplementare è stato poi chiamato tramite un Interaction Reference nel Sequence Diagram della prenotazione del volo.





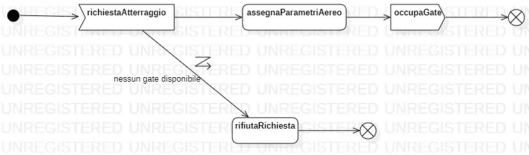


Nella fase di atterraggio di un nuovo aereo il dipendente della torre di controllo deve innanzitutto accedere al sistema, a tal proposito è stato inserito un frammento combinato “loop” per la ripetizione fino a tre volte delle credenziali in caso di inserimento di password errata. L’arrivo di un nuovo aereo è stato rappresentato con un messaggio di cui non è importante il mittente. Quando il nuovo aereo segnala la sua presenza il dipendente deve assegnargli l’ID e il gate in cui sostare. Queste operazioni sono state inserite in un frammento combinato “alt” perché deve essere rispettata la sequenza con la quale son indicati i messaggi.

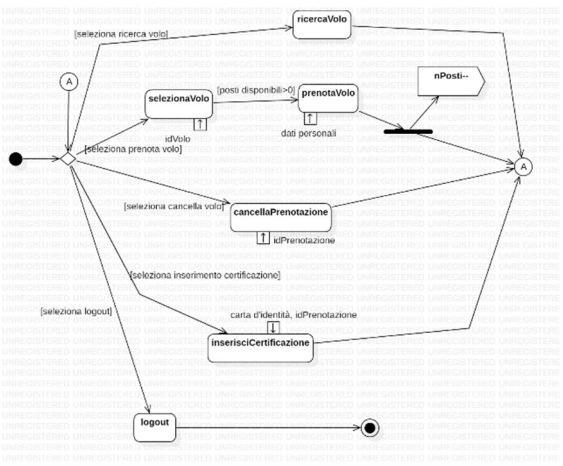


Infine nella rappresentazione delle interazioni della fase di decollo, gli attori sono il dipendente dell’amministrazione e della torre di controllo.
È il dipendente dell’amministrazione che accede al sistema per creare un nuovo volo con le informazioni generali (messaggio di creazione di una nuova lifeline). Dopo che la torre di controllo fornisce il permesso di decollo il dipendente di amministrazione, con un messaggio di distruzione elimina prima l’aereo, liberando di conseguenza un gate, e poi elimina il volo tutte le prenotazioni ad esso associate.

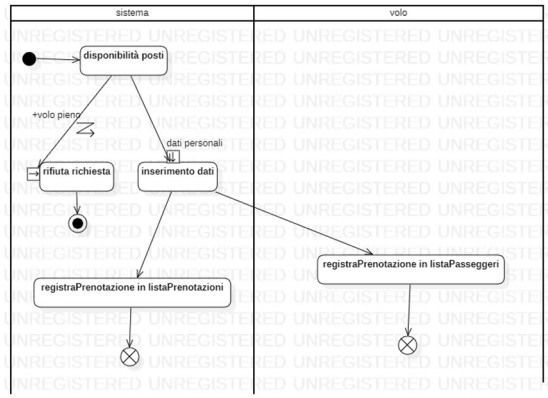
Activity Diagram:
Tramite l’Activity Diagram si può spiegare il flusso di controllo e di dati tra i vari step richiesti per implementare una specifica attività, come parti del codice o specifici metodi.



Nel primo diagramma si modella il metodo atterraggio(): questo metodo inizia quando da input arriva il segnale di richiesta di atterraggio, a questo punto il flusso di esecuzione si divide in due casi:
Nel primo caso ci sono ancora gate disponibili, quindi vengono assegnati i parametri all'aereo e si manda un segnale per occupare un gate. Altrimenti se non ci sono gate disponibili, si rifiuta l'aereo e si termina il flusso.



Nel secondo diagramma si spiega la gestione delle funzionalità offerte al cliente che può decidere tra quattro opzioni: cercare un volo, prenotare un volo, cancellare una prenotazione, inserire la certificazione Cov-19 o fare il logout. Tutte le varie diramazioni, tranne il logout, terminano con il ritorno al nodo di scelta iniziale.
Nel primo caso si sceglie di cercare un volo inserendo una destinazione, il sistema stamperà tutti i voli che portano nel luogo desiderato.
Nel secondo caso, per fare una prenotazione, viene richiesto l'ID di un volo valido, a questo punto verrà confermata la prenotazione e viene rimosso un posto da quelli disponibili nell'aereo, se invece l'aereo è già pieno verrà rifiutata la prenotazione.
Il terzo caso consiste nel cancellare una prenotazione, per farlo verrà richiesto l'ID del biglietto, se corretto la prenotazione sarà rimossa dal sistema.
Come quarto caso si può scegliere di inserire la certificazione, viene richiesta la carta d'identità e l'ID del biglietto, se entrambi gli input corrispondono verrà confermato l'inserimento della certificazione.
Nell' ultimo caso si farà il logout, e quindi si terminerà l'attività di funzionalità del cliente.

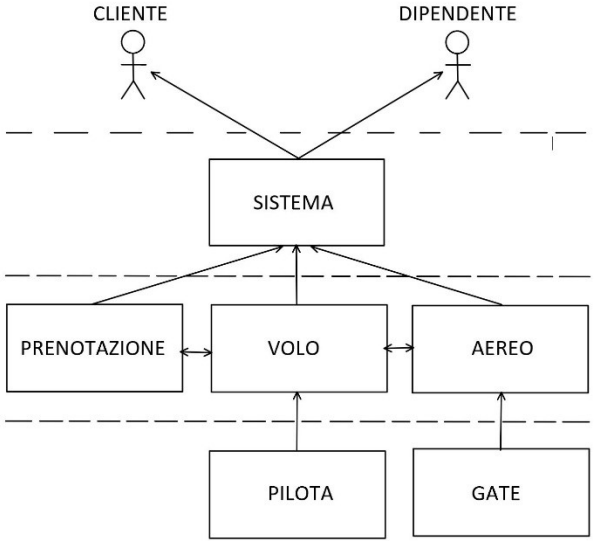


Il terzo Activity diagram illustra nel dettaglio il metodo nuovaPrenotazione. Se c'è disponibilità di posti per il volo, i dati della nuova prenotazione vengono inseriti nel sistema nella lista di tutte le prenotazioni e nella lista dei passeggeri del singolo volo, altrimenti si rifiuta la richiesta di prenotazione.

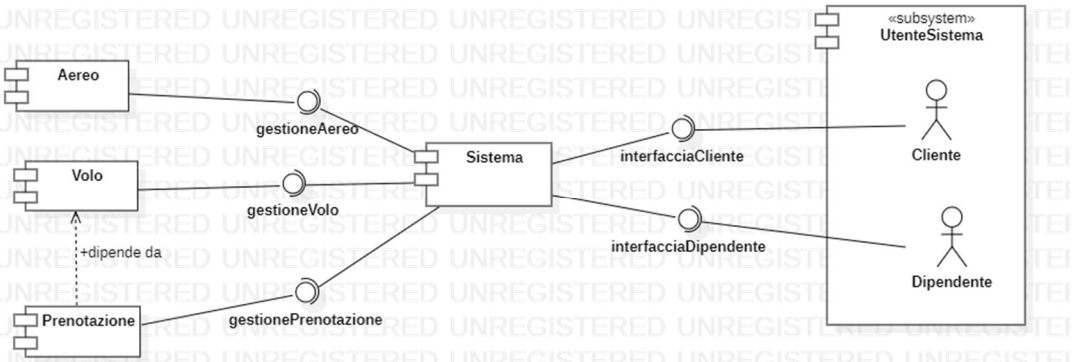
CAP 11

Lo standard IEEE 1471 fornisce una struttura generale per la rappresentazione dell'architettura del software. Gli elementi principali di questo standard sono le persone interessate al sistema, view e viewpoint. Questi ultimi a loro volta sono suddivisi in tre classi (Bass *et al*., 2003): Module viewpoint, Component-and-connector viewpoint, Allocation viewpoint.
Come Module viewpoint, per rappresentare una vista statica del nostro sistema, illustreremo una vista a quattro strati (Layered). Si ricorre a questo tipo di viewpoint quando si vuole rappresentare il sistema su una serie di livelli, dove gli elementi del livello N possono usare gli elementi appartenenti ai livelli inferiori ad N.
Il software è pensato in modo che i clienti e i dipendenti utilizzino lo stesso componente Sistema per accedere alle varie funzionalità.

Il componente a livello più alto è la classe sistema, ovvero un'entità che racchiude funzionalità e metodi degli elementi Volo, Aereo e Prenotazione; quindi, fornisce questi servizi ai clienti e ai dipendenti.
Nel livello centrale si trovano Volo, Aereo e Prenotazione, che forniscono i loro servizi di creazione, cancellazione e modifica degli stessi, al componente Sistema.
Allo stesso modo nell'ultimo livello troviamo: Pilota che fornisce a Volo la possibilità di assegnare nuovi piloti ad un volo specifico e Gate che mostra ad Aereo la disponibilità dei vari gate nell'aeroporto.



Oltre al Module viewpoint costruiamo anche il Component-and-connector viewpoint nel quale si rappresenta la vista dinamica del sistema e, in particolare, come interagiscono tra loro i vari componenti e quali interfacce vengono utilizzate. Tra i componenti si prendono in considerazione le classi Aereo, Volo, Prenotazione. Queste tre interagiscono con la classe sistema fornendo a quest'ultima l'insieme di metodi e informazioni generali che ognuna di esse contiene. La classe sistema ha un ruolo fondamentale perché fa da tramite tra le classi che rappresentano gli oggetti veri e propri del sistema, e il sottosistema dell'utente che accede o come dipendente o come cliente. A seconda della tipologia di utente il sistema rappresenterà un'interfaccia diversa permettendo l'accesso ad alcuni metodi e negandolo ad altri se non si è in possesso delle credenziali adatte.



CAP 12

Poiché per il progetto abbiamo scelto di utilizzare UML e Java, entrambi linguaggi object-oriented, nella fase di design utilizziamo un metodo OOAD. In particolare scegliamo di usare il metodo Booch, seguendo in modo iterativo i seguenti passaggi:

- Identificazione delle classi e oggetti;
- Identificazione della loro semantica e comportamento;
- Identificazione delle relazioni tra le classi;
- Identificazione delle interfacce e implementazioni di classi e oggetti.

Seguendo lo standard IEEE 1016 abbiamo documentato il processo di design come di seguito:

Identification	Sistema
Type	Classe
Purpose & Function	Deve fornire tutti i metodi per: Gestione carburante (metodo addCarburante()) Gestisce passeggeri per tutti i voli (lista Prenotazioni) Protezione dati clienti Permette ricerca voli (metodo ricercaVoli())
Dependencies	Usa le classi Volo, Aereo, Prenotazione, viene usato solo dalla classe Main
Interfaces	Sistema utilizza i metodi delle classi prenotazione, Volo e Aereo. In particolare utilizza i loro costruttori, Volo.modificaVolo(), Prenotazione.inserisciCertificazione(), Aereo.setNeedCarburante()
Processing	L'eccezione gateNonDisponibileException viene gestita nel metodo atterraggio() catturandola con un try-catch e è generata dal tentativo di creazione di un nuovo aereo nel caso in cui non ci siano gate disponibili
Data	Ci sono tre liste: una per i voli, una per gli aerei e una per le prenotazioni. Queste liste vengono gestite internamente a sistema.

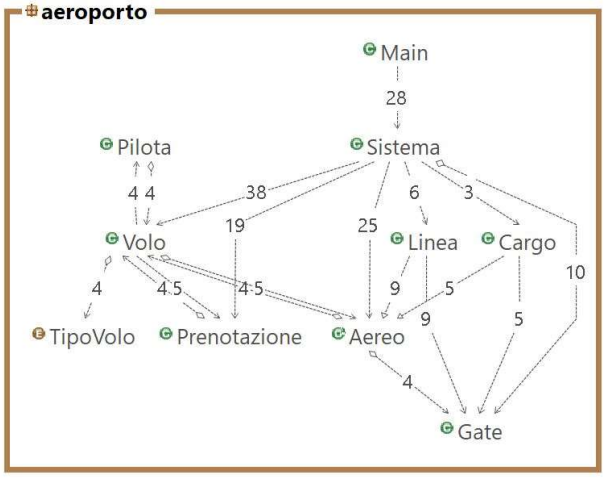
Identification	Piloti	Gate
Type	Classe	Classe
Purpose & Functions	Fornisce metodi che consentiranno l'assegnazione del personale a ogni volo	Fornisce metodi che consentiranno l'assegnazione gate ad Aereo
Dependencies	Usato da Volo	Usato da Aereo
Interfaces	Fornisce il costruttore e metodo isComandante()	Fornisce il costruttore e metodi setIsOccupied() e isOccupied()
Data		Contiene una lista di gate che viene utilizzata dalla classe Aereo per verificare la disponibilità di gate

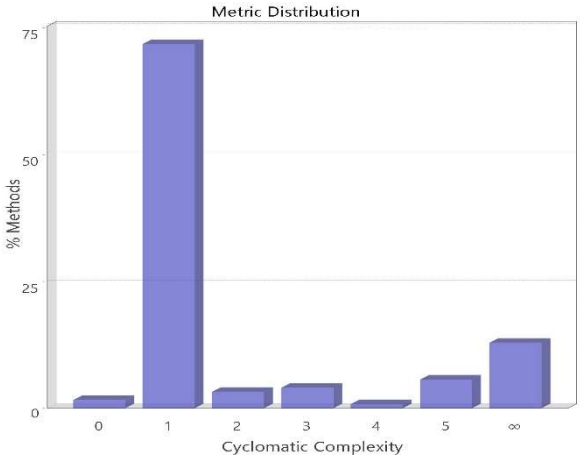
Identification	Volo	Aereo
Type	Classe	Classe astratta
Purpose & Functions	Consente l'assegnazione personale a ogni volo e di Gestire passeggeri sul volo	Assegnazione gate ad aereo Gestione carburante
Dependencies	Viene usata da sistema, usa pilota, prenotazioni	Viene usata da sistema, usa classe Gate
Interfaces	Mette a disposizione della classe Sistema il costruttore, i metodi getNPasseggeri() modificaVolo(), e i metodi per gestire la lista delle prenotazioni.	È una classe astratta, le sottoclassi Linea e Cargo mettono a disposizione di sistema i loro costruttori, i metodi

		modificaCarburante() e getNPasseggeri()
Processing	Il costruttore di Volo lancia e gestisce l'eccezione DataException nel caso in cui la data di arrivo inserita sia prima di quella di partenza, oppure quest'ultima sia antecedente la data odierna.	I costruttori di Linea e Cargo lanciano l'eccezione gateNonDisponibileException che viene poi gestita dalla classe Sistema
Data	Contiene e gestisce due liste: una delle prenotazioni per lo specifico volo e una dei piloti.	

Identification	Prenotazione
Type	Classe
Purpose & Function	Deve fornire tutti i metodi per: Protezione e salvataggio dati clienti
Dependencies	Viene usata dalle classi Sistema e Volo, non usa alcuna classe
Interfaces	Mette a disposizione della classe Sistema il costruttore e il metodo inserisciCertificazione()
Processing	L'eccezione DataException viene lanciata dal costruttore in caso venga inserita una data di nascita successiva a quella odierna. Viene gestita all'interno del costruttore stesso.
Data	Gli attributi contengono le informazioni sensibili del cliente, vengono infatti gestiti come privati e non visibili a nessun altro utente.

Il design del progetto è descritto dal seguente diagramma generato tramite il tool stan4j, in cui come anticipato nel Capitolo 11, si nota che la classe Main è a un livello gerarchico superiore a quello della classe sistema che a sua volta è situata a un livello superiore rispetto alle classi volo aereo e prenotazione, a livello più basso rimane la classe gate che ha soltanto connessioni afferenti.
Il grado di accoppiamento è adeguato e segue il modello a strati descritto nel Capitolo 11.
Inoltre con il tool stan4j calcoliamo la complessità ciclomatica di McCabe e osserviamo che la maggior parte dei metodi hanno una bassa complessità.





Si ricorre all'utilizzo di due design pattern per la risoluzione di alcuni problemi tipici che abbiamo riscontrato durante lo sviluppo del progetto.

Pattern di design:

- Singleton: è fondamentale che la classe sistema venga istanziata una sola volta. Nessun'altra classe (nemmeno il Main) può creare sue istanze che tuttavia devono essere disponibili alle classi utilizzatrici. Nella classe Sistema è presente un attributo statico "ilSistema" che è il riferimento all'unica istanza. Il costruttore di questa classe è privato. Viene implementato il metodo pubblico getIlSistema() che, se possibile, ritorna la variabile di riferimento "ilSistema", in caso contrario crea un'istanza di Sistema salvandola nella variabile "ilSistema" e ritorna la variabile stessa.
- Delegation: è necessario che la classe Sistema utilizzi dei metodi di altre classi (es.: ilSistema.modificaVolo() usa Volo.setDestinazione()). Per renderlo possibile abbiamo collegato la classe "delegator" Sistema alle classi "delegate" (Volo, Prenotazione, Aereo) tramite un'associazione diretta. All'interno della classe Sistema esistono quindi metodi che chiamano al loro interno metodi delle classi "delegate".

CAP 13

Per testare manualmente il codice si possono seguire le seguenti istruzioni che utilizzano tutte le funzionalità implementate dal codice.

1) Avviando il codice viene mostrato il primo menù che permette di scegliere se accedere come clienti, come dipendenti o se spegnere il programma.
2) Se si entra nel menù clienti si potrà vedere che tutte le varie funzionalità restituiscono un output che indica la mancanza sia di voli che prenotazioni, che devono essere creati dal menù dipendenti. Effettuare il logout per tornare alla schermata iniziale.
3) Per entrare nel menù dipendenti verrà richiesta una password. Il software accetterà tre password: 'Amm' (accesso come amministratore), 'Man' (accesso come manutentore aerei), 'TDC' (Accesso come addetto torre di controllo).
4) Se si accede o tramite 'Man' o tramite 'Amm', si noterà che in entrambi è necessario avere almeno un aereo nell'aeroporto per eseguire le funzionalità.
5) Accedendo come 'TDC', si ha la possibilità di far atterrare un aereo seguendo le istruzioni stampate in output. Per far atterrare un aereo si deve specificare se sta arrivando un aereo di linea o cargo.
   Scegliere di far atterrare un aereo di linea (il caso aereo cargo è analogo). L'aereo viene inserito nel sistema dell'aeroporto ed assegnato ad un gate libero, a questo punto sarà richiesto il numero massimo di passeggeri che l'aereo può trasportare, si consiglia di scegliere un numero basso, in modo da controllare in modo semplice la funzionalità che impedisce di prenotare un biglietto di un volo già pieno.
6) Anche se è presente un aereo nell'aeroporto, è impossibile far decollare l'aereo dal menù del dipendente della torre di controllo; infatti, un aereo per poter decollare deve prima aver fatto carburante e deve essere assegnato ad un volo. Effettuare il logout ed entrare nel menù manutenzione (password: 'Man').
7) Selezionare la funzionalità per fare rifornimento, verrà mostrata una lista di aerei che necessitano di carburante. Inserendo uno degli ID in lista si confermerà al sistema di aver fatto carburante all'aereo.
8) Non sarà ancora possibile far decollare l'aereo, infatti è necessario assegnargli prima un volo. Entrare quindi nel menù amministratore tramite la password 'Amm'.
   Avendo un aereo nell'aeroporto sarà possibile utilizzare la funzione per creare un volo, selezionandola si dovrà inserire destinazione, data di partenza e arrivo (le date sono controllate in modo che la partenza non sia prima della data odierna e che

l'arrivo non sia prima della partenza), l'ID dell'aereo che utilizzerà il volo (scelto tra una lista stampata al momento dell'input) e il nome e cognome di comandante e copilota.
Ora nel sistema sarà presente un volo pianificato che resterà nel sistema fino a quando l'aereo assegnato non partirà tramite il comando dal menù torre di controllo.

9) Dal menù amministrazione si può provare a modificare la destinazione del volo inserendo l'ID di un volo esistente (stampati a schermo al momento dell'input) e la nuova destinazione.
10) Prima di far decollare l'aereo è possibile sfruttare questo volo per testare le funzionalità del cliente: effettuare il logout e scegliere di accedere come cliente.
Si può ora cercare i voli programmati per una destinazione inserita in input, se scegliamo la funzione ricerca voli e inseriamo la stessa destinazione inserita durante la pianificazione del volo, vedremo l'ID del volo e la sua data di partenza.
11) Oltre a cercare i voli si può decidere di prenotarne uno. Scegliendo l'apposita funzionalità si vedranno a schermo tutti i voli organizzati e verrà richiesto di inserire l'ID del volo che si desidera prenotare. Inserendo un ID valido sarà possibile compilare tutte le generalità del passeggero, ottenendo così l'ID del biglietto prenotato.
Se si prova ad eseguire un numero di prenotazioni superiori al numero massimo di passeggeri consentiti dall'aereo il sistema bloccherà l'utente avvisando che il volo è al completo.
12) Una volta creata una prenotazione possiamo testare la funzionalità per inserire la certificazione Cov-19 (richiede l'inserimento dell'ID biglietto e della carta d'identità). Questo procedimento attiverà un flag nella prenotazione selezionata che permette al cliente di dichiarare che è in possesso di una certificazione valida.
13) L'ultima funzione che resta da testare è la cancellazione della prenotazione che richiede l'inserimento dell'ID del biglietto da cancellare. Il biglietto viene rimosso automaticamente sia dalla lista dei passeggeri del volo associato, sia dalla lista generale di prenotazioni fatte inerente a tutti i voli dell'aeroporto.
Si consiglia di tenere almeno una prenotazione sul volo in modo da verificare che dopo il decollo tutte le prenotazioni riguardanti il volo decollato vengano rimosse dal sistema.
14) Accedere un'ultima volta come dipendente addetto alla torre di controllo (password: 'TDC'). Dato che l'aereo fatto atterrare in precedenza ha effettuato rifornimento ed è stato assegnato ad un volo, sarà possibile farlo decollare. In seguito si cancellerà automaticamente il volo assegnato e tutte le prenotazioni presenti su di esso.
Dopo il decollo se proviamo ad eseguire le funzioni di inserimento della certificazione o cancellazione della prenotazione dal menù clienti, il sistema comunicherà che non ci sono prenotazioni (cancellazione avvenuta con successo). Allo stesso modo se si prova a prenotare un nuovo volo si noterà che non ci sono voli schedulati. Anche il menù di amministrazione e di manutenzione si troveranno alla situazione di partenza non essendoci aerei nell'aeroporto.
15) Infine come ultimo test è possibile far atterrare il numero massimo di aerei consentiti nell'aeroporto pari a sette. All'arrivo del settimo aereo il sistema non sarà più in grado di accettare atterraggi.

In Eclipse abbiamo implementato dei test automatici tramite Junit sulla maggior parte dei metodi. Non è stato possibile testare la classe main in quanto è costituita principalmente da

richieste di input da tastiera, e dovrà essere testata manualmente usando i passi sopra descritti.
Analizzando la copertura del codice tramite il tool Eclemma si nota che il package aeroporto, ha una copertura del 78%, superiore al valore prefissato pari al 75%.
Il test è composto da un solo metodo che simula l'utilizzo del software. È stato necessario testare tutte le funzionalità in unico metodo in quanto la classe Sistema è un singleton, e utilizzare più metodi sarebbe stato problematico.

CAP 14

Alla fine dell'implementazione del codice e dopo averne testato il funzionamento, abbiamo svolto delle operazioni di manutenzione riguardanti in particolare la facilità di utilizzo del software e la maggior leggibilità del codice.
Dopo la prima implementazione, infatti, il codice era funzionante, ma aveva dei comportamenti poco intuitivi che avrebbero compromesso la facilità di utilizzo. Questo costituiva un problema dato che il programma è pensato per l'utilizzo anche di utenti inesperti.
Un esempio è la funzione per prenotare un volo. Il metodo come prima azione richiede di inserire l'ID di un volo da scegliere tra una lista che viene stampata in output. Nel caso di mancanza di voli veniva comunque richiesto l'inserimento, stampando però una lista vuota. Questo avrebbe potuto creare confusione, infatti l'unico modo per tornare al menù precedente era quello di inserire una stringa di caratteri casuali. In questo modo il sistema non riconoscendo la stringa inserita stampava un messaggio di errore e tornava al menù precedente.
Tramite l'operazione di refactoring è stato inserito un controllo che, in caso non ci siano voli disponibili, non permette l'inserimento della stringa e avvisa immediatamente l'utente che non ci sono voli da prenotare.
Questa operazione è stata svolta in modo simile anche per le funzioni "fai carburante" nel menù manutenzione, "nuovo volo" (nel caso non ci siano aerei disponibili) "modifica destinazione" e "cancella volo" nel menù amministrazione, "decollo aereo" nel menù torre di controllo.
Queste modifiche rendono il codice più intuitivo da utilizzare e restituiscono sempre un output adeguato, in modo da guidare meglio chi utilizza il software.

Oltre a questi cambiamenti è stata svolta anche un'operazione di "pulizia" del codice, in modo che sia più leggibile da nuovi programmatori e per eventuali nuove operazioni di manutenzione. Il codice essendo frutto del lavoro di tre persone distinte, che hanno implementato parti diverse in modo autonomo, risultava disorganizzato e poco leggibile. È stato necessario quindi rielaborarlo in modo da avere un risultato omogeneo e ben commentato.

Un altro lavoro di refactoring è stata l'implementazione di nuovi metodi nella classe Sistema che permettono di esplorare le liste e stamparle.
Prima di questa operazione venivano invocate spesso classi ad un livello gerarchico superiore della classe chiamante. Il fenomeno appena descritto (osservato soprattutto tra i metodi delle classi Aereo, Prenotazione e Volo) generava un aumento del livello di accoppiamento tra le classi, provocando una scarsa manutenibilità.