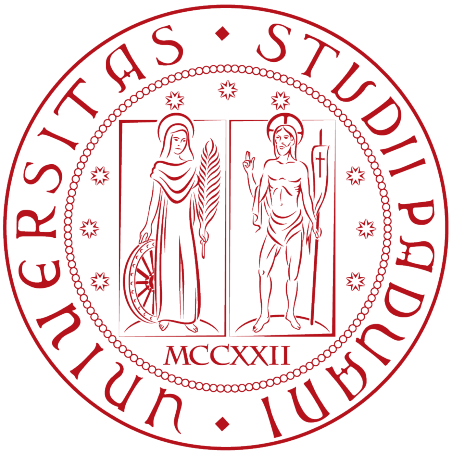
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Un'applicazione web per analisi statistiche su risultati di test

Tesi di laurea triennale

Relatore

Prof.Tullio Vardanega

Laureando

Christian Micheletti

Anno Accademico 2021-2022

Ringraziamenti

Dedicato ai miei famigliari, agli amici, ai colleghi, che mi hanno sostenuto in questi anni.

Ringrazio il mio relatore, il Prof. Tullio Vardanega, per le indicazioni e il prezioso contributo critico apportato durante la stesura.

Villa d'Almè, Luglio 2022 Christian Micheletti

Indice

Capitolo 1

Contesto lavorativo

1.1 L'azienda

1.1.1 Premessa

Per ragioni di trasparenza trovo opportuno fare la seguente premessa. Sono un dipendente presso l'azienda presso cui ho fatto il tirocinio da alcuni anni, alternando lavoro e studio *part-time*. Questo è un progetto con una posizione strategica nel contesto delle politiche aziendali di innovazione. Per ora tengo a precisare che, nonostante siano in parte dovute agli anni di collaborazione con questo gruppo, le informazioni presentate in questo elaborato derivano per la maggior parte dall'esperienza di tirocinio.

1.1.2 Panoramica

L'azienda presso cui ho svolto il tirocinio si chiama *Team Quality S.r.l.*, un'azienda fondata nel 1994 e situata a Villa d'Almè (BG) e con un distaccamento nella città di Treviglio (BG). Si occupa prevalentemente di:

* produrre *software* su commissione, per l'automazione e innovazione dei processi gestionali;
* fornire servizi di supporto sistemistico.

L'azienda conta 21 dipendenti e nel 2020 ha fatturato 2.709.866€[1]. Il personale è diviso in due gruppi, uno dedicato allo sviluppo *software* e uno ai servizi di carattere sistemistico. Io ho svolto il tirocinio nel primo dei due.

La clientela, per quanto riguarda lo sviluppo *software*, si divide in due categorie:

* **privati**, la categoria più numerosa, per i quali sviluppa perlopiù sistemi destinati a uso privato. Tra questi clienti annoverano nomi quali DNV GL, Rulmeca, ABB;
* **enti pubblici**, per i quali sviluppa applicazioni destinate al pubblico. Tra questi clienti spiccano Cosmetica Italia (Confindustria) e INGV, l'Istituto Nazionale di Geologia e Vulcanologia.

Sovente l'azienda prende parte ai convegni organizzati dall'Università degli Studi di Bergamo e dal Politecnico di Milano.

[1] `https://www.reportaziende.it/team_quality_srl_bg`

Come anticipato, la mia attività di tirocinio si è svolta esclusivamente nel gruppo di sviluppo *software*. Le seguenti sezioni presentano le caratteristiche dell'azienda limitatamente all'area di mia partecipazione. Pertanto in seguito mi riferirò "al gruppo" indicando il gruppo di lavoro con cui ho collaborato.

1.2 Metodologie e strumenti

In questa sezione descrivo le prassi aziendali con cui sono entrato in contatto, le tecnologie a supporto e come tali prassi le adoperino. Di seguito, saranno esposti solo i processi a cui ho preso parte, sia durante il tirocinio che al di fuori di esso.

* **gestione di progetto**, composta da:
  - attività di preventivo e consuntivo;
  - pianificazione e gestione del personale;
  - gestione dei rischi.
* **sviluppo**, composto da:
  - analisi;
  - progettazione;
  - codifica;
  - verifica.
* ***quality assurance***.

1.2.1 Gestione di progetto

Attività di preventivo e consuntivo

Dopo aver definito il preventivo iniziale, il progetto viene registrato in un sistema informativo aziendale sviluppato internamente, chiamato SIA. Al progetto vengono associati il preventivo e il personale. Questi membri del gruppo sono così abilitati a segnare le ore lavorate su quel progetto, tracciando automaticamente le attività svolte. Il Responsabile utilizza questi dati per produrre consuntivi di periodo.

Pianificazione

In fase di offerta il Responsabile effetua uno studio di fattibilità, seguito da una pianificazione a grana grossa divisa in fasi sequenziali. Una pianificazione tipica è divisa in queste fasi:

1. sviluppo della banca dati;
2. sviluppo delle funzionalità;
3. configurazione, validazione finale e installazione.

Durante lo svolgimento del progetto il Responsabile rifinisce queste fasi, dividendole in attività. Il Responsabile e l'Analista descrivono tali attività in *ticket* e le inseriscono nel *backlog* di progetto. Successivamente, il Responsabile le organizza in incrementi con durata di una o due settimane (in ore produttive). Questa gestione dei *ticket* viene effettuata con *Jira*.

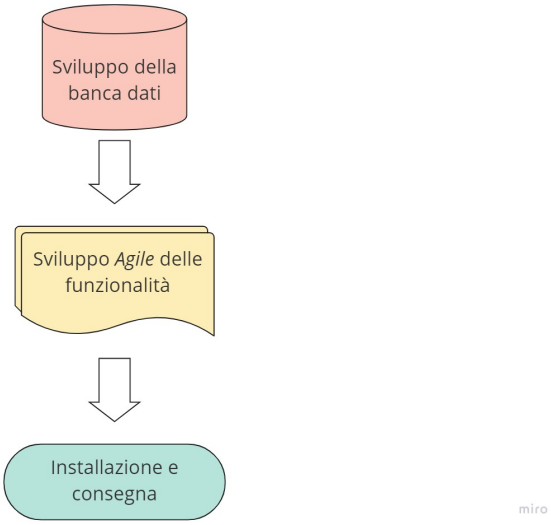


Figura 1.1: La struttura delle attività sequenziali per come sono previste nelle prassi aziendali.

Gestione dei rischi

Per l'attività di gestione e monitoraggio dei rischi, essendo il gruppo molto piccolo (12 persone), i membri non producono documenti formali, ma utilizzano un'istanza della *wiki* di *Redmine* per immagazzinare e catalogare i problemi accaduti e le soluzioni adoperate. Ogni membro del gruppo può aggiornare queste informazioni qualora lo ritenga opportuno.

Versionamento del *software*

Il gruppo gestisce il versionamento dei progetti con *Git*. I *repository* sono ospitati da un *server* interno. Recentemente, il gruppo ha iniziato ad adoperare *GitLab*.

1.2.2 Sviluppo della banca dati

Lo sviluppo della banca dati è la fase di analisi e progettazione preliminare che precede e poggia le fondamenta della vera e propria fase di sviluppo. La metodologia che gli Analisti usano nella fase iniziale è l'analisi orientata agli oggetti, un tipo di analisi che si concentra sugli elementi del dominio applicativo, piuttosto che sulle funzionalità[2]. Entrerò nel merito più avanti, per ora basta sapere che il prodotto atteso di questa fase è il modello del dominio applicativo. A supporto di questa fase gli Analisti usano *Toad Data Modeler* per produrre lo schema relazionale e lo *script* di creazione della banca dati.

A monte di questa fase si inizializzano *client* e *server* dell'applicazione, partendo da *fork* dei progetti "*starter*". In questi progetti vuoti un membro del gruppo esegue degli *script* che, partendo dalla banca dati, generano classi *Java* e *TypeScript* rappresentanti le entità della banca dati e il *boilerplate* necessario per eseguire le operazioni `CRUD` con

[2]Edward Yourdon Peter Coad. *Object-Oriented Analysis, First Editon*. Yourdon Press Computing Series, 1990.

lo strato inferiore. Riguardo questo processo e la natura del codice generato sarò più esaustivo in seguito, per ora basta sapere che il prodotto atteso di questa fase sono i progetti *client* e *server* con il codice necessario per effettuare operazioni `CRUD` con gli strati inferiori.

1.2.3 Sviluppo delle funzionalità

Le attività istanziate in ogni incremento sono:

* analisi;
* progettazione;
* codifica;
* verifica.

Gli incrementi sono divisi per componente da sviluppare, e quindi hanno un obiettivo definito in senso di prodotto. Le attività sono, perciò, circoscritte al componente da produrre.

Analisi

Il Responsabile e l'Analista incaricato eseguono l'analisi delle funzionalità a stretto contatto con il cliente, mediante incontri nei quali presentano le loro proposte. Spesso si recano sul luogo di lavoro del cliente per studiarne il metodo di lavoro, in modo da capire come e dove sia automatizzabile. Durante gli incontri con il cliente, il Responsabile e Analista prendono degli appunti, poi riportati in verbali interni su *Confluence*. Partendo da questi verbali, l'Analista organizza le funzionalità da implementare con *ticket* su *Jira*, e li traccia nei verbali. Gli appunti cartacei diventano allegati a tali verbali e il Responsabile li archivia in faldoni fisici.

Progettazione

L'Analista progetta l'architettura del sistema mediante diagrammi *UML* di *deployment*. Per la progettazione di dettaglio non è richiesta documentazione formale da parte dei Progettisti, tuttavia se lo ritengono opportuno possono preparare degli *UML* da aggiungere come allegato ai *ticket* associati. Il *software* per produrre i diagrammi, a differenza dei *software* elencati in precedenza, non viene specificato: i Progettisti possono usare quello che preferiscono. Tuttavia, ai Progettisti è richiesto di attenersi a un elenco finito di *pattern* ben conosciuti da tutti i membri. Nell'ottica dell'innovazione, vengono accettate anche nuove soluzioni, che però devono essere ben motivate e comunque approvate dal Responsabile.

L'analisi e la progettazione vengono spesso svolti insieme da una sola persona e il prodotto atteso è l'insieme dei *ticket Jira* con gli allegati necessari.

Codifica

Il Responsabile assegna i *ticket* ai Programmatori, che implementano quelle funzionalità secondo le indicazioni. La documentazione richiesta è l'insieme dei commenti sul codice, che seguono gli standard *JavaDoc* per il codice *Java* e *JsDoc* per *TypeScript*. Ci sono diverse situazioni in cui i commenti sono considerati molto importanti, come ad esempio quando c'è un lungo blocco di codice con logica applicativa di non immediata

comprensione. Altri linguaggi di programmazione usati sono *Python, C#, JavaScript* e *Scala*.

Verifica

La verifica del *software* avviene prevalentemente facendo test *e2e* che ne verifichino la correttezza. Di solito i test vengono implementati ed eseguiti sia dal Programmatore che ha implementato quella funzionalità che da un Analista. I test *e2e* vengono automatizzati tramite il *software Selenium IDE*. In certi casi vengono implementati test di unità, automatizzati con *KarmaJS* o *Junit*.

1.2.4 Validazione e consegna

La prassi di validazione più comune è di presentare il sistema al cliente, e se quest'ultimo si dice soddisfatto, si procede all'installazione. Solitamente, i sistemi vengono installati su *server* dedicati, sia esterni che messi a disposizione da *TeamQuality* stessa. L'installazione si avvale di una o più istanze di *Tomcat*, che espongono le `JSP` che fungono da *endpoint* dei progetti.

1.2.5 *Quality assurance*

La qualità dei processi e dei prodotti viene assicurata mediante la conformità alle ISO 9001:2015 e ISO 27001:2013, certificazione per la quale l'azienda si avvale di consulenza esterna.

Tabella 1.1: Riassunto delle attività e degli strumenti

Attività	**Metodo e strumento**
Consuntivo e gestione del personale	Strumento interno
Pianificazione e *ticketing*	*Jira*
Gestione dei rischi	*Redmine*
Versionamento del *software*	*Git*
Progettazione della banca dati	*Toad Data Modeler*
Analisi	Redazione dei verbali con *Confluence*, gestione dei ticket con *Jira*
Progettazione	*UML* come allegati dei *ticket*
Documentazione del codice	*JavaDoc, JsDoc*
Verifica	Test *e2e* con *Selenium IDE*, test di unità con *Karma, Junit*

1.3 Missione aziendale

In generale, l'azienda risponde a esigenze di innovazione delle infrastrutture informatiche, sia *software* che *hardware*. Il gruppo presso cui ho svolto il tirocinio si occupa del primo tipo di infrastrutture. Il tipo principale di processi che il gruppo si appresta ad innovare sono i processi gestionali dei clienti, ma spesso si approccia anche a problemi di domini specifici, per esempio, per l'INGV il gruppo ha sviluppato *software* che richiedeva una certa dimestichezza con l'ambito sismologico. Questa missione, l'innovazione dei sistemi gestionali, non riguarda solo la clientela esterna, ma anche *TeamQuality* stessa e le aziende dello stesso gruppo. Infatti, alcuni dei prodotti *software*

sono nati per l'utilizzo interno, e poi rifiniti per incontrare esigenze esterne, rendendoli più generici.

Le prassi aziendali prevedono modalità definite a priori per il miglioramento e i membri del gruppo dedicano sempre un po' di tempo per cercare e provare nuovi strumenti a supporto dei processi. Ovviamente, per discernere con criterio cosa sia appropriato mantenere occorre molta esperienza in ambito lavorativo e visione d'insieme. Non ci sono dei promotori designati di questa prassi, infatti ogni membro del gruppo può proporre in autonomia gli strumenti. Prassi più efficienti implicano processi aziendali più affidabili e prevedibili.

TeamQuality fa parte del gruppo *IMTEAM*, un gruppo di aziende ognuna specializzata in diversi tipi di servizi, ed è la più vecchia azienda del gruppo, per questo, tra le sorelle, è una sorta di riferimento. Un altro obiettivo di *TeamQuality* è consolidare e uniformare la modalità di sviluppo *software* del gruppo nella sua integrità. Questo obiettivo si colloca nel discorso sull'affidabilità e prevedibilità dei processi aziendali, perché uniformando le prassi di lavoro ogni azienda del gruppo beneficia dell'esperienza collettiva maturata dalle altre.

Diverse tipologie di prodotti e clienti implicano necessariamente strumenti diversi. Può capitare che, per inserirsi con profitto nelle realtà dei clienti, essi diano dei vincoli sulle tecnologie da usare, come *framework* e linguaggi di programmazione. Dalla direzione viene un forte impulso per provare tecnologie diverse, per esempio linguaggi a basso livello e banche dati a grafo. Questo obiettivo può sembrare in contrasto con quelli presentati precedentemente, perché può sembrare che da una parte si voglia ridurre il numero di tecnologie e tecniche adoperate, mentre dall'altra si voglia aumentarlo. Invece, non è una questione di quantità, ma di condivisione e diffusione di prassi che nel corso degli anni hanno raggiunto la maturità. Inoltre, sempre con l'obiettivo di portare nuove competenze, riporto che l'azienda propone spesso tirocini a giovani studenti, specialmente universitari. Questo perché, per esperienza personale, posso dire che i tirocini sono visti principalmente come un modo per formare possibili nuovi collaboratori. Riguardo questo aspetto, entrerò nel merito nel prossimo capitolo.

Riassumendo, a come ho interpretato le politiche di innovazione aziendali, ho individuato tre obiettivi:

1. ammodernare le prassi e le tecnologie di lavoro;
2. consolidare le prassi di lavoro tra le aziende dello stesso gruppo;
3. allargare le tipologie di prodotti da offrire.

Capitolo 2

Il tirocinio

2.1 Obiettivi

2.1.1 Ruolo dei tirocini nelle politiche aziendali

All'interno delle politiche aziendali di innovazione, esposte in §1.3, i tirocini trovano spazio per raggiungere obiettivi sia nel breve che nel lungo termine. In linea di massima, un tirocinio viene visto come un'opportunità per avvicinare un tirocinante all'azienda, con l'obiettivo di rimpinguare l'organico. Tuttavia, *TeamQuality* punta molto sui giovani, specialmente universitari, e li appoggia nel proseguire gli studi anche a livelli avanzati. Per questo, credo che i tirocini abbiano un ruolo nevralgico nelle politiche di innovazione, che va oltre al semplice preludio dell'assunzione.

Per sopperire alla necessità di nuovo personale, l'azienda adopera sia tirocini per studenti, che periodi di prova significativamente più lunghi per tirocinanti più qualificati. Per i periodi più brevi, ai tirocinanti vengono assegnati compiti piccoli e a sé stanti, calibrati per quelle che sono le loro capacità. Solitamente sono mansioni di codifica o di verifica. Per periodi lunghi più di un mese, i tirocinanti vengono maggiormente coinvolti nelle prassi aziendali, di modo che possano decidere con maggior sicurezza se quell'ambiente lavorativo faccia per loro. Questo processo è molto trasparente, perché il gruppo vuole assicurarsi che, qualora il tirocinante decidesse di rimanere, non sia per via di aspettative ingannevoli. In generale ogni tirocinante viene assegnato a un quadro, il quale funge da responsabile e referente per l'attività.

Il tirocinante viene spesso seguito dal responsabile, mentre per attività particolari è affiancato da un membro del gruppo specializzato per quel tipo di attività. Raramente, se l'area di influenza di quelle attività è molto circoscritta, al tirocinante vengono assegnate tecnologie e tecniche in fase di sperimentazione. La tipologia di tirocinio più comune, comunque, è quella di inserire i tirocinanti in un progetto già avviato e assegnargli la responsabilità di una componente piccola o comunque limitata di codice. Durante il tirocinio vengono poste le basi, principalmente per competenze, di una collaborazione più lunga. Gli obiettivi di innovazione inerenti l'efficienza delle prassi aziendali vengono perseguiti introducendo personale con esperienza lavorativa o giovani con un livello di istruzione avanzato.

2.1.2 Natura del progetto di tirocinio

Per le attività di gestione di progetto, le ricadute economiche in caso di fallimento possono essere molto gravose. Il progetto ivi presentato è un progetto *software* completo, ma molto piccolo. Per questo motivo, il tutor ha considerato adeguato il rischio di assegnarne la responsabilità a un membro meno esperto. Il vantaggio che ne deriva è che il sottoscritto potrà gradualmente farsi carico di maggiori responsabilità.

Questo progetto si pone l'obiettivo di automatizzare un processo gestionale del Cliente, rendendolo più efficiente e meno doloroso. È una tipologia di progetto molto comune per questo gruppo di lavoro, con dei risvolti facili da prevedere e possibili situazioni di criticità ben conosciute. Questo progetto, dunque, si presenta come un ambiente facile da controllare ma anche con conseguenze tangibili sulle realtà che incontra. Per questo non mi viene data la piena responsabilità (come poi approfondirò), ma una fetta giusta, che possa essere stimolante senza risultare sopraffacente.

2.2 Il problema affrontato

2.2.1 Analisi preliminare

Il Cliente è una multinazionale operante nella robotica, nell'energia e nell'automazione. Per ragioni di riservatezza aziendale, in questo elaborato non ne riporterò il nome, ma con il solo scopo di dare l'idea della sua dimensione, riporto che il Cliente ha fatturato quasi trenta miliardi di dollari nel 2021. Il prodotto del tirocinio è destinato a una divisione che, partendo da *report* di test su alcune tipologie di prodotti, come ad esempio schede madri, ne effettuano la validazione.

Gli utenti finali appartengono a due categorie:

* i fornitori del Cliente (esterni);
* i validatori presso il Cliente.

La prassi di lavoro che adoperano è la seguente:

1. i fornitori producono i prodotti ed eseguono i test. I risultati vengono riassunti in un *report*;
2. i validatori presso il Cliente raccolgono i *report* e li analizzano per capire se i prodotti testati sono stabili;
3. se anche un solo test fallisce, o se i risultati dei test hanno un'alta deviazione standard, il prodotto non può essere approvato;
4. se ogni test di una singola tipologia di prodotto dà esito positivo e i risultati dei test sono costanti, viene preparato un foglio Excel come allegato della documentazione necessaria per la validazione e il prodotto viene rilasciato sul mercato;
5. quando i test falliscono, i *report* vengono usati per effettuare analisi per cercare di capire dove operare per raddrizzare la produzione. Siccome la metodologia è manuale, queste analisi vengono eseguite meno spesso di quanto vorrebbero.

Questo flusso è illustrato nella figura 2.1. Gli obiettivi finali del prodotto sono i seguenti:

* automatizzare la creazione dell'Excel di validazione;

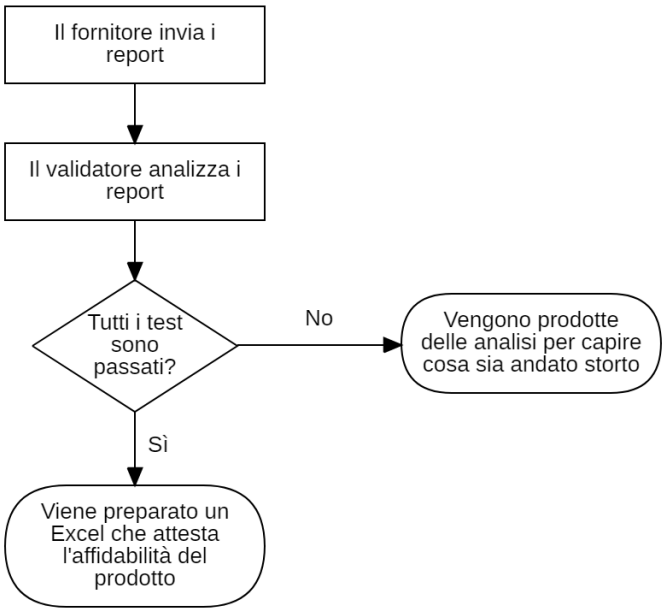


Figura 2.1: Rappresentazione del flusso di lavoro del Cliente

- effettuare analisi statistiche retrospettive sui fallimenti.

Il Cliente ha richiesto che l'applicativo deve essere raggiungibile tramite Internet, anche dai fornitori per caricare i *report* di test. In sostanza, i casi d'uso possono essere riassunti come segue (illustrato in 2.2):

1. i fornitori devono poter caricare i *report* di test nel sistema. Questo ha implicato, in fase di analisi, un'espansione di questo requisito in una gestione comprensiva di versionamento e cancellazione logica dei *report*. Senza il prodotto, i fornitori inviano delle *e-mail* con i *report* in allegato ai validatori. Per coprire alcuni casi limite, i validatori devono poter caricare anch'essi tali documenti;
2. i validatori devono poter scegliere un prodotto per il quale esistono diversi *report* di test e scaricare un foglio Excel dove sia presente l'elenco di tutti i test singoli eseguiti, i valori rilevati nei *report* e svariate statistiche che attestino la stabilità del prodotto;
3. i validatori devono poter individuare quali siano i prodotti per i quali la produzione fallisce più spesso, con lo scopo di analizzare i dati per estrarne delle considerazioni. L'obiettivo di questa analisi, di tipo retrospettivo, è capire in quale punto la produzione abbia fallito.

2.2.2 Requisiti proposti

A valle di un approfondimento con il Cliente, i requisiti per il primo rilascio sono i seguenti.

1. **funzionalità di autenticazione**:

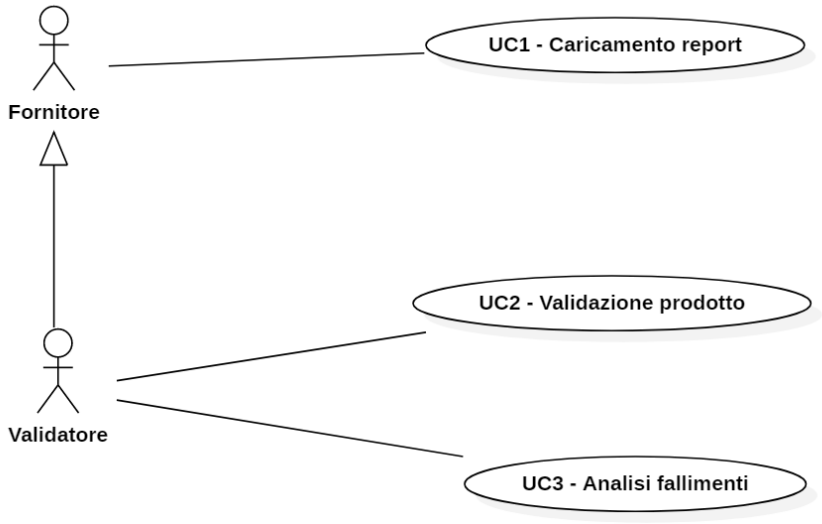


Figura 2.2: Macro-casi d'uso del sistema

   (a) funzionalità di accesso al sistema mediante nome utente e password;
   (b) funzionalità di gestione delle utenze (registrazione, cancellazione e blocco utenze);

2. ***dashboard* di validazione**:
   (a) funzionalità di visualizzazione dei prodotti validabili;
   (b) funzionalità di esportazione del foglio Excel di riassunto dei *report* di test;

3. ***dashboard* di caricamento dei dati**:
   (a) funzionalità di visualizzazione dello stato dei documenti salvati, ovvero in quale misura vi siano stati errori di interpretazione dei *report*;
   (b) funzionalità di caricamento dei *report*;
   (c) funzionalità di accesso ai dati;

4. **funzionalità di sincronizzazione dati**:
   (a) funzionalità di caricamento automatico da *file system*;
   (b) funzionalità di interpretazione dei *report*;
   (c) funzionalità di pulizia dei dati obsoleti.

Inizialmente, era prevista una funzionalità detta di **analisi dei fallimenti**, ma dopo alcuni colloqui con il Cliente, è stata ritenuta meno indispensabile della funzionalità di validazione. Pertanto, per il primo rilascio, questo requisito è stato sostituito dalla funzionalità di validazione. Ho omesso tutte le funzionalità contrattate per i rilasci successivi, perché non fanno parte degli obiettivi del tirocinio. Infine, vi è una richiesta del Cliente, ossia che l'applicativo venga installato su una macchina esterna alla loro rete e che sia raggiungibile tramite Internet.

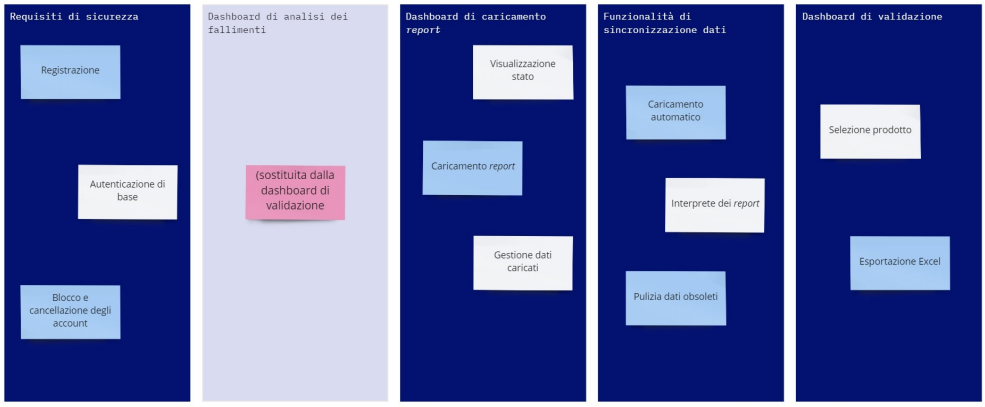


Figura 2.3: Rappresentazione dei requisiti del sistema: le colonne in blu scuro rappresentano i requisiti confermati, mentre il riquadro grigio, la funzionalità di statistiche sui fallimenti, è stata sostituita dai requisiti della *dashboard* di validazione.

2.2.3 Archiettura e *stack* tecnologico

Siccome il Cliente ha richiesto che il prodotto fosse disponibile attraverso Internet, l'architettura proposta è un monolite distribuito (come illustrato in figura 2.4). Il sistema è suddiviso in sei elementi principali:

* il **sistema di autenticazione**;
* il **sistema documentale**;
* la **banca dati**;
* il ***server*** dell'applicazione;
* il ***client*** dell'applicazione;
* il componente di **sincronizzazione**.

Il sistema di autenticazione e il sistema documentale sono dei prodotti aziendali *off-the-shelf*. Gli altri componenti sono i prodotti attesi dalla fase di sviluppo.

Lo *stack* tecnologico proposto è una combinazione di scelte molto comuni per questo gruppo di lavoro, pertanto il sottoscritto aveva già una buona dimestichezza con tali strumenti. Le scelte possono essere riassunte come segue:

* per la banca dati, *MySql 8*;
* per il *server* e il componente di sincronizzazione, *Java 8* e un *framework* interno, chiamato "Cadmo" nella sua versione 3.0;
* per il *client Angular 13*, dunque *TypeScript 4*, *CSS 3*, *HTML 5*.

2.3 Il progetto

2.3.1 Competenze da acquisire

Il ruolo del sottoscritto in questo tirocinio era quello dell'ingegnere del *software*. Più precisamente, le mansioni erano:

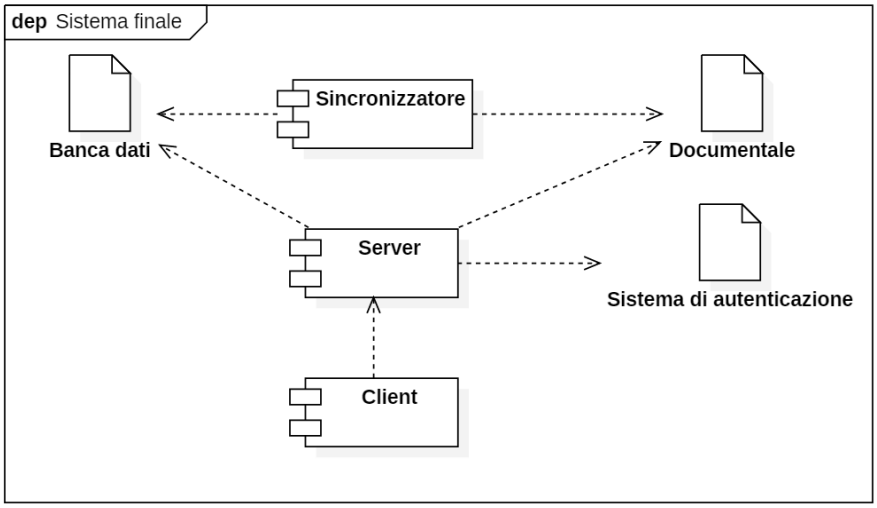


Figura 2.4: Diagramma di *deployment* del sistema.

* Analista;
* Progettista;
* Responsabile di progetto.

A queste, che per il sottoscritto erano novità assolute, si sono aggiunte le consuete mansioni di codifica e verifica, nel ruolo di Programmatore. In ottica formativa, le competenze, hanno toccato le seguenti attività:

* **pianificazione e gestione del personale**;
* **gestione dei rischi**;
* **analisi dei requisiti**;
* **progettazione**.

Ogni attività era pianificata svolgersi, nella sua fase iniziale o almeno la prima volta, con la supervisione del tutor aziendale, un ingegnere del *software* a livello professionale da oltre 30 anni. Era negli accordi che qualora mi fossi dimostrato competente mi avrebbero concesso maggiore autonomia.

2.3.2 Vincoli

Un primo vincolo riguarda il personale disponibile per il progetto. Poiché le priorità del gruppo erano catalizzate da altri progetti più critici, non sarebbero state assegnate risorse a tempo pieno su questo progetto oltre al sottoscritto. Le risorse assegnate avrebbero dunque avuto un monte ore estremamente limitato e sempre e comunque finalizzato al compimento di specifici obiettivi. In altre parole, raggiunti i risultati associati, non sarebbero più state disponibili. Altri vincoli degni di nota sono:

* attenersi all'architettura di sistema proposta al Cliente in fase di offerta (figura 2.4);

* limitarsi allo *stack* tecnologico proposto al Cliente in fase di offerta e successivamente approvato;
* attenersi agli standard aziendali di programmazione, nello specifico limitarsi ai *pattern* già fissati.

Questi vincoli sono applicati perché l'obiettivo del progetto non è fare ricerca o provare soluzioni inedite, ma fare formazione sulla ben rodata metodologia di lavoro del gruppo.

Di seguito riporto la suddivisione delle attività, con gli obiettivi da raggiungere, previste per 320 ore lavorative:

* **Prima settimana**:
  - sviluppo del *mockup* in vista dell'incontro con il Cliente.
* **Seconda settimana**:
  - presentazione del *mockup* al Cliente;
  - analisi del dominio applicativo;
  - sviluppo dell'interprete dei report.
* **Terza settimana**:
  - progettazione dell'interfaccia e dell'esperienza utente;
  - impostazione ambiente di sviluppo *server*;
  - impostazione ambiente di sviluppo *client*;
  - integrare le funzionalità di autenticazione.
* **Quarta settimana**:
  - sviluppo funzionalità di base di accesso ai dati;
  - sviluppo del motore delle statistiche.
* **Quinta settimana**:
  - sviluppo delle statistiche da eseguire.
* **Sesta settimana**:
  - sviluppo dell'importatore manuale;
  - sviluppo dell'esportatore dei dati.
* **Settima settimana**:
  - validazione presso il cliente;
  - installazione e configurazione.
* **Ottava settimana**:
  - sviluppo dell'importatore automatico;
  - redazione del manuale d'uso per i fornitori.

Guardando il piano di lavoro, definito nel dettaglio già prima dell'inizio del tirocinio sarebbe naturale dubitare del mio effettivo contributo ai processi di pianificazione e analisi. Il tutor aveva pianificato queste attività dopo aver definito la struttura del sistema, nello specifico quali fossero i prodotto da consegnare a fine tirocinio. Tuttavia, è spettato a me gestire la pianificazione, partendo dal piano di lavoro come base; e poi definire i requisiti dei singoli *deliverable*, usando come base gli obiettivi ivi elencati.

Per esigenze sia aziendali che del Cliente ho cambiato la disposizione dei requisiti inizialmente previsti. Tuttavia, gli obiettivi del tirocinio non hanno subito variazioni che non siano direttamente riconducibili alle priorità del Cliente, durante il suo corso.

2.3.3 Obiettivi professionali

Questo specifico tirocinio è stato visto come un'opportunità per accelerare la formazione professionale di un dipendente. Le mansioni del sottoscritto si sono sempre limitate a codifica e verifica, mentre il tirocinio prevedeva che alla fine del progetto fossi più avvezzo a:

* gestire la pianificazione di un progetto software;
* gestire i rischi di un progetto software;
* effettuare analisi e progettazione secondo le direttive interne.

Il motore di questo tirocinio è la necessità di avere più personale capace di assumere maggiori responsabilità. Ciò è in linea con il percorso professionale che mi sono figurato. Il mio obiettivo nel medio periodo è di raggiungere una comprensione completa e lucida dei processi dell'ingegneria del *software*, al fine di intraprenderla proficuamente come carriera. Ovviamente ciò richiede molta esperienza sul campo, per questo il tirocinio è stato un punto di partenza per avere più opportunità per inserirmi a livelli di maggiore responsabilità e autonomia decisionale. Infine, una capacità sottovalutata a mio avviso, che mi aspettavo di guadagnare, è quella di finalizzare un progetto. Con ciò intendo essere in grado di definire degli obiettivi, dei criteri di accettazione e un ragionevole intorno entro cui ci si possa dire soddisfatti; una cosa che si può fare solo da un punto di vista più alto.

Rispetto alle diverse offerte che mi sono state proposte, questa era quella con il grado di responsabilità maggiore. Diverse aziende mi hanno proposto di lavorare con mansioni e strumenti che già conosco, altre proposte più allettanti includevano tecnologie veramente interessanti o domini applicativi di mio interesse. La scelta infine si è ridotta a due offerte: il presente tirocinio, e un'altra che prevedeva di utilizzare uno *stack* tecnologico con cui mi piacerebbe molto lavorare in futuro, ma con la mia solita mansione. La scelta che ho preso ha comportato più esperienza con maggiori responsabilità: a posteriori non la rimpiango.

Capitolo 3

Svolgimento di progetto

3.1 Prassi aziendali

3.1.1 Gestione di progetto

Uno degli obiettivi del tirocinio era formare il sottoscritto nelle prassi aziendali di gestione di progetto. Le attività che ho svolto, prima con il tutor, poi in autonomia, comprendono:

* pianificazione;
* gestione del personale;
* gestione dei rischi.

La competenza in queste tre attività è necessaria qualora si voglia intraprendere la gestione di un progetto software con successo. Non mi è stato detto, ma presumo che sia per questo motivo che il tutor ha considerato queste attività oggetto di formazione e valutazione, piuttosto che altre.

Pianificazione

Tra il personale assegnato al progetto, è il Responsabile a svolgere la pianificazione. Il gruppo gestisce la pianificazione con *Jira*, seguendo delle prassi ispirate allo sviluppo

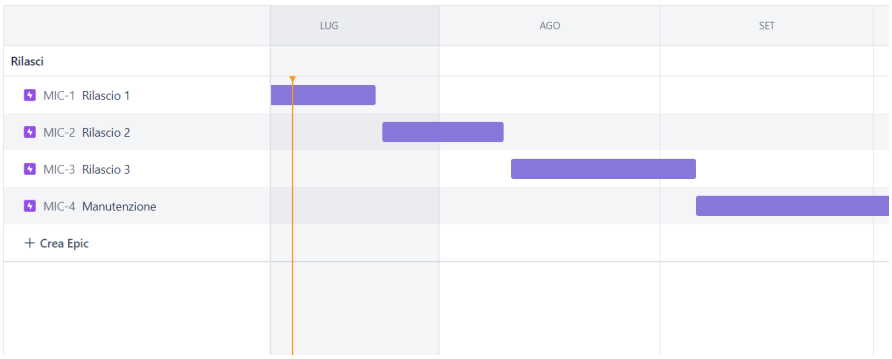


Figura 3.1: Esempio di organizzazione temporale della pianificazione con i *ticket epic*.

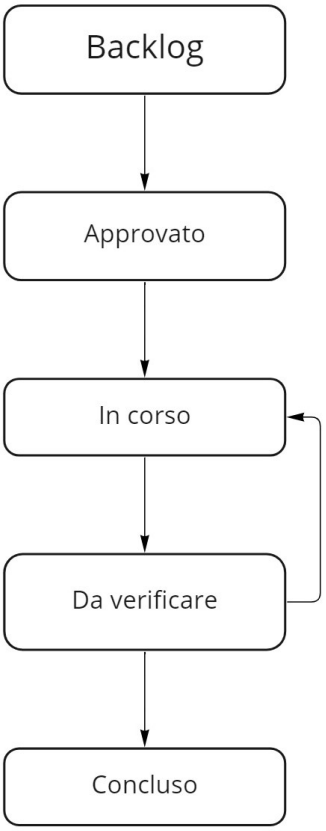


Figura 3.2: Il ciclo di vita di un ticket.

Agile. All'inizio del progetto, il Responsabile definisce i rilasci, rappresentati da *ticket "epic"*. In *Jira* gli *epic* sono visualizzati in un diagramma temporale, simile a un diagramma di Gantt. A tali rilasci vengono poi associati svariati *ticket*, che possono essere storie utente, bachi da risolvere o funzionalità da implementare. Questi *ticket* sono raccolti nel *backlog* di progetto. Il ciclo di vita di un *ticket* ha i seguenti stati, rappresentati da colonne nella *board* di progetto:

1. **da iniziare**: i *ticket* nel backlog;
2. **in corso**: un *ticket* la cui codifica è iniziata dall'assegnatario;
3. **da verificare**: un *ticket* considerato concluso dalla persona a cui è stato assegnato e pertanto richiede una seconda verifica da un altro membro del gruppo;
4. **concluso**: un *ticket* viene chiuso quando un referente per il progetto ne ha validato il risultato.

Gestione dei rischi

La comunicazione informale è caldamente raccomandata come strumento per affrontare difficoltà e situazioni di emergenza. Tuttavia, risulta più efficiente avere uno storico di casi di criticità e i relativi modi di risolverli, da consultare prima di rivolgersi ad altri

membri. Tra le mansioni del Responsabile figura la manutenzione periodica di questa raccolta, organizzata mediante una wiki di *Redmine*.

Gestione del personale

Una mansione che costituisce una fase critica della gestione di progetto è la gestione del personale. Un Responsabile di progetto deve essere in grado di gestire le risorse umane assegnate. Come già accennato, la comunicazione tra i membri del gruppo è perlopiù informale. Le comunicazioni formali avvengono con strumenti quali *e-mail*, appunti su *Confluence*, commenti sui *ticket* di *Jira* vengono comunque utilizzate, e hanno lo scopo di tracciare delle decisioni.

3.1.2 Sviluppo *software*

La fase dello sviluppo, nelle prassi del gruppo, è caratterizzata da due tipi di attività. La prima, per importanza e collocazione temporale, si ispira all'*Analisi Orientata agli Oggetti*. La maggior fonte di informazioni e tecniche è il libro *Object-Oriented Analysis* di Peter Coad. La seconda segue prassi tipiche della metodologia *Agile*, più diffuse. L'obiettivo di seguire queste due diverse metodologie è di rendere lo sviluppo il più resistente possibile agli errori durante le attività di analisi, senza rinunciare alla flessibilità dello sviluppo *Agile*.

Sviluppo orientato agli oggetti

Vi è sempre una fase iniziale dello sviluppo dedicata all'analisi orientata agli oggetti, una metodologia che si concentra sulla natura del dominio del problema. Questa fase precede temporalmente l'inizio dello sviluppo *Agile*, poiché le funzionalità del sistema si baseranno sui risultati dell'analisi orientata agli oggetti. Il prodotto atteso è il modello del dominio. Nel pratico, vengono prodotte la banca dati e gli scheletri degli eseguibili del progetto. Il gruppo adotta delle prassi che permettono di modificare chirurgicamente il risultato di questa fase, senza ripeterla nella sua interezza. Questo perché tornare indietro a questa fase significherebbe ricominciare il progetto da capo. Descriverò le prassi per gli aggiornamenti in seguito.

Questa attività viene eseguita dal Responsabile e dall'Analista incaricato, adoperando alcune tecniche per rappresentare il dominio applicativo secondo diversi strati. Gli strati (*layers*) dell'analisi sono così definiti[1]:

* **oggetti** (*objects*): categorie di entità del mondo reale. Un oggetto in OOA corrisponde a una classe nei linguaggi di programmazione a oggetti;
* **strutture** (*structures*): aggregazioni o classificazioni di oggetti;
* **soggetti** (*subjects*): uno o più esempi di istanze degli oggetti e la natura delle loro connessioni;
* **attributi** (*attributes*): ogni attributo ha lo scopo di descrivere un'istanza di un oggetto;
* **servizi e messaggi** (*services and messages*): un servizio è un processamento che viene eseguito alla ricezione di un messaggio. Riporto il presente punto per completezza, ma attualmente non trova spazio nelle prassi aziendali.

[1] Edward Yourdon Peter Coad. *Object-Oriented Analysis, First Editon*. Yourdon Press Computing Series, 1990.

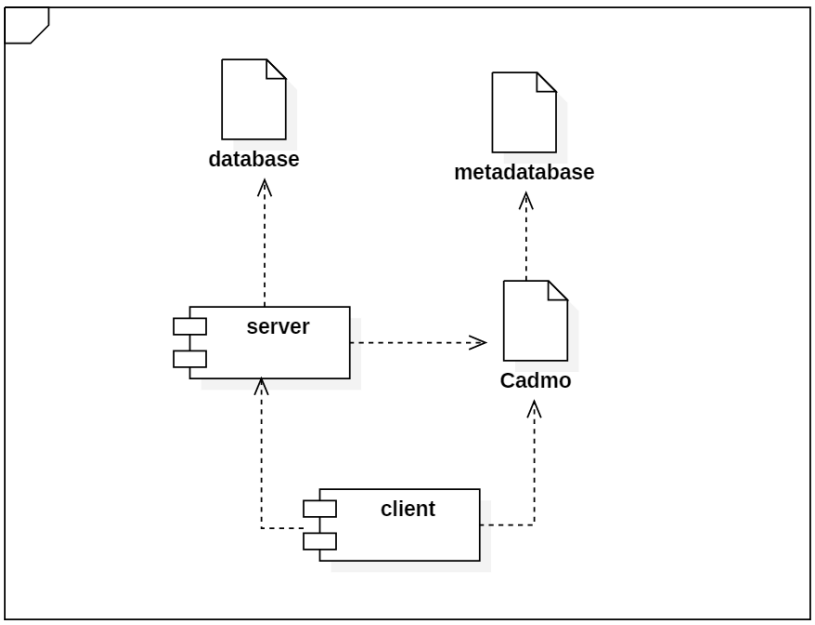


Figura 3.3: Esempio di architettura di un prodotto con Cadmo.

Questi livelli di analisi, specialmente riguardo le interazioni tra gli oggetti, sono ispirati al modello del linguaggio *Smalltalk*[2].

Il risultato di queste tecniche è il modello relazionale della banca dati, prodotto con il *software Toad Data Modeler*, che partendo da un diagramma entità-relazione genera in automatico lo *script* SQL di creazione della banca dati. Ci sono moltissime convenzioni da seguire per la definizione della banca dati, alcune derivano dalla OOA, altre si sono consolidate negli anni. Seguire queste linee guida è cruciale, perché la banca dati e il modello del sistema sono le fondamenta degli eseguibili, prodotti durante le attività di sviluppo. Il gruppo adopera un *framework* sviluppato internamente, **Cadmo**, che permette la generazione di codice *boilerplate* utile per operazioni *CRUD* attraverso *HTTP*.

Cadmo è un *framework* estremamente opinionato scritto in *Java*, creato per costruire applicazioni web con la tecnica del *server-side rendering*. Negli anni le necessità del gruppo hanno virato in una direzione più orientata al *client-side rendering*, pertanto quello che era un motore di generazione di pagine HTML ora espone *endpoint* JSON. Cadmo è una raccolta di classi, funzioni e *script* che negli anni si sono dimostrati spesso utili. Il gruppo adopera dei *template* per inizializzare gli eseguibili *server* e *client*, i quali contengono le librerie comunemente usate, tra cui Cadmo. Tra gli *script* più importanti cito i seguenti:

* gli *script* di generazione delle **classi di persistenza**, ovvero le classi rappresentanti le entità del modello e del *boilerplate* a supporto. Ce n'è uno per ogni linguaggio di programmazione usato dal gruppo. Queste classi vengono create all'interno degli eseguibili vuoti, estratti dai *template*;

[2]Edward Yourdon Peter Coad. *Object-Oriented Analysis, First Editon.* Yourdon Press Computing Series, 1990.

* lo script di creazione del ***metadatabase***, una banca dati che contiene informazioni che il framework usa per validare e gestire gli endpoint.

Tra le altre funzioni di Cadmo, cito la funzione di *scheduling* dei processi periodici. Questo *framework* ha una posizione strategica all'interno delle prassi aziendali, perché è stato creato con lo scopo preciso di supportarne la filosofia dello sviluppo.

Il motivo principale per cui viene analizzato prima il dominio applicativo, piuttosto che le funzionalità, è che i requisiti cambiano più frequentemente degli elementi che compongono una realtà lavorativa. Un requisito può essere frainteso facilmente, le priorità del Cliente possono variare. Ogni possibile funzionalità che riguarda *quel dominio* avrà sempre a che fare con *quegli* oggetti, per questo il lavoro fatto in fase di analisi del dominio applicativo è difficile che si riveli uno spreco di tempo. Il risultato di questa tecnica è avere una *baseline* di codice sulla quale costruire agilmente le varie funzionalità, e dunque limitare i possibili danni dallo "smontamento" di funzionalità non più richieste. Come esporrò in seguito, questo *modus operandi* ha assicurato stabilità al progetto del tirocinio.

Il problema principale con questa prassi è che qualora venga male intesa la natura di una o più entità del sistema, modificarla e tracciare le modifiche a cascata necessarie sono compiti onerosi, critici e introducono facilmente malfunzionamenti imprevisti. Per fronteggiare queste evenienze, sono state studiate delle procedure correttive, riportate nella *wiki* dei rischi. A parte particolari situazioni in cui bisogna comunque fare attenzione, con il supporto degli *script* del *framework*, gran parte del processo è automatizzata. Si effettuano numerosi incontri con il Cliente con l'obiettivo di del loro modo di lavorare, da cui si evincono i requisiti funzionali e non funzionali del sistema. In certi casi, dove il progetto è abbastanza piccolo da permetterlo, il *client* viene costruito con lo scopo iniziale di fornire un *mockup* con scopo dimostrativo. Se invece il progetto è molto grande, il mockup viene costruito con il programma *Balsamiq*, perché in quel caso sarebbe meno costoso.

Riassumendo, questo processo di analisi poggia le fondamenta per il progetto con i seguenti prodotti:

* banca dati;
* *metadatabase*;
* scheletro del server;
* scheletro del client.

Una volta dichiarata conclusa questa fase, si parte con lo sviluppo delle funzionalità del sistema.

Sviluppo *Agile*

La fase di sviluppo, più canonica rispetto ai processi dell'ingegneria del *software*, segue le orme della metodologia *Agile*. Di solito un progetto ha un Responsabile e un Analista incaricato; entrambi hanno funzione di referenti per il progetto. Il Responsabile è incaricato di operazioni di carattere manageriale di analisi, mentre l'Analista è incaricato anch'esso dell'analisi, ma anche della progettazione del codice e dell'esperienza utente e della validazione. I Programmatori assegnati al progetto hanno responsabilità in base alla loro esperienza, partendo da codifica e verifica.

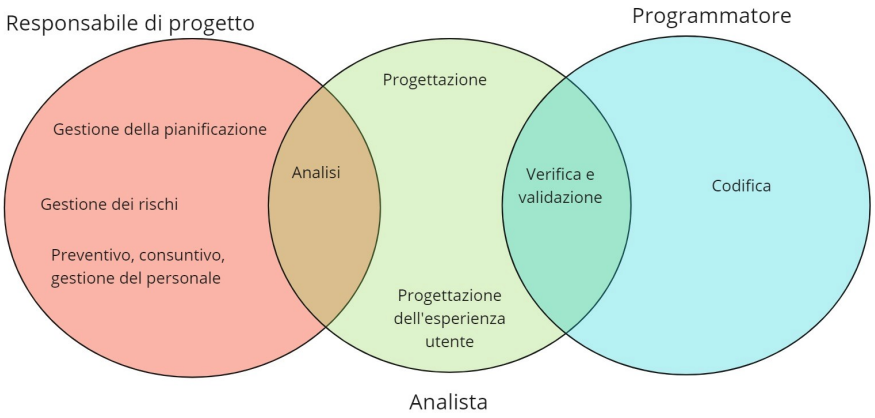


Figura 3.4: Suddivisione delle mansioni per ruolo.

L'analista raggruppa le funzionalità da implementare per aree di interazioni utente che per vari motivi sono correlate. Queste aree sono i prodotti dell'attività di progettazione dell'esperienza utente. Queste aree di interazione sono fondamentalmente gruppi di *ticket*, raggruppati in maniera arbitraria. Come detto in precedenza, tali *ticket* sono organizzati per rilasci. Ogni area di interazioni utente comuni costituisce un'unità di pianificazione che dura da una a tre settimane, paragonabile a uno *sprint* nella metodologia *Scrum*. Di seguito descrivo le prassi di sviluppo di un singolo *sprint*:

1. ogni *sprint* parte con un'analisi delle funzionalità richieste, partendo dagli appunti degli incontri e dalla progettazione dell'esperienza utente. Il prodotto di questa fase sono dei *ticket* di *Jira*, che spiegano nel dettaglio le funzionalità da implementare;

2. viene poi dedicato del tempo per definire l'architettura del codice, che di solito si limita a includere i *pattern* ben conosciuti e adoperati dal gruppo. Chi svolge le mansioni di progettazione, può scrivere delle note inerenti a quali *pattern* usare nei *ticket*;

3. i *ticket* prodotti dalle attività di analisi e progettazione vengono posti nel *backlog*. I Programmatori possono scegliere in autonomia quali *ticket* non "in corso" eseguire, ma generalmente i *ticket* vengono assegnati dal Responsabile. Il Programmatore assegnato implementa quella funzionalità negli eseguibili inizializzati nella fase di sviluppo preliminare. Il Programmatore effettua la verifica del suo stesso codice;

4. quando il Programmatore ritiene di aver soddisfatto il requisito, chiede all'Analista incaricato di verificare la funzionalità. Se la verifica ha esito positivo il *ticket* viene chiuso, altrimenti dà delle indicazioni e il *ticket* viene catalogato come "in corso".

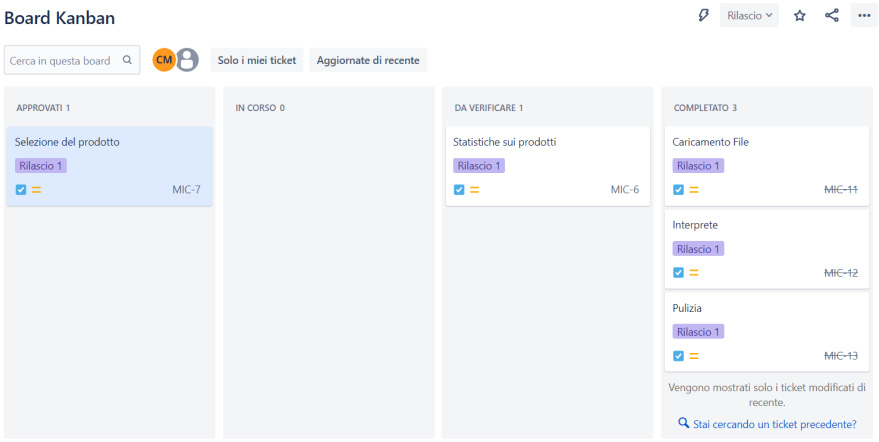


Figura 3.5: Esempio di una bacheca *Jira* in uso. Ogni colonna rappresenta uno stato dei *ticket*. La *board* indica che i requisiti di selezione di prodotto sono previsti per lo *sprint* corrente ma non ancora inizati e i requisiti di statistiche sui prodotti sono in attesa di validazione da parte di un Analista o Responsabile, mentre i requisiti relativi al caricamento *file*, all'interprete e al demone di pulizia sono soddisfatti.

Fornitura

Ogni rilascio è preceduto da una validazione con il Cliente. La modalità è spesso una presentazione, nella quale si simula le interazioni richieste dal punto di vista dell'utente finale. La metodologia di installazione segue dei passi descritti nella wiki dei problemi comuni.

3.2 Sviluppo dell'applicazione

3.2.1 Attività di pianificazione e gestione del personale

All'inizio di ogni settimana, mi sono riservato un po' di tempo (di solito circa un'ora) per svolgere l'attività di modifica della pianificazione, in base al consuntivo della settimana precedente e alle priorità del Cliente.

La pianificazione era fatta utilizzando i *ticket* di *Jira*, fissando la pianificazione con dei *ticket* detti *epic*. All'interno di tali *ticket* non ci sono suddivisioni o pianificazioni temporali. Per facilitare le attività di modifica della pianificazione, in via eccezionale, abbiamo associato un *epic* ad ogni settimana. All'inizio di ogni settimana selezionavo alcuni *ticket* da implementare e li raggruppavo in *sprint* (da intendersi come funzionalità del *software* Jira, non come *sprint* conformi alla tecnica *Scrum*), i quali duravano due o tre giorni. La bacheca del progetto veniva quindi riempita in automatico e successivamente assegnavo i *ticket* al personale, in base alle loro competenze. Gli obiettivi di questa attività erano che a ogni *epic* fosse associato una quantità di obiettivi quantificati in prodotti. Criterio di valutazione sarebbe stato il rapporto tra il numero di ore totali impiegate (quindi sommando le ore di ogni membro) con quelle previste, ossia 432 (320 mie e 112 ripartite tra il tutor e altri due membri del gruppo).

3.2.2 Attività di gestione dei rischi

Alla fine di ogni settimana, ho dedicato un po' di tempo nell'analisi dei problemi incontrati in corso d'opera. Per le prime tre settimane non ci sono stati imprevisti. Durante la quarta settimana ho fatto un errore di valutazione sull'impegno per lo sviluppo dell'esportatore Excel, che richiedeva una buona conoscenza della libreria *Apache Poi*. Avevo messo in conto che avrei dovuto fare formazione per imparare a usarla, ma non ho preventivato tempo sufficiente a questa attività.

Nella quinta settimana invece ho incontrato un problema che, a posteriori, era facilmente prevedibile. Era la settimana della Festa della Repubblica, che cadeva di giovedì e non avevo previsto che le risorse assegnatemi per quell'attività, incluso il tutor, avrebbero fatto il ponte. Avevo previsto che facessero una giornata di ferie, non due e il ritardo accumulato quella settimana era di due giorni. Dall'analisi che ho fatto in quell'occasione è risultato un aspetto importante, ossia che è sempre bene chiedere a inizio mese chi abbia in programma delle ferie. In concomitanza con il tutor aziendale, ho riportato l'accadimento nella *wiki* dei rischi.

Un altro rischio accaduto è uno sbaglio nell'analisi della natura di alcuni elementi del sistema. Come ho già esposto in precedenza, questo modello di sviluppo si basa molto sull'analisi del dominio applicativo, perciò è molto sensibile agli errori che si fanno in quella fase. Si trattava del modo di identificare i *report*. All'inizio il Cliente ci aveva detto che i *report* hanno un identificativo univoco e che i duplicati erano, in ogni caso, difformità dal regolamento interno. Tuttavia, nell'incontro della sesta settimana è emerso che in realtà esisteva un caso in cui gli identificativi potevano essere doppi. Ci hanno poi detto come avremmo potuto identificare dunque i *report* e con queste informazioni ho applicato le direttive di modifica del modello del dominio riportate sulla *wiki* dei rischi.

3.2.3 Attività di consuntivo

Alla fine di ogni settimana, contestualmente alle attività di gestione dei rischi effettuavo, tramite il SIA come esposto in §1.2, le attività di consuntivo. La prassi era la seguente: dal SIA ottenevo i dati, prima filtrando per la settimana, per vedere se ci fossero marcate differenze tra l'impegno previsto e l'impegno consuntivato, indice di pianificazione imprecisa. Infine comparavo il grado di avanzamento con gli obiettivi raggiunti, per avere un rapporto tra prodotti completati e prodotti attesi.

Per la quarta settimana si era verificato un ritardo di una giornata e per la quinta di due giornate. La soluzione è stata spostare la pianificazione degli *epic* avanti di quei giorni. Per la sesta settimana, il bilancio orario era in rosso di cinque giornate. Per rientrare in questo ritardo, il tutor ha assegnato alle attività del progetto un Programmatore in più del previsto per la settima settimana.

3.2.4 Analisi del dominio applicativo

L'analisi del dominio, ispirata all'analisi orientata agli oggetti, è una delle prime attività a cui ho partecipato, svolta nella seconda settimana. Come già esposto, l'obiettivo di questa fase è avere il modello del dominio applicativo definito, per basare su di esso la fase di sviluppo *Agile*. Di seguito, un esempio di come si è svolta l'analisi del modello:

1. inizialmente, io e il tutor abbiamo identificati gli oggetti[3]. Il risultato di questa fase include, ma non si limita a, il prodotto del Cliente e i *report* di test. Senza

[3]Nel senso dato dal Coad, corrispondono con le classi nei linguaggi di programmazione a oggetti.

entrare nel dettaglio, da questa fase sapevamo che la banca dati avrebbe dovuto avere una tabella chiamata `product` e una chiamata `report` (figura 3.6);

2. dopo, abbiamo individuato le strutture[4], per aggregazione e classificazione. Siccome i *report* contengono le informazioni dei punti di test e i risultati, è risultato ovvio che tra `report` e l'entità del singolo test (`test_point`) ci sia una relazione 1:N identificante (figura 3.7);

3. dopo, abbiamo individuati i soggetti[5], che corrispondono in maniera grezza alle relazioni non identificanti. Per esempio, un *report* è associato a un prodotto: pertanto vi sarà una relazione tra i due (figura 3.8). La relazione in questo esempio è identificante per via della natura del *report*: diversamente dalla fase precedente, il risultato dell'analisi dei soggetti è una relazione che può anche non essere identificante;

4. infine, abbiamo descritto i vari oggetti mediante gli attributi (figura 3.9).

Figura 3.6: Risultato dell'analisi degli oggetti del dominio, ovvero le tre tabelle, senza né relazioni né attributi.

Figura 3.7: Risultato dell'analisi delle strutture del dominio, ovvero una relazione 1:N tra `report` e `test_point`. Si tratta di una relazione di aggregazione, poiché i risultati dei test sono contenuti in un *report*.

Figura 3.8: Estratto dell'analisi dei soggetti del dominio, ovvero una relazione 1:N tra `product` e `report`. La relazione è di tipo interazione, poiché un *report* è il risultato di un processo eseguito su un prodotto.

[4]Nel senso dato dal Coad, si tratta di aggregazioni e generalizzazioni.
[5]Nel senso dato dal Coad, risponde alla domanda "con quali oggetti interagisce questo oggetto?".

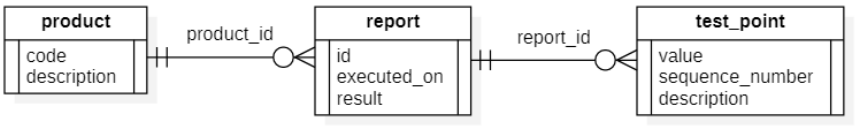


Figura 3.9: Estratto del risultato degli attributi. Ogni attributo corrisponde a un elemento che descrive un report, o una sua proprietà.

Un esempio del risultato finale si trova in figura 3.9. La banca dati reale conta invece una decina di entità, contando anche le entità relazionali.

3.2.5 Analisi dei requisiti

Ho effettuato l'analisi dei requisiti ogni volta che era previsto lo sviluppo di un'applicazione. Emblematico l'incontro presso la sede del proponente, che ha sottolineato l'assenza di un requisito fondamentale, che non avevamo affatto inteso in prima battuta. L'incontro si è svolto presso la loro sede, dopo la presentazione del *mockup*. Lì ci hanno mostrato i prodotti del loro lavoro e ci hanno descritto con degli esempi come li fanno. Una volta tornato in ufficio, ho trascritto gli appunti su *Confluence* e dal contenuto io e il tutor abbiamo definito quali fossero i casi d'uso principali.

Ho poi convertito i casi d'uso in gruppi di *ticket*, tracciati dal documento *Confluence* alla bacheca di *Jira*. Ogni *ticket* inizialmente conteneva solo le informazioni riguardo la funzionalità da implementare. In un secondo momento ho integrato alcune delle funzionalità più complesse con diagrammi *UML* (fase di progettazione). Per le funzionalità per cui occorrevano sia interventi sul *server* che sul *client* le ho descritte in un solo *ticket*, poi suddiviso in sotto-*ticket*.

Per esempio, il motore delle statistiche aveva due componenti d'interfaccia associati: la pagina di dettaglio, corredata di grafici e il riassunto in un foglio Excel. Per questo motivo ho descritto la funzionalità con un *ticket*, con tre sotto-*ticket*:

* uno per il motore, in cui ho specificato quali statistiche dovesse produrre;
* uno per l'interfaccia, in cui ho specificato quali statistiche dovesse presentare e con quale tipo di grafico;
* uno per l'Excel, che avrebbe dovuto contenere l'elenco dei dati.

3.2.6 Progettazione

Durante l'attività di progettazione ho seguito precisamente le direttive aziendali sui *pattern* da usare. Espongo qui due esempi, uno per il *server* in *Java* e uno per il *client* con *Angular*.

Il componente del motore di calcolo doveva poter effettuare, partendo da una lista di punti di test, statistiche quali media, deviazione standard, *trend* (attraverso una regressione lineare) e frequenza dei valori. Per fare questo, mi sono avvalso di una classe chiamata *BigDecimalSummaryStatistics*, che fa parte del *package* `org.eclipse.collections.impl.collector`. In questa occasione, il tutor mi ha fatto notare una convenzione che non conoscevo, ossia che di solito i *package* che si chiamano `impl` sono utilizzati per dare delle implementazioni a metodi generici e quindi

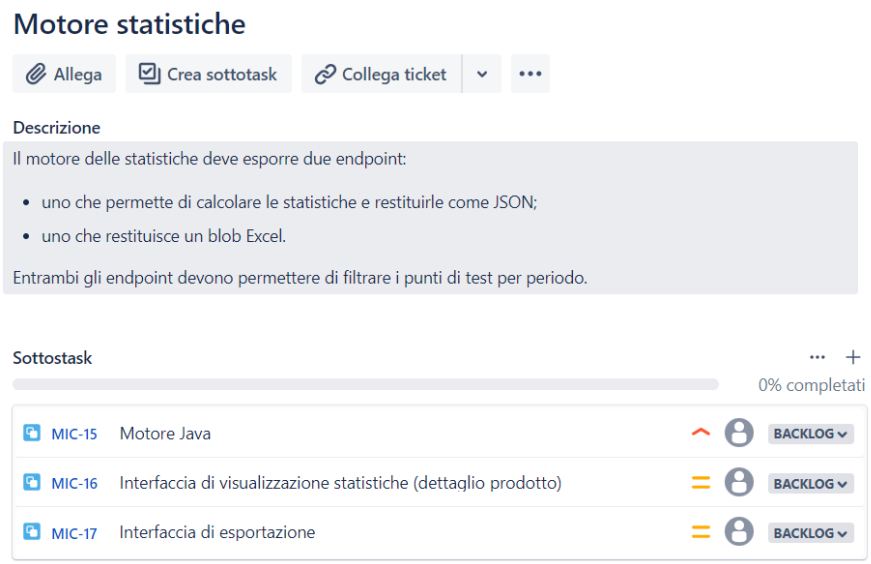


Figura 3.10: Esempio minimale di *ticket*.

questa classe poteva non essere intesa per esse utilizzata direttamente. Il *pattern* usato è un *singleton* che, nonostante non sia più universalmente riconosciuto, internamente è largamente apprezzato (figura 3.11).

Un altro *pattern* di progettazione, lato *client*, meno standard ma comunque utilizzatissimo dal gruppo, è il *pattern* detto *loader-observable*. Questo *pattern* si applica solo a chiamate che dovrebbero restituire sempre la stessa risorsa. Come illustrato in figura 3.12, consiste semplicemente in una funzione, locata in un `Service`, che effettua una chiamata asincrona al *server* che restituisce una `Promise` con il risultato, e che tale risultato venga anche emesso da un `BehaviorSubject` della libreria `rxjs`. In questa maniera, ogni *component Angular* dell'interfaccia che utilizza quella risorsa viene avvisato quando questa è cambiata anche sul *server*.

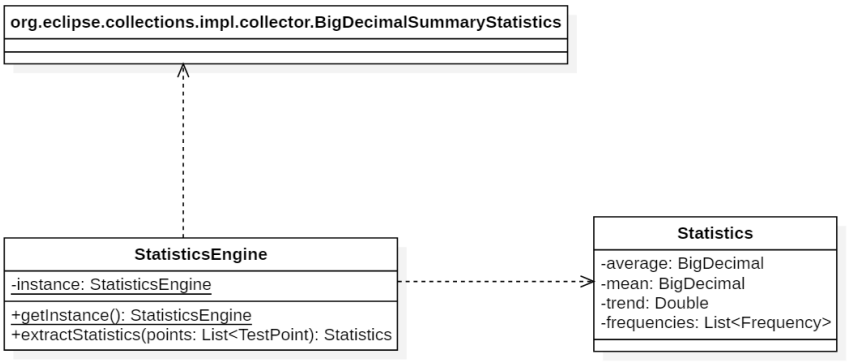


Figura 3.11: Esempio minimale di *UML* della classe *singleton* del motore.

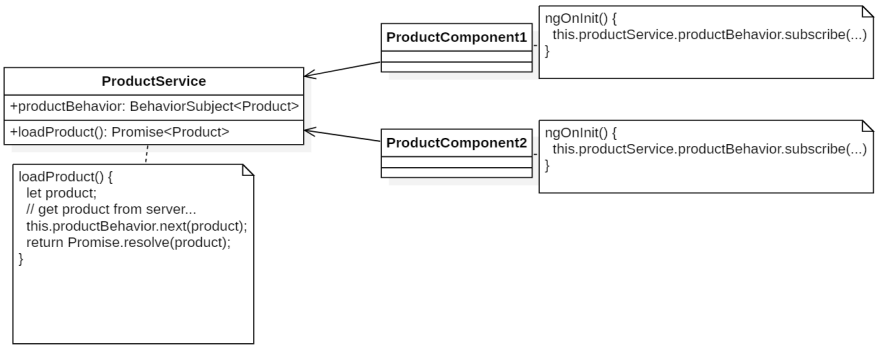


Figura 3.12: Esempio minimale di *pattern loader-observable*.

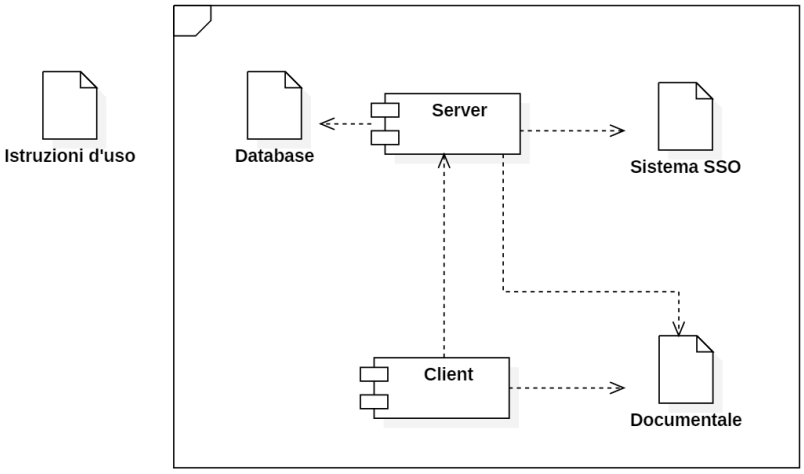


Figura 3.13: I *deliverable* consegnabili alla fine del progetto.

3.2.7 Stesura del manuale d'uso per i fornitori

La stesura del manuale all'inizio mi preoccupava, perché le mie abilità linguistiche sono abbastanza limitate e ho dovuto redigerle in inglese. Alla fine però si è trattato di un piccolo manuale, con tre procedure descritte (caricamento *file*, eliminazione *file*, sostituzione *file*). Le istruzioni hanno consistito in semplici elenchi numerati, che presentavano frasi brevissime e molto semplici. Inoltre, ho corredato tali indicazioni di diverse immagini (*screenshot* dell'applicazione), una per ogni punto. Il tutor si è detto soddisfatto del risultato, per precisione e chiarezza della presentazione. Il manuale è un documento *PDF* redatto in *Word*.

3.3 Risultati finali

Il sistema atteso era inizialmente composto dai seguenti elementi da sviluppare:

* un componente che gestisse i *job* di sincronizzazione;
* il *server* a supporto dell'interfaccia;

∗ l'interfaccia.

Durante le attività di progettazione, il componente di sincronizzazione è stato inglobato nel *server* dell'interfaccia. Durante il corso del progetto la comprensione del dominio applicativo è cambiata, e anche le priorità del Cliente sono mutate, facendoci cambiare i requisiti. La tecnica di effettuare un'analisi del dominio e utilizzare quel risultato come *baseline* per ogni requisito implementato si è rivelata molto efficace. Sebbene sia accaduto uno dei rischi più potenzialmente dannosi, l'infrastruttura del progetto ha retto.

Con lo scopo di dare un'idea sulla dimensione del progetto costruito, elenco i singoli componenti del sistema e riporto il numero di *ticket* (e quindi di requisiti) previsti e completati. Da questo elenco non sono stati considerati i *ticket bug*.

Componente	**Ticket aperti**	**Ticket chiusi**
Banca dati	4	4
Server	19	19
Mockup e poi *Client*	40	40
Integrazione dei componenti *off the shelf*	5	5
Totale	68	68

Tabella 3.1: Bilancio totale dei *ticket*.

In realtà ci sarebbero ancora otto *ticket* ancora aperti, ma questi fanno parte del requisito di analisi dei fallimenti, un requisito che, come ho spiegato in precedenza, è stato poi rimandato a un rilascio successivo. Siccome l'obiettivo di questo tirocinio era di portare un progetto dalla fase iniziale al suo primo rilascio, non li ho considerati nel conto.

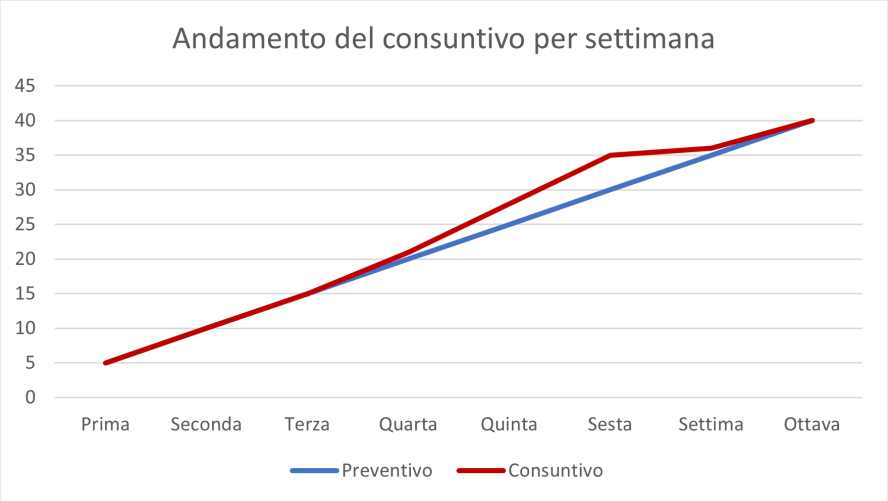


Figura 3.14: Andamento del consuntivo rispetto al preventivo nelle settimane.

Per quanto riguarda le attività di gestione di progetto, all'inizio, per le prime tre settimane, si è svolto tutto in pari con le aspettative. Successivamente abbiamo accumulato ritardi per tre settimane di fila, portando il ritardo complessivo, al suo

picco, a circa cinque giornate di lavoro, quasi una settimana lavorativa. Quando poi, insieme al Cliente, ci siamo resi conto che le priorità non erano state centrate, siamo stati in grado di recuperare grazie alla diligenza mantenuta nelle fasi iniziali del progetto. Inoltre, siamo stati anche quasi fortunati, perché le attività per completare la *dashboard* dei fallimenti erano di mole superiore a quella del requisito di validazione, che ha preso il suo posto. Questo "colpo di fortuna" ha contribuito nel tenere sotto controllo i costi dello sviluppo. In figura 3.14 un grafico che mostra l'andamento di preventivo e consuntivo durante le otto settimane.

Capitolo 4

Conclusioni

4.1 Obiettivi aziendali

Nel complesso, i risultati soddisfano le aspettative iniziali, come rappresentato nel precedente capitolo dalla metrica dei requisiti soddisfatti, quantificata in *ticket*. La documentazione del codice soddisfa gli obiettivi qualitativi, tranne in poche eccezioni.

Non essendoci una metrica precisa per valutare la qualità dei *ticket*, e quindi la qualità dell'analisi e della progettazione, è stato usato un sistema di *feedback* da parte degli interessati: gli assegnatari dei *ticket* ne riportavano poi, a voce in maniera informale, la chiarezza e il grado di dettaglio. L'obiettivo di ciò era determinare se i *ticket* fossero sufficienti per permettere agli assegnatari di lavorare in autonomia. All'inizio gli assegnatari lamentavano una generale mancanza di precisione, corretta gradualmente. Alla fine, il tutor ha considerato positivamente anche questo aspetto.

Per le attività di pianificazione e gestione del personale il tutor ha adoperato come metrica la seguente formula:

$$divergenza\,dalla\,pianificazione = \frac{G_{cs} - G_{ps}}{G_{ps}}$$

dove G_{ps} sono le giornate a preventivo e G_{cs} sono quelle a consuntivo, riferite alla settimana s. All'inizio del progetto il tutor mi aveva detto che la divergenza con la pianificazione non avrebbe mai dovuto raggiungere il 25%; questa è una metrica *standard* nei processi aziendali per valutare le prestazioni della pianificazione. Siccome non ho mai avvicinato quella soglia (figura 3.14), il tutor ha valutato le mie stime dell'impegno per le attività attendibili e, nel complesso, lo svolgimento di questa attività positivamente.

Riguardo la gestione dei rischi, il tutor ha valutato positivamente l'attività, perché rispetto ad alcuni rischi che si sono verificati ho adoperato la *wiki* con profitto, anche se nel frattempo si sono anche verificate delle situazioni di criticità facilmente evitabili.

Riassumendo, per quanto riguarda il discorso formativo per le mansioni di Responsabile e Analista, il tutor si è detto più che soddisfatto. Per quanto riguarda il prodotto, sia il tutor che il Cliente si sono detti soddisfatti dei risultati.

4.2 Obiettivi personali

Ritengo che l'esperienza abbia soddisfatto, nel complesso, gli obiettivi di formazione personale, con l'unica eccezione della progettazione dell'esperienza utente, in cui faccio

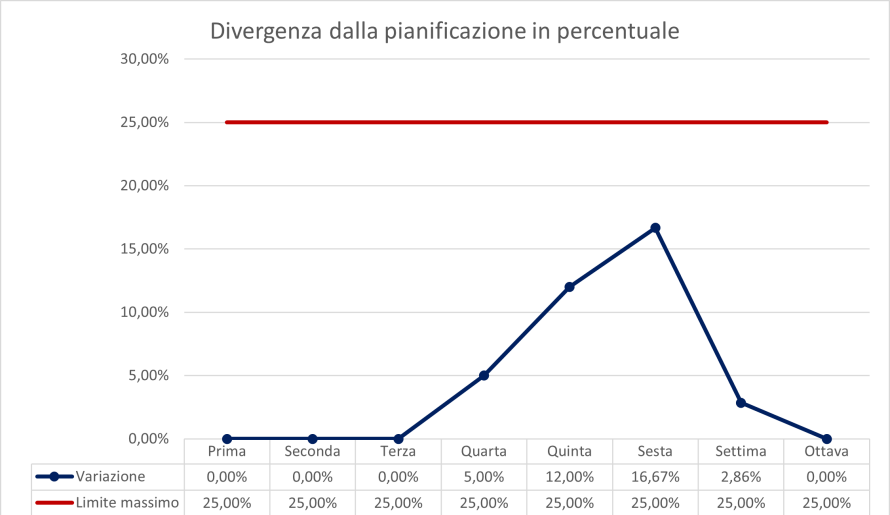


Figura 4.1: Andamento della divergenza dalla pianificazione.

ancora molta fatica. In principio, il mio obiettivo era di iniziare la mia formazione come ingegnere del *software*, così da acquisire le competenze complementari a quelle che già possiedo. Le componenti che mi interessavano maggiormente erano:

* **pianificazione**, con la capacità, come dicevo all'inizio dell'elaborato, di finalizzare un progetto;
* **gestione del personale**, cioè imparare a coordinare diversi membri di un *team* di sviluppo e stimarne correttamente l'impegno;
* **analisi** e le altre competenze necessarie per definire gli obiettivi tangibili di un progetto;
* **progettazione**, per imparare nuove soluzioni tecniche e a comporle per risolvere problemi.

L'analisi dei rischi è stata una proposta del tutor. Inizialmente temevo fosse troppo fuori dalle mie corde, ma alla fine sono soddisfatto di aver fatto anche questa esperienza.

Prima di accettare questo incarico, avevo delle idee molto precise di come effettuare analisi e quali *pattern* usare in fase di progettazione, che derivavano in parte da osservazioni e formazione personali, in parte dall'esperienza universitaria. Un mio obiettivo, al di là dell'apprendimento puro e semplice, era di poter sperimentare e, se avessero avuto successo, inserire nelle prassi aziendali alcune delle mie idee. Questo desiderio nasce dal fatto che alcuni tra le tecniche di analisi e i *pattern* usati sono molto desueti. Sapevo che se mi fossi dimostrato autonomo il tutor mi avrebbe lasciato maggiore autonomia e speravo di avere così un'occasione. Tuttavia, il tutor mi ha caldamente raccomandato di seguire e imparare nel dettaglio le prassi aziendali, perché per introdurre qualcosa di nuovo con profitto bisogna conoscere a fondo ciò che funziona e che quindi è già ben assestato. Razionalmente, mi trovo d'accordo con questo ragionamento, anche se persiste un sentimento di delusione. Ad ogni modo, il tirocinio mi ha fatto guadagnare un nuovo insieme di competenze e abilità e trovo che, per dimensione del progetto, sia stato adeguato e molto significativo come punto di partenza per intraprendere il percorso che ho scelto.

4.3 Difficoltà incontrate

La difficoltà principale che ho incontrato è stato lo svolgimento della progettazione dell'esperienza utente (*UX design*). Da parte della persona incaricata ho trovato ottimo e costante supporto, tuttavia ho sofferto la mancanza di una metodologia ben definita. In questo, penso che delle conoscenze accademiche specifiche possano essere d'aiuto. Degli elementi, per questo scopo, mi sono stati forniti nel corso di Tecnologie Web, che è l'unico corso in cui vedrei ben posizionato questo approfondimento.

Scendendo invece nel dettaglio, ho sentito una mancanza tecnica molto specifica, riguardo il funzionamento dei motori *JavaScript*. Questo discorso non è stato affrontato né in ufficio né in università e molto spesso mi è parso di scrivere codice senza sapere bene cosa implicasse, a livello di esecuzione. Questa sensazione si è tradotta in codice più faticoso da scrivere, perché per la natura del modello del linguaggio le componenti asincrone mi sono sembrate molto difficili da prevedere. Per superare questo problema ho adoperato estensivamente la libreria `rxjs`. È mia opinione che gli studenti beneficerebbero molto nell'approfondire questo linguaggio e su come funzionino i suoi ambienti perché, nonostante le pecche, è innegabile che ormai sia diventato uno strumento quasi necessario da padroneggiare, se si vuole intraprendere la carriera da programmatore. Nello specifico, mi riferisco a come funzioni il sistema a eventi di V8, l'interprete *JavaScript* di *Chrome* e *NodeJS* (figura 4.2) e alle alternative per raggiungere i risultati della programmazione concorrente e distribuita, come i *Web Workers* (figura 4.3). Questo approfondimento potrebbe essere portato nel corso di Tecnologie *Web* che già tratta del linguaggio. Tuttavia ne raccomanda l'uso con parsimonia per buoni motivi, inoltre mi rendo conto che il corso abbia ben altri obiettivi. In alternativa, potrebbe trovare posto nel corso di Altri Paradigmi di Programmazione, dove si studiano modelli di programmazione concorrente e distribuita, tra cui le *Reactive Extension*, base della già citata libreria `rxjs`.

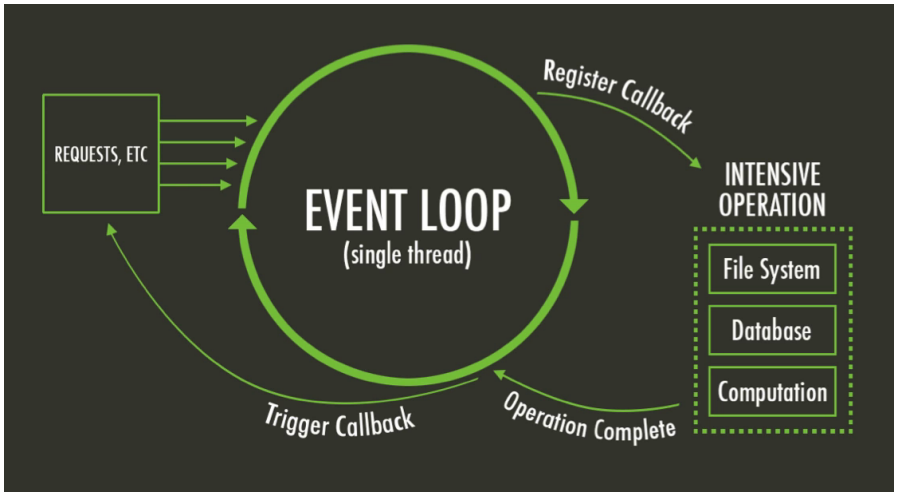


Figura 4.2: Schema del funzionamento del motore *NodeJS*. Il motore, di per se, non permette l'esecuzione di due blocchi di codice in parallelo.

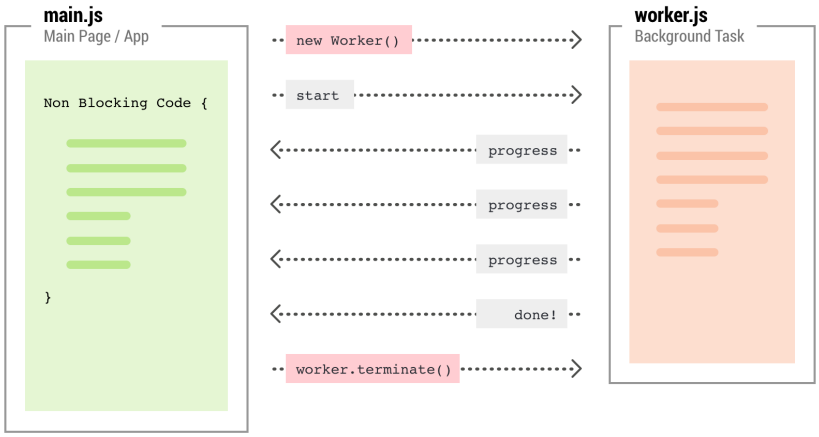


Figura 4.3: Schema del funzionamento di un *Web Worker*. In contrapposizione con il motore V8 (figura 4.2) questa tecnologia permette l'esecuzione parallela di due blocchi di codice.

Nel corso della mia esperienza di studente lavoratore ho trovato diverse discrepanze tra il mondo universitario e quello lavorativo. Esse riguardano principalmente gli obiettivi e il contesto in cui vengono svolte le attività. Una cosa che non viene mai sottolineata abbastanza, sia in ambiente professionale che in ambiente accademico, è che ogni *best practice* non è la migliore in senso assoluto, ma lo è in base alla situazione. O quantomeno, ho notato che dietro ogni scelta, anche quella di non seguirle, c'è una motivazione, che può essere valida nel contesto giusto. Ogni tanto, ho notato la mancanza di attenzione per il retroscena. Un esempio pratico, tratta della normalizzazione delle banche dati. Quando progettavo la banca dati per il progetto del tirocinio ho chiesto un consulto a un mio collega che funge da referente per l'argomento e mi ha spiegato che, in certi casi, è opportuno denormalizzare certe tabelle, per esempio per ragioni di prestazioni. Quando seguii il corso di Basi di Dati, si trattò solo la normalizzazione. Pertanto, mi sono trovato completamente impreparato, non sulla denormalizzazione che in sé è semplice, ma sulla gestione di ciò che questa tecnica comporta. È mia ferma opinione che gli studenti beneficerebbero molto da una visione a tutto tondo, discutendo di problemi del mondo reale, già dai primi corsi, dopo aver posto delle buone basi teoriche.

All'inizio accusavo delle carenze per quanto riguarda le tecnologie, perché in ufficio utilizzavamo tecnologie più complesse e in maniera decisamente più professionale. Tuttavia completando il corso di studi mi sono ricreduto, perché mi sono reso conto che, una volta completati tutti i corsi, avrei avuto[1] le basi per prendere parte con profitto alle attività professionali, anche se con bisogno di qualche ora di formazione sulla specifica tecnologia. Dal punto di vista della gestione di progetto, dell'analisi dei requisiti e della progettazione non mi è mancato niente, sebbene le prassi aziendali fossero abbastanza diverse da come presentate durante il percorso di studi. Le competenze acquisite nel corso di laurea mi hanno aiutato ad adattarmi alle richieste del tirocinio senza troppe difficoltà.

[1] Uso il condizionale perché si tratta di una mia supposizione. Avevo già una certa dimestichezza con gli strumenti utilizzati, dato la mia precedente collaborazione con questo gruppo.