UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

DISEGNO E SVILUPPO DI UN'APPLICAZIONE PER L'ANALISI STORICA DI VARIABILI GESTIONALI

Laureando: *Matteo Salvà 579929 - IF*

Relatore: *Ch.mo Massimo Rumor*

Anno Accademico 2012-2013

Ringraziamenti

Si vuole rivolgere un ringraziamento particolare al relatore Massimo Rumor, per la disponibilità e la professionalità dimostratami durante la stesura di questo elaborato e in generale durante tutto il processo che ha portato a questo lavoro, tirocinio compreso.

Assieme e in parallelo volevo ringraziare anche l'intero staff aziendale per la possibilità concessami, in particolare i tutor, Cristina e Federico, che mi hanno assistito nel periodo di stage formativo, per le informazioni e le conoscenze condivise, oltre che per la competenza e l'appoggio dimostratomi.

Per quanto riguarda la sfera personale, di ringraziamenti ce ne sarebbero molti da fare, ma sono sicuro che ognuna delle persone che vorrei ringraziare ne è consapevole e la mera elencazione in questa sezione potrebbe risultare stucchevole e mielosa.

In ogni caso: GRAZIE.

Sommario

Con la presente si vuole relazionare il lavoro di tirocinio di 500 ore svolto presso la ditta Antea srl sita in Padova. Tale lavoro è stato suddiviso in più parti:

- Una prima parte di analisi, studio e comprensione delle tecniche di ingegneria del software, alcune già studiate in parte teorica nel corso relativo, ma meglio assimilate grazie all'utilizzo di tool concreti nell'ambito della progettazione, anche attraverso piccolissimi elaborati per apprenderne il funzionamento.
- Una seconda parte di analisi dei requisiti, reperiti attraverso alcuni colloqui all'interno dell'azienda con i responsabili del reparto commerciale, che in questo frangente risultavano essere i committenti del progetto.
- Appresi i requisiti richiesti, la terza parte del tirocinio è stata affrontata per reperire informazioni sul gestionale aziendale, considerando che era richiesto un applicativo che potesse interfacciarsi ad esso e reperire alcune informazioni legate alle variabili storiche aziendali.
- Una quarta parte attraverso cui si prende conoscenza delle specifiche tecniche e librerie che sarebbero state poi utilizzate nella stesura del programma attraverso piccoli elaborati di entità molto ridotta.
- Un'ultima parte di stesura di codice vero e proprio e rapida presentazione conclusiva agli utenti finali, per dimostrare il funzionamento del programma e del lavoro svolto.

Lo stage formativo di cui sopra sono elencati gli aspetti cardine è stato completato con profitto nel periodo che va dal 27 Luglio 2010 all'11 Marzo 2011 previo accordo con il relatore Massimo Rumor e i tutor aziendali.

La tesi consiste nella descrizione di tutti questi punti suddividendoli in due macrosezioni che sono lo studio delle tecniche di ingegneria del software e la realizzazione del progetto vero e proprio.

Indice

Capitolo 1

Introduzione

Un ingegnere informatico è un professionista competente nella progettazione, realizzazione e gestione di sistemi informatici complessi e non un semplice utilizzatore di questi strumenti e dei sistemi stessi.

A questo proposito, ciò che caratterizza un ingegnere informatico è sicuramente lo studio e l'utilizzo di quella disciplina che prende il nome di ingegneria del software.

L'ingegneria del software è una disciplina atta a sviluppare un sistema informatico di alta qualità ma senza alcuno spreco, regolando il processo di produzione di un prodotto software in tutte le sue fasi, dalla progettazione e analisi fino alla consegna e distribuzione, passando anche per lo sviluppo. Ciò che caratterizza infatti un ingegnere informatico è la presa di coscienza che la progettazione software non comprende solo il suo mero sviluppo, ma è una attività molto più complessa che prevede, come primo punto, di comprendere il funzionamento dell'intero sistema su cui verrà installato il nuovo prodotto. Un vecchio detto dice che "guardando gli alberi non riuscirai a vedere la foresta", che se portato nel campo dell'informatica esprime molto bene il concetto appena sviluppato: la foresta è il sistema e gli alberi sono gli elementi tecnologici (software incluso) necessari a realizzare il progetto. Se ci si precipita a realizzare gli elementi tecnologici prima di comprendere il sistema, si commetteranno inevitabilmente degli errori che deluderanno il cliente e comprometteranno la riuscita del progetto.[5]

Lo scopo di questa tesi vuole essere il consolidamento delle tecniche di ingegneria del software e la comprensione degli strumenti adeguati a seguire quelli che sono gli insegnamenti di questa disciplina.

1.1 Obiettivi del tirocinio

Attraverso lo stage formativo in esame, la ditta Antea srl, tramite le figure dei tutor aziendali, intende preparare e formare il tirocinante in quelle che sono alcune caratteristiche e conoscenze ritenute fondamentali in ambito lavorativo, ampliando e dando un significato più concreto alle teorie studiate durante il corso di studio.

In particolare gli obiettivi primari sono quelli di osservare e studiare i processi e gli strumenti di una azienda di software, dando particolare rilievo e attenzione agli strumenti legati alla metodologia di programmazione agile nota come Extreme Programming (o processo XP), ai sistemi di versionamento, ai sistemi di build, al bugtracking, allo sviluppo di test e alla collaborazione in team di sviluppo. L'utilizzo di tali strumenti è da intendersi teso, come ultimo fine, allo sviluppo di applicazioni e progetti atti a raggiungere fattori qualitativi elevati che seguano requisiti standard e che pertanto siano maggiormente apprezzati dai committenti dei progetti stessi.

Per raggiungere la conoscenza di tali processi e meccanismi, al tirocinante, è richiesta inoltre la realizzazione di un progetto software in tutte le sue fasi, dalla raccolta dei requisiti al deployment dell'applicativo, utilizzando i più opportuni tra gli strumenti studiati nella prima parte di analisi.

Capitolo 2

La ditta

Antea, fondata nel 1989, è specializzata nello sviluppo di soluzioni software altamente flessibili e nella fornitura di servizi integrati, che le permettono di essere competitiva in diversi settori industriali quali, Chimico, Petrolchimico, Energia e Utility.

Come azienda privata, dispone di una rete di vendita, di consulenza e di supporto ramificata sul territorio nazionale ed internazionale. Oltre alla sede centrale di Padova e agli uffici regionali di Houston, dell'Arabia Saudita e dell'Algeria, si avvale anche di partner locali in Egitto, in Germania, in Iran e nel mercato nigeriano.

Antea sviluppa con i propri clienti dei veri e propri rapporti di partnership che vanno ben oltre i normali rapporti Cliente/Fornitore. Grazie a queste collaborazioni fondate su stima e fiducia reciproca, ha partecipato a diversi progetti in cui il grande valore aggiunto sono le specifiche e le indicazioni fornite dai clienti.

Le soluzioni software sono sviluppate totalmente ed esclusivamente dai propri collaboratori, garantendo elevata flessibilità e personalizzazione.

Antea assicura ai propri clienti l'impegno delle tecnologie più avanzate nello sviluppo dei propri prodotti e la disponibilità di un pacchetto completo di servizi al fine di ottimizzare il monitoraggio delle attività manutentive e ispettive, contribuendo alla sicurezza del personale e alla sostenibilità delle attività per l'ambiente. Il 3D è il passo tecnologico per una più intuitiva gestione degli asset, per una condivisione della realtà, per facilitare lo scambio di informazioni e per accelerare attività di training.[1]

Parte I

Studio e Analisi delle Tecniche di Ingegneria del Software

Capitolo 3

Elementi di ingegneria del software aziendali

Il software, come un qualsiasi altro oggetto, è il risultato finale di una produzione. In tal senso si può affermare che se nella produzione di un'automobile sono presenti varie fasi come l'acquisto di materie prime, la lavorazione, la catena di montaggio con macchinari atti ad automatizzare il lavoro per renderlo più agevole, più veloce, e quindi più economico, anche per la produzione di software è necessario svolgere più attività, per le quali tuttavia vengono messi a disposizione strumenti che facilitano, non poco, il lavoro del programmatore.

Anche lo sviluppo e l'utilizzo di questi strumenti, il cui scopo è quello di semplificare il lavoro di produzione, massimizzando la qualità del prodotto finito, rientrano in quella categoria denominata ingegnerizzazione del software.

Di seguito saranno elencati alcuni degli aspetti principali dell'ingegneria del software su cui il tirocinante è stato invitato a riflettere nella prima parte del tirocinio e una rapida panoramica del prodotto specifico utilizzato da Antea mettendone in risalto i motivi della scelta.

3.1 Processo XP

Alla base e al principio di una produzione software è necessario stabilire una procedura di azione che aiuti gli sviluppatori ad ottenere un risultato di alta qualità in tempi prefissati.

Nel corso degli anni sono stati sviluppati molti modelli di processi (a cascata, incrementale, prototipale e molti altri) tuttavia dal 2001 ha inizia-

to a prendere piede una metodologia di sviluppo chiamata processo agile, la quale si è dimostrata particolarmente efficiente nel momento in cui i requisiti del processo non fossero particolarmente chiari, o comunque fossero estremamente mutevoli e i tempi di analisi, design e testing non fossero facilmente definibili. Infatti, il processo Agile fa dell'adattabilità il proprio punto di forza privilegiando in particolar modo la pronta risposta ai cambiamenti rispetto all'esecuzione di un piano fissato. Ciò è reso possibile dal fatto che il cliente viene talmente coinvolto nello sviluppo del processo da entrare a far parte del team di sviluppo; questo se da un lato comporta problemi in quanto il cliente richiederà sempre maggiori modifiche e funzionalità, dall'altra risulta un vantaggio perché si riduce il rischio di fallimento,considerando che anche lui sta lavorando al progetto dandone continue specifiche e testandolo in prima persona.

Tra le metodologie Agili, che si sono sviluppate nel tempo, la più diffusa è sicuramente il processo XP (Extreme Programming). Le caratteristiche di questo processo sono sintetizzabili in dodici punti che ne fanno un vero e proprio vade mecum:

1. Pair Programming: Una coppia di programmatori scrive il codice assieme; sebbene possa sembrare poco produttivo, al contrario porta numerosi vantaggi, tra cui il fatto che a prendere le decisioni sono due cervelli, due persone conoscono il codice ed inoltre il codice è costantemente controllato.

2. Refactoring:Consiste nella tecnica di migliorare il codice senza alterarne le funzionalità. Codice più semplice e leggibile implica codice più duraturo.

3. Simple Design: Il codice scritto dovrebbe essere il più semplice che possa funzionare, dovrebbe essere facilmente testabile, non contenere duplicati , evidenziare facilmente quale era l'intento del programmatore, spiegabile velocemente a eventuali nuovi membri del team.

4. Collective Code Ownership: Tutte le persone del team sono responsabili di tutto il codice, pertanto possono attuare modifiche e migliorie ad esso.

5. Continuous Integration: Gli sviluppatori dovrebbero integrare e inviare il codice nella repository molto spesso, evitando così il diversificarsi o il frammentarsi del codice stesso, rendendo più veloce ed economico l'assemblaggio dell'intero programma. Queste integrazioni vengono effettuate tramite server dedicati, in Antea viene utilizzato Hudson.

6. On-Site Customer: come tutti i processi agili anche xp richiede che il cliente faccia parte del team di sviluppo.

7. Small Release: Rilasci rapidi e ravvicinati di codice permettono un feedback immediato del cliente oltre a stabilire quali siano le funzionalità necessarie e quali no.

8. 40-Hour week: Un programmatore non dovrebbe lavorare troppo; Il numero delle ore non è importante tuttavia programmatori stanchi commettono più errori e pertanto richiedono maggior tempo per il refactoring.

9. Coding Standard: Il codice dovrebbe essere formattato in modo tale da raggiungere una formattazione standard. In questo modo si rende il codice coerente e facilmente leggibile e modificabile per l'intero gruppo di programmatori. La formattazione standard semplifica ed incoraggia il collective code ownership.

10. System Metaphor: Dovrebbe essere creata una sorta di mappa del codice che permetta ai programmatori di conoscere e vedere dove posizionare nuovi pezzi di codice.

11. The Planning Game: è una riunione di pianificazione che si verifica periodicamente (tipicamente una volta a settimana); l'idea è per ogni progetto di partire con un piano rudimentale e svilupparlo quando le idee si fanno più chiare.

12. Testing: Il testing è un'attività particolarmente importante e ne è riservata una intera sezione(3.4).[3, 5]

XP è uno dei processi che viene particolarmente seguito in Antea, o, comunque, di cui ne vengono presi in considerazione molti aspetti, sebbene, eventualmente, si decida di attuare una metodologia differente.

3.2 Ide

Lo strumento che mette maggiormente in risalto quelle che sono le caratteristiche dell'ingegneria del software è il cosiddetto IDE (Integrated Development Environment).

L'IDE è un software che aiuta lo sviluppo del codice e comprende un editor di codice sorgente, un compilatore e/o un interprete, un tool di building

automatico e un debugger. A volte sono integrati con un sistema di controllo di versione e con tool per semplificare la costruzione delle GUI. Alcuni IDE, rivolti allo sviluppo di software orientati agli oggetti, comprendono anche un navigatore di classi, un analizzatore di oggetti e un diagramma della gerarchia delle classi.[2]

Le motivazioni per cui l'utilizzo di un IDE sia da preferire a quello del blocco note sono molteplici. Prima tra tutte il fatto che l'IDE aiuta notevolmente a sviluppare quelle che sono le direttive principali utilizzate soprattutto nei processi agili appena analizzati (xp in primis), ma valide in molte metodologie di sviluppo, in particolar modo quanto concerne il collective code ownership, il coding standard, il refactoring e il testing.

L'IDE che viene utilizzato in azienda è NetBeans poiché oltre ad essere free e open source incrementa costantemente l'editor per Java, il linguaggio più usato in Antea, attraverso molte funzionalità e una vasta gamma di strumenti, modelli ed esempi. Inoltre soddisfa in pieno quelle che sono le esigenze di progetto elencate nel paragrafo precedente, infatti, a livello di codice, rende automatica la formattazione, fa combaciare parole e parentesi, rende disponibile modelli di codice e suggerisce codice da implementare. Per quanto riguarda la caratteristica del simple design, offre differenti modalità con cui visualizzare l'insieme di tutti i dati del progetto stesso (classi, interfacce, ecc.), per rendere agevole la loro gestione.

L'IDE NetBeans rende anche semplice e agevolata l'integrazione con i vari strumenti di versionamento come SVN, Mercurial e Git.[4]

3.3 Sistemi di Versionamento

Poiché, nella realizzazione di software, i cambiamenti sono inevitabili, uno degli obiettivi principali dell'ingegneria del software è quello di accrescere la facilità con cui trattare i cambiamenti e ridurre il carico di lavoro necessario in occasione dei cambiamenti stessi, soprattutto considerando il fatto che la maggior parte dei cambiamenti software è giustificata.

Per gli ingegneri l'obiettivo è quello di lavorare in modo efficace. Ciò vuol dire che non devono interferire tra loro nella stesura e nella revisione di codice senza motivo. Tuttavia hanno anche il compito di comunicare e coordinarsi in modo efficiente. In particolare, essi utilizzano strumenti che li aiutano a realizzare un prodotto software coerente, in cui le modifiche vengono propagate nel lavoro degli altri ingegneri tramite una fusione dei file, e anche, in caso di modifiche simultanee, a risolvere conflitti e unire le modifiche. Tramite questi strumenti, viene mantenuto un elenco cronologico

dell'evoluzione di tutti i componenti del sistema e una registrazione dei motivi che hanno spinto al cambiamento, oltre ad una registrazione di ciò che è stato effettivamente modificato.

A mano a mano che il progetto procede, verranno create più versioni dei singoli prodotti e ciascuna di queste dovrà essere memorizzata in un archivio (repository), il quale deve salvarle tutte in modo da consentire una gestione efficace delle release del prodotto e fare si che gli sviluppatori possano tornare alle versioni precedenti durante le fasi di collaudo e debugging. L'archivio deve anche mantenere un audit trail che stabilisca ulteriori informazioni riguardanti da chi, quando e perché è stata effettuata una modifica.[5]

Gli strumenti atti a svolgere questo insieme di funzioni sono i cosiddetti sistemi di versionamento.

Di tali strumenti ne esistono un vasta quantità, tuttavia si è posta l'attenzione in particolare su due di questi, SVN, storicamente in uso all'interno dell'azienda e Mercurial, ultimamente utilizzato in Antea.

La differenza sostanziale che esiste tra questi due sistemi è che dove SVN costituisce una tipica architettura client-server in cui c'è un server centrale per memorizzare le revisioni di un progetto, Mercurial è un sistema completamente distribuito, che permette di dare a ciascuno sviluppatore una copia dell'intero sviluppo del progetto. In questo modo ogni programmatore lavora indipendentemente dalla rete o dall'accesso al server centrale, rendendo revisioni, fusioni, diramazioni più veloci ed economiche.

Chiaramente così, però, Mercurial occupa molto più spazio su disco rispetto a SVN ma allo stesso tempo ha un indice molto elevato di fault-tolerance, in quanto sono presenti più copie della repository di progetto, una per ogni sviluppatore.

3.4 Unit testing

Come ogni prodotto anche il software necessita di essere collaudato per individuare gli errori che sono stati inavvertitamente, ma sicuramente, commessi durante le sue fasi di progettazione e di realizzazione.

Per collaudare un software possono essere utilizzate diverse strategie e tecnologie, tuttavia quella presa in considerazione è il collaudo per unità, ovvero lo unit testing. Come si può intuire lo unit test si concentra sulla verifica della più piccola unità di progettazione software che, nel caso di progettazione ad oggetti è il metodo.

Gli unit tests agevolano anche il refactoring, infatti, dopo qualche piccolo cambiamento, la verifica del superamento dei test implica che i cambiamenti apportati alla struttura non hanno introdotto cambiamenti funzionali.

Questa tipologia di test funziona perfettamente con i processi agili, in particolare XP che ne fa anche uno dei suoi pilastri e punti di forza, tanto da avere una metodologia di utilizzo particolare, il TDD (Test Driven Development), sviluppata da Kent Beck nel 2003.

Secondo Beck la procedura da seguire può essere riassunta in 5 punti:

1. Aggiungere velocemente un test.
2. Eseguire tutti i test e vedere quello nuovo fallire.
3. Apportare qualche piccola modifica.
4. Eseguire tutti i test e vederli passare tutti con successo.
5. Fare un refactoring per eliminare le duplicazioni.[6]

La differenza sostanziale di questa metodologia di testing rispetto alle altre è quella di creare i test ancora prima di scrivere il codice. Questa tecnica comporta alcuni vantaggi, infatti, in questo modo lo unit test aiuta lo sviluppatore a considerare cosa realmente sia necessario far svolgere ad un determinato metodo poiché i requisiti saranno strettamente fissati dal test: non ci possono essere fraintendimenti su una specifica scritta in forma di codice eseguibile. Inoltre il programmatore avrà un feedback costante del suo lavoro.

Sebbene sia possibile svolgere lo unit test anche manualmente, normalmente è consigliato un approccio automatico tramite dei framework specifici: ne esistono diverse quantità e ciascuno è specifico per un linguaggio di programmazione. Per quanto riguarda il progetto si è utilizzato JUnit, in generale utilizzato anche nei vari progetti gestiti in Antea. Le motivazioni per cui è stato scelto JUnit sono da ricercarsi nel fatto che è sicuramente uno tra i più diffusi framework java e che la sua integrazione in Netbeans risulta piuttosto semplice oltre che completa.

3.5 Sistemi di Building

Un sistema di build è un insieme di script e file di configurazione che permettono di portare a termine in maniera automatica operazioni quali build, test, pacchettizzazione, rilascio e installazione del proprio software. Un sistema

di build è il vero e proprio motore dello sviluppo software. Infatti, lo sviluppo del software è un'attività complessa che coinvolge molteplici attività quali: controlli del codice sorgente, generazione di codice, controlli automatici di codice sorgente, generazione di documentazione, compilazione, linking, unit testing, testing d'integrazione, pacchettizzazione, creazione di versioni binarie, rilascio del codice sorgente, distribuzione e report.

Detto questo, si può ricondurre la produzione del software a un'attività suddivisa in quattro fasi:

1. Gli sviluppatori scrivono codice sorgente e i contenuti (grafica, testi, modelli, ecc.)
2. Gli scritti originali (artifacts) vengono trasformati in prodotti finiti (eseguibili, siti web, documenti, installer, ecc.)
3. I prodotti finiti vengono testati.
4. I prodotti finiti vengono distribuiti e rilasciati.

Un buon sistema di build automatico dovrebbe prendersi carico dei punti 2-4, rendendo molto più rapido e meno oneroso il compito dei vari componenti del team di progetto.[7]

In Antea, il sistema di build utilizzato per i progetti Java è Maven, il quale nasce come una sorta di evoluzione di Ant, il sistema di build più famoso e diffuso, soprattutto per rendere standard le modalità di build del software, per dare una chiara definizione su cosa consista il progetto che si vuole sviluppare, per facilitare la pubblicazione delle informazioni del progetto e per disporre di una soluzione di condivisione di pacchetti Jar attraverso molteplici e differenti progetti. La caratteristica principale per cui è stato scelto questo tipo di sistema è sicuramente la capacità che ha Maven di gestire le dipendenze. Infatti la forza di Maven è che una volta definite, queste vengono scaricate e aggiornate automaticamente rendendo molto semplice il lavoro di rilascio al programmatore.

3.6 Debugging

Il debugging è una conseguenza del collaudo; ossia, quando un caso di prova fallisce, tramite il debugging si procede all'eliminazione dell'errore. Attraverso l'esecuzione dei casi di prova del collaudo lo sviluppatore ha un'indicazione di come si presenti l'errore, tuttavia non necessariamente ne conosce immediatamente le cause, considerando che spesso causa ed errore non hanno una relazione ovvia.

Esistono diverse metodologie di debugging, tuttavia lo scopo principale è sempre il medesimo, ovvero scoprire e correggere la causa dell'errore nel software.

Per fare ciò si può semplicemente inserire nel programma righe che stampino messaggi per tracciare l'andamento del programma o i valori delle variabili. Tale metodo non scova l'errore tuttavia lo sviluppatore può capire dove si sia insinuato all'interno dell'algoritmo, procedendo passo passo insieme ad esso. Questa metodologia tuttavia ha un limite non indifferente, ovvero fa perdere notevole tempo al programmatore perché oltre a dover aggiungere righe di codice, inutili allo sviluppo del programma, è poi costretto a cancellarle.

I moderni ambienti di sviluppo contengono specifici programmi detti debugger atti a svolgere questa operazione di verifica dell'esecuzione del programma in maniera autonoma, senza l'aggiunta di altro inutile codice.

Grazie al debugger in sostanza è permesso far eseguire il programma fino ad un punto deciso dallo sviluppatore in cui il programma si arresta momentaneamente. A questo punto è possibile ispezionare le variabili, eseguire il programma una riga alla volta o far procede l'esecuzione del programma a piena velocità fino al raggiungimento del breakpoint successivo. Per quanto riguarda Antea il debugger utilizzato è quello nativo della IDE Netbeans.[8, 5]

3.6.1 Sistemi di Bug Tracking

Uno strumento di ingegneria del software particolarmente importante e strettamente legato al processo di debugging è sicuramente il sistema di bug tracking. Questo strumento è utilizzato per tenere traccia delle segnalazioni di bug all'interno del codice in modo da tenere sotto controllo eventuali errori e la loro riproducibilità.

In sostanza il sistema di bug tracking consiste in un database in cui sono registrati tutti gli elementi descrittivi di un bug quali data di apparizione, gravità, stato e programmatore ad esso associato.

Per progetti di una certa entità il team di sviluppo è composto da più persone che lavorano contemporaneamente allo stesso progetto, introducendo ciascuno feature differenti. È possibile, probabile, che mentre si sta sviluppando un particolare pezzo di codice ci si accorga di bug, collegati ad altre parti di codice, che sono state precedentemente sviluppate dallo stesso sviluppatore o da altri. Per evitare di perdere la concentrazione su ciò che si sta facendo, la cosa migliore è quella di postare nel bug tracking il bug riscontrato, sempre che il bug non precluda il raggiungimento della parte di software

su cui si sta attualmente lavorando. In questo modo, l'errore è messo in comune con tutti gli altri membri del team e vari scenari sono realizzabili: i più comuni sono che chi ha l'idea di come risolvere si può accollare tale compito, oppure solitamente se lo accolla chi effettivamente ha introdotto il bug in quanto ha una conoscenza diretta del codice da lui scritto. Il sistema di bug tracking utilizzato da Antea è Redmine, software libero ed open source basato su web, che fornisce funzionalità integrate di gestione dei progetti, gestione dei problemi, il supporto per molteplici opzioni di controllo di versione e soprattutto sostiene progetti multipli.

Con questa si conclude la rapida lista di strumenti analizzati, non sono tutti quelli previsti dall'ingegneria del software, ma sicuramente sono fondamentali allo sviluppo di software di buona qualità.

Parte II

Il progetto Halite

Capitolo 4

Analisi dei requisiti

L'idea del progetto è nata per venire incontro a quelle che erano le esigenze del reparto commerciale della ditta. In particolar modo, era richiesto di poter ricevere dal database aziendale informazioni utili per quanto concerne i dati storici aziendali su commesse, ordini, clienti, articoli e risorse umane.

Il database gestionale in Antea prende il nome di Salgemma e pertanto l'applicativo sviluppato è stato chiamato Halite che non è altro che un nome alternativo con cui viene chiamato il cloruro di sodio, ovvero Salgemma.

L'applicazione è stata chiaramente creata ad hoc per il particolare database a disposizione e pertanto funziona solo se connesso ad esso.

Procediamo quindi nell'analisi dell'applicativo.

4.1 I requisiti aziendali

Le necessità della ditta erano quelle di ottenere un programma che potesse facilmente reperire informazioni dal database gestionale, senza il rischio di compromettere l'integrità dei dati ivi contenuti, che, fino all'installazione di Halite, erano accessibili solo attraverso un piuttosto complesso e macchinoso software di interfacciamento col database stesso.

Per ottenere tali informazioni è stato richiesto uno schema delle esigenze ai committenti (Figura 4.1). Tale schema è stato creato secondo lo schema tipico delle user stories dell'extreme programming: le user stories sono la descrizione delle funzionalità che, a detta del cliente, il sistema deve implementare. Le user stories non devono essere troppo lunghe (circa tre frasi per funzionalità), devono essere scritte dai committenti e pertanto devono presentarsi senza sintassi tecniche.

	SALGEMMA	
#		NOTE
1	COMMESSA: redditività di tutte le commesse con costo, ricavo, tempo e guadagno in valore assoluto e %	Arco di tempo di default sarà: 1Gennaio - 31Dicembre. Possibilità di selezionare un arco di tempo casuale e ottere la lista delle commesse piu redditive
2	Redditività per cliente (tutti gli ENI, Polimeri, etc) con le info e note come al punto 1	
3	Redditività totale di tutte le commesse aggregate con le info e note come al punto 1	
	COMMESSA: nella lista delle commesse redditive (vale per punto 1-2 e 3), vedere quali risorse sono coinvolte	
4	Vedere le attività più redditive sia in termini assoluti che %, assegnate in tutte le commesse	
5	Grafico commessa:Non so sé fattibile, ma si potrebbe legare con i punti 1-2 e 3, cioè quando ho esempio l'elenco delle commesse piu redditive, sotto compare il grafico per l'arco di tempo selezionato.	

Figura 4.1: Requisiti di Halite

Come si evince dallo schema le richieste sono piuttosto semplici e ben suddivise: i primi tre punti richiedono il reperimento di determinate informazioni dal database, seguendo determinati criteri indicati nel punto 1 e nel campo note. I successivi due punti sembrano una richiesta di dettagli dalle informazioni già reperite, mentre l'ultimo punto è estremamente chiaro e richiede la creazione di grafici sempre partendo dalle prime tre specifiche.

Ottenuto lo schema, sono stati svolti alcuni colloqui per chiarire alcuni punti e cercare di ottenere informazioni che potessero far comprendere meglio quali fossero le reali esigenze del cliente, cercando di scovare anche le esigenze che il cliente dava per scontate. Durante questi colloqui, oltre alle specificazioni sulle esigenze già descritte, è emerso come l'esportazione nei formati excel e .csv potesse rappresentare una funzionalità secondaria di grande valore, nel momento in cui questa fosse possibile.

Ottenute le informazioni necessarie si è proseguito con la fase successiva: la produzione.

Capitolo 5

Produzione

A seguito della raccolta dei requisiti si è passati alla fase successiva, che può essere definita come fase produttiva. Tale fase può essere suddivisa in due sezioni che sono: lo studio del gestionale e la progettazione dell'applicativo con conseguente scrittura di codice.

5.1 Il gestionale aziendale

L'analisi delle esigenze ha evidenziato come il progetto fosse inscindibilmente legato al gestionale aziendale. Pertanto, prima di cimentarsi nella progettazione e scrittura del programma, è stato necessario prendere confidenza con l'ambiente che maggiormente caratterizzava l'intero progetto, ovvero il database gestionale della ditta: Salgemma.

Non mi è dato sapere se tale database fosse documentato oppure no, ma mi è stata fornita l'implementazione della base di dati in modo tale da capire come funzionasse e mettermi alla prova anche in questo esercizio. Per tale ragione mi sono apprestato a svolgere il reverse engineering del gestionale stesso.

5.1.1 Cos'è il reverse engineering

Il reverse engineering richiama l'immagine della pietra filosofale nella tradizione alchemica: a partire da un programma non strutturato e non documentato si ottiene una documentazione completa del programma. Il reverse engineering, tuttavia, non è una scienza perfetta e sebbene sia possibile estrarre informazioni strutturali del codice sorgente, il livello di astrazione, il grado di completezza della documentazione prodotto, il livello di interazione

tra gli strumenti e l'analista e la direzione di un processo sono altamente variabili. Ciò significa che, sebbene sia possibile capire come funzioni un prodotto finito, tornare indietro e crearne una documentazione dettagliata, uguale a quella che si sarebbe dovuta redigere del programma nel momento in cui esso è stato sviluppato, è praticamente impossibile.[5]

5.1.2 Reverse engineering di Salgemma

Il reverse engineering può essere effettuato su qualunque prodotto software, non necessariamente un programma, in particolare in questo caso è stato affrontato il reverse engineering di un database. In sostanza partendo dall'ispezione di una copia fisica del database si è risaliti allo schema logico e successivamente a quello concettuale, ovvero allo schema ER (Entità-Relazione). Lo schema ER non è completo, mancano tutti gli attributi delle varie entità, che tuttavia sono stati omessi per motivi di spazio in quanto comunque essi sono ben visibili nello schema logico.

I benefici dello svolgere il reverse engineering del database gestionale si sono rivelati immediatamente nel momento in cui sono state create le query per il programma, infatti, una descrizione di tutti gli attributi e di tutte le relazioni ha permesso il facile reperimento delle informazioni indispensabili nella scrittura delle query stesse, anche delle più complesse.

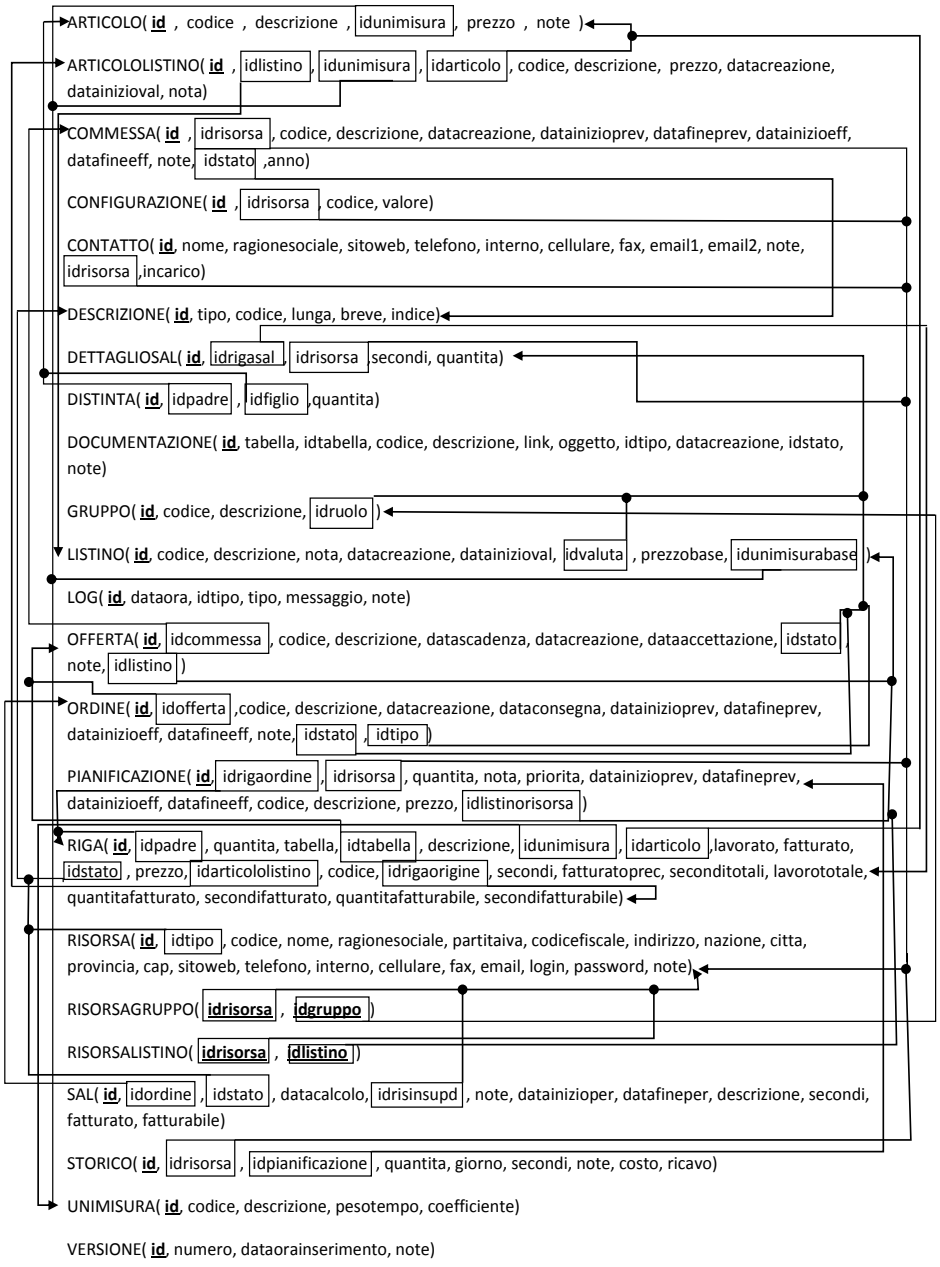


Figura 5.1: Schema Logico Salgemma

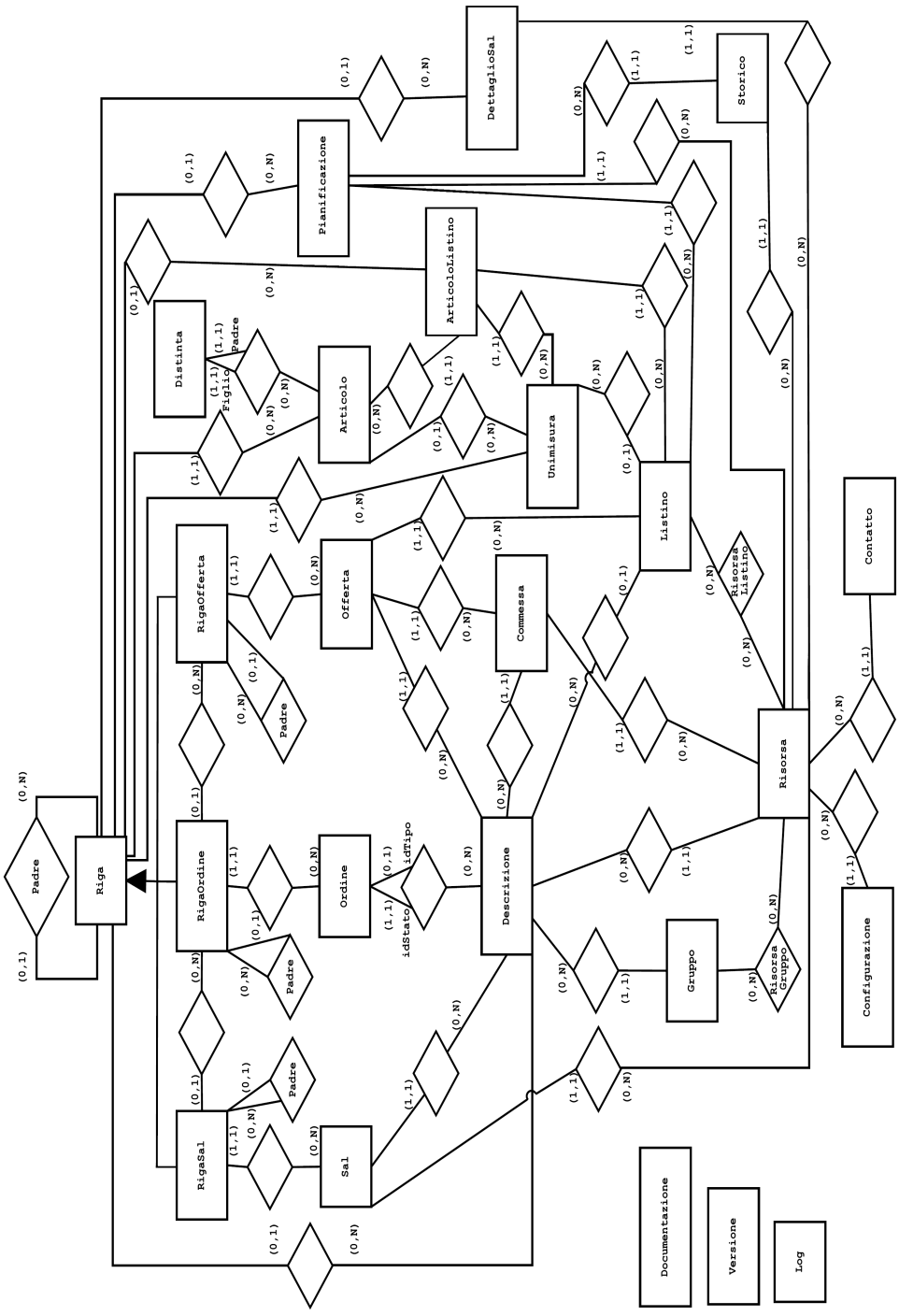


Figura 5.2: Schema ER Salgemma

5.2 La stesura del codice

Ultimata la parte di reverse engineering si è passati alla progettazione del prodotto e alla conseguente scrittura di codice vero e proprio. In realtà, per prendere confidenza con quelle che sarebbero state le librerie utilizzate per le funzionalità base del programma, sono stati scritti altri pezzi di codice, ininfluenti e inutili per il programma in sé, ma, utili per cimentarsi con sintassi completamente nuove.

5.2.1 Metodologia utilizzata

Innanzitutto, prima di spiegare come sia stato scritto il codice e quali siano state le librerie effettivamente utilizzate, è necessario descrivere quella che è stata la metodologia di sviluppo utilizzata nella progettazione del software in questione.

Sebbene nella prima parte fosse stata studiata la metodologia agile, e in particolare xp, non è stata questa la metodologia effettivamente usata, bensì quella incrementale.

Il modello incrementale prevede uno sviluppo del software diviso a stadi. Il primo stadio consiste spesso in un prodotto base, cioè un prodotto che soddisfa i requisiti fondamentali, tralasciando diverse caratteristiche supplementari (alcune note, altre sconosciute). Questo prodotto viene saggiato dal cliente (o sottoposto a una revisione approfondita). In seguito alla valutazione del cliente, si stende un piano per il successivo stadio. Tale piano deve curarsi di due aspetti: le modifiche al prodotto tese a soddisfare meglio quelle che sono le esigenze del cliente e l'inserimento di nuove funzioni. Il processo si ripete a mano a mano che si completano i vari stadi, fino a giungere al prodotto finale.[5]

Questo di fatto è stato il processo utilizzato, infatti, in primo luogo sono state fornite le funzionalità principali di Halite, in secondo luogo si è passati allo sviluppo delle funzionalità secondarie, quali il miglioramento dell'interfaccia utente, con le modifiche precedentemente analizzate delle funzionalità delle tabelle, oltre a grafici ed esportazioni.

5.2.2 Linguaggio e librerie

Il progetto è stato completamente sviluppato in Java, con il JDK versione 6, sia perché è praticamente l'unico linguaggio di programmazione da me affrontato durante il corso di studi sia perché è il linguaggio utilizzato

maggiormente all'interno dell'azienda, anche se non è l'unico, in quanto, a seconda delle richieste, vengono sfruttati altri linguaggi di programmazione.

Grazie all'utilizzo di Maven, di cui ricordiamo il punto forte essere la gestione delle dipendenze, e alla sua facile integrazione con l'IDE Netbeans, è possibile avere all'interno dell'IDE stessa anche un grafico di tutte le dipendenze del progetto, pertanto ne visualizziamo il contenuto per mostrare quelle che sono le librerie principali utilizzate nello sviluppo del software in esame.

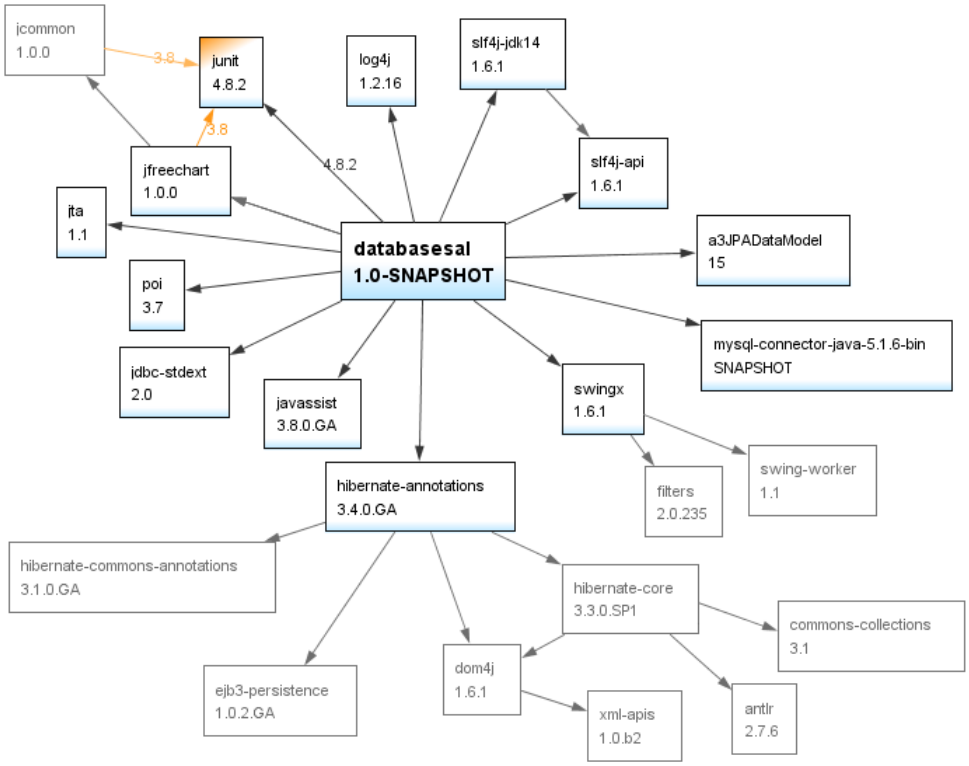


Figura 5.3: Diagramma delle dipendenze Maven

A prima vista dello schema si nota l'encomiabile lavoro svolto dal sistema di building che mette in risalto quelle che sono delle discrepanze di versione tra quelle che sono le dipendenze richieste da jfreechart, il quale richiederebbe la versione 3.8 di junit e il junit effettivamente richiesto ed installato nelle librerie del progetto che è alla versione 4.8.2. Il sistema di building però capisce che la versione 4.8.2 è più recente e scarica solo quella, sebbene

metta in mostra che potrebbero esserci degli errori dettati proprio da questa discrepanza.

Come si può notare dal grafico le librerie fondamentali, utilizzate nella realizzazione del software, sono: log4j, slf4j, a3JPADataModel,mysql-connector, swingx, hibernate-annotations, javassist, jdbc, poi, jta, jfreechart e junit.

In realtà, sebbene siano tutte dipendenze dirette del software in analisi, non sono tutte librerie principali, ma sono librerie necessarie per il corretto funzionamento di quelli che realmente sono gli strumenti software utilizzati ovvero Hibernate, Jdbc, jUnit, poi, jfreechart, log4j e swingx. Possiamo quindi suddividere ed analizzare le dipendenze del grafo in vari sottogruppi a seconda della dipendenza principale e dello scopo per cui vengono importate ed utilizzate, mettendo in risalto quelle che sono le caratteristiche cardini delle varie librerie.

Database: Per quanto concerne la struttura interna del programma, quindi principalmente la connessione del programma nei confronti del database gestionale, sono stati utilizzati due strumenti: Jdbc e Hibernate.

Jdbc (Java DataBase Connectivity) è lo standard industriale per la connessione a database indipendenti tra il linguaggio di programmazione Java e una vasta gamma di database, indipendentemente dal particolare DBMS utilizzato. Questa tecnologia permette quindi di scrivere codice una volta soltanto e farlo funzionare anche nel momento in cui si vada a sostituire il DBMS utilizzato per i dati aziendali. Le funzionalità principali di Jdbc sono quelle di interfacciarsi e stabilire una connessione con un database, mandare ad esso codice SQL e processare i risultati ottenuti.[9]

Hibernate è uno strumento di mapping tra oggetti e relazioni (ORM) in ambiente di sviluppo Java, ciò significa che permette di mappare una rappresentazione di dati da un modello ad oggetti ad un modello relazionale con uno schema basato su SQL. Oltre al mapping, Hibernate si prende cura anche delle query e strutture per il recupero dei dati, riducendo il tempo di sviluppo speso utilizzando Jdbc e le normali query sql.

In sostanza, le tabelle del database vengono trasformate in classi persistenti Java e le operazioni di insert, delete e update vengono fatte utilizzando normali metodi getter e setter delle variabili di istanza della classe stessa. Riportiamo un esempio relativo ad Halite: come si evince dallo schema logico(5.1) Articolo ha come attributi i campi Id, Codice, Descrizione, IdUniMisura, Prezzo, Note; invece di utilizzare le laboriose query sql per recuperare informazioni su di essa è stata creata la classe Articolo che prevede le va-

riabili id, codice, descrizione, idUniMisura, prezzo e note, e tali campi sono accessibili tramite i metodi get e set della classe stessa come si può notare dall'estratto della classe:

```
@Entity
@Table(name = "articolo", uniqueConstraints = {
@javax.persistence.UniqueConstraint(columnNames = {"codice"})})
public class Articolo extends BaseEntity
implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Basic(optional = false)
@Column(name = "id")
private Integer id;

@Basic(optional = false)
@Column(name = "codice", unique = true, length = 45)
private String codice;

@Column(name = "descrizione", length = 255)
private String descrizione;

@Column(name = "prezzo")
private BigDecimal prezzo;

@Column(name = "note", length = 255)
private String note;

@JoinColumn(name = "idunimisura", referencedColumnName = "id")
@ManyToOne(fetch = FetchType.LAZY)
private Unimisura idunimisura;

public Articolo(){
}

public Articolo(Integer id) {
this.id = id;
}

public Articolo(Integer id, String codice) {
this.id = id;
this.codice = codice;
}

public Integer getId() {
return this.id;
}

public String getCodice() {
return this.codice;
}

public String getDescrizione() {
return this.descrizione;
```

```
}

public BigDecimal getPrezzo() {
return this.prezzo;
}

public String getNote() {
return this.note;
}

public Unimisura getIdunimisura() {
return this.idunimisura;
}
```

Hibernate come ciascuno degli ORM richiede dei metadati che gestiscano le trasformazioni dei dati da una rappresentazione all'altra. A questi metadati provvedono le cosiddette Hibernate Annotations che sono le parole precedute dal simbolo "@" nell'estratto di codice appena esaminato. Per far individuare ad Hibernate quali siano le classi in cui ricercare le annotazioni è richiesto di aggiungere al progetto un file di configurazione di Hibernate il quale deve chiamarsi hibernate.cfg e deve essere in formato xml. La forma sostanziale del file di configurazione è la seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration -3.0.dtd">
<hibernate-configuration>
  <session-factory>
   <property name="hbm2dll.auto">validate</property>
   <mapping class="com.anteash.salt.entities.Articolo"/>
  </session-factory>
</hibernate-configuration>
```

Anche le query in Hibernate possono subire un cambiamento. Infatti, sebbene le normali query sql possano essere utilizzate e richieste al database è possibile anche creare query in formato HQL (Hibernate Query Language) il quale non differisce molto dal formato SQL se non dal fatto che è completamente orientato agli oggetti e, pertanto, comprende nozioni come ereditarietà, polimorfismo e associazioni. Nel caso si decida di svolgere le query con questo linguaggio, Hibernate genererà il codice SQL necessario ad interrogare il database. Anche in questo caso potrebbe essere esemplificativo riportare le due query presenti nel codice di Halite riguardanti sempre l'entità Articolo:

```
@NamedQueries({
@NamedQuery(name = "ArticoliProficuiRange",
```

```
query="select new com.anteash.salt.gui.Join"
+"(storico.idpianificazione.idrigaordine.idarticolo.codice,"
+"storico.idpianificazione.idrigaordine.idarticolo.descrizione, "
+"sum(storico.costo), sum(storico.ricavo), sum(storico.secondi),"
+"sum(storico.quantita),(sum(storico.ricavo)-sum(storico.costo)),"
+"((sum(storico.ricavo)-sum(storico.costo))/sum(storico.ricavo))*100) "
+"from Storico as storico "
+"where giorno>=:inizio and giorno<=:fine "
+"group by storico.idpianificazione.idrigaordine.idarticolo.codice "
+"order by (sum(storico.ricavo)-sum(storico.costo)) desc "),
@NamedQuery(name = "ArticoliNonProficuiRange",
query = "select new com.anteash.salt.gui.Join(articolo.codice,"
+"articolo.descrizione) "
+"from Articolo as articolo "
+ "where articolo.id not in ("
+ "select storico.idpianificazione.idrigaordine.idarticolo.id "
+ "from Storico as storico "
+ "where giorno>=:inizio and giorno<=:fine)"
+ "group by articolo.codice ")})
```

Come si può notare non viene fatto un join vero e proprio tra le tabelle del database bensì viene creata una lista di istanze di Join, che rappresentano nuove istanze di una classe sviluppata appositamente, che riceve come parametri gli attributi che si vogliono reperire, passando da una entità all'altra attraverso gli attributi di chiave esterna.

Oltre al reperimento semplificato delle informazioni nelle operazioni di join, grazie ad Hibernate (e in realtà anche a Jdbc) viene semplificato anche il meccanismo di creazione e utilizzo delle query. Infatti attraverso la annotazione @NamedQuery visibile nel listato è possibile creare appunto una namedquery, che in sostanza è una query che permette di parametrizzare alcuni campi (nell'esempio where giorno>=:inizio and giorno <=:fine; inizio e fine sono due parametri). In questo modo non è necessario scrivere una query ogni volta che si voglia utilizzarla con un determinato set di parametri ma è sufficiente definirla una volta sola mantenendola parametrizzata e far riempire runtime il set di parametri con valori concreti. Grazie a questa parametrizzazione delle query si possono ottenere vantaggi importanti, in particolare si incrementano le prestazioni del sistema in quanto la query viene creata la prima volta, tenuta in memoria e utilizzata nel momento del bisogno migliorando e facilitando così la riusabilità della query stessa e rendendo anche più facile il mantenimento del codice: infatti, se ci si accorge di un errore nella query è sufficiente cambiarlo una volta sola per tutte.

Questa non voleva essere una guida esaustiva di Hibernate ma semplicemente ne voleva mettere in risalto alcune delle caratteristiche principali di questo importante strumento.[10]

In questo paragrafo database sono da annoverare anche le librerie slf4j, mysql-connector, javassist, jta che sono utilizzate internamente proprio da Hibernate per svolgere al meglio il proprio compito. La libreria a3JPADataModel è una libreria interna di Antea che presenta le classi Java con getter e setter dei vari attributi del database di Salgemma, infatti essendo queste già utilizzate in altri progetti era dispendioso ed inutile ricrearle.

Log: Fare il log di una applicazione può essere un metodo di basso livello per farne il debug. Infatti, potrebbe essere anche l'unico metodo per fare il debug della applicazione che si sta sviluppando considerando che non sempre i tool di debug sono disponibili, come spesso accade nel caso di un'applicazione distribuita. Nel caso di Halite il log viene creato per dare chiarimenti su cosa potrebbe essere andato storto nel caso in cui il programma si fosse arrestato in maniera insolita.

La libreria che viene utilizzata per il log è log4j, sebbene ce ne sia una standard nel linguaggio Java, principalmente perché presenta potenzialità maggiori che la libreria standard non supporta e pertanto risulta sicuramente più completa. La metodologia di funzionamento è piuttosto banale e assomiglia molto a quella del normale flusso di standard output, tuttavia grazie ad un file di proprietà associato al progetto ne vengono definiti e modificati i vari parametri, in particolare specifica dove si voglia fare il log, ovvero su quale appender (console, file,mail,...), eventualmente su quale file e in che formato, ovvero con quale layout.

```
# Settaggio livello e appender

log4j.rootLogger= info, fileout

# Configurazione fileout

log4j.appender.fileout=org.apache.log4j.FileAppender
log4j.appender.fileout.File=HaliteLog.log
log4j.appender.fileout.layout=org.apache.log4j.PatternLayout

# Configurazione Pattern fileout

log4j.appender.fileout.layout.ConversionPattern=%d %5p (%F:%L) - %m%n
```

Test: Le motivazioni per cui si necessita fare dei test sono state già affrontate, qui si vuole fare un po' più riferimento a come si sviluppa uno unit test. Per prima cosa è necessario importare nel progetto la libreria per junit che nel nostro caso consiste della libreria junit 4.8.2.

Grazie all'integrazione in Netbeans creare uno unit test è molto semplice, infatti è sufficiente creare lo scheletro della classe che si vuole realizzare, senza implementare i metodi, e poi lasciare a Netbeans il compito di creare la classe che testi questa classe scheletro. La classe creata sarà anche essa a forma di scheletro, ovvero saranno presenti le intestazioni dei metodi di test per ciascun metodo della classe di partenza. Ora, all'interno di ciascun caso di prova vengono svolti alcuni calcoli, relativi a condizioni che si ritengono vere, atti a testare quello che si prevede (se si usa la metodologia TDD) o che si crede debba svolgere quel determinato metodo e si invoca quel metodo, poi si demanda ad un metodo di test, generalmente assertEquals ma non è l'unico, di confrontare valori previsti e valori realmente ottenuti. Svolgendo i test una barra di completamento visualizzerà la percentuale di test superati.

Interfaccia: Per quanto riguarda l'interfaccia utente di Halite, oltre alla classe swing di java è stato utilizzato il pacchetto org.jdesktop.swingx. Questo pacchetto in realtà si può considerare come una estensione della classe swing, poiché include componenti nuovi ed avanzati che provvedono alle funzionalità che vengono richieste nelle applicazioni per clienti. In sostanza è grazie a questa classe che sono stati raggiunte funzionalità secondarie relative all'interfaccia utente, come la possibilità di riorganizzare le colonne, ordinarle, nasconderle e mostrarle, ecc. Le modalità di funzionamento di questa libreria non differiscono rispetto a quella standard di java, l'unica differenza sta nell'utilizzare gli oggetti di questa classe.

Per quanto riguarda l'interfaccia grafica da sottolineare è la facilità con cui questa possa essere sviluppata grazie ai tool integrati nell'IDE, che permettono di visualizzare, aggiungere e rimuovere gli oggetti a livello grafico con la creazione automatica di codice lasciata all'IDE stesso. Questa facilitazione è anche possibile con gli strumenti del pacchetto Swingx, previa semplice aggiunta del pacchetto jar nelle librerie dell'IDE.

Grafici: La creazione di grafici in Halite è stata resa possibile grazie all'utilizzo della libreria JFreeChart. JFreeChart è una libreria gratuita per la piattaforma Java che permette la creazione di grafici a torta, istogrammi, diagrammi Cartesiani, grafici di dispersione, diagrammi di Gantt e molti altri, oltre a consentire l'esportazione nei formati PNG, JPEG, PDF o SVG, poter effettuare zoom interattivi, creazione del pannello degli strumenti automatico ed essere utilizzati in qualunque tipo di applicativo, sia esso un'applicazione, una servlet, un applet o una pagina JSP.[11]

Grazie a questo strumento la creazione di grafici base è davvero semplicissima, infatti si può scomporre lo sviluppo in tre fasi: creare un dataset con i dati da visualizzare, creare un oggetto freechart che crea il grafico e visualizzare il grafico ottenuto. Per quanto riguarda Halite il procedimento è stato svolto in due metodi differenti, per i primi due punti si è creato il metodo "faiGrafico", mentre per il terzo è stato creato il metodo "setGrafico" che semplicemente inserisce in un pannello il grafico creato col metodo "faiGrafico".

```
public JFreeChart faiGrafico(String s, JXTable tab) {
DefaultCategoryDataset ds = new DefaultCategoryDataset();
for (int i = 0; i < tab.getRowCount(); i++) {
 ds.addValue((BigDecimal)(tab.getValueAt(i,
 tab.convertColumnIndexToView(5))),"Guadagno Assoluto",
 (String) tab.getValueAt(i, tab.convertColumnIndexToView(0)));
 }
JFreeChart fc = ChartFactory.createBarChart3D("Grafico "+s,s,
"Guadagno Assoluto",ds,PlotOrientation.VERTICAL, true, true, false);
CategoryPlot capl = fc.getCategoryPlot();
CategoryAxis ax = capl.getDomainAxis();
ax.setCategoryLabelPositions(
 CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 5.0));
CategoryItemRenderer cir = capl.getRenderer();
cir.setItemLabelsVisible(true);
return fc;
}
```

Esportazioni: Le esportazioni sono state gestite tramite una libreria di Apache: Apache.poi. L'intento di Apache POI è di creare e mantenere delle API per la manipolazione di molti formati basati sugli standard Office Open XML e i formati OLE 2 di Microsoft (xls, doc, ppt,...).

Anche in questo caso, come nel caso dei grafici, l'uso di questa libreria facilita enormemente il lavoro del programmatore. Infatti, una volta creato il foglio di calcolo, tramite gli appositi metodi, sarà sufficiente prelevare il dato dalla tabella e settarlo nella cella del foglio per ottenere una esportazione perfetta, provvedendo infine alla scrittura sul file attraverso il metodo "write".[12]

```
public void esportaExcel(File f, PannelloTabelle panTab) {
HSSFWorkbook wb = new HSSFWorkbook();
esportaTabellaXls(wb, panTab);
try {
 FileOutputStream fos = new FileOutputStream(f);
 wb.write(fos);
 fos.close();
 } catch (IOException ex) {
```

```
logger.error(ex.getCause().getMessage());
  }
}
```

In linea di massima questo è il comportamento principale adottato in Halite, il metodo "esportaTabellaXLS" svolge le operazioni di copia dalla tabella passata come parametro al foglio di calcolo appena creato.

5.2.3 JavaDoc

Una delle attività che non può assolutamente mancare nella progettazione e nello sviluppo di un progetto software è sicuramente la documentazione. La documentazione ha diverse sfaccettature, diversi ambiti di utilizzo, diversi destinatari. Anche la tesi che si sta svolgendo, di fatto, appartiene alla documentazione del progetto svolto. Una delle caratteristiche spesso tralasciate ma assolutamente vitali nella vita di un software è la documentazione del codice che si sta sviluppando. Infatti, essa assume particolare importanza nel momento in cui si sta procedendo alla stesura di codice in un team di sviluppo, in quanto ciò che si sta creando, molto probabilmente dovrà o potrà essere usato da un altro sviluppatore a cui verrà sicuramente semplificato il compito se, assieme al codice sorgente, è presente una buona documentazione di cosa effettivamente faccia quel determinato listato di programma. La documentazione, tuttavia, non è fondamentale solo in caso di collaborazione. Un prodotto software necessita di essere mantenuto, ed inoltre è possibile che venga richiesto di implementare nuove funzionalità o semplicemente dopo molto tempo ci si accorge di un bug nel funzionamento che ai tempi della stesura era passato inosservato. Rileggere tutto il codice sorgente, specialmente se non vengono adottati standard di scrittura potrebbe risultare estremamente noioso, complesso, lento e pertanto costoso.

In Java esiste uno strumento molto utile per la documentazione in forma standardizzata del codice sorgente: Javadoc. Javadoc analizza le dichiarazioni di classi, interfacce, metodi, costruttori e se trova dei commenti formattati in un determinato modo, li processa e li trasforma automaticamente in un insieme chiaro ed elegante di paginazione HTML.

Un esempio estratto dal codice di Halite di commento di documentazione che Javadoc analizza è il seguente, tratto dal metodo di estrazione in foglio excel precedentemente analizzato:

```
/** Riempie il file passato come parametro come un file excel, creando
*  un foglio con il nome del PannelloTabelle passato come parametro e
*  riempiendolo con i dati contenuti nella tabella presente nello stesso
*  pannello.
```

```
*
* @param f File su cui scrivere
* @param panTab PannelloTabelle da cui ricavare i dati da esportare
*/
public void esportaExcel(File f, PannelloTabelle panTab) {
...}
```

Come risulta dalla figura 5.4, la documentazione risultante è appunto un documento HTML, del tutto analogo a quello della documentazione standard di java.

com.anteash.salt.gui

Class RichiesteDaTabelle

java.lang.Object
  └ com.anteash.salt.gui.RichiesteDaTabelle

public class RichiesteDaTabelle
extends Object

Author:
Matteo

Constructor Summary

RichiesteDaTabelle()

Method Summary

void	esportaExcel(File f, PannelloTabelle panTab) Riempie il file passato come parametro come un file excel, creando un foglio con il nome del PannelloTabelle passato come parametro e riempiendolo con i dati contenuti nella tabella presente nello stesso pannello.

Figura 5.4: Documentazione Javadoc

Capitolo 6

Risultati raggiunti

Al termine dei vari cicli di produzione, quando il prodotto soddisfava tutti i requisiti iniziali, è stata effettuata una riunione con i committenti e gli users finali nella quale sono state evidenziate le modalità di funzionamento di Halite e in generale il lavoro svolto. Infine si è passati al deployment con l'installazione del prodotto nei computer dei vari utenti.

Ai fini di esporre al meglio i risultati raggiunti, quindi l'applicazione che è stata consegnata ai clienti, è utile differenziare e analizzare separatamente quelle che, fin dalla raccolta e dalla analisi dei requisiti, si erano dimostrate essere le funzionalità principali e quelle secondarie del progetto.

6.1 Funzionalità principali

Tramite Halite le informazioni sono reperibili in un modo che si può definire quasi deduttivo, ovvero partendo dal generale possono poi essere analizzate maggiormente nel dettaglio.

In primo luogo, quindi, le query inviate al database sono di carattere generale e ritornano, per uno specificato range di date:

- L'insieme di tutte le commesse attive;
- L'insieme di tutte le commesse attive raggruppate per cliente;
- L'insieme di tutti gli ordini attivi;
- L'insieme di tutti gli articoli;
- L'insieme di tutte le risorse;
- Il totale complessivo della ditta.

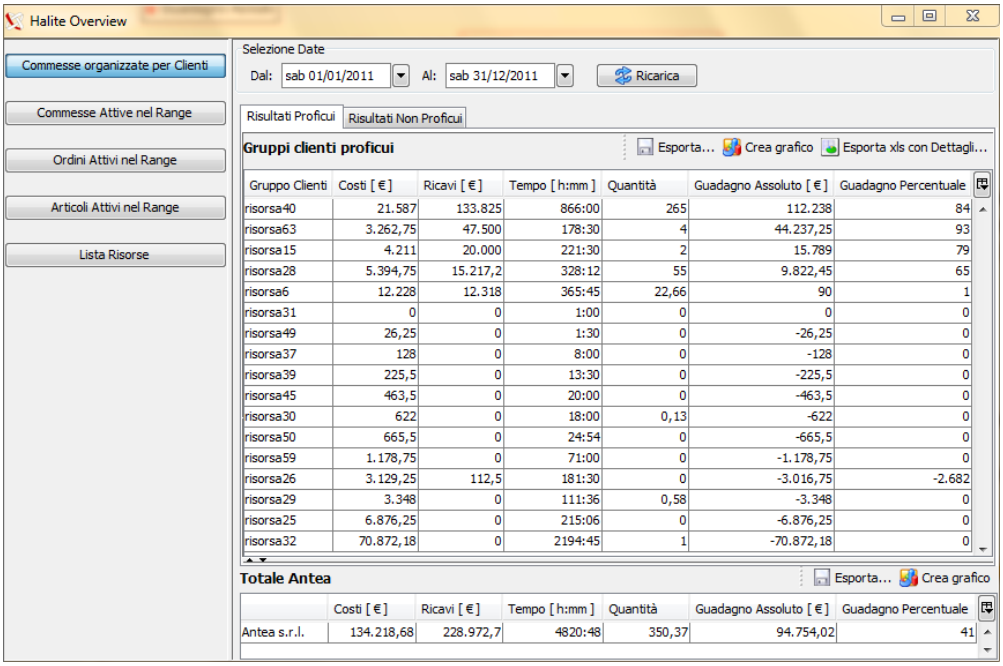


Figura 6.1: Richieste principali

Come si può notare dalla figura 6.1, i risultati della query richiesta vengono visualizzati in forma tabulare mettendo in risalto quelli che sono gli aspetti economico/gestionali quindi costi, ricavi, tempo, quantità, guadagno assoluto e guadagno percentuale sui ricavi.

Non necessariamente tutte le commesse, gli ordini, le risorse o gli articoli attivi risultano essere proficui nel periodo stabilito. È probabile, infatti, che esse siano trascurate, volontariamente o involontariamente da parte dell'azienda per motivi interni. Halite, pertanto, distingue in due tabelle le risposte proficue da quelle non proficue della query richiesta, tramite l'utilizzo di due tab separate (Figura 6.2).

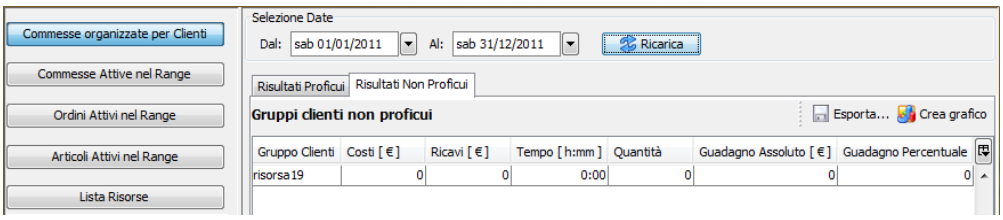


Figura 6.2: Risultati non proficui

Per ciascuna delle informazioni reperite è poi possibile scendere in dettaglio e analizzare, a seconda della prima query richiesta:

- L'insieme di tutte le risorse impiegate in una determinata commessa;
- L'insieme di tutte le risorse che lavorano per un determinato cliente;
- L'insieme di tutte le risorse che lavorano per un particolare ordine;
- L'insieme di tutti gli ordini a cui un determinato articolo è collegato;
- L'insieme di tutte le commesse in cui lavora una determinata risorsa.

Anche queste informazioni di dettaglio sono raggruppate in tabelle che vengono visualizzate da Halite nel momento in cui si effettua un doppio click del mouse su una delle righe della tabella principale, quella desiderata(Figura 6.3).

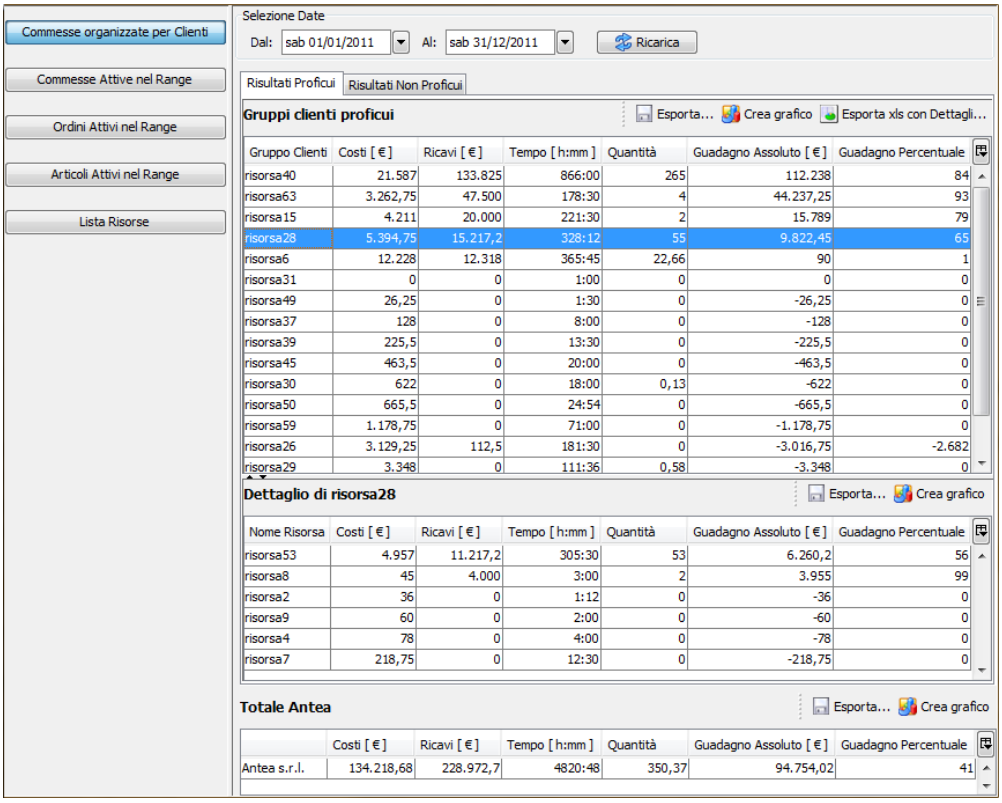


Figura 6.3: Richieste secondarie

6.2 Funzionalità secondarie

L'applicazione rispondeva, con questi strumenti, a quelle che erano le esigenze principali del cliente, tuttavia Halite è stata arricchita con dei moduli che si sono rivelati molto utili per facilitare il lavoro degli utenti finali.

Per prima cosa sono state rese dinamiche le tabelle di visualizzazione, ovvero, è stato reso possibile modificare l'ordine delle colonne, modificare l'ordinamento delle righe tramite pressione sull'header della colonna di cui si voglia mettere in risalto il risultato maggiore o minore, nascondere intere colonne di cui non si necessiti in quel particolare frangente, rimostrare intere colonne precedentemente nascoste ed autodimensionare le celle a seconda dei valori in esse contenuti (Figura 6.4).

Sono stati poi aggiunti moduli per la creazione di grafici e per l'esportazione delle tabelle in formati .xls e .csv.

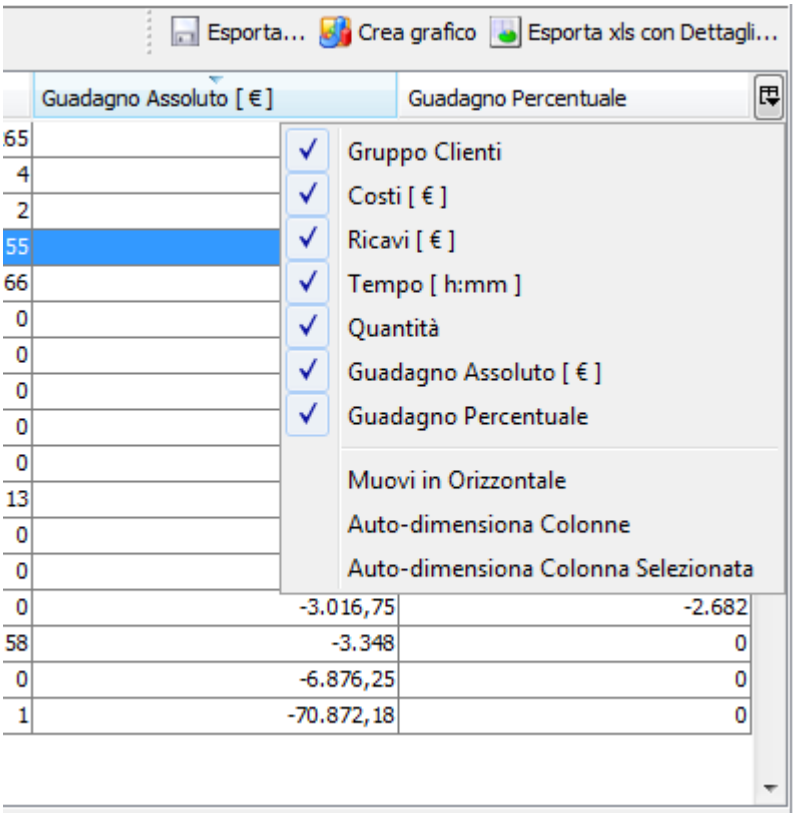


Figura 6.4: Funzionalità Interfaccia

6.2.1 Grafici

La lettura di tabelle può rivelarsi spesso complessa e comunque non certamente intuitiva in quanto è necessario leggere tutti i dati per poterne estrapolare informazioni utili.

Con Halite, soprattutto per facilitare le operazioni di confronto, è stata creata la funzionalità "crea grafico", che permette la visualizzazione tramite istogramma dei dati relativi ai guadagni assoluti, per tutte le query sopra riportate (Figura 6.5).

Del grafico si possono modificare alcune caratteristiche, ingrandire o rimpicciolire gli assi, modificarne l'orientamento da verticale a orizzontale e viceversa, lo si può salvare in formato .png o mandarlo direttamente in stampa.

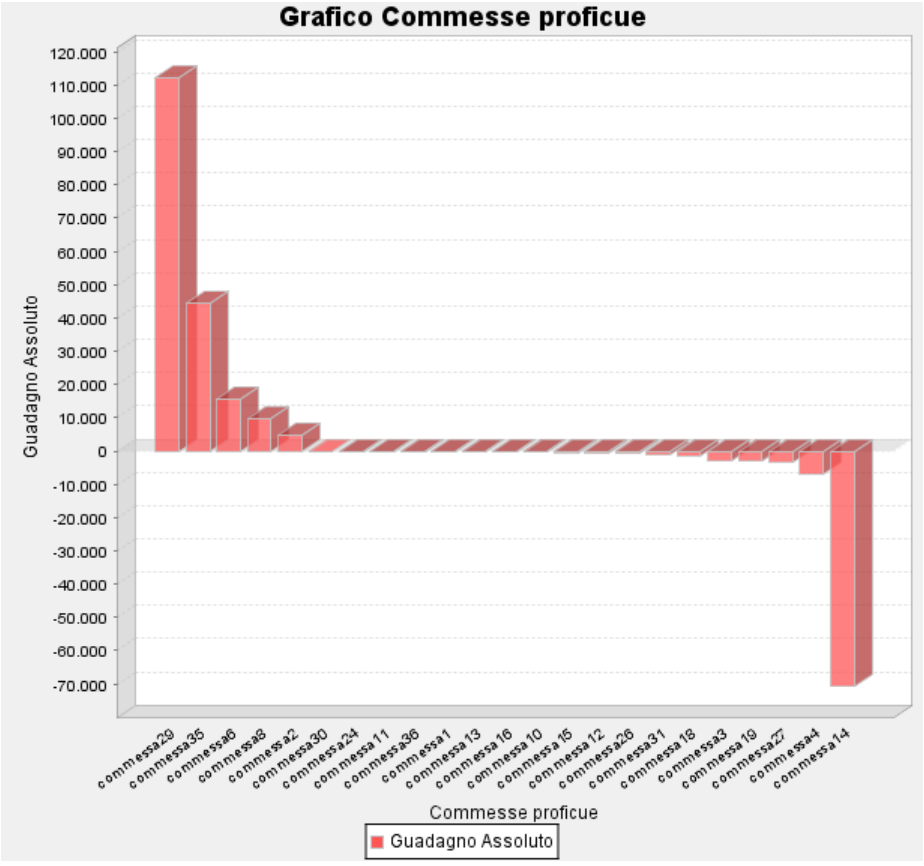


Figura 6.5: Grafico Commesse proficue

6.2.2 Esportazioni

Ai fini commerciali, avere dei dati per poterli esclusivamente consultare non è quasi mai sufficiente; questi devono essere analizzati, valutati, sommati ed eventualmente modificati e scambiati. Per raggiungere tali obiettivi è stata creata la funzione di esportazione: ogni tabella può essere esportata nei formati .csv e .xls in modo tale da poter effettuare tutte le operazioni necessarie, utilizzando i consolidati programmi di calcolo quali Excel, LibreOffice, eccetera.

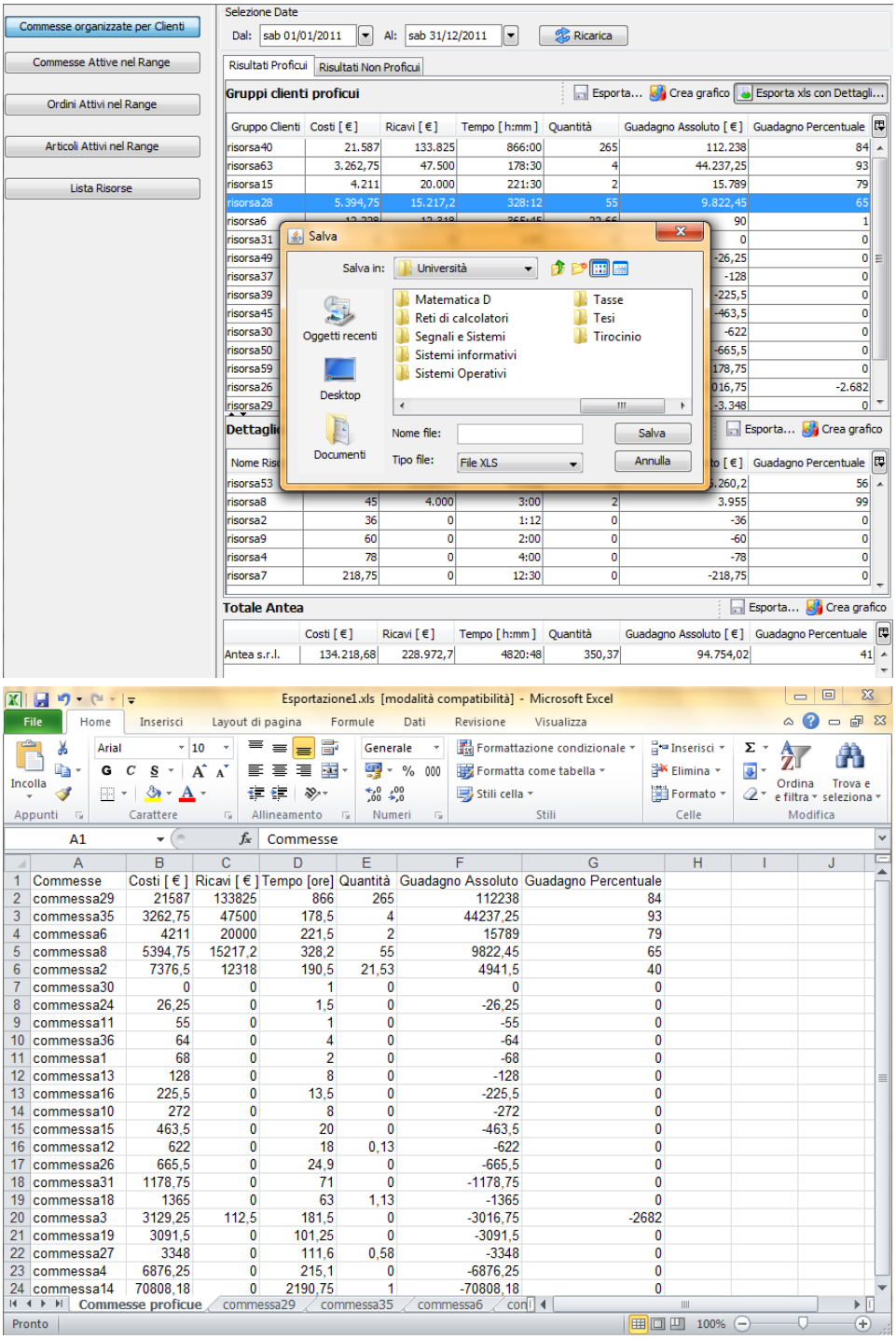
Halite mette a disposizione due modalità di esportazione che sono:

- L'esportazione della singola tabella;
- L'esportazione totale delle tabelle con i relativi dettagli.

Per quanto riguarda l'esportazione singola, si ha che il foglio di calcolo viene riempito con gli headers e i valori delle colonne mostrate della singola tabella che si è deciso di esportare, sia essa la tabella principale, la tabella che esprime i dettagli o la piccola tabella che mette in risalto il totale aziendale, e tale esportazione può essere fatta sia in formato .xls che .csv.

L'esportazione totale invece può essere effettuata solo dalla tabella principale, quella dal carattere più generale, e l'esportazione avviene solo in formato .xls. Nel foglio di calcolo esportato, oltre all'esportazione singola della tabella principale, esattamente analoga all'esportazione singola della stessa tabella, saranno presenti tanti fogli di calcolo quante sono le righe della tabella principale e su ciascuno di questi sarà presente l'esportazione della tabella di dettaglio relativo alla riga della tabella principale ad essa associata(Figura 6.6).

Per concludere la presentazione delle attività di Halite si aggiunge che tutta l'applicazione, in quanto presenta dati interni all'azienda relativi al bilancio della stessa, è stata protetta da un meccanismo di login che si ricollega al meccanismo di autenticazione del Database gestionale stesso (Figura 6.7). Inoltre viene creato un file di log in caso di errori, per cercare se possibile di ristabilire il corretto funzionamento.



Commesse	Costi [€]	Ricavi [€]	Tempo [ore]	Quantità	Guadagno Assoluto	Guadagno Percentuale
commessa29	21587	133825	866	265	112238	84
commessa35	3262,75	47500	178,5	4	44237,25	93
commessa6	4211	20000	221,5	2	15789	79
commessa8	5394,75	15217,2	328,2	55	9822,45	65
commessa2	7376,5	12318	190,5	21,53	4941,5	40
commessa30	0	0	1	0	0	0
commessa24	26,25	0	1,5	0	-26,25	0
commessa11	55	0	1	0	-55	0
commessa36	64	0	4	0	-64	0
commessa1	68	0	2	0	-68	0
commessa13	128	0	8	0	-128	0
commessa16	225,5	0	13,5	0	-225,5	0
commessa10	272	0	8	0	-272	0
commessa15	463,5	0	20	0	-463,5	0
commessa12	622	0	18	0,13	-622	0
commessa26	665,5	0	24,9	0	-665,5	0
commessa31	1178,75	0	71	0	-1178,75	0
commessa18	1365	0	63	1,13	-1365	0
commessa3	3129,25	112,5	181,5	0	-3016,75	-2682
commessa19	3091,5	0	101,25	0	-3091,5	0
commessa27	3348	0	111,6	0,58	-3348	0
commessa4	6876,25	0	215,1	0	-6876,25	0
commessa14	70808,18	0	2190,75	1	-70808,18	0

Figura 6.6: Esportazione in Excel

Figura 6.7: Login ad Halite

Capitolo 7

Considerazioni finali

7.1 Possibili migliorie

Un prodotto software, se ben fatto, non termina mai di crescere ed accrescere le proprie specifiche, migliorandosi continuamente. Semplicemente ad un certo punto si decide di impiegare le risorse in altri progetti terminando così l'evoluzione di un determinato prodotto.

Così come ogni programma, anche Halite presenta questa caratteristica e le migliorie apportabili potrebbero essere molteplici e tra le più disparate: dallo sviluppo di una interfaccia utente più accattivante fino all'aggiunta di modalità per modificare, organizzare, sistemare ed eventualmente elaborare i dati presenti, senza dover ricorrere a programmi esterni.

Anche l'aspetto dei grafici potrebbe essere migliorato. Infatti, attualmente il grafico viene riempito dai valori relativi ai "guadagni assoluti", in quanto queste erano le specifiche, tuttavia potrebbe risultare utile permettere all'utente di scegliere quale sia il parametro di cui visualizzare l'andamento grafico, e anche quale tipologia di grafico visualizzare (a torta piuttosto che un istogramma).

Una modifica che potrebbe essere particolarmente interessante e vantaggiosa è quella di trasferire l'intera applicazione a livello web. Infatti, nell'ormai era cloud, degli smartphone e dei tablet, essere in grado di raggiungere ed ottenere informazioni, anche di carattere lavorativo, in qualsiasi posto ci si trovi, in viaggio come in ufficio o a casa, può sicuramente risultare un vantaggio non indifferente, soprattutto da un punto di vista economico, per chi usufruisce del servizio e in generale per l'intera azienda.

7.2 Difficoltà incontrate

Halite è un progetto relativamente semplice, e le difficoltà riscontrate, a dire il vero, non sono state moltissime. Sicuramente, la causa delle difficoltà è la poca dimestichezza con gli strumenti che sono stati utilizzati: infatti, sebbene tutti gli strumenti citati in questo capitolo siano di notevole importanza per un ingegnere informatico, è necessario che l'ingegnere stesso impari a conoscere le funzionalità dello strumento prima di poterne usufruire al meglio e con notevoli vantaggi. Negli anni dell'università i progetti svolti sono stati pochi e di dimensione ridotta pertanto non si era mai programmato niente che comportasse l'uso di librerie esterne a quelle standard di Java e non si era mai presa in considerazione la possibilità di sviluppare codice in un ambiente di sviluppo integrato piuttosto che nel normale blocco note.

Sono quindi convinto che l'inesperienza sia stata la causa delle maggiori difficoltà riscontrate nella stesura del progetto, anche perché si può affermare che, di fatto, questo sia stato il mio primo progetto sviluppato in toto, dalla raccolta delle specifiche fino al rilascio dell'applicativo.

Capitolo 8

Conclusioni

Halite è uno strumento che ancora oggi, dopo quasi due anni dall'avvenuto rilascio, è apprezzato ed utilizzato per gli scopi e gli obiettivi previsti e recuperati durante il reperimento delle specifiche. Questo mi fa credere di aver adempiuto al meglio a quelle che erano, appunto, le richieste del committente.

Il tirocinio è stato sicuramente un'esperienza molto positiva e incredibilmente formante. Infatti, se da un lato alcune delle ricerche della prima parte di analisi sono state un semplice ripasso delle nozioni apprese durante vari insegnamenti del corso di laurea, in particolare Ingegneria del Software, dall'altro sono state approfondite delle voci che nei vari corsi non erano state analizzate o che erano state menzionate solo in parte.

Cimentarsi con la produzione di un progetto da capo a fondo credo sia stata una pratica particolarmente formativa. Ho potuto assistere a molte dinamiche aziendali ed inoltre sperimentare e toccare con mano tutti i vari strumenti di cui si è semplicemente sentito parlare. L'utilizzo di tali strumenti è stato molto interessante e mi ha dato prova concreta di ciò che era stato studiato e analizzato, consolidando così le mie conoscenze. Ritengo infatti che, magari non nell'immediato ma a distanza di tempo considerevole, un'attività svolta in maniera pratica rimanga molto più impressa di uno studio svolto e pertanto la conoscenza di quel determinato strumento piuttosto che delle motivazioni che ne sostengono l'utilizzo sia più facilmente assimilata e comporti un bagaglio culturale che difficilmente verrà dimenticato.

Elenco delle figure

Bibliografia

[1] http://www.anteash.com

[2] http://it.wikipedia.org

[3] http://www.extremeprogramming.org

[4] http://netbeans.org

[5] Roger S. Pressman - Principi di Ingegneria del Software, quinta edizione McGraw-Hill

[6] Kent Beck - Test Driven Development: By exemple, Addison-Wesley

[7] Gigi Sayfan - A build system for complex projects, articolo per Dr. Dobb's Journal del 7 Luglio 2009

[8] Cay Horstmann - Concetti di informatica e fondamenti di Java, Apogeo

[9] http://www.oracle.com

[10] http://www.hibernate.org

[11] http://www.jfree.org

[12] http://poi.apache.org