Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione basata su UML e implementazione basata su Python di un sistema informativo per la gestione di una cartolibreria

UML-based design and Python-based implementation of an information system for the management of a stationery shop

Relatore

Prof. Domenico Ursino

Candidato

Kevin Giannattasio

Anno Accademico 2020-2021

Indice

Elenco delle figure

Introduzione

Agli albori dell'informatica, negli anni '50, i paesi industrializzati hanno iniziato un lungo e radicale cambiamento tecnologico in virtù della nascita dei primi calcolatori elettronici. Questi ultimi vennero denominati computer, dal latino "computare", che significa, letteralmente, "fare di conto". Infatti, i computer vennero concepiti per automatizzare alcune capacità della mente umana, come, ad esempio, il calcolo e la memorizzazione dei dati, potenziandone la portata e applicandoli alla soluzione di particolari problemi scientifici ed ingegneristici.

Questo avvenne grazie allo sviluppo di software, ovvero insiemi di programmi scritti in uno dei linguaggi di programmazione tra i tanti che nacquero nel XX secolo.

Ben presto ci si rese conto della potenzialità dei software, i quali vennero adoperati all'interno delle aziende per automatizzare alcuni processi lavorativi, per eseguire calcoli finanziari, per memorizzare digitalmente grandi quantità di informazioni, e per tantissime altre funzionalità.

Oltre all'industria del lavoro, i software acquisirono importanza anche in quella dell'intrattenimento. Basti pensare alle prime televisioni, ai telefoni cellulari, oppure ai cabinati dei videogiochi arcade.

Nel mercato dei software la richiesta era molto alta, ma le persone effettivamente in grado di sviluppare erano poche, e solitamente poco competenti. I programmatori erano perlopiù autodidatti e provenienti da settori scientifico-disciplinari non puramente informatici. Inoltre, non esistevano tecniche di progettazione standard per la creazione dei prodotti software, bensì l'intero sviluppo consisteva nella mera programmazione. Ciò portava facilmente a sforare il budget ed il tempo preventivati e, soprattutto, a sviluppare dei prodotti di scarsa qualità, che non rispettavano i requisiti prefissati e che avevano errori e malfunzionamenti di ogni sorta.

Queste problematiche erano note all'interno delle società di allora, le quali, di conseguenza, decisero di movimentarsi all'insegna dello sviluppo tecnologico. Nella seconda metà dello scorso secolo vennero effettuati studi e conferenze che decretarono la nascita di una nuova disciplina ingegneristica, denominata *ingegneria del software*, che prevedeva l'utilizzo di tecniche standard per la produzione di software di alta qualità, nel rispetto dei tempi di consegna, del budget prefissato e dei requisiti richiesti dai committenti.

Tutt'oggi questa disciplina è in continuo sviluppo, e prevede un insieme di fasi da dover attraversare a seconda del processo software scelto.

In questa tesi si intende porre particolare attenzione sulla tipologia di sviluppo "plan-driven", che prevede una dettagliata e meticolosa fase di progettazione a monte dell'implementazione del software.

In particolare, vengono presentate ed esplicate alcune tra tutte le tappe fondamentali che sono state attraversate nello sviluppo "plan-driven" di un software standalone, adibito alla gestione di una cartolibreria realmente esistente.

Pertanto, si è deciso di strutturare la tesi nel seguente modo:

- Nel Capitolo 1 viene introdotta l'ingegneria del software, facendo riferimento ai momenti storici che ne hanno favorito la nascita. Successivamente, viene analizzato il processo "plan-driven", utilizzato per lo sviluppo del prodotto software in questione.
- Nel Capitolo 2 viene riportata la documentazione relativa alla fase di specifica del software, durante la quale sono stati dedotti i requisiti da dover implementare nel sistema.
- Nel Capitolo 3 è presente la documentazione relativa alla progettazione del database sviluppato per introdurre la persistenza dei dati nel software prodotto.
- Nel Capitolo 4 vengono riportati i casi d'uso estrapolati dai requisiti funzionali del sistema.
- Nel Capitolo 5 viene trattato il linguaggio di programmazione utilizzato per la fase d'implementazione, ossia Python. Successivamente, sono riportati ed esplicati i punti chiave del codice utilizzato.
- Nel Capitolo 6 viene eseguita un'analisi di fine progetto, per comprendere quali sono gli aspetti positivi e quelli negativi del software prodotto.
- Nel Capitolo 7 sono riportate le conclusioni in merito al lavoro svolto.

1

Gestione “plan-driven” di un progetto software

L’ingegneria del software è una disciplina piuttosto recente. Agli albori dell’informatica le competenze richieste per generare i prodotti software erano ristrette alla mera programmazione. L’assenza di sufficienti nozioni a supporto dello sviluppo di tali prodotti fece sì che venissero implementati software di infima qualità, in tempi e costi smisurati. Il che ha portato all’insorgere della così detta “crisi del software”. In questo capitolo viene mostrato il modo con cui si è arrivati a concepire i principi che stanno alla base dell’ingegneria del software così com’è conosciuta oggi. Nello specifico, ci si concentra maggiormente sul processo software di tipo “plan-driven”, tramite cui l’implementazione di un prodotto software non è altro che la diretta conseguenza della sua progettazione.

1.1 L’ingegneria del software

L’ingegneria del software è definita come la disciplina tecnologica e gestionale che riguarda la produzione sistematica, oltre che la manutenzione, dei prodotti software che vengono sviluppati e modificati entro tempi e costi preventivati.

Allo stato attuale, il software è diffuso su larga scala in tutti i settori lavorativi, oltre che in molti contesti sociali. Infatti, l’economia di tutti i paesi industrializzati dipende dal software. Basti pensare che, agli inizi degli anni ’80, negli Stati Uniti d’America, le spese per lo sviluppo dei programmi applicativi comprendevano all’incirca il 2% del Prodotto Interno Lordo (PIL).

Nelle società moderne, la produzione e la distribuzione industriale sono completamente computerizzate, così come il sistema finanziario. L’intrattenimento, come musica, cinema e televisione, fa uso intensivo del software. Questa stessa tesi è stata scritta utilizzando un’opportuna applicazione.

L’impulso maggiore all’evoluzione dell’ingegneria del software è fornito, senza dubbio, dallo sviluppo delle tecnologie della rete. Internet è, a tutti gli effetti, una della maggiori rivoluzioni tecnologiche e culturali dell’ultimo millennio. Il crollo dei costi dell’hardware, la grande competitività tra le aziende software e l’uso del web inducono l’ingegneria del software a ridefinire i suoi principi: tecnologie, metodologie e culture. Internet e l’open source rappresentano due pilastri imprescindibili che

costringono tutti i produttori a ridefinire le proprie strategie. E lo sviluppo del software non può esimersi dall'adeguarsi a tali nuovi paradigmi.

1.1.1 La crisi del software

Agli albori dell'informatica, negli anni '50, gli sviluppatori sono pochi e provenienti da altre discipline ingegneristiche o scientifiche. Le competenze informatiche si riducono alla sola conoscenza dei linguaggi di programmazione. Le metodologie di sviluppo sono totalmente assenti, e le competenze sono acquisite sul campo. Lo sviluppo del software si riduce dunque alla mera programmazione. Le uniche discipline applicate sono quelle degli algoritmi, delle strutture dati e dei linguaggi di programmazione.

Molti grandi progetti software realizzati in quegli anni per risolvere complesse problematiche scientifiche e applicative falliscono miseramente, non riuscendo a sviluppare software affidabile e capace di soddisfare i requisiti. Inoltre, i tempi di consegna non sono mai rispettati e i costi aumentano a dismisura. La problematica, conosciuta come “crisi del software”, affonda le sue radici nell'inesistenza di una pratica ingegneristica consolidata, con basi scientifiche e metodologicamente definite. Si decide, allora, di avviare una fase di studio e ricerca per creare i fondamenti di una scienza ingegneristica, battezzata “Software Engineering”.

L'ingegneria del software nasce ufficialmente tra il 1968 e il 1969, in occasione di due conferenze NATO tenutesi a Garmisch, in Germania, e a Roma. La nascita ufficiale dell'ingegneria del software avvia importanti studi e attività di ricerca nel campo delle metodologie, dei metodi e delle tecniche per il software. Si definisce una nuova disciplina su base teorica e scientifica portando l'ingegneria del software alla stregua degli altri rami ingegneristici.

Ad oggi, la conoscenza dei linguaggi di programmazione, dei sistemi operativi e di quelli di gestione è considerata la competenza necessaria e sufficiente per programmare, o produrre codice. Sviluppare software, invece, è ben più complesso e richiede anche competenze metodologiche relative all'ingegneria dei requisiti, alla progettazione del software, al test e collaudo ed alle discipline collaterali.

1.1.2 Software Engineering Body of Knowledge (SWEBOK)

Diversamente da quanto stabilito per altri settori dell'ingegneria, a riguardo del software non sono definite formalmente le competenze necessarie per ricoprire i vari ruoli previsti. Ancora oggi la professione di ingegnere del software non è riconosciuta legalmente e non esiste un albo professionale.

Tuttavia, ciò non esime le organizzazioni software dal definire al loro interno ruoli con rispettive responsabilità, autorità e competenze.

Alla professione dell'Ingegnere del software è richiesto un insieme di competenze base identificate come “Software Engineering Body of Knowledge (SWEBOK)”, da acquisire presso università, scuole di specializzazione o corsi ad hoc organizzati dalle aziende. La versione corrente della guida emessa nel 2004 consta di dieci competenze di base (vedi Figura 1.1) che sono:

1. Software Requirements;

2. Software Design;
3. Software Construction;
4. Software Testing;
5. Software Maintenance;
6. Software Configuration Management;
7. Software Engineering Management;
8. Software Engineering Process;
9. Software Tolls and Methods;
10. Software Quality.

Le prime cinque competenze si riferiscono ai processi primari dello sviluppo software, mentre le altre cinque ai processi di supporto.

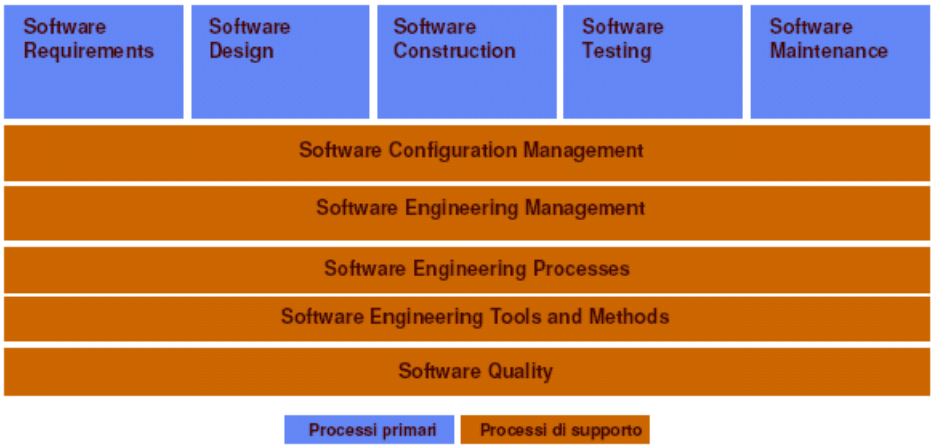

Figura 1.1. Le dieci competenze base dell'Ingegneria del software

1.2 Il processo di sviluppo software

L'ingegneria del software è finalizzata alla realizzazione di un prodotto. Un prodotto software è definito come i programmi, le procedure, l'eventuale documentazione associata, e i dati relativi all'operatività di un sistema di elaborazione. Ogni software è riferibile ad una delle seguenti categorie:

- Software di sistema, ovvero una collezione di programmi al servizio di altri (ad esempio, i compilatori).
- Software real-time, che sorveglia, analizza e controlla eventi esterni.
- Software gestionale, per l'elaborazione dei dati aziendali (ad esempio, Enterprice Resource Planning - ERP).
- Software scientifico e/o per l'ingegneria (ad esempio, Computer Aided Design - CAD).

- Software Embedded, ovvero incorporato nei sistemi (ad esempio, in automobili o lavatrici).
- Software web-based, in cui tutte le funzioni sono accessibili tramite un normale web browser, come Firefox, Chrome o Explorer.
- Software per personal computer, come Microsoft Office, gli antivirus, i videogiochi, etc.
- Software per l’intelligenza artificiale, per il riconoscimento vocale, l’apprendimento dei robot, etc.

Ma al di là della sua applicazione, un software deve essere sviluppato in modo che sia quanto più possibile di qualità, in accordo con quattro proprietà da dover rispettare:

- *Manutenibilità*, in quanto deve essere modificabile in modo da soddisfare nuovi requisiti.
- *Affidabilità*, poiché in caso di guasto non deve produrre danni fisici od economici.
- *Efficienza*, in quanto non deve fare uso indiscriminato di memoria e tempo di calcolo.
- *Facilità di utilizzo*, quindi deve essere corredato di un’interfaccia utente.

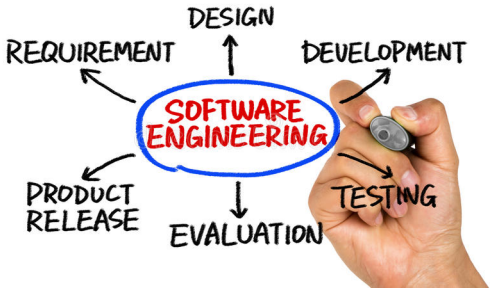

Figura 1.2. Il processo dell’ingegneria del software

Per far sì che un prodotto rispetti tali caratteristiche occorre affidarsi ad un corretto processo software, ovvero quella serie di attività necessarie alla realizzazione di un prodotto software nei tempi, costi e caratteristiche desiderate. Il ciclo di vita di un processo può variare a seconda del prodotto software da dover realizzare. Tuttavia, in generale, vengono sempre percorse quattro fasi fondamentali (vedi Figura 1.2), nel seguente ordine:

1. *Specifica del software*, ovvero quella fase in cui vengono definite le funzionalità e i vincoli operativi da dover rispettare.
2. *Progettazione e implementazione del software*, dove avviene l’effettiva realizzazione del prodotto.
3. *Validazione del software*, per accertarsi che il cliente ne sia soddisfatto.
4. *Evoluzione del software*, il quale dopo il rilascio deve essere continuamente modificato per adattarsi a nuove esigenze o per porre soluzione a problemi di diversa natura.

Il modo con cui vengono sviluppate queste quattro fasi determina il tipo di processo che si sta realizzando.

Ad un estremo esiste lo sviluppo agile, nato all'inizio del secolo attuale, che fonda la sua filosofia sull'implementazione del software, a discapito della fase di progettazione, che è ridotta al minimo o completamente assente. Questa caratteristica rende lo sviluppo agile particolarmente adatto a progetti di piccola o media dimensione (ad esempio, i siti web), non critici e con requisiti che cambiano rapidamente.

I processi agili seguono un modello di sviluppo detto incrementale, o iterativo (vedi Figura 1.3), tale per cui le quattro fasi fondamentali di un processo, precedentemente introdotte, vengono percorse ciclicamente più volte, producendo ad ogni *incremento* una versione, o *release*, aggiornata del software. Ognuna di queste release viene prontamente testata, mediante test automatici, utilizzati a supporto dello sviluppo, come test di regressione, che mirano, quindi, a verificare l'insorgere di eventuali comportamenti anomali in seguito all'integrazione di un nuovo componente all'interno del software che si sta implementando. Infine, la versione prodotta viene sottoposta al committente, o ad un suo rappresentante, che deve trovarsi a stretto contatto con il team di sviluppo, al fine di fornire un feedback immediato ed, eventualmente, introdurre nuovi requisiti al prodotto che si sta sviluppando.

I punti di forza del processo agile diventano debolezze nel momento in cui si decide di applicarlo per lo sviluppo di un software di grandi dimensioni e/o che dovrà operare in un ambiente critico, dove un qualsiasi guasto o malfunzionamento può portare a danni più o meno importanti all'interno dell'azienda che lo utilizza. Infatti, mediante lo sviluppo incrementale, si ha una documentazione di progettazione quasi (se non del tutto) inesistente, il che comporta un costo di manutenzione del software più elevato, soprattutto se il team che avrà il compito di far evolvere il software è diverso dal team che lo ha prodotto. Inoltre, non è possibile garantire la validità dei requisiti, se questi vengono continuamente modificati o se vengono introdotti nuovi requisiti in fase di implementazione. Infine, i costi per il rappresentante del cliente, che sia costantemente a contatto con il team di sviluppo, diventano molto alti da sostenere per l'azienda che commissiona il software, se questo richiede molto lavoro prima di essere rilasciato.

1.2.1 I processi "plan-driven"

All'estremo opposto dei processi agile ci sono quelli "plan-driven", detti anche processi guidati da piani, dove tutte le attività sono pianificate in anticipo e il loro avanzamento è misurato rispetto a quanto previsto dal piano. Questo tipo di approccio, di solito, è conveniente laddove si intende progettare e sviluppare sistemi di grandi dimensioni, oppure critici.

Un sistema critico (in inglese critical system) è un generico sistema che, in caso di fallimento nel suo funzionamento, può provocare danni non considerati accettabili, quali gravi danni economici, ambientali o di vita.

Si pensi, per esempio, ad un sistema per il controllo di una navicella spaziale, dove il minimo errore può rappresentare una seria minaccia per la vita degli uomini a bordo. L'uso di un approccio agile, dove la pianificazione è ridotta al minimo, sarebbe rischioso a livelli inaccettabili per un software di questo tipo. Inoltre, occorrerebbe pianificare con estrema accortezza l'hardware e il software da utilizzare

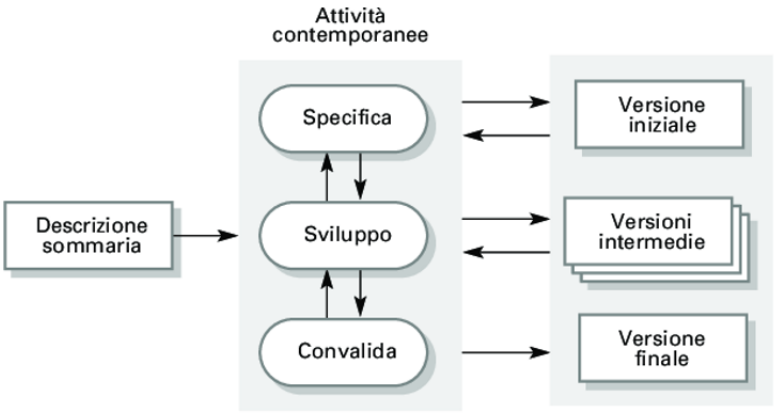

Figura 1.3. Schema del modello di sviluppo incrementale

per produrre il sistema. Rientrano in questa categoria di sistemi i dispositivi medici, i sistemi avionistici, razzi o missili, e gli impianti nucleari.

Inoltre, è preferibile utilizzare un processo “plan-driven” anche per software di grandi dimensioni, utilizzati in ambienti non critici. Un esempio calzante potrebbe essere quello di un software gestionale per una banca, dove un malfunzionamento può rivelarsi disagiante per i fruitori dell’applicazione bancaria (gli utenti), ma anche per i proprietari della banca, che saranno soggetti a delle perdite economiche più o meno grandi, a seconda di quanto tempo sarà necessario per risolvere il disservizio. Quanto più sarà accurata la progettazione del software, tanto minori saranno le probabilità di incorrere in malfunzionamenti di ogni sorta. Come se non bastasse, i prodotti software di grandi dimensioni tipicamente sono costosi, costituendo un vero e proprio investimento da parte dell’azienda committente. Per questo motivo si vuole agevolare quanto più possibile il mantenimento e il processo evolutivo di questi prodotti. Senza una corretta fase di progettazione ed un’abbondante documentazione del software sarebbe più complicato poter lavorare al processo evolutivo da parte del team di sviluppo incaricato. Mantenere ed evolvere il software diverrebbe, quindi, notevolmente più costoso, sia economicamente, che in termini temporali.

1.2.2 Il modello a cascata

I processi “plan-driven” seguono un modello di sviluppo a cascata (waterfall model), la cui diffusione ha avuto inizio negli anni ’70 ed è tutt’oggi ancora molto utilizzato. Secondo il modello a cascata le quattro attività fondamentali di specifica, progettazione/implementazione, convalida ed evoluzione del software vengono trattate come fasi distinte del processo (vedi Figura 1.4). Ogni attività deve essere completata prima che si possa passare alla successiva; essa riceve come input l’output dell’attività precedente.

La prima di queste attività è la specifica del software, svolta da quella branca dell’ingegneria del software che prende il nome di ingegneria dei requisiti. A monte

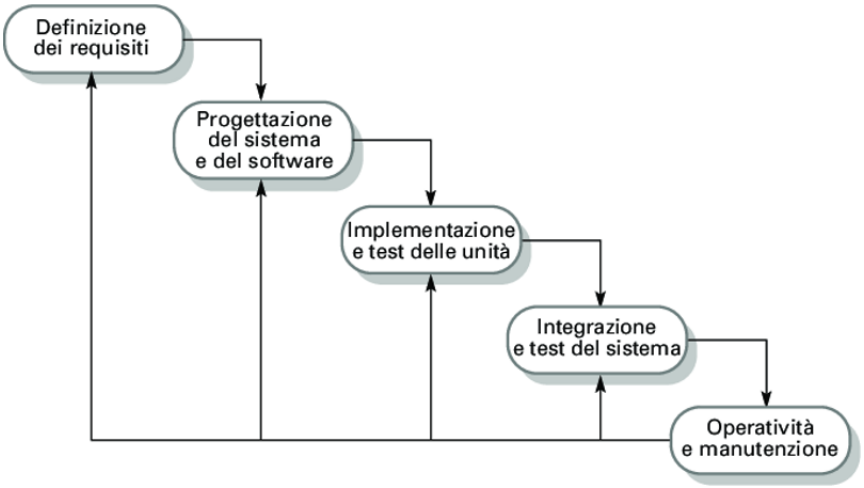

Figura 1.4. Schema del modello di sviluppo a cascata

di questa fase è tipicamente previsto uno studio di fattibilità per verificare che sia effettivamente realistico, sia tecnicamente che economicamente, produrre il software richiesto. Se lo studio di fattibilità viene superato con successo vengono eseguite le seguenti tre sottofasi fondamentali, al fine di produrre un output per l'attività successiva:

1. La deduzione e analisi dei requisiti prevede il confronto diretto con gli stakeholder, ovvero tutti i soggetti attivamente coinvolti nel progetto. Questo confronto può avvenire mediante un'intervista (aperta o chiusa), per comprendere quali sono le funzionalità che debba possedere il software che si intende produrre, oppure attraverso l'etnografia, cioè l'osservazione del dominio applicativo nel quale andrà ad operare il software.
2. Nella specifica dei requisiti le informazioni raccolte dalla fase precedente vengono strutturate e categorizzate all'interno di un documento. La strutturazione dell'informazione prevede la suddivisione dei requisiti in funzionali (per descrivere che cosa il sistema deve fare) e non funzionali (per imporre i vincoli da rispettare nello sviluppo del sistema).
3. Infine la convalida dei requisiti consiste nel verificare se il documento prodotto può essere utilizzato come input della fase di progettazione. A tale scopo, vengono effettuati controlli di validità, consistenza, completezza e realismo dei requisiti.

In genere, la specifica del software termina dopo più iterazioni delle suddette sottofasi. In effetti, gli stakeholder tendono a cambiare facilmente opinione sulle funzionalità da introdurre, non avendo le idee chiare sul software di cui necessitano (vedi Figura 1.5). Inoltre, i requisiti di stakeholder differenti sono, talvolta, in contraddizione tra loro.

Seguendo il modello a cascata, una volta completata la fase di specifica del software si passa alla sua progettazione. Questa attività consiste a sua volta in più sottofasi:

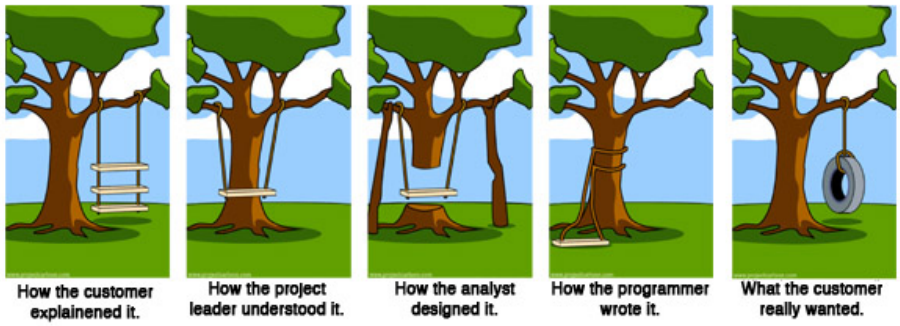

Figura 1.5. Quello di cui hanno effettivamente bisogno gli stakeholder è spesso diverso da quello che chiedono

- La progettazione dell’architettura consiste nello stabilire la struttura complessiva del software, ovvero quali sono i moduli che lo compongono e come essi comunicano tra loro.
- Nella progettazione del database vengono stabilite le strutture dati del sistema e come queste devono essere rappresentate all’interno del database.
- La progettazione dell’interfaccia utente è utile per poter stabilire un facile accesso alle funzionalità del software da parte degli utenti. Infatti, è bene che l’interfaccia di un software fornisca una user experience (ovvero una relazione tra il prodotto e la persona che lo utilizza) piacevole, e che invogli gli utenti al suo utilizzo.
- La fase di progettazione e scelta dei componenti consiste nel verificare se esistono dei componenti di software precedenti che siano riutilizzabili (in accordo con la *Component Based Software Engineering*), da integrare tra loro all’interno del progetto, in modo da risparmiare tempo e parte del budget per lo sviluppo. Laddove questi componenti non vengono trovati, o non sono integrabili tra loro, sarà, ovviamente, necessaria una progettazione da zero. In questa fase viene utilizzato UML, acronimo di *Unified Model Language*, ovvero un linguaggio di modellazione basato sul paradigma orientato agli oggetti, capace di fornire sia una rappresentazione strutturale che dinamica del progetto che si sta sviluppando.

L’output della fase di progettazione è un insieme di documenti e diagrammi dettagliati che descrivono in modo accurato il sistema; essi saranno utilizzati come riferimento diretto per l’implementazione del software.

Grazie all’utilizzo del modello a cascata la fase di implementazione costituisce la diretta conseguenza della progettazione; in questo modo il codice finale sarà tanto più di qualità quanto più lo sono stati il documento e i diagrammi prodotti nella progettazione. Ovviamente non si può dire lo stesso per il modello di sviluppo incrementale, dove l’implementazione avviene in assenza di linee guida, a causa della mancanza di un’accurata progettazione.

La terza attività consiste nel testing del sistema. In realtà, il concetto di testing fa parte di una categoria più ampia, ossia quella di verifica e convalida (*Software verification and validation*).

Verificare un software significa controllare che questo non manifesti bug, ovvero comportamenti anomali in seguito a determinate sequenze di input, e che soddisfi i requisiti stabiliti nella corrispondente documentazione.

D'altro canto, la convalida del software ha un'accezione più generica di quella della verifica, e mira a soddisfare le aspettative dei clienti. In quest'ottica, oltre ai meccanismi interni che governano il prodotto software, ha particolare valenza anche la user experience offerta.

Quello del testing è un argomento molto vasto. In effetti, esso viene utilizzato a supporto della fase di sviluppo del software, attraverso gli unit test, i test delle componenti ed i test di sistema, tutti svolti dal team di sviluppo.

Inoltre, deve essere effettuato il cosiddetto test di rilascio, a scopo di convincere il committente che il software prodotto sia pronto all'uso. Questo viene eseguito mediante i test dei requisiti ed il test delle prestazioni. Il primo è un test quantitativo che verifica la realizzazione di ogni requisito, utilizzando a supporto gli use case diagram (ovvero i diagrammi dei casi d'uso) di UML. Il secondo è un test qualitativo che utilizza approcci come gli *stress test* per verificare che le funzionalità introdotte nel software vengono attuate in tempi adeguati, utilizzando una quantità di risorse hardware accettabili.

Infine, esiste il test degli utenti (gli alpha test e i beta test) dove il software viene messo a disposizione di un'utenza limitata, che utilizza quindi le funzionalità del prodotto ed effettua segnalazioni agli sviluppatori in caso di problemi.

Il 60% dei costi di un prodotto software di solito vengono sostenuti solo in seguito al suo rilascio. Infatti, i sistemi software si adattano ed evolvono continuamente durante il loro ciclo di vita, dal rilascio della prima versione fino all'ultima.

Esistono tre diverse tipologie di manutenzione:

- La manutenzione correttiva mira a correggere errori e vulnerabilità del software che vengono scoperti in seguito al suo rilascio. Il costo di manutenzione da sostenere in questo caso è a carico del team di sviluppo.
- La manutenzione adattiva serve ad applicare delle modifiche al software a seguito di cambiamenti dell'ambiente, come il sistema operativo o l'hardware. Generalmente questo tipo di manutenzione è a carico del cliente.
- La manutenzione perfettiva invece serve per introdurre modifiche che migliorino le qualità del software, come ad esempio nuove funzionalità (ovvero requisiti che sorgono in seguito al suo rilascio). Questo tipo di manutenzione è sempre a carico del cliente.

In genere quest'ultima è il tipo di manutenzione a cui è più comunemente soggetto un software. Il motivo è dato dalla presenza dei cosiddetti sistemi legacy, ovvero sistemi software vecchi, che non sono stati dismessi poiché chi li utilizza non può o non intende farlo, per ragioni diverse. Il motivo principale è che sostituire un sistema implica un periodo di dismissione, che comporta una perdita di denaro che non si vuole sostenere.

Quindi, piuttosto che la sostituzione, per i sistemi legacy spesso si applicano altre soluzioni, come quella del *re-engeneering* (o reingegnerizzazione), che prevede la progettazione e implementazione di nuovi componenti che diano al sistema la capacità di potersi interfacciare con i sistemi moderni. La reingnerizzazione del software potrebbe richiedere una nuova documentazione del sistema, la rifattorizzazione della

sua architettura, la traduzione dei programmi in un linguaggio di programmazione moderno, la modifica e l’aggiornamento della sua struttura e dei suoi dati.

Qui termina il *workflow* del processo software messo in atto seguendo un modello a cascata. Ovviamente, a scopo introduttivo, è stato tutto riassunto al minimo, tralasciando molti argomenti importanti che, invece, andrebbero approfonditi da parte di chi fosse interessato a questa disciplina informatica. Un concetto molto importante che non è stato menzionato è quello dei *design pattern* (o pattern progettuali), definiti come “una soluzione progettuale generale ad un problema ricorrente”. Esistono 23 principali design pattern, ampiamente descritti nel libro *Design Patterns: Elements of Reusable Object-Oriented Software*, pubblicato nel 1994 da E. Gamma, R. Helm, R. Johnson e J. Vlissides (ricordati anche come la *Gang of Four*). In Figura 1.6 viene mostrata la copertina originale del libro.

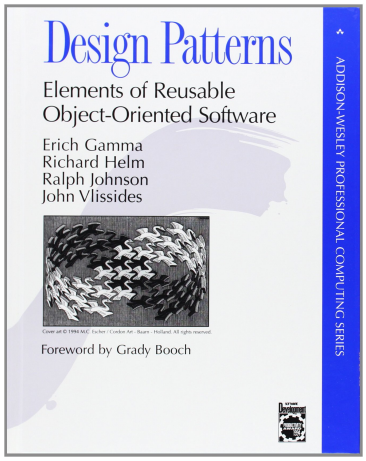

Figura 1.6. Design Patterns: Elements of Reusable Object-Oriented Software

2

Descrizione del contesto di riferimento e analisi dei requisiti

In questo capitolo, sarà trattato un caso reale, adoperando un processo "plan driven" per lo sviluppo di un software stand-alone destinato alla gestione di una cartolibreria. L'attività in questione è denominata Compucart, ed è situata in Molise, in un paese chiamato San Martino in Pensilis. Viene riportata tutta la documentazione relativa alla fase di specifica software della suddetta cartolibreria. In dettaglio, è ampiamente discusso tutto il processo che parte dalla raccolta di informazioni, attraverso un'intervista aperta, fino ad arrivare all'estrapolazione dei requisiti, e la loro successiva convalida.

2.1 Intervista aperta

Il primo passo da compiere è quello di dedurre le funzionalità di cui deve essere dotato il sistema. A tale scopo, nel progetto di riferimento, si possono percorrere tre strade:

1. L'intervista, possibilmente aperta, per ottenere una descrizione dettagliata del contesto di riferimento.
2. L'etnografia, per comprendere il dominio applicativo nel quale andrà a collocarsi il software, allo scopo di prevenire eventuali problematiche che potrebbero sorgere in fase di sviluppo.
3. Sia l'intervista aperta che l'etnografia, in modo da avere una visione completa e dettagliata sulle dinamiche che coinvolgono la cartolibreria, riducendo al minimo le probabilità di dover fronteggiare defezioni dei requisiti nelle fasi successive.

La scelta per la quale si era optato inizialmente era la terza, ossia quella di effettuare un'etnografia, in modo da avere a disposizione ulteriori informazioni a supporto dell'intervista.

I titolari, o stakeholder, della cartolibreria si sono fatti trovare molto disponibili per sostenere un colloquio. D'altro canto però, hanno preferito evitare il sopralluogo, principalmente per le restrizioni che erano in vigore al momento della fase di specifica del software, a causa della pandemia di COVID-19.

Pertanto, l'unico strumento del quale si è potuto fruire per la deduzione dei requisiti è stata la sola intervista, comprendente un incontro di persona con i diretti

interessati, e ulteriori contatti telefonici finalizzati a risolvere i vari dubbi sorti dopo l'incontro.

Nel seguito è riportato un sunto delle parti più importanti dell'intervista, in forma di discorso diretto, con le domande effettuate e le corrispondenti risposte.

- *Parlateci del vostro lavoro, di che attività si tratta?*
  "La nostra è a tutti gli effetti una cartolibreria. Quindi, vendiamo principalmente articoli di cancelleria e libri scolastici. Secondariamente, offriamo servizi di fax, fotocopie, stampe, mail, plastificazione di documenti, rilegatura di tesi e incartamento di libri. Anche se può suonare atipico per una cartolibreria, occasionalmente vendiamo articoli informatici, e forniamo assistenza tecnica ai computer."
- *Qual è la vostra routine e che orari di lavoro avete?*
  "Lavoriamo tutta la settimana, ad esclusione della domenica, dalle 7:30 alle 13:00 la mattina, mentre nel pomeriggio dalle 16:00 alle 20:00. Abbiamo differenti categorie di clienti, richiedenti diversi tipi di servizi, per cui non abbiamo una routine fissa. Tutto dipende dai clienti della giornata."
- *Quali aspetti della vostra attività ritenete opportuno informatizzare?*
  "La nostra principale necessità è quella di poter gestire in maniera semplice e veloce il duplice processo di acquisto all'ingrosso degli articoli dai fornitori, e la loro corrispondente vendita ai clienti. A tale scopo avremmo bisogno di tenere sotto controllo in ogni istante tutti gli articoli presenti in magazzino. In questo modo possiamo sapere se e quando è necessario ordinare altra merce. Inoltre, vorremmo poter tenere traccia delle vendite, mediante la registrazione delle fatture e degli scontrini, con un sistema collegato direttamente al registratore di cassa."
- *È necessario memorizzare i dati dei clienti all'interno del software che utilizzerete?*
  "Riteniamo di sì, dal momento che i clienti hanno la possibilità di chiedere il rilascio di una fattura, e a tale scopo è necessario conoscere i loro dati. In particolare, distinguiamo i clienti in due categorie separate sulla base del possesso o meno della partita IVA. In quest'ultimo caso si tratta spesso di clienti che comprano per conto di una ditta."
- *Avete dei dipendenti di cui volete gestire lo stipendio mediante il software?*
  "Allo stato attuale non abbiamo dipendenti. Tuttavia, in alcuni periodi, capita che necessitiamo che qualcuno ci dia una mano con il lavoro. In ogni caso, sono sempre contratti a termine, almeno per il momento. Però, riteniamo che potrebbe essere utile avere una sezione del software dedicata alla gestione dei dipendenti."
- *In che rapporti siete con i vostri fornitori?*
  "Per ogni tipologia di merce sappiamo a chi rivolgerci. Abbiamo una lista di numeri di telefono ed e-mail da utilizzare per metterci in contatto con i nostri fornitori. Tali contatti avvengono tipicamente ogni mese, ma in realtà dipende dalla domanda dei clienti. Per esempio, nel periodo estivo che precede l'inizio delle scuole, riceviamo molte più richieste di libri scolastici rispetto all'inverno inoltrato."
- *In che modo volete poter gestire le informazioni sulle fatture e sugli scontrini?*
  "Innanzitutto, ci sono sostanziali differenze tra i due. Gli scontrini non hanno un intestatario, ma devono comunque essere rilasciati con un numero identificativo e

la lista di articoli acquistati. Inoltre, l'erogazione dello scontrino si può annullare nel caso in cui ci fossero refusi. Le fatture, invece, sono diverse, in quanto devono obbligatoriamente possedere un intestatario, oltre che tutti i dati descritti nell'articolo 21, comma 2, del D.P.R. 633/1972. Per legge, al momento della sua registrazione, ogni fattura deve essere inviata all'agenzia delle entrate. Quindi, in caso di errori, non è possibile sostituirla senza farne direttamente richiesta all'agenzia."

- *In generale, quali sono i dati che vi trovate a dover gestire durante il lavoro?*
  "I dati che dobbiamo utilizzare più spesso sono quelli relativi alla merce. Ad ogni articolo devono essere associati una breve descrizione ed il prezzo, oltre che la quantità presente in magazzino. Qualora un certo numero minimo di articoli dovesse mancare, sarebbe necessario contattare subito i nostri fornitori. Quindi dobbiamo avere a disposizione tutti i loro recapiti. Inoltre, come vi abbiamo già detto, vorremmo gestire lo storico delle vendite, attraverso la memorizzazione degli scontrini e delle fatture erogate. Infine, vi abbiamo già parlato delle informazioni di cui vorremmo disporre circa i dipendenti ed i clienti."

2.2 Descrizione del contesto

Le informazioni estrapolate dalla precedente intervista saranno ora descritte in maniera formale, al fine di avere un discorso completo a cui fare affidamento per la deduzione dei requisiti.

Compucart è una cartolibreria gestita da una coppia di titolari, in un piccolo paese del basso Molise, chiamato San Martino in Pensilis.

L'attività è a disposizione di una vasta clientela, distinguibile in due categorie: si parla, infatti, di clienti privati e clienti possessori di partita IVA, i quali tipicamente effettuano acquisti per conto di un'azienda. I servizi offerti riguardano principalmente la vendita di articoli di cancelleria e libreria, oltre che la messa a disposizione di sistemi di stampa, fax, fotocopie, mail, plastificazione, rilegatura e incartamento.

All'interno del processo lavorativo, i titolari dell'attività entrano prima di tutto in contatto con i loro fornitori, per poter acquisire all'ingrosso la merce che sarà, poi, venduta ai clienti. Tale merce consiste in articoli di ogni genere, le cui disponibilità in magazzino dovranno essere monitorabili in ogni momento.

Inoltre, il magazzino viene gestito con un meccanismo di carico/scarico. Queste due operazioni dovranno poter essere effettuate manualmente, ma anche automaticamente, in quanto ad ogni acquisto presso un fornitore corrisponderà un'operazione di carico, e ad ogni erogazione di una ricevuta dovrà essere effettuato lo scarico degli articoli venduti. Le ricevute erogabili sono di due tipi. Su richiesta del cliente, in seguito ad un suo acquisto, deve essere possibile effettuare una fattura, contenente i dati del cliente stesso (ai sensi dell'art. 21, comma 2, D.P.R. 633/1972), oltre che il numero identificativo, la data di erogazione e la lista della merce acquistata. In seguito alla presenza di errori all'interno di una fattura, non è possibile effettuarne l'annullamento senza contattare direttamente l' agenzia delle entrate. In alternativa ci sono gli scontrini che, a differenza delle fatture, si possono annullare senza mettere in atto alcun processo burocratico.

I lavoratori della cartolibreria sono i titolari stessi che, però, non escludono la possibilità di assumere del personale in futuro. In tal caso, oltre ai recapiti e alle mansioni dei dipendenti, sarà necessario tener conto delle rispettive retribuzioni.

Il software che si intende sviluppare, per poter gestire tutte queste funzionalità, dovrà interfacciarsi con il registratore di cassa, e con il sistema elettronico di fatturazione. Inoltre, dovrà essere presente un database esterno grazie al quale è possibile avere dei dati persistenti da poter utilizzare nei processi elaborativi.

2.3 Glossario dei termini

Nella descrizione del contesto, sono stati sottolineati i termini che si ritiene ambigui, per i quali è quindi necessaria una spiegazione.

Sono stati sottolineati in verde i termini legati al business. Invece, è stato utilizzato il blu per i termini tecnici.

Nelle Tabelle 2.1 e 2.2 sono riportate le parole sottolineate, al fine di essere esplicate con delle descrizioni, onde evitare incomprensioni con gli stakeholder e con il team di sviluppo.

Termine	*Descrizione*	*Tipo*	*Sinonimi*
Cartolibreria	Luogo adibito alla vendita di libri oltre che di articoli di cartoleria	Business	Attività
Clienti privati	Clienti non possessori di partita IVA	Business	Clienti
Partita IVA	Numero identificativo di un soggetto che esercita un'attività	Business	-
Articoli	Oggetti destinati alla vendita	Business	Merce
Cancelleria	Materiale scrittorio destinato a scuole, uffici o studi	Business	-
Libreria	Luogo adibito alla vendita di libri	Business	-
Stampa	Produzione di scritti o disegni in più copie, attraverso strumenti	Business	-
Fax	Strumento collegato ad un apparecchio telefonico per l'invio di telecopie	Business	Telecopiatrice
Mail	Messaggio di posta elettronica	Business	E-mail
Plastificazione	Procedimento con il quale si avvolgono libri o quaderni al fine di aumentarne la durata	Business	-
Rilegatura	Procedimento di assemblamento di libri	Business	Legatura
Incartamento	Avvolgimento in un involucro di carta provvisorio	Business	-
Fornitori	Industriale o commerciante che fornisce i propri prodotti ad un negozio, ufficio o privato	Business	Produttore
Ingrosso	Commercio intermedio tra la vendita da parte del produttore e la compera da parte del venditore al minuto	Business	-
Merce	Oggetti destinati alla vendita di carta provvisorio	Business	Articoli

Tabella 2.1. Glossario dei termini per il contesto di riferimento (prima parte)

Termine	*Descrizione*	*Tipo*	*Sinonimi*
Magazzino	Locale adibito al deposito di merci e di materiali vari	Business	-
Carico	Aumento della quantità di un tipo di merce in magazzino	Tecnico	-
Scarico	Diminuzione della quantità di un tipo di merce in magazzino	Tecnico	-
Ricevuta	Dichiarazione comprovante l'acquisto di articoli, rilasciata dal venditore	Business	Fattura, scontrino
Fattura	Documento di vendita di articoli, contenente i dati dell'acquirente	Business	Ricevuta
Numero identificativo	Numero utilizzato per riconoscere univocamente un dato	Tecnico	ID
Agenzia delle entrate	Ente pubblico che vigila sulla corretta applicazione della legislazione tributaria	Business	-
Scontrini	Documento che attesta l'effettuato pagamento di un'acquisto	Business	Ricevute
Recapiti	Dati con i quali contattare una persona	Business	-
Mansioni	Insieme di compiti eseguiti da un dipendente a lavoro	Business	Ruolo
Dipendenti	Chi svolge la propria mansione alle dipendenze di altri	Business	Lavoratori
Retribuzioni	Compenso corrisposto in merito ad una prestazione di lavoro	Business	Salario
Registratore di cassa	Dispositivo meccanico o elettronico per la memorizzazione delle transazioni	Business	-
Sistema elettronico di fatturazione	Sistema gestito dall' agenzia delle entrate per poter gestire le fatture delle imprese	Business	-
Database	Sistema di archiviazione dei dati	Tecnico	Base di dati
Dati persistenti	Dati in grado di sopravvivere all'esecuzione del programma che li ha generati	Tecnico	Dati permanenti
Processi elaborativi	Insieme di operazioni che utilizzano dati al fine di produrre un risultato	Tecnico	-

Tabella 2.2. Glossario dei termini per il contesto di riferimento (seconda parte)

2.4 Diagramma dei sistemi

Il gestionale che si intende sviluppare è un software stand-alone, ossia capace di funzionare da solo, in maniera indipendente da altri sistemi. Tuttavia, per poterne garantire un utilizzo efficiente, dovrà essere inevitabilmente messo in comunicazione con altri sistemi esterni. Se ne ipotizzano almeno quattro, rappresentati nel diagramma di Figura 2.1.

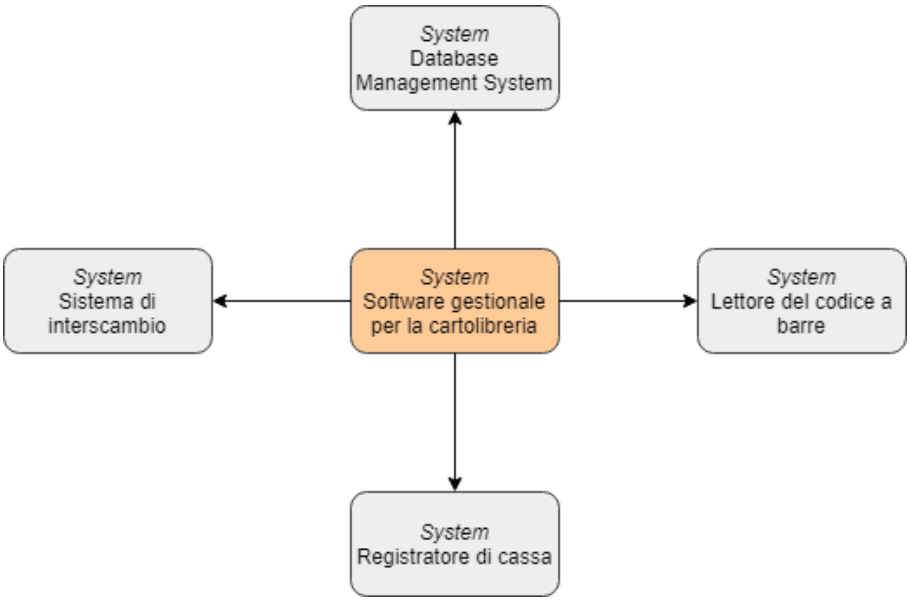

Figura 2.1. Diagramma dei sistemi per il sistema da realizzare

Il primo di questi è un DBMS (acronimo di Database Management System), ovvero un sistema che consente la creazione, manipolazione e interrogazione efficiente di database. Ovviamente, sarebbe possibile memorizzare le informazioni del software su file. Tuttavia, l'utilizzo di un database garantisce la presenza di un modello strutturato per la rappresentazione dei dati, in cui tutte le elaborazioni avvengono in maniera semplice e veloce. Si rimanda al prossimo capitolo per la documentazione della fase di progettazione della base di dati.

Gli altri sistemi del diagramma riguardano il processo di fatturazione. Al momento dell'acquisto da parte di un cliente sarà necessario fornire al software tutti i codici a barre dei prodotti da inserire nella ricevuta, prima che avvenga la sua emissione. Per evitare un meccanismo di inserimento manuale di tali codici, per ogni articolo venduto, si potrebbe far interfacciare il sistema con il lettore del codice a barre.

Inoltre, si potrebbe mettere in comunicazione il software con il registratore di cassa della cartolibreria, in modo tale che, ad ogni processo di vendita, la corrispondente ricevuta venga simultaneamente stampata dal registratore di cassa e memorizzata all'interno del database. È possibile sviluppare questo meccanismo mediante l'utilizzo del design pattern Observer.

Infine, la cartolibreria deve dichiarare tutti i pagamenti percepiti dalle vendite. A tale scopo, è bene che il software gestionale sia in comunicazione con il sistema di interscambio dell'agenzia delle entrate. Quest'ultimo è un sistema elettronico di fatturazione, in cui tutte le fatture (gli scontrini sono esclusi da questo processo) devono pervenire, in modo da essere verificate, ed in seguito inviate verso le amministrazioni pubbliche destinatarie, o verso committenti privati.

2.5 Requisiti di sistema

A questo punto, si ritiene di disporre delle informazioni sufficienti per poter esprimere in maniera completa tutti i requisiti di sistema. Prima di essere opportunamente strutturati in una tabella, questi sono stati suddivisi in requisiti funzionali e non funzionali. Rientrano nel primo gruppo i requisiti che descrivono le funzionalità che il sistema deve rispettare. Invece, fanno parte della seconda categoria i vincoli e le proprietà del sistema. In modo non completamente corretto, ma utile ai fini della comprensione, si può dire che i requisiti funzionali rispondono alla domanda su *"che cosa deve fare il sistema?"*, mentre quelli non funzionali rispondono a *"come deve essere sviluppato il sistema?"*.

2.5.1 Requisiti funzionali

A fronte dell'intervista condotta con gli stakeholder, risulta intuitivo suddividere i requisiti funzionali in aree separate, una per ogni aspetto della cartolibreria. Tali aree riguardano la gestione dei dipendenti, dei fornitori, del magazzino e degli articoli. Poi è richiesta la gestione dei clienti, che possono essere suddivisi sulla base del possesso o meno la partita IVA. Infine, vi è l'aspetto della contabilità, che comprende, quindi, gli scontrini e le fatture.

Per ciascuna di queste aree, le funzionalità che devono essere introdotte riguardano principalmente operazioni di inserimento, visualizzazione, modifica ed eliminazione.

Alla luce di queste considerazioni, sono stati dedotti i requisiti funzionali riportati riportati nelle Tabelle 2.3 e 2.4.

Requisiti	*Descrizione*
RF1 inserimento dipendente	Il sistema deve gestire l'inserimento di un nuovo dipendente.
RF2 visualizzazione dipendente	Il sistema deve gestire la visualizzazione di un dipendente.
RF3 modifica dipendente	Il sistema deve gestire la visualizzazione dipendente.
RF4 eliminazione dipendente	Il sistema deve gestire l'eliminazione di un dipendente.
RF5 inserimento fornitore	Il sistema deve gestire l'inserimento di un nuovo fornitore.
RF6 visualizzazione fornitore	Il sistema deve gestire la visualizzazione di un fornitore.
RF7 modifica fornitore	Il sistema deve gestire la modifica di un fornitore.
RF8 eliminazione fornitore	Il sistema deve gestire l'eliminazione di un fornitore.
RF9 inserimento articolo	Il sistema deve gestire l'inserimento di un nuovo articolo.
RF10 visualizzazione articolo	Il sistema deve gestire la visualizzazione di un articolo.
RF11 modifica articolo	Il sistema deve gestire la modifica di un articolo.
RF12 eliminazione articolo	Il sistema deve gestire l'eliminazione di un articolo.

Tabella 2.3. Requisiti funzionali del sistema (prima parte)

Requisiti	*Descrizione*
RF13 carico magazzino	Il sistema deve gestire il carico di merce in magazzino.
RF14 scarico magazzino	Il sistema deve gestire lo scarico di merce in magazzino.
RF15 inserimento cliente	Il sistema deve gestire l'inserimento di un nuovo cliente.
RF16 visualizzazione cliente	Il sistema deve gestire la visualizzazione di un cliente.
RF17 modifica cliente	Il sistema deve gestire la modifica di un cliente.
RF18 eliminazione cliente	Il sistema deve gestire l'eliminazione di un cliente.
RF19 inserimento clientePIVA	Il sistema deve gestire l'inserimento di un nuovo cliente dotato di partita IVA.
RF20 visualizzazione clientePIVA	Il sistema deve gestire la visualizzazione di un cliente dotato di partita IVA.
RF21 modifica clientePIVA	Il sistema deve gestire la modifica di un cliente dotato di partita IVA.
RF22 eliminazione clientePIVA	Il sistema deve gestire l'eliminazione di un cliente dotato di partita IVA.
RF23 creazione scontrino	Il sistema deve gestire la creazione di un nuovo scontrino.
RF24 visualizzazione scontrino	Il sistema deve gestire la visualizzazione di uno scontrino.
RF25 eliminazione scontrino	Il sistema deve gestire l'eliminazione di uno scontrino.
RF26 creazione fattura	Il sistema deve gestire la creazione di una nuova fattura.
RF27 visualizzazione fattura	Il sistema deve gestire la visualizzazione di una fattura.

Tabella 2.4. Requisiti funzionali del sistema (seconda parte)

2.5.2 Requisiti non funzionali

Dalle informazioni raccolte si comprende che gli stakeholder vogliono un software che, oltre a fornire tutta una serie di funzionalità per la gestione della cartolibreria, debba essere semplice da usare. A tale scopo si ritiene necessaria la presenza di un'interfaccia utente, possibilmente user friendly, mediante la quale si possono eseguire tutte le operazioni del software.

Un altro punto che è stato esplicitamente spiegato dai titolari della cartolibreria riguarda le fatture. Per l'emissione delle suddette deve essere rispettato l'articolo 21, comma 2, D.P.R. 633/1972. In particolare, su ogni fattura deve essere documentata la data di emissione, il numero di partita IVA, l'ammontare della spesa, la ragione sociale o il nominativo dell'acquirente, oltre ad un numero identificativo.

Inoltre, è necessaria la presenza di un numero identificativo anche per gli scontrini, i dipendenti, i clienti, e i fornitori memorizzati nel database del sistema. Invece, gli articoli sono univocamente identificabili grazie al loro codice a barre.

Infine, per la fase di implementazione, si vuole utilizzare un linguaggio di programmazione moderno, che sia costantemente arricchito di nuove librerie, e allo stesso tempo facile da utilizzare, in modo da minimizzare i tempi per la creazione del prodotto software. Python è un linguaggio di programmazione di alto livello, orientato agli oggetti, ed è allo stesso tempo portabile e facile da usare, oltre ad

essere tra i linguaggi di programmazione attualmente più conosciuti. Quest'ultima è una caratteristica da non trascurare per le future manutenzioni, qualora il team che si occuperà all'evoluzione del software sarà diverso da quello che lo ha implementato. Si decide quindi di utilizzare Python (in particolare la Versione 3.8.0) per l'implementazione del software gestionale.

Tutti i requisiti non funzionali sono riportati nella Tabella 2.5.

Requisiti	*Descrizione*
RNF1 interfaccia grafica	Il sistema deve disporre di un interfaccia grafica semplice da utilizzare.
RNF2 gestione delle fatture	Le fatture dovranno essere rappresentate ai sensi dell'articolo 21, comma 2, D.P.R. 633/1972.
RNF3 id	Il sistema dovrà impiegare identificativi univoci per la rappresentazione delle informazioni.
RNF4 implementazione	Il sistema dovrà essere realizzato mediante l'utilizzo della tecnologia Python 3.

Tabella 2.5. Requisiti non funzionali del sistema

2.6 Convalida dei requisiti

È importante dare molta attenzione alla fase di specifica dei requisiti, poiché essa rappresenta il punto di partenza di tutto il progetto. Ogni errore che viene fatto in questa fase si ripercuote sull'intero processo software. Ovviamente, vi è sempre la possibilità di porre delle modifiche ai requisiti. Però, quanto più tardi esse vengono apportate, tanto maggiori sono i costi da dover sostenere, sia in termini temporali che economici.

Inoltre, seguendo la filosofia dello sviluppo "plan-driven", non si può passare alla fase successiva del processo software senza la certezza di aver terminato quella corrente. Per questo motivo è necessaria la convalida dei requisiti, ovvero una serie di controlli che mirano a verificare che i requisiti estrapolati definiscono il sistema realmente voluto dagli stakeholder.

In questo caso specifico, grazie alla convalida dei requisiti, si può affermare che questi ultimi sono:

- *validi*, in quanto riflettono le reali esigenze degli stakeholder;
- *consistenti*, dal momento che non ci sono requisiti in conflitto con altri;
- *completi*, poiché sono state definite tutte le funzioni richieste dal sistema, senza trascurarne nessuna;
- *reali*, cioè tutti i requisiti dedotti dai titolari della cartolibreria sono implementabili in tempi e costi ragionevoli.

3

Progettazione del database

In questo capitolo si vuole descrivere l'insieme delle procedure adoperate per la progettazione di un sistema moderno di persistenza dei dati, da dover utilizzare come supporto per il software gestionale che si intende sviluppare. Nello specifico, attraverso la progettazione concettuale, viene prodotto uno schema E-R iniziale, il quale è una rappresentazione grezza degli elementi del database. Pertanto, sarà necessario, in seguito a determinate operazioni di raffinamento, tipiche della fase di progettazione logica, produrre un modello logico, semplice ed efficiente, da poter utilizzare in fase di progettazione fisica per poter effettivamente codificare il database in SQL.

3.1 Progettazione concettuale

La progettazione concettuale è quella fase della progettazione di un database che consiste nell'identificazione e rappresentazione dei dati, attraverso l'utilizzo di schemi opportuni, in modo indipendente dal sistema utilizzato. Si tratta di un livello di progettazione più vicino all'uomo rispetto alla macchina. Quindi, deve essere realizzato con strumenti e linguaggi comprensibili a tutti, non solo agli specialisti.

Nella fase preliminare di progettazione, la rappresentazione dei dati avviene mediante l'utilizzo di un modello concettuale, il quale è un insieme di classi e di associazioni tra classi che descrivono gli elementi del mondo reale. Questo modello ha l'obiettivo di esprimere il significato dei termini utilizzati dagli esperti del dominio, e di trovare le giuste relazioni tra concetti differenti.

Il modello concettuale più utilizzato è il modello E-R (cioè il modello Entità-Relazione), formato dai seguenti costrutti:

- Le *entità*, utilizzate per rappresentare classi di oggetti che hanno proprietà comuni ed esistenza autonoma. Le occorrenze, o istanze, sono oggetti che esistono indipendentemente dalle proprietà dell'entità a cui appartengono. In uno schema, ogni entità ha un nome che la identifica univocamente e viene rappresentata graficamente tramite un rettangolo, con il nome dell'entità al suo interno.
- Le *relazioni*, anche dette *associazioni*, rappresentano un legame tra due o più entità. Ognuna di esse viene rappresentata graficamente da un rombo contenente il nome dell'associazione. Così come per gli oggetti delle entità, anche quelli che appartengono a una relazione sono denominati occorrenze, o istanze.

- Gli *attributi* sono proprietà che accomunano tutte le istanze di un'entità o di una relazione. Ogni attributo è rappresentato con un'ellisse al cui interno viene specificato il suo nome. Esistono anche gli attributi composti, che sono costituiti da più attributi semplici.
- Le *cardinalità* vengono specificate per ciascuna entità che partecipa a una relazione, e indicano quante volte un'occorrenza di una di queste entità può essere legata ad occorrenze delle altre entità coinvolte nell'associazione. Anche un attributo può avere una cardinalità rispetto all'entità alla quale si riferisce. In tal caso si parla di *attributo multivalore*.
- Gli *identificatori* costituiscono un sottoinsieme degli attributi di un'entità che identificano in maniera univoca ogni occorrenza dell'entità stessa.
- Le *generalizzazioni* rappresentano dei legami logici esistenti tra un'entità genitore e più entità figlie, in modo tale che queste ultime siano una *specializzazione* del genitore, ereditandone, allo stesso tempo, tutti i costrutti che vi fanno riferimento. Da questo punto in avanti, ci si riferirà unicamente alle cosiddette *generalizzazioni esclusive*, nelle quali ogni occorrenza dell'entità genitore è, a sua volta, occorrenza di al più una delle entità figlie.

Il primo passo da compiere, nella progettazione concettuale del database della cartolibreria, consiste nell'identificazione di una strategia che possa essere ritenuta la più adatta per questo progetto specifico. L'applicazione di tale strategia deve essere finalizzata alla costruzione di uno schema E-R che rispetti le caratteristiche di correttezza, completezza, leggibilità e minimalità.

Qualora il modello E-R prodotto non dovesse rispettare tali caratteristiche, sarà necessario effettuare delle modifiche, in modo tale da avere un punto di partenza che sia il più semplice possibile da gestire nella fase di progettazione logica.

3.1.1 Sviluppo dello scheletro di partenza

Si intende utilizzare una strategia di progettazione *top-down* (a discapito delle strategie di sviluppo *bottom-up* e *inside-out*).

Seguendo un approccio top-down per la costruzione del modello concettuale, il processo di progettazione parte dall'identificazione di concetti molto astratti e generici, che vengono poi arricchiti e approfonditi in raffinamenti successivi. In questo modo si ha un costante controllo della complessità dei costrutti.

Tipicamente, quando si adotta una strategia top-down, il problema principale da dover fronteggiare è quello di avere una visione generale del sistema da rappresentare.

Nel caso specifico della cartolibreria, questo problema viene ampiamente superato poiché, già in fase di deduzione dei requisiti, sono state identificate le otto aree principali del software gestionale: dipendenti, fornitori, articoli, magazzino, clienti privati, clienti con partita IVA, scontrini e fatture. Queste stesse otto aree, a loro volta, possono costituire il punto di partenza della progettazione del database. Infatti, per ognuna di esse, sarà necessario memorizzarne e gestirne le informazioni.

Si identificano, quindi, otto entità principali, riportate in Figura 3.1.

Da tale schema si nota che l'entità *Cliente privato* e *Cliente con partita IVA* possono essere specializzazioni di un'unica entità *Cliente*.

Figura 3.1. Le otto entità di partenza per lo sviluppo del modello concettuale

Inoltre, anche le entità *Fattura* e *Scontrino* sono generalizzabili in un'entità *Ricevuta*. Però, in questo caso, il discorso è più complesso, poiché gli stakeholder, nell'intervista, hanno richiesto la memorizzazione delle fatture rilasciate dalla cartolibreria agli acquirenti, ma anche la memorizzazione delle fatture rilasciate dai fornitori verso la cartolibreria, al momento del rifornimento della merce. Per questo motivo, si introduce un'altra entità, denominata *Fattura di carico*.

Applicando le suddette generalizzazioni, si ottiene lo schema in Figura 3.2.

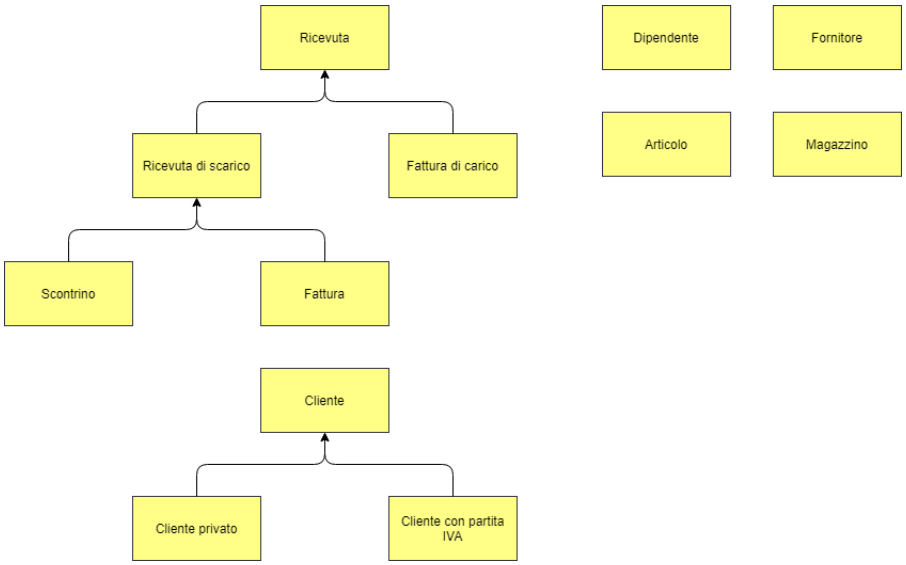

Figura 3.2. Le otto entità di partenza per lo sviluppo del modello concettuale, con l'aggiunta delle generalizzazioni

A questo punto, è necessario trovare delle associazioni che leghino quanto più

possibile le entità di partenza.

Ogni ricevuta di scarico viene erogata dalla cartolibreria, in seguito ad un *Pagamento* da parte di un cliente. Invece, una fattura di carico viene rilasciata alla cartolibreria dopo il *Rifornimento* di merce da parte di un fornitore.

Inoltre, ogni volta che viene emessa una ricevuta, dovrà essere automaticamente eseguita un'azione di *Carico* o di *Scarico* del magazzino. Si ricorda che queste due operazioni corrispondono alla diminuzione e all'aumento della quantità di merce in magazzino.

In aggiunta, il magazzino è associabile ad un determinato fornitore per mezzo dell'*Articolo* che viene acquisito dalla cartolibreria. In questo modo, si trasforma l'entità *Articolo* in un'associazione tra le entità *Fornitore* e *Magazzino*.

Infine, non è stato possibile identificare alcuna associazione che leghi *Dipendente* con qualsiasi altra entità.

Alla luce di queste relazioni, si giunge allo schema in Figura 3.3.

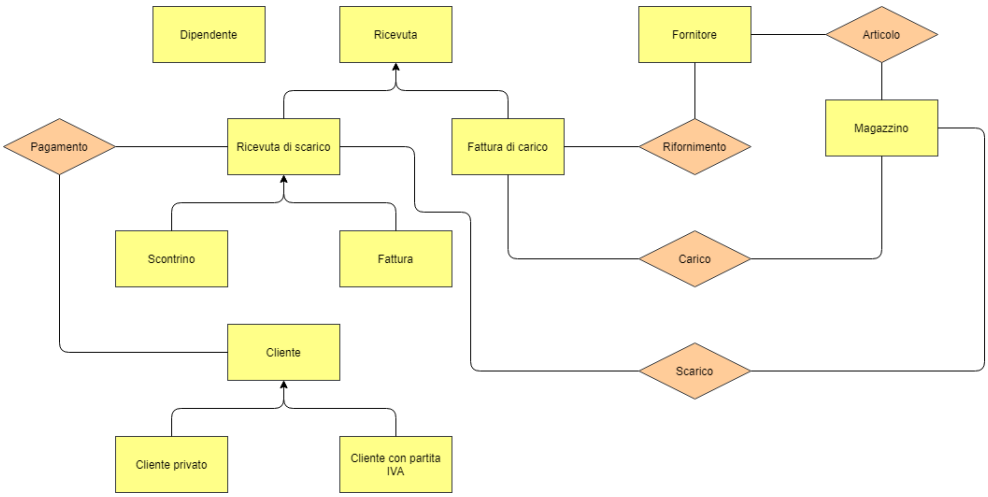

Figura 3.3. Modello E-R completo di entità e relazioni

Per completare il modello E-R, non resta che aggiungere le rispettive cardinalità e gli attributi, ottenendo, come risultato finale, quello in Figura 3.4.

In particolare, si noti che, per la rappresentazione degli identificatori, sono stati utilizzati dei cerchi neri, mentre per gli altri attributi sono stati usati dei cerchi bianchi. Inoltre, le entità *Cliente* e *Fornitore* posseggono due attributi multivalore, i quali non possono essere rappresentati all'interno di un database relazionale. Per questo motivo, sarà necessaria una ristrutturazione di tali attributi nella fase di progettazione logica.

3.1.2 Analisi di qualità del modello E-R

Dopo aver realizzato lo schema concettuale del database, è necessario effettuarne una verifica della qualità. Solo in questo modo si può avere la garanzia, nei paragrafi

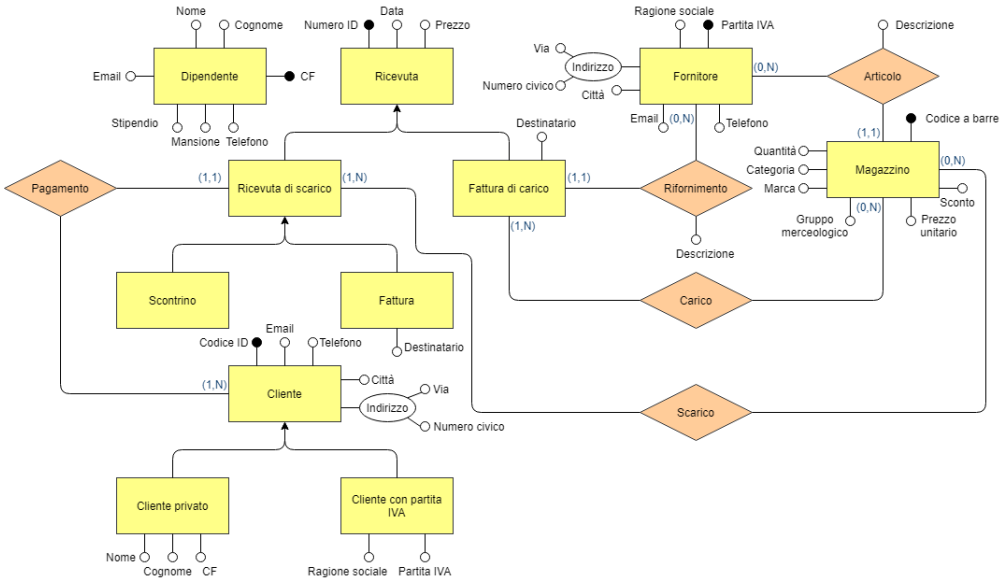

Figura 3.4. Modello concettuale finale

successivi, di partire da un modello semplice da tradurre nello schema logico. A tale scopo, si verifica che il modello ottenuto rispetta le seguenti proprietà:

- *Correttezza*, visto che la realtà di interesse è rappresentata mediante uno schema che utilizza nel modo corretto, sia a livello sintattico che semantico, i costrutti del modello E-R.
- *Completezza*, perché sono stati rappresentati tutti i dati degli aspetti descritti in fase di specifica del software, senza nessuna eccezione.
- *Leggibilità*, infatti lo schema concettuale è stato rappresentato nel modo più ordinato possibile. Inoltre, i nomi dei costrutti utilizzati sono stati presi direttamente dal lessico degli stakeholder, in modo da evitare eventuali incomprensioni.
- *Minimalità*, poiché non sono presenti cicli o ridondanze nel modello concettuale finale.

Questa breve analisi di qualità, seppur indicativa, identifica uno schema ben tracciato. Dunque, risulta essere una buona base di partenza per il prossimo passo della progettazione.

3.1.3 Dizionario dei dati

Prima di proseguire con la progettazione del database, si ritiene necessario descrivere approfonditamente i costrutti utilizzati nella creazione del modello concettuale, in modo da potervi far riferimento senza preoccuparsi di eventuali incomprensioni, o interpretazioni errate, da parte dei lettori della documentazione.

In Tabella 3.1 sono contenute le descrizioni delle entità dello schema E-R; mentre le associazioni si trovano in Tabella 3.2.

Nome	*Descrizione*	*Attributi*	*Identificatore*
Dipendente	Lavoratore retribuito della cartolibreria	Mansione (stringa), Nome (stringa), Cognome (stringa), Email (stringa), Telefono (numerico), CF (stringa), Stipendio (numerico),	CF (stringa)
Ricevuta	Dichiarazione di un pagamento avvenuto	Numero ID (numerico), Data (data), Prezzo (numerico)	Numero ID (numerico)
Ricevuta di scarico	Attestazione di un acquisto da parte di un cliente	-	-
Fattura di carico	Attestazione di un acquisto di merce presso un fornitore	Destinatario (stringa)	-
Scontrino	Attestazione di un acquisto da parte di un cliente che può o meno possedere una partita IVA	-	-
Fattura	Attestazione di un acquisto da parte di un cliente possessore di partita IVA	Destinatario (stringa)	-
Cliente	Persona che ha effettuato almeno un acquisto in cartolibreria	Codice ID (numerico), Email (stringa), Telefono (numerico), Città (stringa), Indirizzo (indirizzo)	Codice ID (numerico)
Cliente privato	Cliente non dotato di una partita IVA	Nome (stringa), Cognome (stringa) CF (stringa)	-
Cliente con partita IVA	Cliente dotato di una partita IVA	Ragione sociale (stringa), Partita IVA (numerico)	-
Fornitore	Commerciante o industriale che vende all'ingrosso	Partita IVA (numerico), Email (stringa) Ragione sociale (stringa), Città (stringa), Indirizzo (indirizzo), Telefono (numerico)	Partita IVA (numerico)
Magazzino	Articoli adibiti alla vendita	Codice a barre (numerico), Gruppo merceologico (stringa), Marca (stringa), Categoria (stringa), Prezzo (numerico), Quantità (numerico)	Codice a barre (numerico)

Tabella 3.1. Dizionario dei dati relativo alle entità

Nome	*Descrizione*	*Attributi*	*Identificatore*
Pagamento	Associa ogni ricevuta di scarico al cliente che l'ha effettuata	Ricevuta di scarico (1,1) Cliente (1,N)	-
Articolo	Associa un fornitore ad un elemento presente in magazzino	Fornitore (0,N) Magazzino (1,1)	Descrizione (stringa)
Rifornimento	Associa un fornitore ad una fattura per l'acquisto di merce	Fornitore (0,N) Fattura di carico (1,1)	Descrizione (stringa)
Carico	Associa una fattura agli elementi caricati in magazzino	Fattura di carico (1,N) Magazzino (0,N)	-
Scarico	Associa una ricevuta agli elementi scaricati dal magazzino	Ricevuta di carico (1,N) Magazzino (0,N)	-

Tabella 3.2. Dizionario dei dati relativo alle associazioni

3.2 Progettazione logica

La progettazione logica del database consiste nella traduzione dello schema concettuale in quello logico, il quale ha l'obbiettivo di rappresentare gli stessi dati in maniera corretta ed efficiente. A tale scopo, è necessario effettuare l'analisi delle prestazioni sotto entrambi i punti di vista, temporale e spaziale.

Lo schema logico, a differenza di quello concettuale, rispecchia il modello dei dati scelto. In questo caso si vuole utilizzare un modello relazionale, il cui assunto fondamentale è che i dati vengano rappresentati all'interno di *relazioni*, e manipolati con gli operatori dell'algebra relazionale o del calcolo relazionale.

In ogni caso, lo schema logico risultante è indipendente dallo specifico DBMS (Database Management System) che verrà scelto al termine della progettazione logica.

3.2.1 Tavola dei volumi

Al fine di conoscere, ed eventualmente migliorare, le prestazioni della base di dati che si sta progettando, è necessario misurarne i parametri di tempo e spazio. Per tempo si intende il numero delle occorrenze di entità, e associazioni, visitate durante l'esecuzione di una determinata operazione (o requisito).

Misurare lo spazio significa, invece, effettuare una previsione del numero di occorrenze (medio o massimo) previste per ogni entità e associazione.

Quest'ultimo parametro viene misurato attraverso la tavola dei volumi.

In Tabella 3.3 sono riportati i volumi medi attesi, per le entità e le associazioni del database che si sta progettando, calcolati su dieci anni di utilizzo. Ovviamente, i valori inseriti, pur essendo coerenti tra loro, sono puramente indicativi.

3.2.2 Tavola delle operazioni

Quando viene svolta una determinata operazione, o requisito, bisogna considerare, oltre al numero di costrutti che vengono percorsi, anche la frequenza con cui questa si verifica. Infatti, quando si calcola l'efficienza temporale di un'operazione, bisogna tener conto di tutte le occorrenze visitate, dette anche *accessi*, e moltiplicare tali valori per la frequenza con la quale l'operazione in esame viene svolta.

I valori medi stimati per le frequenze relative ad ogni operazione (o requisito) del progetto in esame sono mostrate nella tavola delle operazioni, in Tabella 3.4. Anche in questo caso, così come per la tavola dei volumi, sono stati utilizzati dei valori stimati, in quanto è impossibile conoscere le frequenze dei requisiti con estrema precisione, senza aver analizzato il modo di lavorare degli stakeholder per un lasso di tempo molto lungo (dell'ordine degli anni).

3.2.3 Analisi delle ridondanze

Nella fase di progettazione concettuale, si è voluto realizzare uno schema E-R che rispetti, tra le altre proprietà, la minimalità, ovvero l'assenza di ridondanze.

Concetto	*Tipo*	*Volume*
Ricevuta	E	72000 (69600 + 2400)
Fattura di carico	E	2400
Ricevuta di scarico	E	69600 (46400 + 23200)
Scontrino	E	46400
Fattura	E	23200
Pagamento	R	69600
Cliente	E	4640 (3000 + 1640)
Cliente privato	E	3000
Cliente con partita IVA	E	1640
Dipendente	E	10
Rifornimento	R	2400
Fornitore	E	50
Articolo	R	3000
Magazzino	E	3000
Carico	R	2400
Scarico	R	69600

Tabella 3.3. La tavola dei volumi del database della cartolibreria

Una ridondanza, in un modello E-R, è definita come un'informazione significativa, che è, allo stesso tempo, derivabile da altre. La ripetizione di una medesima informazione può portare ad un'inconsistenza dei dati, qualora ci fosse un'anomalia di aggiornamento nel database. In altre parole, se una modifica di un dato non dovesse avvenire uniformemente per tutte le sue ridondanze, non sarebbe possibile identificare quale, tra queste, rappresenta l'informazione corretta. Nel caso peggiore, due dati ridondanti potrebbero essere in totale contrasto tra loro.

Inoltre, anche se può risultare un problema minore, bisogna considerare che l'impiego di dati ridondanti comporta un maggiore utilizzo della memoria.

Tuttavia, nonostante gli svantaggi, in alcuni casi la presenza di ridondanze può essere conveniente.

Nel progetto di riferimento, partendo dalla visualizzazione di un articolo in magazzino, non è possibile conoscere il fornitore di origine senza accedere all'entità specifica. Allora, per velocizzare questo processo, si può ipotizzare l'introduzione volontaria di un attributo ridondante *Fornitore* nell'entità *Magazzino*. Prima di questo, però, bisogna verificare se la presenza di tale ridondanza sia effettivamente conveniente dal punto di vista computazionale.

A tale scopo, si fa affidamento alle tavole degli accessi, le quali misurano il numero di occorrenze visitate durante lo svolgimento di un determinato requisito.

L'introduzione della ridondanza in questione interessa quattro operazioni: RF5

Requisito	*Frequenza*
RF1 inserimento dipendente	1 volta all'annoa
RF2 visualizzazione dipendente	1 volta al mese
RF3 modifica dipendente	1 volta all'anno
RF4 eliminazione dipendente	1 volta all'anno
RF5 inserimento fornitore	5 volte all'anno
RF6 visualizzazione fornitore	1 volta al mese
RF7 modifica fornitore	1 volta all'anno
RF8 eliminazione fornitore	1 volta all'anno
RF9 inserimento articolo	1 volta al mese
RF10 visualizzazione articolo	1 volta a settimana
RF11 modifica articolo	1 volta al mese
RF12 eliminazione articolo	1 volta al mese
RF13 carico magazzino	240 volte l'anno
RF14 scarico magazzino	580 volte l'anno
RF15 inserimento cliente	25 volte al mese
RF16 visualizzazione cliente	1500 volte l'anno
RF17 modifica cliente	1 volte al mese
RF18 eliminazione cliente	1 volta l'anno
RF19 inserimento clientePIVA	164 volte l'anno
RF20 visualizzazione clientePIVA	820 volte l'anno
RF21 modifica clientePIVA	1 volta al mese
RF22 eliminazione clientePIVA	1 volta l'anno
RF23 creazione scontrino	4640 volte l'anno
RF24 visualizzazione scontrino	1 volta l'anno
RF25 eliminazione scontrino	1 volta a settimana
RF26 creazione fattura	2320 volte l'anno
RF27 visualizzazione fattura	2320 volte l'anno

Tabella 3.4. La tavola delle operazioni del database della cartolibreria

inserimento fornitore, RF6 visualizzazione fornitore, RF7 modifica fornitore e RF8 eliminazione fornitore. Quindi, sarà necessario calcolare i costi dei suddetti requisiti in presenza e in assenza dell'attributo *Fornitore* nell'entità *Magazzino*, al fine di verificare se tale derivazione convenga realmente.

Nelle Tabelle 3.5, 3.7, 3.9 e 3.11 sono riportate le quattro tavole degli accessi dei

suddetti requisiti, in assenza della ridondanza. Invece, nelle Tabelle 3.6, 3.8, 3.10 e 3.12 sono riportate le tavole degli accessi, delle medesime operazioni, nell'ipotesi in cui tale ridondanza sia presente. In ognuna di queste tabelle si può notare che, per i valori degli accessi dei singoli costrutti, è stato fatto riferimento alla tavola dei volumi precedentemente prodotta. Inoltre, si tenga in conto che, per poter effettuare delle misurazioni corrette, prima di calcolare i costi, bisogna modulare ogni accesso con il tipo di operazione eseguita. Nello specifico, le uniche due operazioni che caratterizzano un database sono la scrittura (S) e la lettura (L), di cui la prima è più complessa della seconda. Infatti, si ritiene ogni operazione di scrittura pari a due volte un'operazione di lettura.

Si consideri come requisito esemplificativo quello dell'inserimento di un nuovo fornitore (RF5). Nella Tabella 3.5 si verifica che, in assenza dell'attributo ridondante *Fornitore* nell'entità *Magazzino*, è necessaria un'operazione di scrittura sull'entità *Fornitore*, e ulteriori 60 operazioni di scrittura sull'associazione *Articolo*. Quest'ultimo numero deriva dal fatto che, dalla tavola dei volumi, in Tabella 3.3, il volume medio atteso dei fornitori memorizzati, su un arco temporale di 10 anni, è di 50 unità, mentre il volume di articoli in magazzino è stimato a 3000 unità. Quindi, ci si aspetta che ad ogni fornitore corrispondano mediamente $3000/50 = 60$ articoli in magazzino. Se ogni scrittura vale come due operazioni di lettura, allora il requisito RF5, in assenza della ridondanza, costa esattamente 60 S * 2 + 1 S * 2 = 122 L ad ogni sua esecuzione. Però, come è già stato spiegato in precedenza, bisogna anche considerare la frequenza con cui viene eseguito un determinato requisito. In effetti, dalla tavola delle operazioni, in Tabella 3.4, si può verificare che il requisito RF5 avviene mediamente 5 volte all'anno. Quindi, il costo totale per l'inserimento di un nuovo fornitore nel database è esattamente di 122 L * 5 = 610 L/anno.

Invece, applicando lo stesso procedimento, in presenza della ridondanza, dalla Tabella 3.5, si nota che, accanto alle due operazioni sopraccitate di scrittura sull'entità *Fornitore*, e di 60 altre scritture sulla relazione *Articolo*, si aggiungono 3000 letture su *Articolo*, 3000 letture su *Magazzino* e 60 scritture su *Magazzino*. Questo significa che il singolo inserimento di un nuovo fornitore costa 1 S * 2 + 60 S * 2 + 3000 L + 3000 L + 60 S * 2 = 6242 L, che moltiplicate per la frequenza diventano 6242 L * 5 = 31210 L/anno.

Se l'analisi fosse limitata unicamente all'operazione RF5, si arriverebbe alla conclusione che la presenza della derivazione sia altamente svantaggiosa. Infatti, in tal caso, si otterrebbe un costo complessivo di 31210 operazioni di lettura all'anno, che sono molte più di 610, cioè quelle che si avrebbero nel caso in cui non venisse impiegata la ridondanza.

Nonostante ciò, nel calcolo dei costi totali, bisogna considerare anche quelli relativi alle operazioni RF6, RF7 e RF8.

In Tabella 3.13 sono riportati tutti i costi dei quattro requisiti analizzati, nel caso di assenza della ridondanza. In Tabella 3.14 vengono, invece, mostrati gli stessi costi, ma in sua presenza.

Dalle suddette tabelle, si fa notare che, limitando l'analisi della ridondanza alle operazioni RF5, RF7 e RF8, sarebbe di gran lunga più conveniente la sua assenza. Infatti, in tal caso, si avrebbero, in assenza della derivazione, 3834 L/anno totali, contro 40734 L/anno nel caso in cui questa è presente.

Tuttavia, l'operazione RF6, cioè la visualizzazione del fornitore, è decisamente

più frequente delle altre tre. Difatti, come si può notare dalle sopraccitate tabelle, in assenza della ridondanza, si ottengono complessivamente 75846 L/anno, mentre in sua presenza se ne hanno 40746, ovvero il 46% in meno.

Alla luce di questi calcoli, si conclude l'analisi verificando che la presenza dell'attributo ridondante *Fornitore* nell'entità *Magazzino* è, in effetti, conveniente. Pertanto, si decide di introdurlo all'interno del modello E-R.

In Figura 3.5 è mostrato lo schema E-R con l'aggiunta della ridondanza.

Concetto	*Costrutto*	*Accessi*	*Tipo*
Fornitore	E	1	S
Articolo	R	60	S

Tabella 3.5. Tavola degli accessi di RF5 in assenza della ridondanza

Concetto	*Costrutto*	*Accessi*	*Tipo*
Fornitore	E	1	S
Articolo	R	60	S
Articolo	R	3000	L
Magazzino	E	3000	L
Magazzino	E	60	S

Tabella 3.6. Tavola degli accessi di RF5 in presenza della ridondanza

Concetto	*Costrutto*	*Accessi*	*Tipo*
Magazzino	E	1	L
Articolo	R	3000	L
Fornitore	E	3000	L

Tabella 3.7. Tavola degli accessi di RF6 in assenza della ridondanza

3.2.4 Ristrutturazione dello schema concettuale

Per poter passare alla fase di progettazione fisica, e quindi all'implementazione vera e propria della base di dati, è necessario produrre uno schema logico che

Concetto	*Costrutto*	*Accessi*	*Tipo*
Magazzino	E	1	L

Tabella 3.8. Tavola degli accessi di RF6 in presenza della ridondanza

Concetto	*Costrutto*	*Accessi*	*Tipo*
Fornitore	E	50	L
Fornitore	E	1	S

Tabella 3.9. Tavola degli accessi di RF7 in assenza della ridondanza

Concetto	*Costrutto*	*Accessi*	*Tipo*
Fornitore	E	50	L
Fornitore	E	1	S
Articolo	R	60	L
Magazzino	E	3000	L
Magazzino	E	60	S

Tabella 3.10. Tavola degli accessi di RF7 in presenza della ridondanza

Concetto	*Costrutto*	*Accessi*	*Tipo*
Fornitore	E	50	L
Fornitore	E	1	S
Articolo	R	3000	L
Articolo	R	60	S

Tabella 3.11. Tavola degli accessi di RF8 in assenza della ridondanza

possa rappresentare, il più fedelmente possibile, a livello di astrazione, il modello relazionale.

Per questa ragione, è inevitabile passare per la ristrutturazione dello schema concettuale, la quale consiste in una serie di operazioni volte al raffinamento del modello E-R utilizzato. Nonostante l'analisi delle ridondanze stessa fosse finalizzata al miglioramento della qualità del suddetto modello, ci sono altre procedure da utilizzare per tradurlo in uno schema logico finale.

Solitamente, nei progetti di grande dimensione, sono richiesti dei partizionamenti e accorpamenti tra costrutti, al fine di migliorare l'efficienza delle operazioni. Ad esempio, se un determinato requisito prevede l'accesso a due istanze separate di due costrutti differenti, se ne può valutare l'accorpamento in un unico diagramma. D'altro canto, se due diverse operazioni prevedono di accedere ad istanze diverse di

Concetto	*Costrutto*	*Accessi*	*Tipo*
Fornitore	E	50	L
Fornitore	E	1	S
Articolo	R	3000	L
Articolo	R	60	S
Magazzino	E	3000	L
Magazzino	E	60	S

Tabella 3.12. Tavola degli accessi di RF8 in presenza della ridondanza

Operazione	*Costo*	*Frequenza (annuale)*	*Totale*
RF5 inserimento fornitore	122 L	5 volte l'anno	610 L/anno
RF6 visualizzazione fornitore	6001 L	12 volte l'anno	72012 L/anno
RF7 modifica fornitore	52 L	1 volta l'anno	52 L/anno
RF8 elimina fornitore	3172 L	1 volta l'anno	3172 L/anno

Tabella 3.13. Calcolo dei costi in assenza della ridondanza

Operazione	*Costo*	*Frequenza (annuale)*	*Totale*
RF5 inserimento fornitore	6242 L	5 volte l'anno	31210 L/anno
RF6 visualizzazione fornitore	1 L	12 volte l'anno	12 L/anno
RF7 modifica fornitore	3232 L	1 volta l'anno	3232 L/anno
RF8 elimina fornitore	6292 L	1 volta l'anno	6292 L/anno

Tabella 3.14. Calcolo dei costi in presenza della ridondanza

una stessa entità o associazione, si può pensare ad un suo partizionamento in due costrutti a sé stanti.

Nel progetto trattato non è presente nessuna necessità di accorpamento o partizionamento poiché i requisiti funzionali di sistema sono stati suddivisi in macroaree, che poi sono state utilizzate come scheletro di partenza per lo schema E-R prodotto.

Invece, nel progetto di riferimento, vanno ristrutturate le gerarchie, che sono dei costrutti non rappresentabili in un modello relazionale. A seconda dei casi, si hanno tre possibili metodi previsti per il trattamento di una gerarchia:

- L'accorpamento delle entità figlie nel genitore conviene se gli accessi a tali costrutti sono contestuali, cioè se nell'esecuzione di una determinata operazione è necessario l'accesso a tutte le entità figlie.
- L'accorpamento dell'entità genitore nelle figlie conviene se gli accessi a queste ultime sono distinti, cioè se nessuna operazione, tra quelle definite nel database,

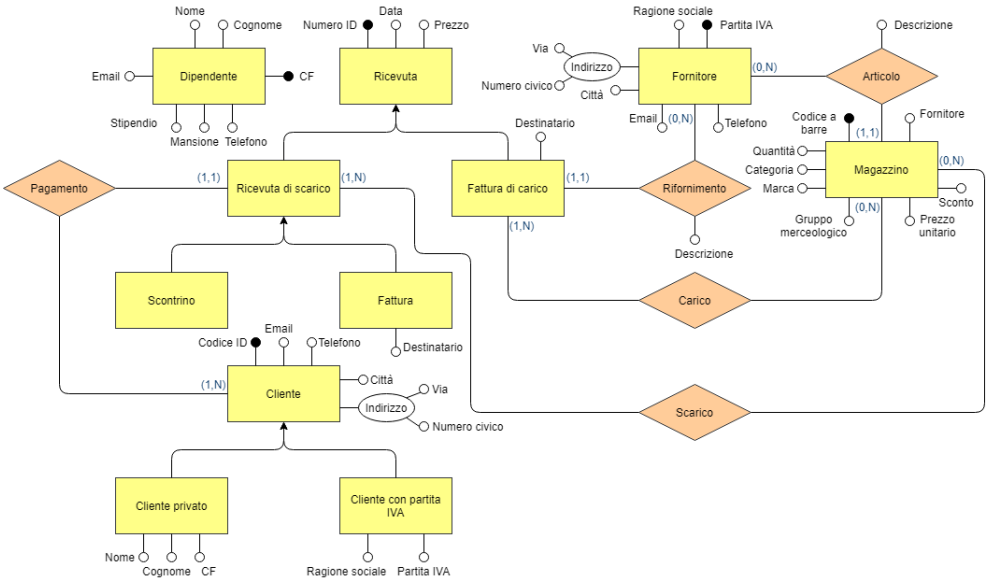

Figura 3.5. Modello concettuale finale con l'aggiunta della ridondanza

coinvolge le istanze del genitore ma non quelle delle figlie.

- La sostituzione della generalizzazione con delle relazioni conviene se gli accessi alle entità figlie sono separati dagli accessi al genitore, cioè qualora non ci si dovesse trovare in nessuno dei due casi precedenti.

Dalla Figura 3.5 si nota subito la presenza di tre generalizzazioni.

La prima di queste ha come entità genitore *Ricevuta*, la quale però non viene utilizzata in nessun requisito tra quelli dedotti, né presenta associazioni che la leghino con altre relazioni. Quindi, è il caso perfetto per attuare un accorpamento dell'entità genitore nelle figlie (vedi Figura 3.6).

La seconda generalizzazione da dover trattare è quella relativa all'entità genitore *Ricevuta di scarico*. In questo caso, si è presa in considerazione la tabella delle operazioni, dove si può notare la presenza di requisiti specifici per gli scontrini e per le fatture. Nessuna operazione, invece, prevede il coinvolgimento dell'entità genitore. Sulla base di questa verifica, si è deciso accorpare l'entità genitore nelle figlie. Però, in seguito all'accorpamento, è necessario utilizzare due associazioni, anziché una sola, che leghi l'entità *Cliente* alle due entità *Scontrino* e *Fattura*. Si ottiene, quindi, il risultato in Figura 3.7.

Si decide di accorpare l'entità *Cliente* nelle sue due specializzazioni *Cliente privato* e *Cliente con partita IVA* per la stessa ragione del caso precedente, ottenendo il risultato in Figura 3.8.

Oltre alle generalizzazioni, anche gli attributi composti rappresentano dei costrutti che non si possono rappresentare all'interno di un database relazionale.

L'unico modo con cui è possibile ristrutturare un attributo composto è quello di scinderlo nei suoi singoli elementi. Nel progetto di riferimento, i tre attributi

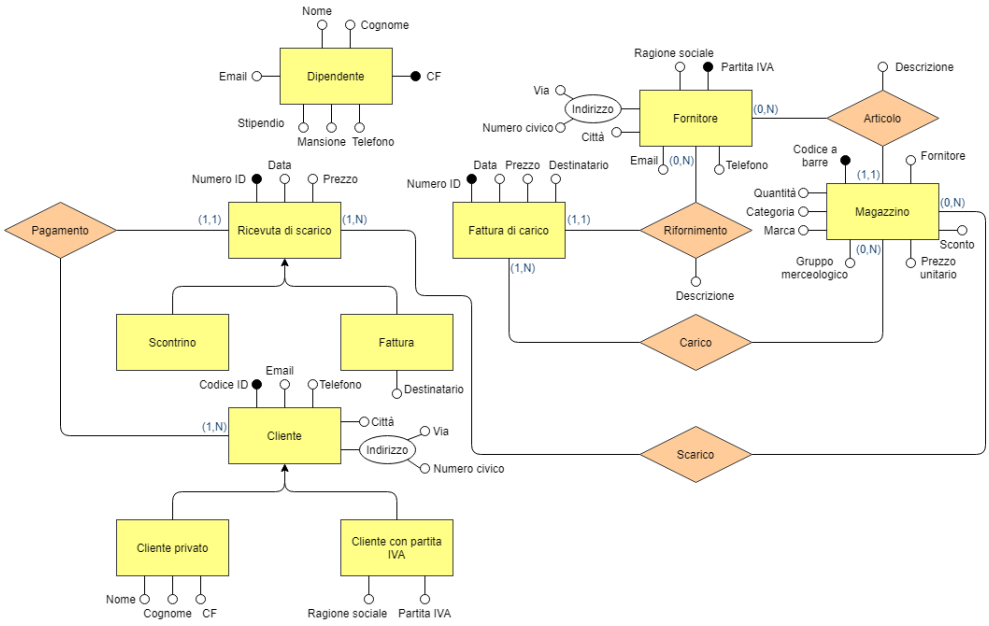

Figura 3.6. Modello concettuale ristrutturato della generalizzazione *Ricevuta*

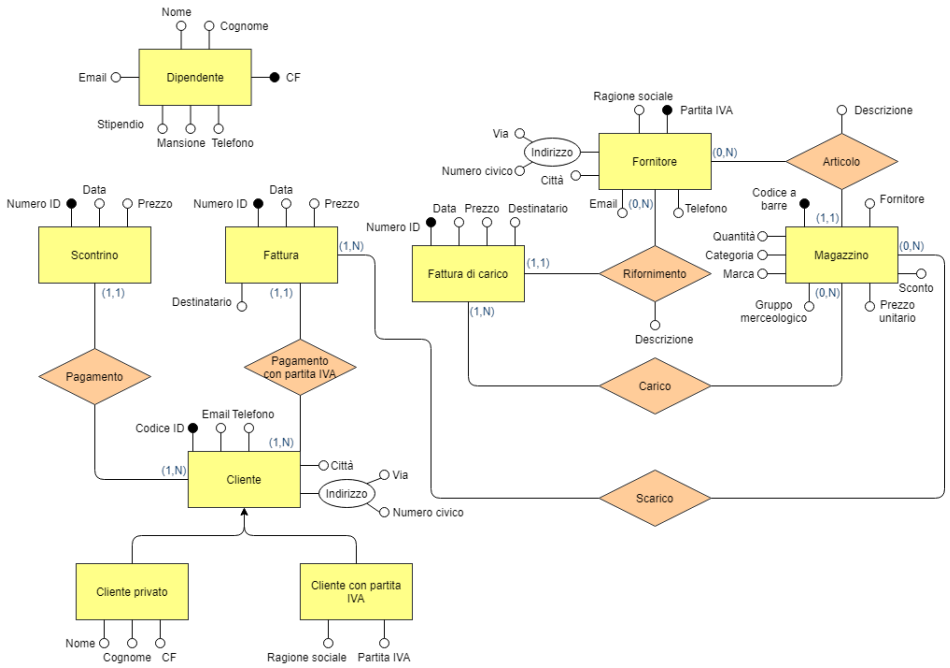

Figura 3.7. Modello concettuale ristrutturato della generalizzazione *Ricevuta di scarico*

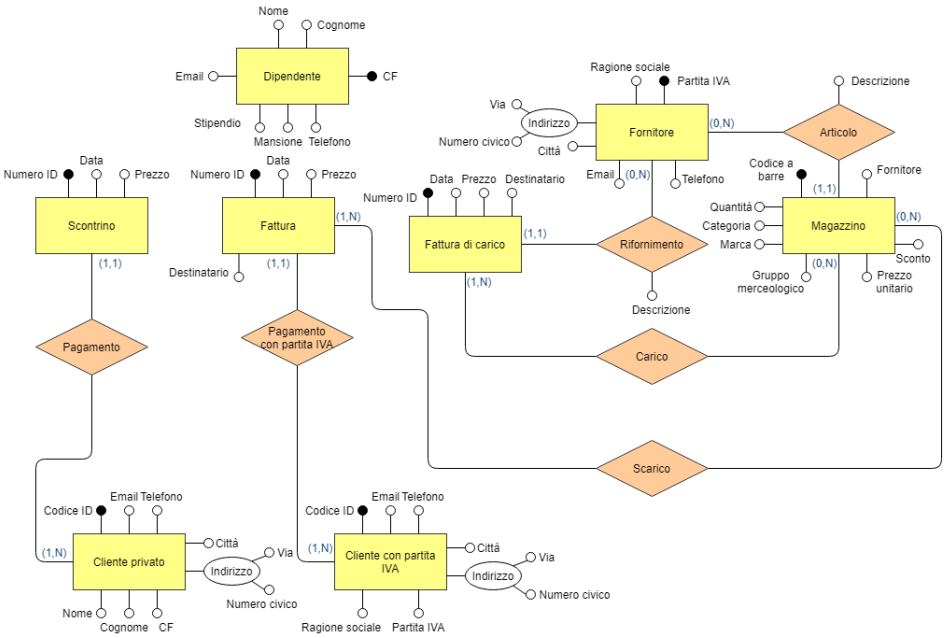

Figura 3.8. Modello concettuale ristrutturato della generalizzazione *Cliente*

composti, denominati *Indirizzo*, si possono scindere negli attributi *Via* e *Numero civico*.

Alla luce di tutte queste modifiche, si giunge allo schema logico finale, rappresentato in Figura 3.9.

Da una breve analisi, si conclude che il modello logico prodotto è, allo stesso tempo, efficiente rispetto alle operazioni definite, e semplice, essendo formato da costrutti meno complessi rispetto a quelli che erano presenti nel modello concettuale di partenza.

3.3 Progettazione fisica con MySQL

L'ultima fase di progettazione di una base di dati è la progettazione fisica. Essa che consiste fondamentalmente in tre attività, nel seguente ordine:

1. *Scelta delle strutture di memorizzazione*. Inoltre, in progetti più complessi di quello attuale, è necessaria anche una scelta delle strutture ausiliarie di accesso ai dati, o *indici*. Questi elementi servono per rendere più efficiente l'accesso ai dati contenuti in tabelle usate di frequente. Le strutture di memorizzazione e di accesso sono valutate tra quelle messe a disposizione dal DBMS scelto.
2. *Traduzione del modello logico in uno schema fisico dei dati* contenente le definizioni delle tabelle, dei relativi vincoli di integrità e delle viste espresse in SQL.

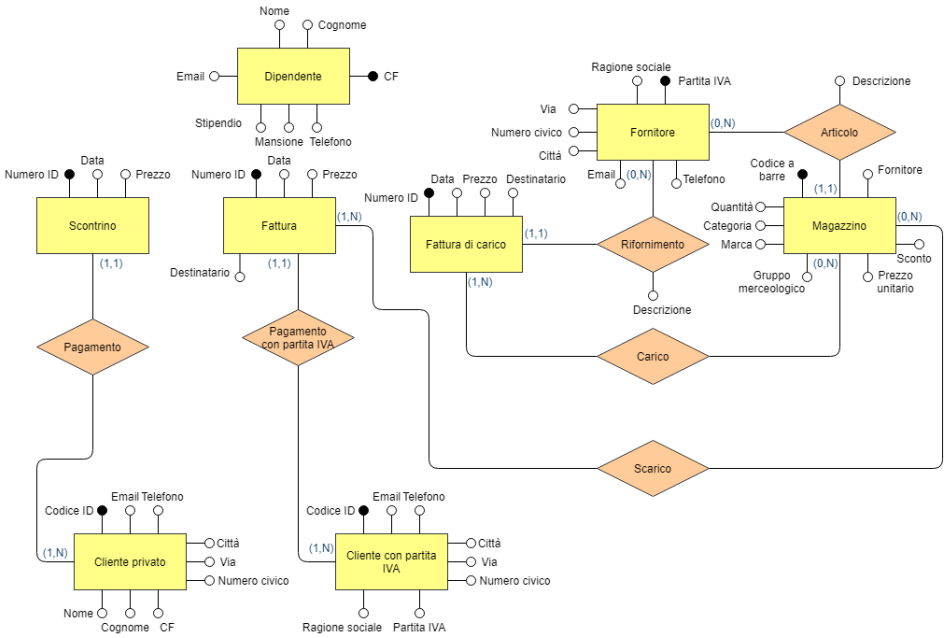

Figura 3.9. Modello logico finale

3. *Implementazione, mediante SQL, delle transazioni*, ovvero sequenze di operazioni che, se eseguite nel modo corretto, producono una variazione dello stato del database.

Tuttavia, prima di queste attività, è necessario scegliere un DBMS (Database Management System) che implementi il modello di dati dello schema logico. MySQL è un RDBMS (Relational Database Management System) tra i più diffusi degli ultimi anni, composto da un client a riga di comando e un server, entrambi disponibili per quasi tutti i sistemi operativi moderni. Tutti i modelli relazionali vengono normalmente utilizzati con il linguaggio di interrogazione SQL (Structured Query Language), il quale è l'unione di quattro tipi diversi di linguaggio:

- Data Definition Language (DDT), per la creazione e la modifica degli schemi dei database.
- Data Manipulation Language (DML), per l'inserimento, la modifica e la gestione dei dati memorizzati.
- Data Query Language (DQL), per l'interrogazione dei dati memorizzati.
- Data Control Language (DCL), per la creazione e la gestione degli strumenti di controllo e di accesso ai dati (gli indici).

Nella progettazione fisica del database che si sta sviluppando si vuole utilizzare MySQL, il quale, essendo un RDBMS, gestisce database relazionali, che sono, dunque, costituiti da *relazioni*, cioè tabelle (anche se l'utilizzo del termine "tabella" non è propriamente corretto).

Perciò, il primo passo da compiere è la scelta delle strutture di memorizzazione da utilizzare. A partire dallo schema logico, in Figura 3.9, si vuole associare ogni

costrutto ad una relazione, o tabella, ad eccezion fatta per le associazioni *Pagamento* e *Pagamento con partita IVA*, le quali mettono in relazione le loro entità tramite azioni che le caratterizzano, e non per mezzo di un insieme di dati che le accomunano. In effetti, si potrebbe comunque decidere la traduzione dei suddetti costrutti in tabelle; tuttavia, così facendo, si andrebbero a memorizzare dei dati inutili per il software gestionale che si vuole sviluppare.

Fatta questa considerazione, in Tabella 3.15 viene mostrata la traduzione dello schema logico nel modello relazionale.

Entità/Relazione	*Traduzione*
Dipendente	Dipendente(CF, Nome, Cognome, Email, Telefono, Mansione, Stipendio)
Scontrino	Scontrino(Numero ID, Data, Prezzo)
Fattura	Fattura(Numero ID, Data, Prezzo, Destinatario)
Cliente privato	Cliente(Codice ID, Nome, Cognome, CF, Citta, Via, Numero_civico)
Cliente con partita IVA	ClientePIVA(Codice ID, Ragione_sociale, Partita_IVA, Email, Telefono, Citta, Via, Numero_civico)
Fattura di carico	Fattura_carico(Numero ID, Data, Prezzo, Destinatario)
Rifornimento	Rifornimento(Numero ID, Partita_IVA, Descrizione)
Fornitore	Fornitore(Partita_IVA, Ragione_sociale, Telefono, Email, Citta, Via, Numero_civico)
Articolo	Articolo(Codice_barre, Partita_IVA, Descrizione)
Magazzino	Magazzino(Codice_barre, Fornitore, Gruppo_merceologico, Categoria, Marca, Prezzo_unitario, Scontro, Quantita)
Carico	Carico(Codice_barre, Numero_ID)
Scarico	Scarico(Codice_barre, Numero_ID)

Tabella 3.15. Traduzione dello schema logico nel modello relazionale

Nella suddetta tabella, le sottolineature della seconda colonna rappresentano le *chiavi primarie*, cioè un insieme di attributi che permettono di individuare univocamente una tupla, o record, di una relazione.

Definito l'insieme di tabelle da implementare, si dispone di informazioni a sufficienza per la creazione fisica del database, e per l'inserimento di tutte le sue operazioni precedentemente dedotte.

Quindi, si può utilizzare SQL per la codifica del database della cartolibreria con le sue corrispondenti tabelle, da dover popolare. Infine, lo stesso linguaggio potrà essere utilizzato per definire un insieme di *query*, utile all'attuazione dei requisiti funzionali.

4

Progettazione dei casi d'uso

In questo capitolo vengono utilizzati i requisiti funzionali dedotti in fase di specifica del software per estrapolare i casi d'uso del sistema. Inoltre, ogni singolo caso d'uso viene dettagliatamente spiegato, al fine di arricchire la documentazione del software con delle linee guida da poter seguire nella fase di progettazione del sistema.

4.1 Diagramma dei casi d'uso

Il caso d'uso è una tecnica usata nei processi di ingegneria del software per descrivere, in maniera esaustiva e non ambigua, uno scenario di interazione, cioè un dialogo, tra il sistema e i suoi utilizzatori.

Questo concetto è stato introdotto per la prima volta da Ivar Jacobson, nel 1992, all'interno della sua software house, conosciuta come Objectory AB. Tale modello standard si è rivelato molto importante al fine di poter esprimere, in maniera schematica e dettagliata, i modi in cui un sistema deve essere utilizzato.

Si può ricorrere ai casi d'uso a supporto della fase di deduzione dei requisiti, ma essi si rivelano più utili per la progettazione del sistema. Infatti, la conoscenza dei passi da dover percorrere, nella realizzazione di uno specifico requisito funzionale, fornisce agli ingegneri del software delle linee guida da poter seguire per definire le fasi preliminari della progettazione del sistema.

Il linguaggio di modellazione UML (Unified Model Language) mette a disposizione, tra i vari costrutti, anche i diagrammi dei casi d'uso (UCD), i quali sono dedicati alla descrizione delle funzioni o dei servizi offerti da un sistema, così come vengono percepiti dai suoi utilizzatori (gli attori).

In termini pratici, questi diagrammi vengono utilizzati per descrivere, in maniera approfondita, i requisiti funzionali del sistema. Inoltre, ad ogni caso d'uso corrisponde un diagramma di attività ed un diagramma di sequenza, che vengono utilizzati in UML per descrivere la dinamicità del sistema.

Il primo passo da compiere, nello sviluppo del software gestionale di interesse, consiste nella formulazione dei diagrammi dei casi d'uso, in modo che questi possano fungere da punto di partenza per la progettazione del sistema, dall'architettura ai singoli componenti.

In fase di analisi dei requisiti sono state definite otto aree di interesse della cartolibreria da dover gestire. Ad ognuna di queste corrisponde un certo insieme di requisiti funzionali, per un totale di ventisette. Tali requisiti vengono usati, a loro volta, per giungere alla formulazione dei casi d'uso mostrati in Figura 4.1.

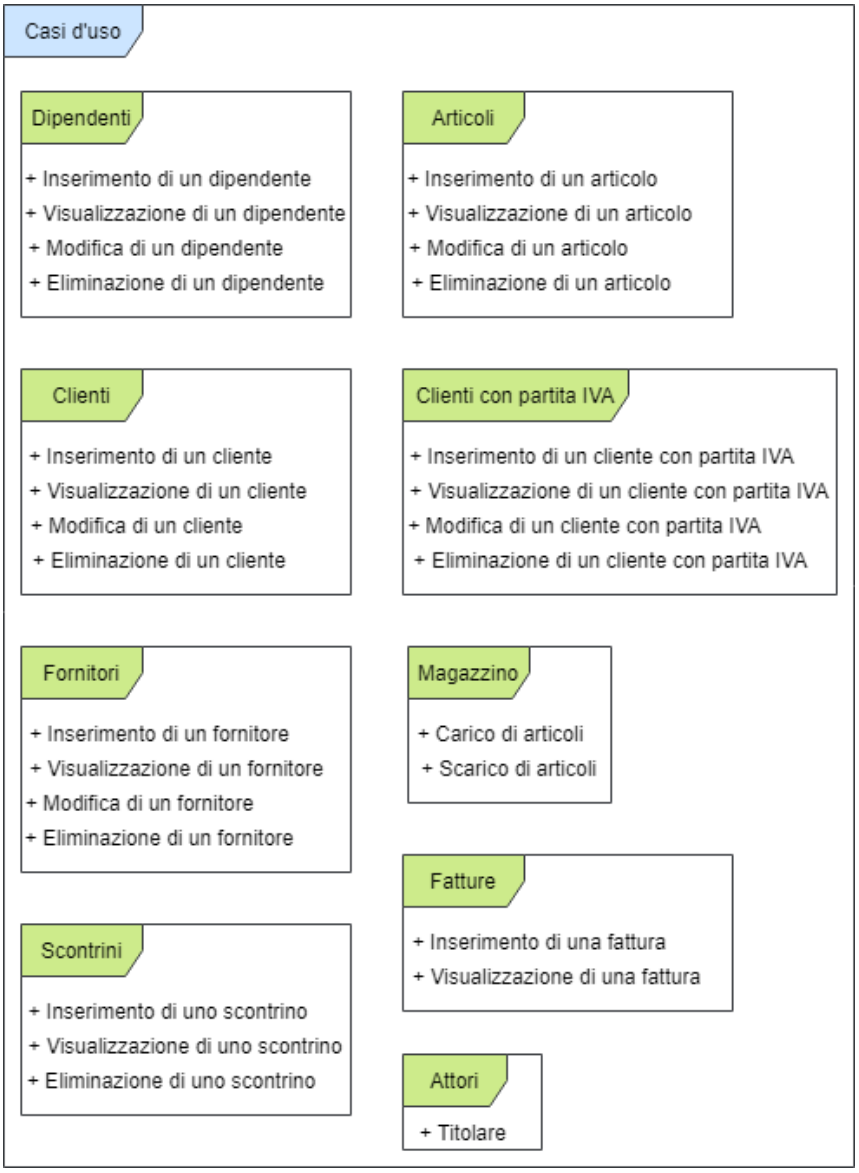

Figura 4.1. I casi d'uso del sistema della cartolibreria

A causa della sua dimensione, il diagramma dei casi d'uso complessivo verrà partizionato, in base alle aree funzionali, in otto diagrammi separati. In seguito,

ognuna di queste parti verrà presentata e trattata dettagliatamente.

4.1.1 Attori

Nei diagrammi dei casi d'uso, un attore rappresenta un ruolo coperto da un certo insieme di entità interagenti con il sistema, come, ad esempio, utenti umani, organizzazioni e sistemi esterni. Ogni attore è rappresentato graficamente da un'icona di uno *stickman*, cioè un uomo stilizzato.

Il software che si vuole sviluppare è pensato per essere utilizzato dal titolare della cartolibreria, ragion per cui quest'ultimo è l'unico attore che interagisce con il sistema. Tuttavia, egli può decidere di essere sostituito, in parte o in tutte le sue funzioni, dai suoi dipendenti. In tal senso, i titolari ed i dipendenti che interagiscono con il software vengono considerati come istanze di un unico attore *Titolare* (vedi Figura 4.2).

Figura 4.2. L'unico attore presente nel diagramma dei casi d'uso

4.1.2 Dipendenti

Gli stakeholder della cartolibreria, allo stato attuale, non hanno dipendenti. Tuttavia, non si precludono la possibilità di averne in futuro, anche solo a tempo determinato. Quindi, il software gestionale dovrà mettere a disposizione degli utilizzatori tutte le funzionalità necessarie per la gestione dei dati di uno o più dipendenti. Infatti, per tale area funzionale, si hanno a disposizione quattro casi d'uso, mostrati in Figura 4.3.

Inserimento di un dipendente

Il caso d'uso di inserimento di un dipendente viene attuato dal titolare dell'attività in seguito ad ogni nuova assunzione.

- *Precondizioni*: il dipendente non è presente nel sistema.
- *Postcondizioni*: il dipendente è presente nel sistema, oppure non è stato possibile aggiungerlo.
- *Sequenza principale degli eventi*:
  1. Il titolare della cartolibreria assume un nuovo dipendente, quindi decide di inserirlo nel software.
  2. Il sistema mostra l'interfaccia per l'inserimento dei dati del nuovo dipendente.
  3. Il titolare inserisce tutte le informazioni relative al dipendente da dover memorizzare.

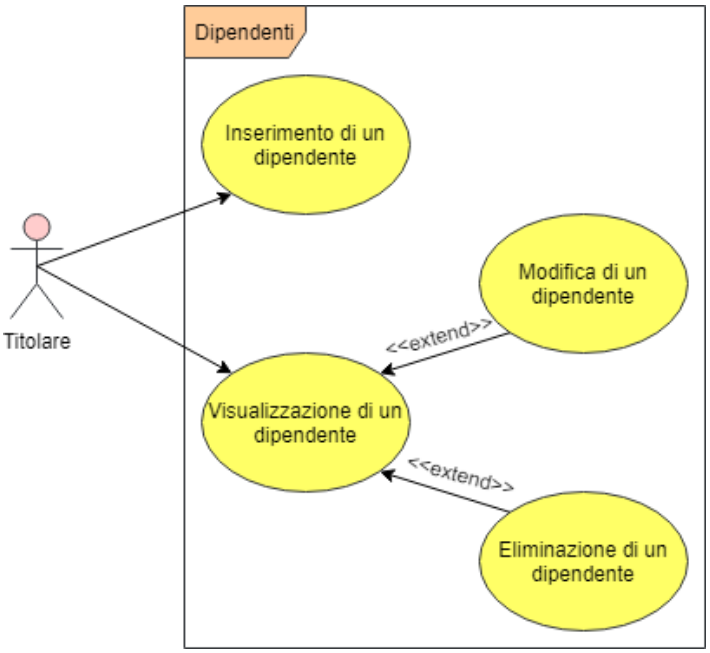

Figura 4.3. Diagramma dei casi d'uso dei dipendenti

4. Il titolare termina l'inserimento dei dati del dipendente.
5. Il nuovo dipendente viene correttamente inserito all'interno del sistema.

- *Sequenza alternativa degli eventi*:
  5. I dati inseriti dal titolare sono incompleti o errati.
  6. L'inserimento del nuovo dipendente fallisce.

Visualizzazione di un dipendente

Il caso d'uso di visualizzazione di un dipendente memorizzato nel sistema si verifica quando il titolare della cartolibreria vuole conoscere i dati di un dipendente, al fine di modificarli, eliminarli o utilizzarli per altri scopi.

- *Precondizioni*: il dipendente è presente nel sistema.
- *Postcondizioni*: -
- *Sequenza principale degli eventi*:
  1. Il titolare dell'attività vuole visualizzare le informazioni di un dipendente.
  2. Il sistema cerca il dipendente da dover visualizzare.
  3. Il sistema mostra sullo schermo i dati del dipendente cercato.
- *Sequenza alternativa degli eventi*: -

Modifica di un dipendente

Il caso d'uso di modifica di un dipendente del sistema viene effettuato dal titolare dell'attività quando è necessario modificare i dati di un dipendente, come ad esempio

i contatti, la mansione, lo stipendio, ma non il valore identificativo utilizzato per memorizzarlo.

- *Precondizioni*: il dipendente è presente nel sistema.
- *Postcondizioni*: le informazioni del dipendente sono state modificate correttamente, oppure non è stato possibile applicare la modifica.
- *Sequenza principale degli eventi*:
  1. Il titolare della cartolibreria vuole modificare i dati di un dipendente.
  2. Il sistema cerca il dipendente da visualizzare per la modifica delle sue informazioni.
  3. Il sistema mostra sullo schermo i dati del dipendente da modificare.
  4. Il titolare effettua la procedura per l'applicazione delle modifiche desiderate.
  5. Il sistema aggiorna i dati del dipendente.
- *Sequenza alternativa degli eventi*:
  5. I dati inseriti dal titolare sono incompleti o errati.
  6. L'operazione di modifica del dipendente fallisce.

Eliminazione di un dipendente

Il caso d'uso di eliminazione di un dipendente presente nel sistema viene applicato dal titolare della cartolibreria quando non è più necessario mantenere le informazioni di un dipendente memorizzato, a causa del fatto che questa persona non lavora più all'interno dell'attività.

- *Precondizioni*: il dipendente è presente nel sistema.
- *Postcondizioni*: il dipendente è stato eliminato dal sistema.
- *Sequenza principale degli eventi*:
  1. Il titolare dell'attività vuole eliminare un dipendente dal sistema.
  2. Il sistema cerca il dipendente da visualizzare per la sua eliminazione.
  3. Il sistema mostra sullo schermo i dati del dipendente da eliminare.
  4. Il titolare avvia la procedura di eliminazione del dipendente.
  5. i dati del dipendente vengono eliminati dal sistema.
- *Sequenza alternativa degli eventi*: -

4.1.3 Articoli

Gli articoli sono elementi importanti da dover manipolare all'interno del software gestionale, dal momento che essi rappresentano i beni con cui l'attività in questione produce valore.

Gli articoli sono acquisiti all'ingrosso presso uno dei fornitori, e in seguito vengono messi in magazzino, in attesa della loro vendita ai clienti. Tenendo conto di questo ciclo di vita, i casi d'uso relativi agli articoli sono costituiti, come per i dipendenti, dalle operazioni di inserimento, visualizzazione, modifica ed eliminazione (Figura 4.4). Infatti, le descrizioni di tali operazioni, sono analoghe a quelle dei dipendenti. Pertanto, si rimanda alla Sezione 4.1.2.

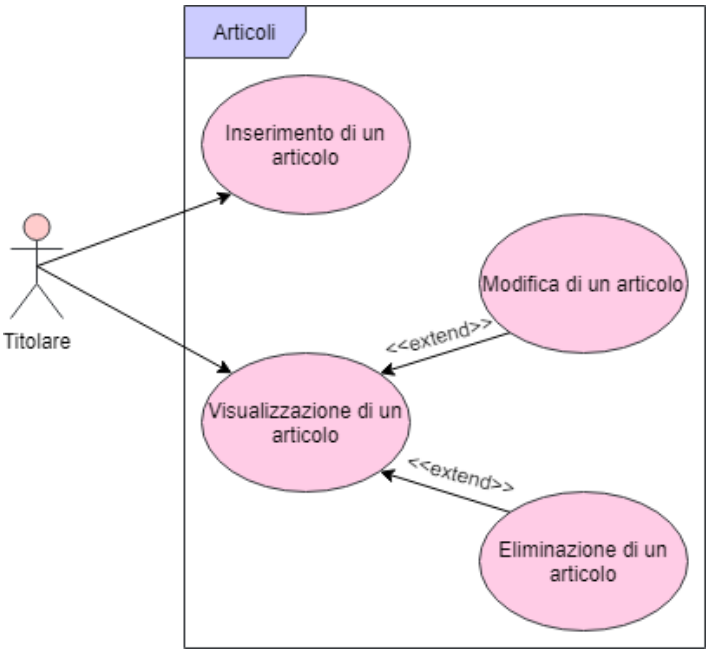

Figura 4.4. Diagramma dei casi d'uso degli articoli

4.1.4 Clienti privati e clienti con partita IVA

La cartolibreria è rivolta a tutte le tipologie di clienti, dagli studenti ai lavoratori, dai privati alle istituzioni pubbliche. In seguito ad ogni acquisto, per legge, deve essere prodotta una ricevuta, che, nella maggior parte dei casi, consiste in uno scontrino. Tuttavia, a volte, è necessaria l'erogazione di una fattura, all'interno della quale devono essere inseriti i dati dell'acquirente.

Se il cliente in questione visita abitualmente la cartolibreria, per evitare di dover inserire manualmente i suoi dati ogni volta che effettua un acquisto, lo si può memorizzare all'interno del sistema.

I clienti che richiedono il rilascio di una fattura sono categorizzabili in due gruppi: i clienti normali, che effettuano acquisti a scopo privato, e i clienti dotati di partita IVA, che solitamente comprano articoli per conto della loro azienda. Nonostante entrambe le categorie di clienti possano richiedere una fattura in seguito ad un determinato acquisto, la loro memorizzazione in due gruppi differenti è ritenuta necessaria poiché coloro che non possiedono una partita IVA possono comunque richiedere una fattura utilizzando il loro codice fiscale.

Al di là delle differenziazioni, i casi d'uso dei clienti normali (Figura 4.5) e quelli relativi ai clienti con partita IVA (Figura 4.6) sono gli stessi descritti per gli articoli ed i dipendenti. Quindi, per averne una descrizione dettagliata, si rimanda alla Sezione 4.1.2.

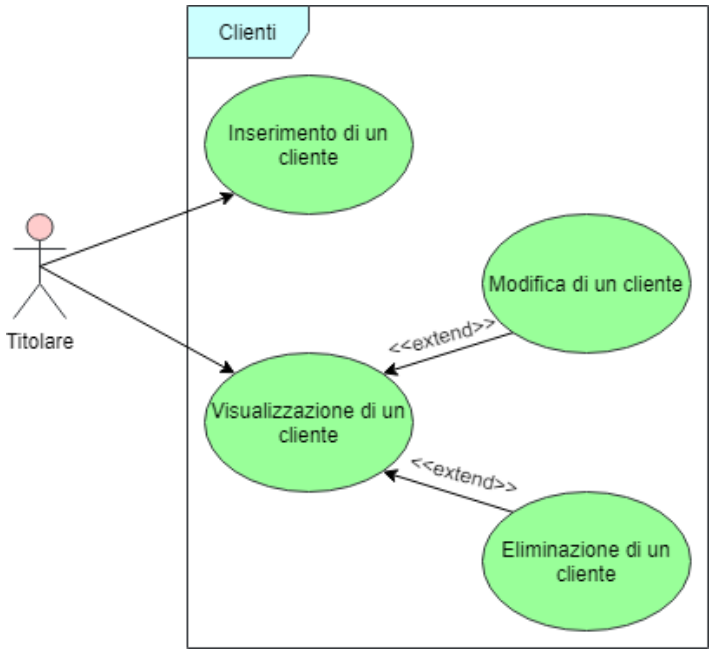

Figura 4.5. Diagramma dei casi d'uso dei clienti normali

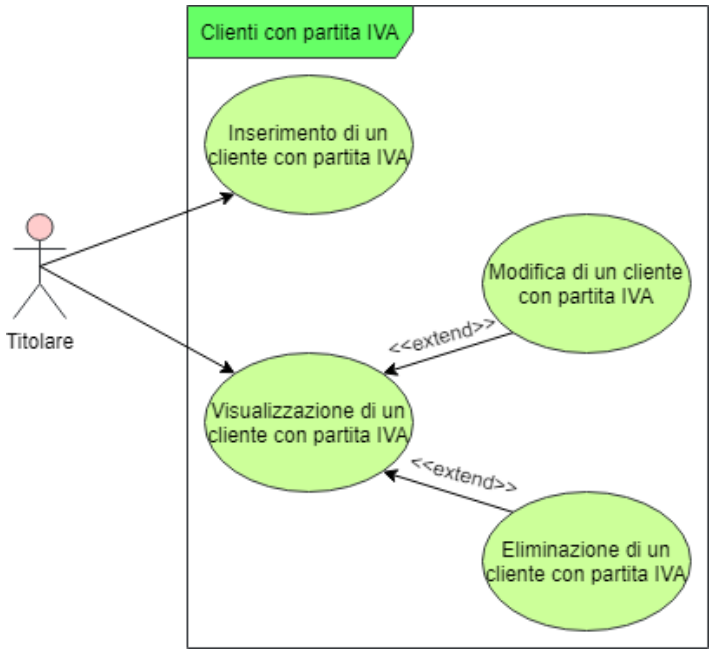

Figura 4.6. Diagramma dei casi d'uso dei clienti dotati di partita IVA

4.1.5 Fornitori

I fornitori sono industriali o commercianti che distribuiscono all'ingrosso gli articoli che la cartolibreria, a sua volta, vende ai propri clienti. Periodicamente, gli stakeholder dell'attività, sulla base delle disponibilità degli articoli in magazzino, contattano i propri fornitori al fine di accordarsi sulla merce da acquisire. A tale scopo, in fase di raccolta dei requisiti, i titolari della cartolibreria hanno espresso la necessità di memorizzare nel sistema i propri fornitori. Quindi, anche in questo caso, come per le precedenti aree di interesse, sono necessarie le medesime funzionalità di inserimento, visualizzazione, modifica ed eliminazione dei fornitori (Figura 4.7). Pertanto si rimanda alla Sezione 4.1.2 per una descrizione dettagliata dei casi d'uso.

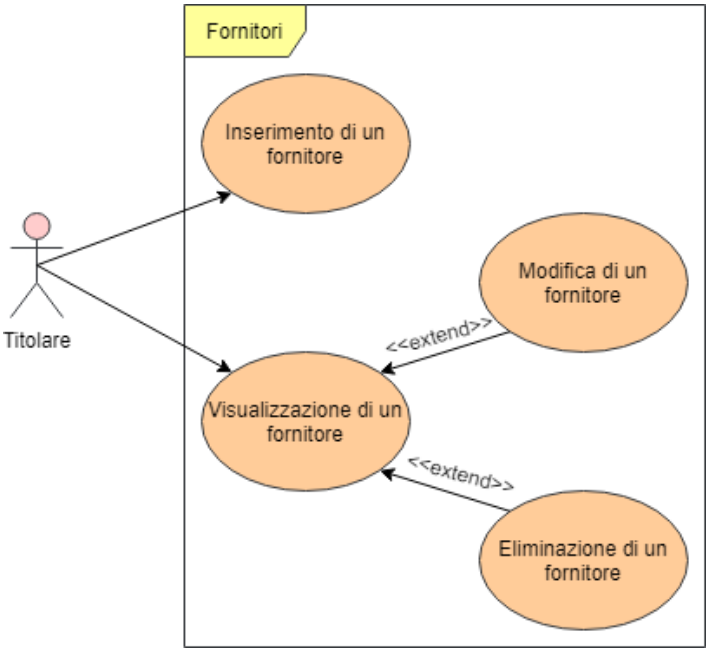

Figura 4.7. Diagramma dei casi d'uso dei fornitori

4.1.6 Scontrini

Per tener conto dello storico delle vendite, gli stakeholder hanno richiesto la memorizzazione nel sistema delle informazioni di tutti gli scontrini rilasciati.

Tipicamente, in seguito all'acquisto di uno o più articoli da parte di un cliente, è richiesta l'emissione di uno scontrino cartaceo per dichiarare il compimento della transazione. Se la ricevuta emessa dovesse presentare un refuso, allora non sarebbe possibile una sua modifica, bensì bisognerebbe cancellare lo scontrino, per poi emetterlo nuovamente con i dati corretti.

Tenendo conto di questa dinamica, i casi d'uso relativi agli scontrini sono quelli presenti in Figura 4.8.

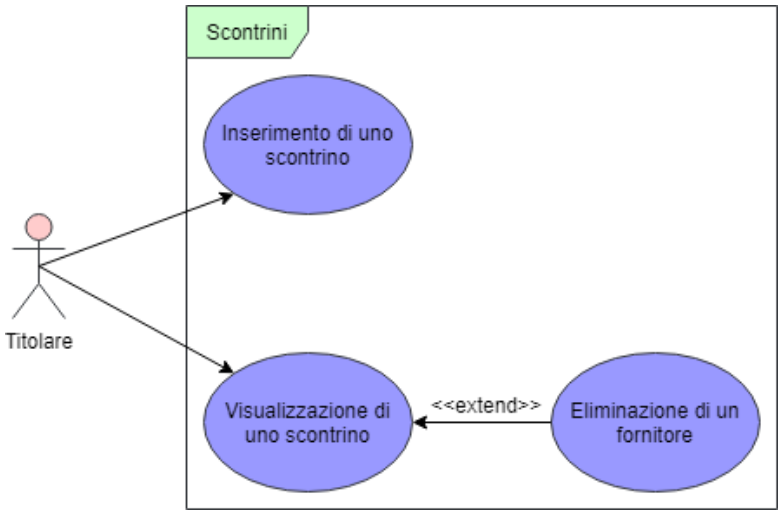

Figura 4.8. Diagramma dei casi d'uso degli scontrini

Inserimento di uno scontrino

In seguito ad una transazione, il titolare dell'attività attua l'inserimento di uno scontrino nel sistema.

- *Precondizioni*: un cliente effettua un pagamento.
- *Postcondizioni*: lo scontrino è presente nel sistema, oppure non è stato possibile aggiungerlo.
- *Sequenza principale degli eventi*:
  1. Il titolare della cartolibreria vuole memorizzare lo scontrino nel sistema.
  2. Il sistema mostra l'interfaccia per l'inserimento dei dati degli articoli venduti.
  3. Il titolare inserisce tutti i dati degli articoli venduti.
  4. Il titolare conclude l'inserimento.
  5. Il sistema scarica dal magazzino gli articoli inseriti nello scontrino.
  6. Lo scontrino viene correttamente memorizzato nel sistema.
- *Sequenza alternativa degli eventi*:
  5. I dati inseriti dal titolare sono incompleti o errati.
  6. L'inserimento dello scontrino fallisce.

Visualizzazione di uno scontrino

Il caso d'uso di visualizzazione di uno scontrino si verifica quando il titolare dell'attività vuole controllarne i dati, alla ricerca di eventuali errori o per altri scopi.

- *Precondizioni*: lo scontrino è presente nel sistema.
- *Postcondizioni*: -
- *Sequenza principale degli eventi*:
  1. Il titolare dell'attività vuole visualizzare le informazioni di uno scontrino.

2. Il titolare seleziona dal sistema lo scontrino cercato.
3. Il sistema mostra sullo schermo i dati dello scontrino.

- *Sequenza alternativa degli eventi*: -

Eliminazione di uno scontrino

Il caso d'uso di eliminazione di uno scontrino viene applicato quando è necessario un suo annullamento a causa di un errore di digitazione.

- *Precondizioni*: lo scontrino è presente nel sistema.
- *Postcondizioni*: lo scontrino è stato eliminato dal sistema.
- *Sequenza principale degli eventi*:
  1. Il titolare dell'attività vuole eliminare uno scontrino presente nel sistema.
  2. Il titolare seleziona dal sistema lo scontrino da eliminare.
  3. Il sistema mostra sullo schermo i dati dello scontrino da eliminare.
  4. Il titolare avvia la procedura di eliminazione dello scontrino.
  5. Gli articoli presenti nello scontrino vengono caricati in magazzino.
  6. Lo scontrino viene eliminato dal sistema.
- *Sequenza alternativa degli eventi*: -

4.1.7 Fatture

Una fattura può essere rilasciata sia ai clienti dotati di partita IVA che ai clienti normali, mediante l'utilizzo del codice fiscale.

Ogni fattura, in seguito ad una sua emissione, viene automaticamente mandata all'Agenzia delle Entrate. Questo significa che, a differenza degli scontrini, non è possibile richiederne un annullamento senza mettere in atto un processo burocratico che coinvolge l'agenzia stessa.

Quindi, ci sono unicamente due casi d'uso relativi alle fatture, mostrati in Figura 4.9.

Inserimento di una fattura

Il caso d'uso di inserimento di una fattura viene attivato quando un cliente che effettua un acquisto richiede il rilascio di una fattura.

- *Precondizioni*: un cliente effettua un pagamento.
- *Postcondizioni*: la fattura è presente nel sistema, oppure non è stato possibile inserirla.
- *Sequenza principale degli eventi*:
  1. Il titolare della cartolibreria vuole memorizzare una fattura nel sistema.
  2. Il sistema mostra l'interfaccia per l'inserimento dei dati della fattura.
  3. Il titolare inserisce i dati del cliente che ha effettuato l'acquisto.
  4. Il titolare inserisce tutti i dati degli articoli venduti.
  5. Il titolare conclude l'inserimento.
  6. Il sistema scarica dal magazzino gli articoli inseriti nella fattura.
  7. La fattura viene correttamente memorizzata nel sistema.
- *Sequenza alternativa degli eventi*:
  6. I dati inseriti dal titolare sono incompleti o errati.
  7. L'inserimento della fattura fallisce.

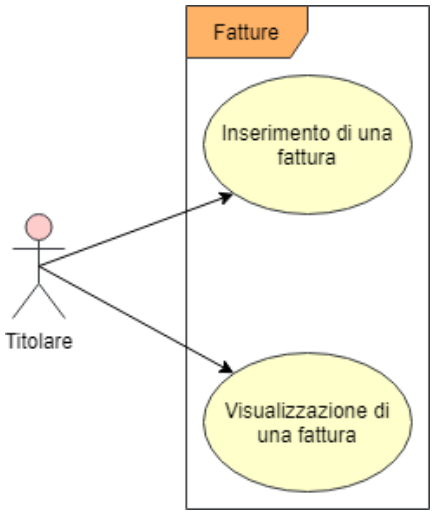

Figura 4.9. Diagramma dei casi d'uso delle fatture

Visualizzazione di una fattura

Il caso d'uso di visualizzazione di una fattura memorizzata nel sistema si verifica quando il titolare dell'attività vuole controllarne i dati.

- *Precondizioni*: la fattura è presente nel sistema.
- *Postcondizioni*: -
- *Sequenza principale degli eventi*:
  1. Il titolare dell'attività vuole visualizzare le informazioni di una fattura.
  2. Il titolare seleziona dal sistema la fattura cercata.
  3. Il sistema mostra sullo schermo i dati della fattura.
- *Sequenza alternativa degli eventi*: -

4.1.8 Magazzino

Il magazzino è una raccolta di articoli disponibili alla vendita. In seguito ad ogni rifornimento, gli articoli acquisiti all'ingrosso vengono caricati in magazzino, quindi aumentano. Al contrario, quando un cliente effettua un acquisto, è necessario scaricare dal magazzino, ovvero diminuire, le quantità degli articoli venduti. Quindi, in quest'area di interesse, sono presenti due casi d'uso, mostrati in Figura 4.10.

Carico di articoli

L'operazione di carico viene eseguita ogniqualvolta che è necessario aumentare le disponibilità degli articoli presenti in magazzino, quindi ad ogni rifornimento della merce.

- *Precondizioni*: l'articolo da caricare è presente nel sistema.
- *Postcondizioni*: la quantità di un articolo disponibile alla vendita è aumentata.

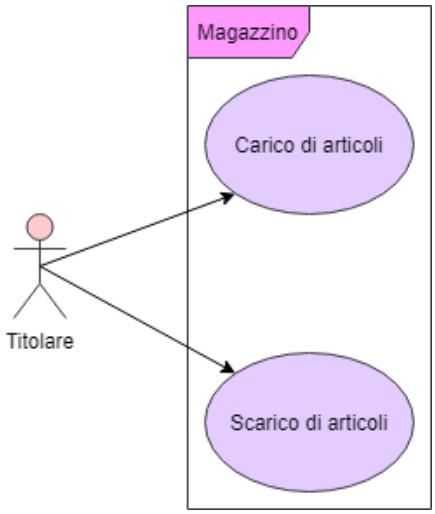

Figura 4.10. Diagramma dei casi d'uso del magazzino

- *Sequenza principale degli eventi*:
  1. Il titolare dell'attività acquisisce merce presso un fornitore.
  2. Il titolare interagisce con il sistema per richiedere l'operazione di carico.
  3. Il sistema mostra sullo schermo l'interfaccia di inserimento dei dati dell'operazione di carico.
  4. Il titolare inserisce l'articolo da dover caricare in magazzino.
  5. Il titolare inserisce la quantità di articolo acquisita dal fornitore.
  6. Il titolare termina l'inserimento dei dati del carico merce.
  7. Il sistema aggiorna la quantità dell'articolo caricato nel magazzino.
- *Sequenza alternativa degli eventi*: -

Scarico di articoli

L'operazione di scarico viene eseguita ogni volta che è necessario diminuire le disponibilità degli articoli presenti in magazzino, quindi in seguito ad ogni vendita.

- *Precondizioni*: l'articolo da scaricare è presente nel sistema.
- *Postcondizioni*: la quantità dell'articolo disponibile alla vendita è diminuita, oppure non è stato possibile effettuare lo scarico.
- *Sequenza principale degli eventi*:
  1. Il titolare della cartolibreria vuole diminuire la quantità di un articolo dal magazzino, in seguito ad una vendita.
  2. Il titolare interagisce con il sistema per richiedere l'operazione di scarico.
  3. Il sistema mostra sullo schermo l'interfaccia di inserimento dei dati dell'operazione di scarico.
  4. Il titolare inserisce l'articolo venduto ad un cliente.
  5. Il titolare inserisce la quantità di articolo da dover scaricare dal magazzino.
  6. Il titolare termina l'inserimento dei dati dello scarico merce.
  7. Il sistema aggiorna la quantità dell'articolo scaricato dal magazzino.

- *Sequenza alternativa degli eventi*:
  7. La quantità inserita dal titolare è maggiore di quella effettivamente presente in magazzino.
  8. L'operazione di scarico fallisce.

4.2 Matrice di mapping

Un software di qualità deve inevitabilmente rispettare tutti i requisiti funzionali prefissati in fase di specifica del software. Inoltre, si è visto come vengono utilizzati i casi d'uso per rappresentare il modo con cui si vogliono implementare tali requisiti nel sistema.

Per poter verificare se questi vengono o meno rispettati, si utilizza una matrice di mapping nella quale vengono mostrate le associazioni tra i requisiti e gli scenari del sistema.

In Figura 4.11 è mostrata la matrice di mapping dei requisiti del software gestionale che si intende sviluppare, da cui si può verificare che ad ogni requisito corrisponde almeno un caso d'uso, e viceversa.

Matrice di mapping Requisiti/Casi d'uso	RF1 Inserimento dipendente	RF2 Visualizzazione dipendente	RF3 Modifica dipendente	RF4 Eliminazione dipendente	RF5 Inserimento fornitore	RF6 Visualizzazione fornitore	RF7 Modifica fornitore	RF8 Eliminazione fornitore	RF9 Inserimento articolo	RF10 Visualizzazione articolo	RF11 Modifica articolo	RF12 Eliminazione articolo	RF13 Carico magazzino	RF14 Scarico magazzino	RF15 Inserimento cliente	RF16 Visualizzazione cliente	RF17 Modifica cliente	RF18 Eliminazione cliente	RF19 Inserimento clientePIVA	RF20 Visualizzazione	RF21 Modifica clientePIVA	RF22 Eliminazione clientePIVA	RF23 Creazione scontrino	RF24 Visualizzazione scontrino	RF25 Eliminazione scontrino	RF26 Creazione fattura	RF27 Visualizzazione fattura
Inserimento di un dipendente	X																										
Visualizzazione di un dipendente		X																									
Modifica di un dipendente		X	X																								
Eliminazione di un dipendente		X		X																							
Inserimento di un fornitore					X																						
Visualizzazione di un fornitore						X																					
Modifica di un fornitore						X	X																				
Eliminazione di un fornitore						X		X																			
Inserimento di un articolo									X																		
Visualizzazione di un articolo										X																	
Modifica di un articolo										X	X																
Eliminazione di un articolo										X		X															
Carico di articoli													X													X	
Scarico di articoli														X									X			X	
Inserimento di un cliente															X												
Visualizzazione di un cliente																X											
Modifica di un cliente																X	X										
Eliminazione di un cliente																X		X									
Inserimento di un cliente con partita IVA																			X								
Visualizzazione di un cliente con partita IVA																				X							
Modifica di un cliente con partita IVA																				X	X						
Eliminazione di un cliente con partita IVA																				X		X					
Inserimento di uno scontrino																							X				
Visualizzazione di uno scontrino																								X			
Eliminazione di uno scontrino																								X	X		
Inserimento di una fattura																										X	
Visualizzazione di una fattura																											X

Figura 4.11. La matrice di mapping tra i requisiti funzionali ed i casi d'uso del sistema

5

Implementazione

Nel presente capitolo vengono introdotti il linguaggio di programmazione Python e la libreria PyQt, utilizzata per lo sviluppo delle interfacce grafiche. Inoltre, si esplicano i punti più importanti del codice utilizzato per l'implementazione del software destinato alla cartolibreria in esame.

5.1 Python

In fase di specifica del software, è stato estrapolato il requisito non funzionale *RNF4 implementazione*, che prevede l'utilizzo del linguaggio di programmazione Python per lo sviluppo del software.

Python è un linguaggio di programmazione moderno, di più "alto livello" rispetto agli altri linguaggi conosciuti, dalla sintassi semplice e potente, che ne facilita l'apprendimento. Gli ambiti di applicazione di questo linguaggio sono svariati: sviluppo di siti o applicazioni web e desktop, realizzazione di interfacce grafiche, amministrazione di sistema, calcolo scientifico, creazione e gestione di database, sviluppo di giochi, programmazione di grafica 3D, etc.

Negli anni '80, al National Research Institute for Mathematics and Computer Science (CWI) di Amsterdam, alcuni ricercatori, tra cui Guido Van Rossum, hanno sviluppato un linguaggio di nome ABC, molto potente ed elegante, che, fin da subito, divenne popolare nel mondo Unix.

Qualche anno dopo (fine anni ottanta) Guido Van Rossum ha avuto una serie di idee mirate al miglioramento di ABC e, pertanto, si mise a lavorare allo sviluppo di un nuovo linguaggio, a cui diede il nome di Python, in onore di Monty Python's Flying Circus, una serie televisiva britannica andata in onda dal 1969 al 1974.

Python è il linguaggio attualmente più in voga, come si può notare dalla Figura 5.1, ed è tuttora in crescita grazie ad una serie di vantaggi:

- Si tratta di un linguaggio completamente gratuito, ed è possibile usarlo e distribuirlo senza restrizioni di copyright. Per questa ragione, da oltre 25 anni, Python ha una comunità molto attiva, e riceve costantemente miglioramenti che lo mantengono aggiornato e al passo con i tempi.

- Python è un linguaggio multi-paradigma. Infatti, supporta sia la programmazione procedurale, sia quella ad oggetti, includendo funzionalità come l'ereditarietà singola e multipla, l'overloading degli operatori, e il duck typing. Inoltre, supporta anche diversi elementi della programmazione funzionale, come iteratori e generatori.
- È un linguaggio portabile, cioè utilizzabile su diverse piattaforme, come Unix, Linux, Windows, DOS, Macintosh, sistemi real-time, OS/2, cellulari Android e iOS. Questo è possibile perché Python è un linguaggio interpretato, in cui lo stesso codice può essere eseguito su qualsiasi piattaforma che abbia l'interprete installato.
- Python, è al tempo stesso, semplice e potente. La sintassi, i diversi moduli e le funzioni che sono già inclusi nel linguaggio sono consistenti, intuitivi, e facili da imparare.
- Ogni installazione di Python include la *standard library*, cioè una collezione di oltre 200 moduli per svolgere i compiti più disparati, come, ad esempio, l'interazione con il sistema operativo e il file system. Inoltre, il Python Package Index (PyPI) consente di scaricare ed installare migliaia di moduli aggiuntivi creati e mantenuti dalla comunità.
- Prima della loro interpretazione, i programmi scritti in Python vengono automaticamente compilati in un formato chiamato bytecode. Questo formato è più compatto ed efficiente, e garantisce, quindi, prestazioni elevate.
- Python è un linguaggio di alto livello, che adotta un meccanismo di garbage collection che si occupa automaticamente dell'allocazione e del rilascio della memoria. Questo consente al progammatore di utilizzare liberamente le variabili, senza doversi preoccupare di allocare e rilasciare spazi di memoria manualmente (cosa che è, invece, necessaria in linguaggi di più basso livello, come C o C++).

Nonostante tutti questi vantaggi, come ogni altro linguaggio di programmazione, Python ha dei difetti. Il suo problema più grande è la lentezza delle prestazioni quando viene utilizzato per sviluppare applicazioni complesse. Inoltre, Python è limitato nella programmazione concorrente, o multithreading, rispetto ad altri linguaggi di più basso livello. Infine, non è un linguaggio fortemente tipizzato, il che può essere un vantaggio, poiché in questo modo si può definire una variabile ancora prima di conoscerne il tipo, ma è anche uno svantaggio, dal momento che si hanno maggiori probabilità di generare errori o bug nelle applicazioni.

Tuttavia, per risolvere queste problematiche, si può utilizzare Python con l'estensione di altri linguaggi, come C (Cython) o Java (Jython).

5.1.1 Libreria PyQt5 per l'interfaccia grafica

Uno dei problemi principali nel quale ci si imbatte quando si vuole sviluppare un software user friendly, è quello di produrre un'interfaccia grafica, dal momento che essa necessita di competenze diverse rispetto a quelle che servono per lo sviluppo degli altri aspetti del software. Di conseguenza, in molti casi, riuscire a sviluppare un'interfaccia grafica user friendly risulta essere un ostacolo piuttosto difficile da superare per un programmatore.

Però, si è visto come Python abbia una comunità estremamente attiva, che lavora continuamente per produrre e mantenere nuove librerie. Tra queste, ne esistono

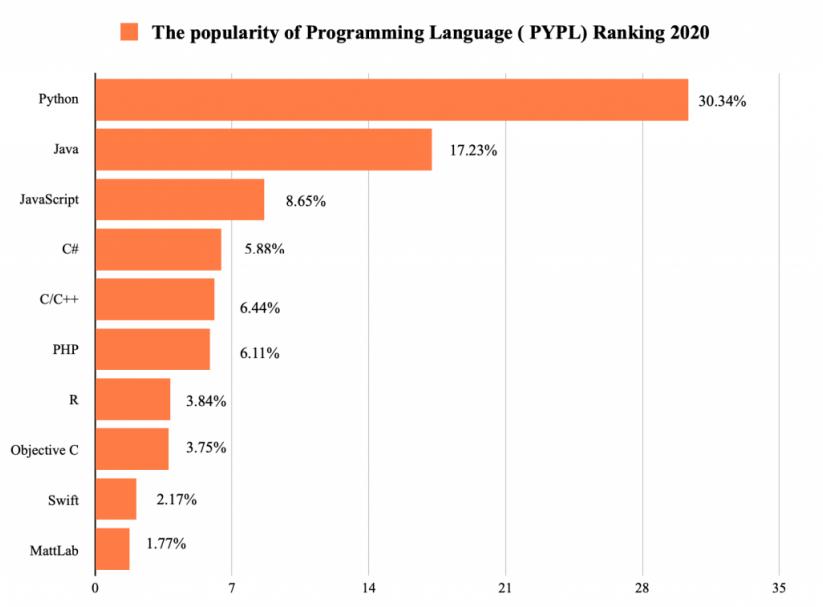

Figura 5.1. I linguaggi di programmazione più popolari del 2020

alcune create appositamente per scrivere interfacce grafiche in maniera semplice ed intuitiva. Una delle più utilizzate è PyQt, che permette di utilizzare Python insieme al toolkit Qt (disponibile anche per molti altri linguaggi di programmazione). Essa si basa sull'utilizzo dei *widget*, ossia elementi grafici con cui viene facilitata l'interazione degli utenti con il software utilizzato.

Prima di poter utilizzare PyQt5 è necessario installarlo nel proprio ambiente di sviluppo, digitando, nella console di Python, il codice `pip install PyQt5`. Dopo qualche secondo di attesa, il pacchetto è pronto all'utilizzo.

5.1.2 Home del software

Nel progetto in esame, all'avvio del software, si vuole ottenere una finestra ridimensionabile, a partire dalla quale deve essere possibile navigare velocemente tra tutte le sezioni disponibili. Quindi, è necessario scrivere il main dell'applicazione istanziando al proprio interno un oggetto della classe `Home`, cioè della pagina principale, dopo che quest'ultima è stata implementata (il codice è mostrato in Figura 5.2).

All'interno di `Home` si vogliono avere tanti `QTabWidget` quante sono le aree di interesse del software: gli scontrini, le fatture, il magazzino, gli articoli, i fornitori, i dipendenti, i clienti normali e quelli dotati di partita IVA. In questo modo è possibile accedere velocemente ad una specifica area, cliccando sul widget corrispondente (il codice si trova in Figura 5.3).

Raffinando il codice mostrato, si può giungere facilmente ad un risultato simile a quello presente in Figura 5.4.

```python
if __name__ == '__main__':
    app = QApplication(sys.argv)
    vista_home = Home()
    vista_home.show()
    sys.exit(app.exec_())

```

Figura 5.2. Il main del software

```python
        layout = QVBoxLayout()
        self.setLayout(layout)

        tabs = QTabWidget()
        tabs.addTab(self.Home(), "Home")
        tabs.addTab(self.Scontrini(), "Scontrini")
        tabs.addTab(self.Fatture(), "Fatture")
        tabs.addTab(self.Magazzino(), "Magazzino")
        tabs.addTab(self.Articoli(), "Articoli")
        tabs.addTab(self.Clienti(), "Clienti")
        tabs.addTab(self.ClientiPIva(), "Clienti PIVA")
        tabs.addTab(self.Fornitori(), "Fornitori")
        tabs.addTab(self.Dipendenti(), "Dipendenti")

        layout.addWidget(tabs)
```

Figura 5.3. La home del software

Figura 5.4. L'interfaccia utente del software

5.2 Architettura MVC

Prima dell'implementazione del software gestionale in questione, è stato fatto un approfondito lavoro di progettazione, che non è stato riportato in questa tesi.

Tuttavia, quando si segue un approccio di sviluppo "plan-driven", il modo con cui viene implementato un software è la diretta conseguenza di come è stato progettato. In particolare, la progettazione ed organizzazione della struttura complessiva di un software, cioè la sua progettazione architetturale, influenza il modo con cui vengono implementati i vari moduli. Per questo motivo, in fase di progettazione di un software, è importante saper scegliere il pattern architetturale più adatto da utilizzare.

Nel caso della cartolibreria, si vuole implementare un prodotto di ottima qualità, con una buona organizzazione del codice, che possa mirare al favoreggiamento di eventuali manutenzioni future, qualora fossero necessarie la risoluzione di malfunzionamenti oppure l'aggiunta di nuove specifiche. A tale scopo, si è optato per l'utilizzo del pattern architetturale Model-View-Controller (MVC), tramite il quale si è portati ad una suddivisione modulare delle componenti del software che si sta sviluppando. Infatti, secondo questo pattern architetturale, le componenti del software vengono suddivise in tre moduli principali:

- Il *model* fornisce i metodi per accedere ai dati utili al software.
- La *view* si occupa di visualizzare i dati del model e di gestire l'interazione degli utenti con l'interfaccia dell'applicazione.
- Il *controller* riceve i comandi dell'utente e li attua modificando lo stato degli altri due componenti.

Questi tre moduli comunicano continuamente tra loro durante il ciclo di vita di un software. Nello specifico, in Figura 5.5, sono indicate le associazioni dirette, tramite le frecce continue, mentre le frecce tratteggiate ne identificano le associazioni indirette, che possono essere messe in atto tramite l'utilizzo del design pattern Observer.

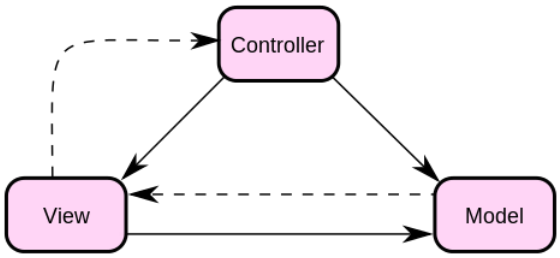

Figura 5.5. Le associazioni tra i componenti dell'architettura MVC

In questo modo, si avrà un codice organizzato e facilmente manutenibile. Inoltre, grazie al fatto di avere il model ed il controller separati dalle viste, viene favorito il riuso del codice.

Utilizzando questo pattern architetturale, durante la fase di implementazione, ogni singola area del software è suddivisa nelle sue tre componenti di model, view e controller, come si può notare per i dipendenti, in Figura 5.6.

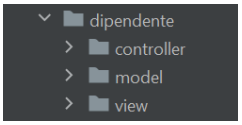

Figura 5.6. Suddivisione del codice dei dipendenti nelle componenti di model, view e controller

Ognuno di questi moduli è implementato in maniera molto simile in tutte le aree del software. A causa di ciò, si riportano, a scopo esemplificativo, unicamente le parti più significative dell'implementazione dei dipendenti. Le stesse considerazioni sono valide anche per le altre aree, a discapito di quella relativa alla gestione del magazzino, che viene trattata in una sezione apposita.

5.2.1 View

La view è quella componente dell'architettura MVC che si occupa della visualizzazione dei dati del software, all'interno di un'interfaccia grafica con la quale un utente può interagire. In alcuni casi, essa può perfettamente combaciare con il front-end di un'applicazione.

Nell'implementazione del prodotto software in esame, dal momento che non sono richieste funzioni elaborative complesse a livello di controller, né particolari manipolazioni dei dati nel model, la maggior parte del codice risiede nelle view.

Per poter accedere rapidamente ad un determinato dipendente, si vuole sviluppare un'interfaccia grafica contenente, in maniera organizzata, tutti i dipendenti del sistema. Di conseguenza, bisogna manipolare una lista di dipendenti, la quale richiede, quindi, l'implementazione di interfacce, processi elaborativi e modello dei dati, in maniera separata rispetto alle componenti della cartella *dipendente* di Figura 5.6. A tale scopo, come è mostrato in Figura 5.7, si crea una cartella specifica per gestire la lista dei dipendenti.

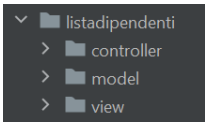

Figura 5.7. La cartella *listidipendenti*, creata per poter gestire separatamente i singoli dipendenti dalla lista nella quale sono contenuti

Da parte dell'utilizzatore del software, il modo migliore per gestire graficamente una lista di dipendenti è quello di utilizzare una tabella in grado di contenerla. Quindi, nella view corrispondente, si inizializza un `QTableWidget`, ossia l'elemento grafico di una tabella. In seguito, tramite l'utilizzo di un ciclo *for-in*, si visitano tutti gli elementi della lista, al fine di mostrarli all'interno del widget inizializzato (Figura 5.8). Grazie a questo processo, utilizzando il codice della libreria PyQt5, si ottiene una tabella simile a quella mostrata in Figura 5.9.

Partendo dal risultato ottenuto, per poter accedere alle informazioni di un determinato dipendente, deve essere messo in atto un opportuno meccanismo di ricerca. A tal fine, si utilizza una `QLineEdit`, ossia l'elemento grafico di un riquadro, mediante il quale si vuole introdurre la possibilità di cercare uno specifico dipendente attraverso il suo nome e/o cognome. Inoltre, di lato a questo riquadro, per poter annullare il meccanismo di ricerca, si vuole inserire un pulsante (`QPushButton`), il quale, ogni volta che viene premuto, fa sì che vengano restituiti nuovamente tutti gli elementi contenuti nella lista dei dipendenti. L'implementazione di queste due componenti grafiche è rappresentata in Figura 5.10, mentre il risultato che si ottiene

```
147     def show_table_view_items(self, item_list):
148         i = 0
149         for dipendente in item_list:
150             item = QTableWidgetItem(str(dipendente.codice_id))
151             self.table_view.setItem(i, 0, item)
152             item = QTableWidgetItem(str(dipendente.nome))
153             self.table_view.setItem(i, 1, item)
154             item = QTableWidgetItem(str(dipendente.cognome))
155             self.table_view.setItem(i, 2, item)
156             item = QTableWidgetItem(str(dipendente.cf))
157             self.table_view.setItem(i, 3, item)
158             item = QTableWidgetItem(str(dipendente.email))
159             self.table_view.setItem(i, 4, item)
160             item = QTableWidgetItem(str(dipendente.telefono))
161             self.table_view.setItem(i, 5, item)
162             item = QTableWidgetItem(str(dipendente.mansione))
163             self.table_view.setItem(i, 6, item)
164             item = QTableWidgetItem("€"+str(dipendente.stipendio_mensile))
165             self.table_view.setItem(i, 7, item)
166             i = i + 1
```

Figura 5.8. Funzione utilizzata per inserire ogni elemento della lista dei dipendenti all'interno della `QTableWidget`

Codice ID	Nome	Cognome	Codice Fiscale	Email	Telefono	Mansione	Stipendio Mensile
1	Mario	Rossi	RSSMRA80A01L113T	rossimario@gmail.it	3432748962	cassiere	€1470
2	Luigi	Bruni	BRNLGI72F90T785P	luigibruni@gmail.it	3779515876	addetto alle pulizie	€1125

Figura 5.9. Una `QTableWidget` modificata in modo tale da dare ad esso l'aspetto desiderato, necessario per poter contenere la lista dei dipendenti del sistema

è mostrato in Figura 5.11. Invece, nel momento in cui si digita una stringa sulla barra di ricerca e si preme il tasto invio, viene attivato l'algoritmo in Figura 5.12, che crea una nuova lista nella quale vengono inseriti i dipendenti trovati in seguito al processo di ricerca. Tale lista viene passata come parametro della funzione in Figura 5.8, al fine di essere mostrata nella tabella.

A partire dalla lista dei dipendenti, deve essere possibile selezionare i singoli elementi, al fine di manipolarli, in accordo con il diagramma dei casi d'uso. A tale scopo, si è pensato all'implementazione dei pulsanti relativi alle operazioni di inserimento, visualizzazione, ed eliminazione di un determinato dipendente. Con l'aggiunta di questi elementi grafici, si giunge al risultato complessivo in Figura 5.13.

Selezionando un qualsiasi dipendente, con il doppio click o con il pulsante di visualizzazione, viene creata un'istanza della classe *VistaDipendente*, presente nella view della cartella *dipendente* (Figura 5.6). L'implementazione di questa classe con-

```
23          self.navbar_layout = QHBoxLayout()
24
25          self.show_all_button = QPushButton("Mostra tutto")
26          self.show_all_button.clicked.connect(self.update_table_view)
27          self.navbar_layout.addWidget(self.show_all_button)
28
29          self.search_label = QLabel("Cerca tra i dipendenti:")
30          self.navbar_layout.addWidget(self.search_label)
31          self.search_bar = QLineEdit("Nome Cognome")
32          self.navbar_layout.addWidget(self.search_bar)
33
34          self.navbar_layout.addStretch()
35          self.search_bar.returnPressed.connect(self.filter_dipendenti)
```

Figura 5.10. L'implementazione della barra di ricerca dei dipendenti

Figura 5.11. Il risultato dell'implementazione della barra di ricerca dei dipendenti

```
84      def filter_dipendenti(self):
85          self.table_view.clearContents()
86          self.table_view.model().removeRows(0, self.table_view.rowCount())
87
88          filter_list = []
89          for dipendente in self.controller.get_lista_dipendenti():
90
91              if self.search_bar.text().upper() in dipendente.nome.upper() + \
92                      " " + dipendente.cognome.upper() \
93                      or dipendente.nome.upper() + " " + dipendente.cognome.upper() \
94                      in self.search_bar.text().upper():
95                  filter_list.append(dipendente)
96
97          self.table_view.setRowCount(len(filter_list))
98          self.table_view.setColumnCount(8)
99          self.show_table_view_items(filter_list)
```

Figura 5.12. Algoritmo utilizzato per cercare un dipendente specifico all'interno della lista nella quale è contenuto

siste in un insieme di elementi grafici, disposti in un layout a griglia (`QGridLayout`), finalizzati alla visualizzazione, modifica ed eliminazione dei dati di un dipendente. La porzione iniziale del codice corrispondente a questi elementi è mostrata in Figura 5.14, mentre il risultato grafico che si ottiene in output è quello presente in Figura 5.15.

5.2.2 Model

Per permettere l'interfacciamento tra gli utenti di un prodotto software ed i suoi dati, è necessario che questi vengano mappati all'interno di oggetti specifici, nell'ambiente di sviluppo integrato che si sta utilizzando. Solo in questo modo è possibile

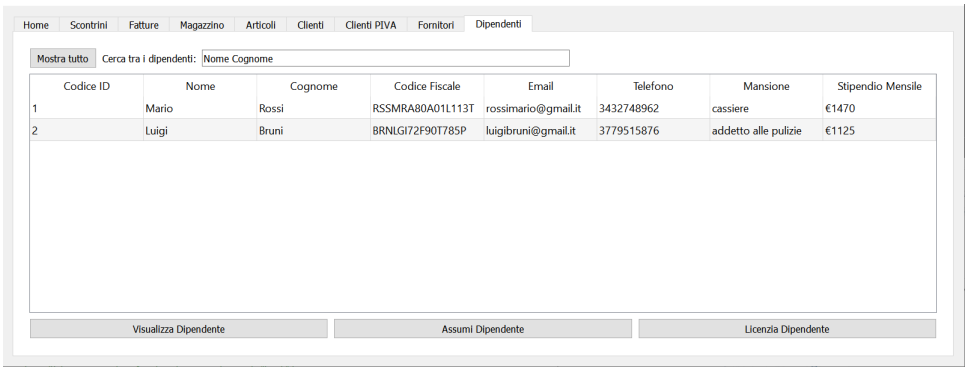

Figura 5.13. Risultato ottenuto dall'implementazione della view relativa alla lista dei dipendenti

```
20          grid_layout = QGridLayout()
21
22          label_codice_id = QLabel("Codice ID: " + str(self.controller.get_codice_id_dipendente()))
23          grid_layout.addWidget(label_codice_id, 0, 0)
24
25          self.label_nome = QLabel("Nome: " + str(self.controller.get_nome_dipendente()))
26          grid_layout.addWidget(self.label_nome, 1, 0)
27
28          button_modifica_nome = QPushButton("Modifica Nome")
29          button_modifica_nome.clicked.connect(lambda: self.show_modifica_dipendente("Modifica Nome"))
30          grid_layout.addWidget(button_modifica_nome, 1, 1)
```

Figura 5.14. Parte dell'implementazione degli elementi grafici utili alla manipolazione di un singolo dipendente

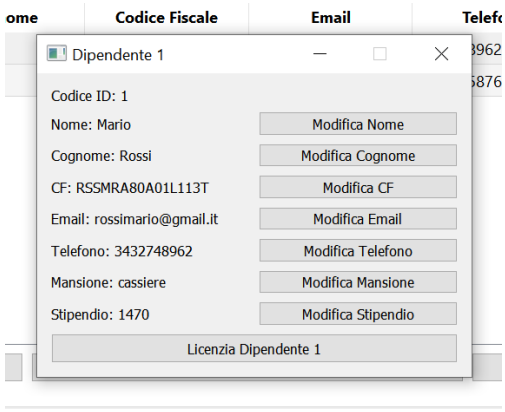

Figura 5.15. Interfaccia grafica per poter gestire un determinato dipendente

manipolarli tramite funzioni complesse, con l'obbiettivo di produrre un risultato da passare alla view per la sua visualizzazione.

Pertanto, tutte le informazioni da dover gestire nel software dedicato alla cartolibreria devono essere mappate all'interno di oggetti di una classe del model.

Riprendendo l'esempio dei dipendenti, si è visto come sia la cartella *dipendente* (Figura 5.6), sia *listadipendenti* (Figura 5.7) hanno un proprio model. Infatti, si fa riferimento al primo di questi quando bisogna manipolare i dati di un singolo dipendente, ad esempio, nel caso in cui fosse necessario accedere allo stipendio o al codice fiscale di un dipendente specifico. Invece, si fa riferimento al model presente in *listadipendenti* quando si ha la necessità di accedere ad un dipendente che è contenuto all'interno della lista. Date queste considerazioni, la classe del model dei singoli dipendenti deve essere implementata come in Figura 5.16. In essa si può notare che ogni singolo dato viene memorizzato all'interno di un attributo di classe specifico. Invece, il model relativo alla lista dei dipendenti è costituito unicamente da una lista di elementi, sulla quale possono essere eseguiti dei metodi specifici, come quelli presenti in Figura 5.17.

```
5   class Dipendente():
6       def __init__(self, codice_id, nome, cognome, cf,
7                    email, telefono, mansione, stipendio_mensile):
8           super(Dipendente, self).__init__()
9
10          self.codice_id = codice_id
11          self.nome = nome
12          self.cognome = cognome
13          self.cf = cf
14          self.email = email
15          self.telefono = telefono
16          self.mansione = mansione
17          self.stipendio_mensile = stipendio_mensile
```

Figura 5.16. Gli attributi utilizzati per mappare ogni dipendente presente nel sistema

```
46      def aggiungi_dipendente(self, dipendente):
47          self.lista_dipendenti.append(dipendente)
48
49      def get_dipendente_by_id(self, id):
50          for dipendente in self.lista_dipendenti:
51              if id == dipendente.codice_id:
52                  return dipendente
53
54      def rimuovi_dipendente_by_id(self, id):
55          for dipendente in self.lista_dipendenti:
56              if id == dipendente.codice_id:
57                  self.lista_dipendenti.remove(dipendente)
```

Figura 5.17. Alcuni metodi presenti nel model della lista dei dipendenti

5.2.3 Controller

Il controller è quel componente dell'architettura MVC che serve per elaborare i comandi dati da un utente, in seguito alla sua interazione con l'interfaccia grafica, e modificare, di conseguenza, il model e/o la view.

Nel software che si sta sviluppando non sono presenti elaborazioni particolarmente complesse, eccetto nei casi d'uso del magazzino, la cui implementazione però, viene trattata nella prossima sezione.

Riprendendo l'esempio dei dipendenti, come per gli altri moduli dell'architettura, anche in questo caso, è necessaria l'implementazione dei due controller relativi alle cartelle *dipendente* (Figura 5.6) e *listadipendenti* (Figura 5.7). Il primo di questi, viene utilizzato per inizializzare un oggetto del model corrispondente, sul quale possono essere applicati i metodi di classe rappresentati in Figura 5.18.

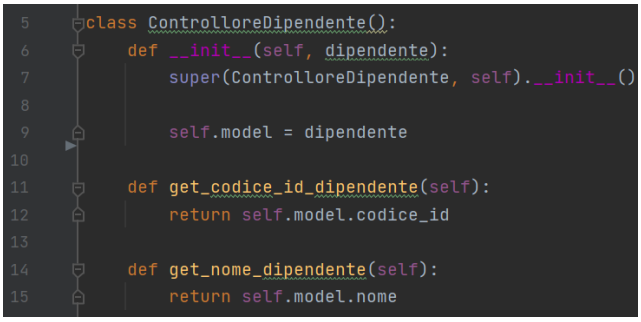

Figura 5.18. Porzione iniziale della classe relativa al controller dei dipendenti

Il secondo controller, relativo alla cartella *listadipendenti*, viene utilizzato per eseguire operazioni simili al primo, ma relativi alla lista dei dipendenti, e non ai suoi singoli elementi. In Figura 5.19 è riportata la porzione iniziale di quest'ultima classe.

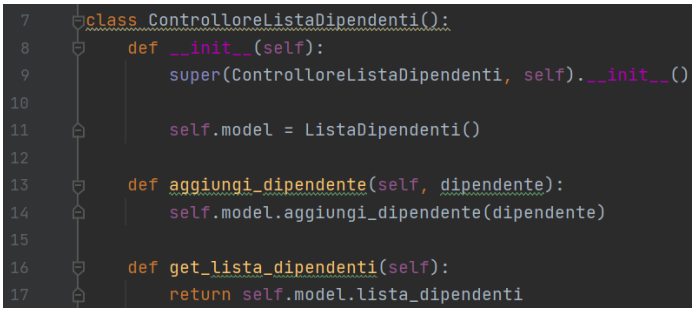

Figura 5.19. Porzione iniziale della classe relativa al controller dei dipendenti

5.3 Gestione del magazzino

I casi d'uso della gestione del magazzino sono diversi da quelli relativi alle altre aree del software. Infatti, le funzionalità che devono essere introdotte, in questo caso, riguardano il controllo della quantità degli articoli disponibili per la vendita.

Una volta che un nuovo articolo viene inserito all'interno del sistema, la sua quantità viene impostata di default a zero. Per poterla incrementare, è necessaria l'esecuzione di un'operazione di carico del magazzino. Invece, quando la quantità di un articolo è maggiore di zero, sarà possibile eseguire un'operazione di scarico.

Ogni volta che la cartolibreria acquisisce merce, il fornitore deve rilasciare una fattura di carico (Figura 5.20) che, quando viene memorizzata nel sistema, attiva la corrispondente funzione, mostrata in Figura 5.21. Quest'ultima consiste in un algoritmo che cerca l'articolo da caricare, all'interno della lista nella quale è contenuto, e somma la sua attuale quantità con quella che è stata inserita nella fattura. Inoltre, tale funzione restituisce al chiamante un valore booleano, che indica se l'algoritmo è stato eseguito correttamente o meno.

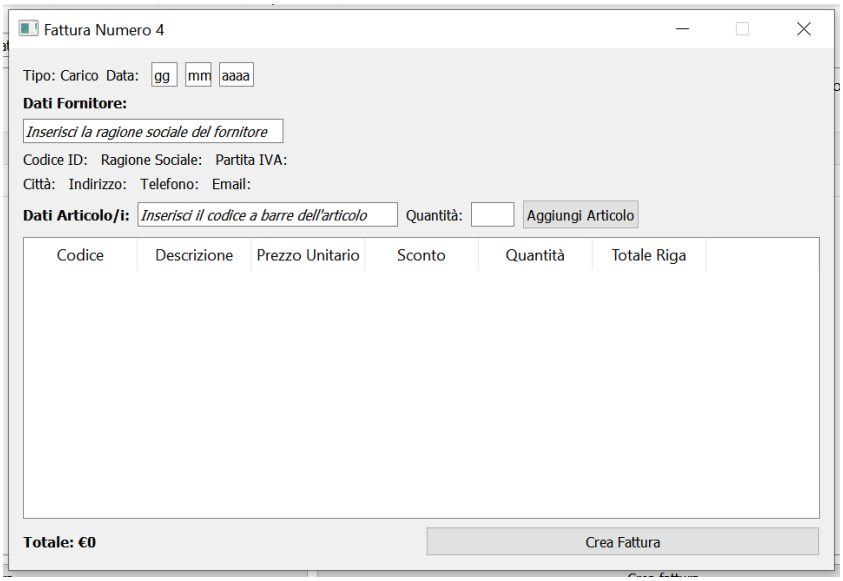

Figura 5.20. Interfaccia grafica di una fattura di carico, scritta con la libreria PyQt5

Nel momento in cui un cliente acquista uno o più articoli, deve essere memorizzato nel sistema uno scontrino (Figura 5.22) o una fattura di scarico (Figura 5.23). In seguito al click sul pulsante corrispondente alla creazione della ricevuta, deve essere conseguentemente eseguita la funzione di scarico (Figura 5.24). Anche in questo caso, come per la funzione di carico, si attiva un algoritmo che ricerca l'articolo da scaricare all'interno della lista nella quale è contenuto. Tuttavia, è necessario un passo aggiuntivo, ossia verificare che la quantità presente in magazzino sia maggiore o uguale di quella inserita nella ricevuta. In caso contrario, la funzione restituirà il valore booleano `False`, e l'operazione fallirà.

```
34      def inserimento_carico (self, codice_barre, stock):
35          for articolo in self.get_lista_articoli():
36              if codice_barre == articolo.codice:
37                  controller_articolo = ControlloreArticolo(articolo)
38                  getstock=controller_articolo.get_stock_articolo()
39                  totstock= getstock + int(stock)
40                  controller_articolo.set_stock_articolo(totstock)
41                  return True
42          return False
```

Figura 5.21. Funzione del controller che si attiva ad ogni operazione di carico

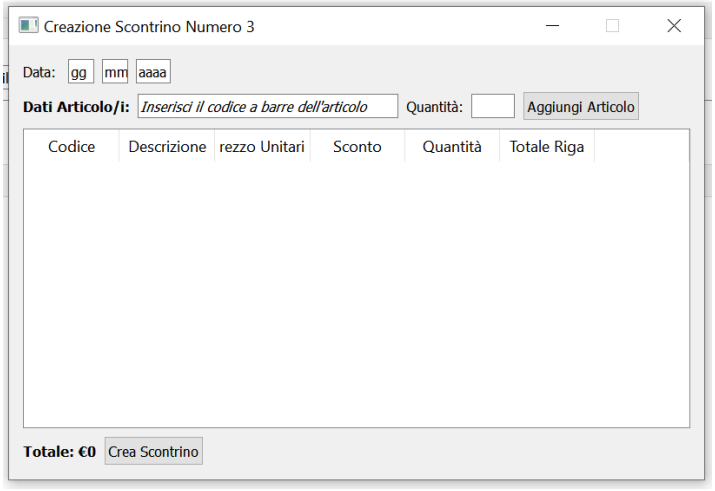

Figura 5.22. Interfaccia grafica di uno scontrino, sviluppata tramite la libreria PyQt5

Si ritiene necessaria l'implementazione delle operazioni di carico e scarico anche quando non viene generata nessuna fattura o scontrino. Infatti, al momento dell'installazione del software, ci si aspetta che nella cartolibreria sia già presente della merce acquisita e pronta per essere venduta ai clienti. Pertanto, per gli stakeholder risulta più comodo aumentare la quantità degli articoli, evitando di dover inserire tutte le fatture precedentemente emesse dai fornitori. Inoltre, per motivi diversi, potrebbe essere necessario togliere dalla vendita determinati articoli, e quindi doverne diminuire la quantità all'interno del software. Alla luce di queste considerazioni, si introduce la possibilità di eseguire le operazioni di carico e scarico con dei pulsanti appositi, implementati nell'area del software relativa al magazzino. In Figura 5.25 è rappresentata l'interfaccia per l'esecuzione manuale dell'operazione di carico, a partire dalla quale, una volta inseriti il codice a barre e la quantità dell'articolo, in seguito al click sul bottone "*Carica articolo*" viene eseguita la funzione mostrata in Figura 5.21. Il discorso è analogo per l'operazione manuale di scarico.

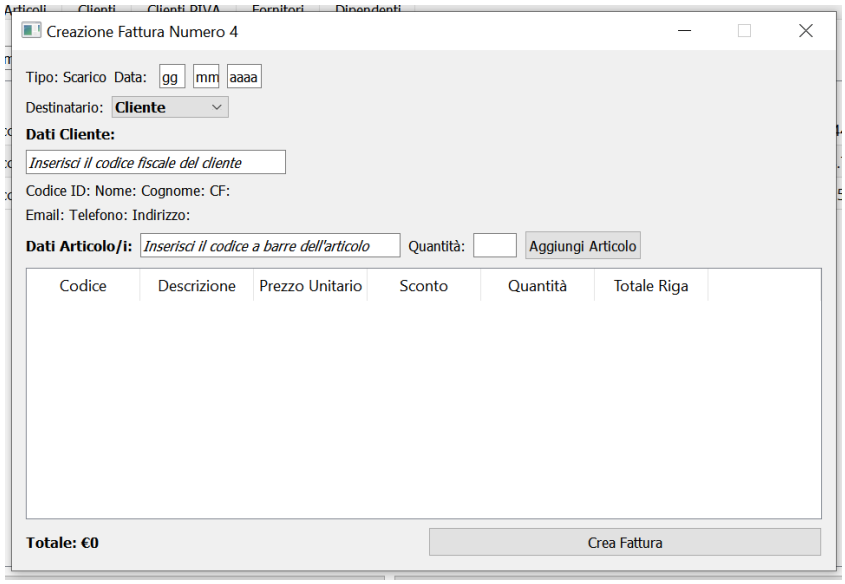

Figura 5.23. Interfaccia grafica di una fattura di scarico, sviluppata tramite la libreria PyQt5

```
49      def scarico (self, codice_barre, stock):
50          for articolo in self.get_lista_articoli():
51              if codice_barre == articolo.codice:
52                  controller_articolo = ControlloreArticolo(articolo)
53                  getstock = controller_articolo.get_stock_articolo()
54                  totstock = getstock - int(stock)
55                  if totstock < 0:
56                      return False
57                  controller_articolo.set_stock_articolo(totstock)
58                  return True
59          return False
```

Figura 5.24. Funzione del controller che si attiva ad ogni operazione di scarico

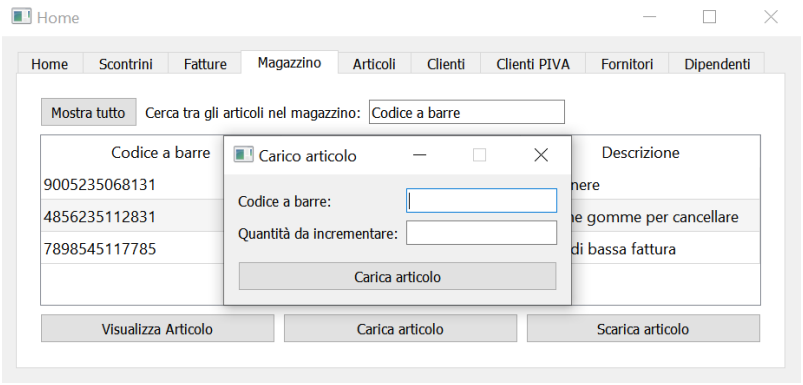

Figura 5.25. Interfaccia grafica per l'esecuzione manuale dell'operazione di carico

6

Discussione in merito al lavoro svolto

In questo capitolo viene applicata una SWOT Analysis del software implementato, con l'obbiettivo di verificarne gli aspetti positivi e negativi. Successivamente, vengono descritte le lezioni apprese dal lavoro svolto, al fine di estrapolare delle best practise da poter utilizzare nei progetti futuri simili.

6.1 SWOT Analysis

L'analisi SWOT (in inglese, *SWOT Analysis*) è uno strumento di pianificazione strategica usato per valutare i punti di forza (*Strengths*), le debolezze (*Weaknesses*), le opportunità (*Opportunities*) e le minacce (*Threats*) di un progetto, in un'impresa, o in ogni altra situazione in cui un'organizzazione o un individuo debba prendere una decisione per il raggiungimento di un obiettivo.

I punti di forza e le debolezze sono caratteristiche intrinseche ad un progetto, o ad un'azienda. Infatti, sono aspetti sui quali si dispone di un certo controllo, e che si è in grado di migliorare qualora risultassero troppo fragili. Invece, le opportunità e le minacce sono questioni esterne ad un progetto, ma che incidono comunque sul suo ciclo di vita. Per questo motivo, è possibile cogliere le opportunità o proteggersi dalle minacce, ma non è possibile modificare nessuna delle due.

Al fine di rappresentare graficamente la suddetta schematizzazione, si può considerare una matrice 2×2, chiamata *matrice SWOT* (Figura 6.1), in cui i fattori interni ed esterni che hanno un potenziale impatto, positivo o negativo, sul progetto che si è voluto realizzare, sono opportunamente identificati ed organizzati.

Il software gestionale prodotto, oltre ad essere stato implementato per scopi didattici, si trova nella sua primissima versione. Ciò significa che quest'ultimo, nella pratica, avrà sicuramente dei problemi da dover risolvere e degli aspetti da poter migliorare. Allora, si può pensare di applicare un'analisi SWOT al fine di disporre di una documentazione da poter utilizzare con l'obbiettivo di facilitare eventuali evoluzioni future.

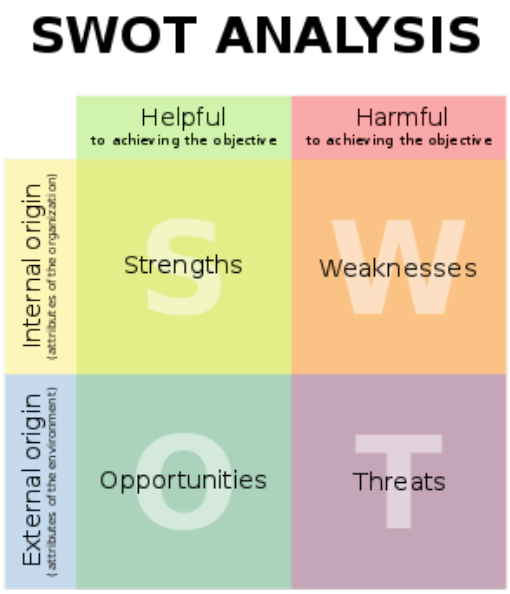

Figura 6.1. La matrice SWOT, utilizzata per schematizzare il risultato dell'analisi SWOT.

6.1.1 Punti di forza

Una cartolibreria può essere vista banalmente come un magazzino in cui gli articoli entrano ed escono continuamente, in seguito ai processi di acquisizione della merce all'ingrosso, e di vendita ai clienti.

Fin da subito, si è voluto sviluppare un prodotto software che fosse focalizzato su questa dinamica, rendendola efficiente, e al tempo stesso, semplice da eseguire per gli stakeholder. Si ritiene, pertanto, che il suo punto di forza principale sia l'insieme di automatismi che vengono attivati nella gestione del magazzino, quando si devono gestire gli articoli in entrata e quelli in uscita. Infatti, ogniqualvolta si vuole inserire una ricevuta nel sistema, gli articoli che la caratterizzano devono essere già stati registrati in esso. Inoltre, ad ogni vendita, la ricevuta rilasciata viene memorizzata solo se in magazzino ci sono sufficienti articoli da scaricare.

Dal punto di vista tecnico, il vantaggio del software sviluppato è rappresentato dalla sua modularità. Con l'utilizzo del pattern architetturale MVC si è potuto implementare un software modulare, quindi costituito da più blocchi di codice che comunichino tra loro. In questo modo è stata favorita la sua manutenibilità, poiché, qualora fosse necessario introdurre dei cambiamenti nel codice implementato, basterebbe modificare unicamente i componenti interessati, lasciando gli altri inalterati.

6.1.2 Punti di debolezza

La maggiore debolezza del software sviluppato è, molto probabilmente, il suo aspetto grafico. In effetti, nonostante il software in questione sia facile da usare, la presenza

di un'interfaccia grafica che utilizzi gli elementi nativi del sistema operativo su cui essa viene avviata, non risulta particolarmente gradevole dal punto di vista estetico.

Di conseguenza, in futuro, si può pensare di migliorare questo aspetto modificando le view del software, continuando ad utilizzare le funzionalità della libreria PyQt5, oppure utilizzando altri toolkit. Qualora dovesse essere necessario, si potrebbe pensare di chiedere agli stakeholder un piccolo investimento monetario, per consentire l'implementazione della GUI (Graphical User Interface) attraverso i framework più moderni (come, ad esempio, Delphi VCL), e quindi, garantire inevitabilmente un miglioramento estetico del software utilizzato.

6.1.3 Opportunità

Il software sviluppato, grazie alla sua architettura modulare, può avere molte opportunità di miglioramento, a partire dall'introduzione di piccoli dettagli fino all'aggiunta di nuove funzionalità e alla comunicazione con altri sistemi. In effetti, nella fase di specifica del software, nel Capitolo 2, si è ipotizzato di mettere in comunicazione il software stesso con quattro sistemi esterni: il DBMS (Database Management System), il lettore del codice a barre, il registratore di cassa e il sistema di interscambio dell'agenzia delle entrate. In questa tesi, è stato progettato un unico aspetto tra quelli presentati, ossia il DBMS, nel Capitolo 3. Tuttavia, ci si aspetta di implementare, nelle release future, tutte le interfacce necessarie per consentire la comunicazione del software con gli altri tre sistemi esterni, a partire dal lettore del codice a barre, senza il quale si ha che il processo di inserimento degli articoli all'interno di una ricevuta deve essere realizzato manualmente, risultando piuttosto lento.

6.1.4 Minacce

Il prodotto in questione è stato sviluppato unicamente per gli stakeholder che ne hanno commissionato il progetto. Per questo motivo, si tratta di un software che è pronto ad essere utilizzato nell'istante in cui viene installato nel sistema della cartolibreria.

Questo vantaggio può avere un lato negativo, ovvero non esiste un modo con cui controllare le autorizzazioni di chi utilizza il software. Quindi, se dovesse essere inavvertitamente scaricato un malware sulla stessa macchina nella quale è installato anche il software gestionale, allora i suoi dati sarebbero soggetti a seria minaccia.

Inoltre, qualora venissero introdotte delle funzionalità che prevedano di utilizzare la rete internet per la comunicazione del software con un altro sistema, sarebbe necessario pianificare dei meccanismi di sicurezza per proteggere la cartolibreria da eventuali attacchi informatici.

6.2 Lezioni apprese

Alla fine della realizzazione di un prodotto software, è buona norma dedicare del tempo per analizzare il lavoro svolto, con lo scopo di redigere un documento delle

lezioni apprese (*lesson learned*). Tipicamente, questo documento viene messo in una raccolta, detta *registro delle lezioni apprese* (*lesson learned register*), che, con il tempo, costituisce un vero e proprio know-how dell'azienda che lo utilizza.

Documentare le lezioni apprese, durante o dopo la conclusione di un progetto, significa investire del tempo per analizzare e documentare i successi e gli insuccessi del progetto svolto, in modo da trarne delle conclusioni mirate a rendere l'organizzazione dei prossimi progetti sempre più efficace ed efficiente, rispettando i tempi ed il budget assegnato.

Purtroppo, si tratta di un'abitudine che spesso viene trascurata in favore dell'avanzare costante del progetto al quale si sta lavorando.

Tuttavia, l'utilizzo di uno storico che possa documentare e registrare le lezioni apprese aiuta il project manager ed il suo team di lavoro ad aumentare la capacità realizzativa futura, ed evitare di ripetere gli errori che hanno avuto un impatto negativo nei progetti passati. Infatti, tale documentazione richiede un investimento di tempo, che, però, nel lungo periodo, si traduce in un risparmio di tempo, il quale è determinato proprio dalla capacità di non ripetere gli stessi errori, e di utilizzare ciò che produce dei risultati positivi.

Secondo il report PPM Benchmark study, di Axelos, circa la metà dei project manager che non effettua, o effettua raramente, revisioni di progetto ha un'esperienza di insuccesso nell'arco di un anno, rispetto al 34% di coloro che effettuano sempre o spesso revisioni. Questo fa comprendere che il concetto di lezioni apprese dovrebbe essere parte integrante della gestione di un progetto.

Il prodotto sviluppato è un software gestionale per permettere l'esecuzione, in maniera facile ed efficiente, delle funzionalità richieste dai titolari della cartolibreria Compucart. Per lo sviluppo del prodotto in questione, è stato utilizzato un processo software "plan-driven". Quindi, a monte della sua implementazione, è stato fatto un meticoloso lavoro di raccolta dei requisiti e di progettazione. In questo modo si sono potute portare alla luce, e poi risolvere, eventuali problematiche, prima della fase d'implementazione, riducendo al minimo i danni causati dall'apporto di modifiche.

In fase di specifica del software ci si è affidati ad un'intervista aperta, che da sola, però, si è rivelata insufficiente per la deduzione dei requisiti. Quindi, per ottenere delucidazioni, si sono dimostrati necessari successivi contatti telefonici con gli stakeholder. Si sarebbero potuti evitare tali confronti nel caso in cui fosse stata effettuata un'etnografia a supporto dell'intervista.

Invece, in fase di progettazione, si è optato per l'utilizzo del pattern architetturale MVC, che si è rivelato un grosso vantaggio poiché ha permesso la suddivisione del prodotto software in più moduli, uno per ogni suo aspetto. In questo modo, si è potuto ottenere un codice ordinato, leggibile, modulare, e facilmente manutenibile, tutte caratteristiche tipiche di un software ben realizzato.

Infine, Python è un linguaggio di programmazione semplice da utilizzare, grazie al quale si è potuto realizzare il software in questione nel minor tempo possibile. Si tratta del linguaggio di programmazione attualmente più in voga; perciò, risulta semplice riuscire a trovare un programmatore che sia in grado di leggerlo ed utilizzarlo per il processo di evoluzione.

7

Conclusioni

In questa tesi è stato adoperato il processo "plan-driven" dell'ingegneria del software con l'obiettivo di progettare ed implementare un sistema standalone per la gestione di una cartolibreria.

A tale scopo, il primo passo affrontato è stato quello di introdurre i concetti cardine dell'ingegneria del software, ripercorrendo rapidamente la sua storia, e spiegando perché è stata necessaria la nascita di questa disciplina informatica per lo sviluppo dei prodotti software moderni. In questo modo, è stato possibile spiegare che cosa sono i processi "plan-driven", fornendo esempi di quando risultano vantaggiosi rispetto ai più moderni processi agili.

In seguito a questa introduzione, si è proceduto alla documentazione del progetto in esame, partendo dalla fase di specifica del software, e raccogliendo tutte le informazioni da utilizzare per l'estrapolazione dei requisiti del sistema. Questa fase è stata particolarmente rilevante per comprendere quali dovevano essere le funzionalità da mettere a disposizione degli stakeholder nel software da implementare. Infatti, a partire dai requisiti del sistema, si giunge alla fase di progettazione del software, la quale rappresenta il fulcro dei processi "plan-driven".

Nonostante sia stata effettuata una progettazione completa del software in questione, nella presente tesi sono state documentate unicamente la progettazione del database e quella relativa ai casi d'uso.

Il sistema gestionale ha preso vita solo in seguito alla fase di implementazione del software, in cui è stato discusso sinteticamente il linguaggio di programmazione sul quale si è fatto affidamento, cioè Python, e sono stati esplicati, per motivi di brevità, i punti salienti del codice utilizzato.

L'ultima attività effettuata è stata quella relativa all'analisi del lavoro svolto. Anche se quest'ultima fase non viene presa in considerazione in molti processi software moderni, a causa del fatto che incide sul tempo ed il budget preventivati per il progetto, risulta comunque un passo critico nella determinazione del successo o dell'insuccesso dei progetti futuri.

Il prodotto finale è un software gestionale completamente funzionante che rispetta i requisiti richiesti, pronto per essere installato ed utilizzato all'interno del sistema della cartolibreria Compucart.

Tuttavia, il ciclo di vita di un prodotto software non termina con il suo rilascio. Questo significa che il software sviluppato dovrà essere soggetto a manutenzioni

future, finalizzate alla risoluzione di bug e/o problematiche varie che inevitabilmente sorgeranno con il suo utilizzo. In particolare, ci si aspetta che gli stakeholder richiedano l'introduzione di nuovi requisiti per sopperire alla mancanza di alcune funzionalità che potrebbero velocizzare il processo lavorativo.

Pertanto, si ritiene di avere a disposizione una documentazione sufficiente da utilizzare per i futuri processi di evoluzione del software.

Riferimenti bibliografici

1. PyQt5 5.15.4. `https://pypi.org/project/PyQt5/`, 2021.
2. Sistema di interscambio. `https://www.agenziaentrate.gov.it/portale/web/guest/schede/fabbricatiterreni/sistema-di-interscambio/come-aderire-al-sistema`, 2021.
3. C. Batini, B. Pernici, and G. Santucci. *Sistemi informativi. Vol. 5: Sistemi distribuiti*. FrancoAngeli, 2004.
4. M. Beri. *Python 3: Guida tascabile al linguaggio di Google, Star Wars e la NASA*. Apogeo, 2010.
5. M. Buttu. *Programmare con Python: Guida completa*. Edizioni LSWR, 2014.
6. N. Ceder. *Python: Guida alla sintassi, alle funzionalità avanzate e all'analisi dei dati*. Apogeo, 2019.
7. A. De Lucia and G. Tortora. *Metamorphos: Methods and Tools for Migrating Software Systems Towards Web and Service Oriented Architectures : Experimental Evaluation, Usability and Technology Transfer*. Rubbettino Editore, 2010.
8. A.B. Downey. *Pensare in Python*. Egea, 2018.
9. L. Favre. *UML e Unified Process*. Liliana Favre, 2003.
10. M. Fowler. *UML distilled. Guida rapida al linguaggio di modellazione standard*. Pearson Education (Pearson), 1997.
11. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
12. IEEE Computer Society (IEEE). Guide to the Software Engineering Body of Knowledge (SWEBOK Guide): Version 3.0. 2020.
13. K.A. Lambert. *Programmazione in Python*. Maggiori Editore, 2012.
14. M.A. Lutz. *Imparare Python*. Hoepli, 2008.
15. D. Mandrioli, M. Jazayeri, and C. Ghezzi. *Ingegneria del software: fondamenti e principi*. Pearson Education (Pearson), 2004.
16. R.C. Martin. *Clean Architecture: Guida per diventare abili progettisti di architetture software*. Apogeo, 2017.
17. R.C. Martin. *Clean Code: Guida per diventare bravi artigiani nello sviluppo agile di software*. Apogeo, 2018.
18. R.D. Necaise and C.S. Horstmann. *Concetti di informatica e fondamenti di Python*. Maggiori Editore, 2013.
19. M.T. Nygard. *L'arte del Rilascio: Progettazione e deploy di software che funziona*. Apogeo, 2018.
20. R. Pooley and P. Stevens. *Usare UML. Ingegneria del software con oggetti e componenti*. Pearson Education (Pearson), 2007.

21. R.S. Pressman. *Principi di ingegneria del software*. La Feltrinelli, 1991.
22. C. Savy and L. Sansone. *Ingegneria del software. Tecniche di sviluppo del software*. Liguori Editore, 1994.
23. I. Sommerville. *Software Engineering*. Pearson Education (Pearson), 2015.
24. A. Sweigart. *Automatizzare le cose noiose con Python: Programmazione pratica per principianti assoluti*. Edizioni LSWR, 2015.
25. J.D. Ullman. *Basi di dati e basi di conoscenza*. Jackson Libri, 1991.

Ringraziamenti

Ricordo ancora, senza alcuna nostalgia, i miei primi giorni di Università. Allora la mia impressione era quella di trovarmi al cospetto di una montagna apparentemente troppo alta da poter superare in soli tre anni.

Tuttavia, anche se, inizialmente, come gran parte dei miei colleghi, ho fatto fatica a comprendere le nozioni spiegate durante le lezioni, non mi sono voluto arrendere ai primi ostacoli.

Dopo un primo anno traballante sono riuscito a prendere il ritmo e a non fermarmi fino al superamento di tutti gli esami.

Il merito di ciò è da dare, innanzitutto, ai professori della facoltà, i quali si sono sempre dimostrati estremamente competenti e disponibili per chiarimenti, ma anche alla mia famiglia ed ai miei amici, per il loro supporto morale. In merito a questo, ritengo necessari alcuni ringraziamenti.

Ringrazio prima di tutto i miei famigliari, ed in particolare i miei genitori, per i loro sacrifici economici fatti affinché io abbia potuto frequentare l'Università.

Ringrazio tutti i miei colleghi, ed in particolare Antonio, Francesco, Carmen, Francesca e Chiara, per essere stati sempre disponibili a condividere appunti e a darmi consigli ogniqualvolta ne avessi avuto bisogno.

Ringrazio il Rappresentante degli studenti della facoltà di Ingegneria, Alex, per avermi sempre aiutato tutte le volte che sono sorti problemi in merito a questioni legate all'università.

Ringrazio i miei due ex coinquilini Antonio e Fabrizio, per non avermi mai fatto annoiare, e per non aver creato nessun dissidio durante la convivenza.

Ringrazio tutti i miei amici molisani, ed, in particolare, Luca, Simone, Giuseppe, Piergiuseppe, Razvan, Angelo, Manuel, Marialuisa, Patrizia e Sara, per aver portato pazienza nei periodi in cui non mi sono fatto vivo a causa dei miei impegni. Insieme a loro ringrazio Lorenzo, Carlo, Oscar e Matteo, per i loro preziosi consigli di carriera.

Un ringraziamento speciale va al relatore di questa tesi, il Professore Domenico Ursino, per la sua grande pazienza, ed ancor più grande disponibilità.

Inoltre, non potrei dimenticare di ringraziare me stesso, per aver sempre percorso la strada più difficile poiché è quella che un giorno si rivelerà la più redditizia.

Infine, vorrei concludere questa tesi ringraziando l'Università Politecnica delle Marche, che mi ha insegnato, al di là della moltitudine di nozioni tecniche relative al percorso di studi, ad avere la consapevolezza che se mi impegno a sufficien-

za, con determinazione e perseveranza, nessuna montagna è troppo alta per essere sormontata.