Progetto:

**iSpesa**

Titolo del documento:

**Report Finale**

Informazioni Documento

|  |  |  |  |
| --- | --- | --- | --- |
| Nome Documento | *iSpesa\_Report\_Finale\_D5* | Numero  Documento | D5 |
| Descrizione | Report Finale del progetto: organizzazione del lavoro, ruoli, tempo complessivo e di ciascun membro dedicato al progetto, criticità, autovalutazione | | |

Indice

[Scopo del documento 3](#_Toc1)

[1. Approcci all’Ingegneria del Software 4](#_Toc2)

[2. Organizzazione del Lavoro 5](#_Toc3)

[3. Ruoli e Attività 6](#_Toc4)

[4. Carico e distribuzione del lavoro 6](#_Toc5)

[5. Criticità 7](#_Toc6)

[6. Autovalutazione 7](#_Toc7)

# Scopo del documento

Il seguente documento è un report finale che testimonia come il nostro team ha lavorato allo sviluppo del progetto.

Dopo aver descritto gli approcci all’Ingegneria del Software, verrà descritta l’organizzazione e la suddivisione del lavoro tra i vari membri del team, dopodiché mostreremo le ore di lavoro complessive ed individuali per i vari deliverables.

Successivamente elencheremo le eventuali criticità e problematiche riscontrate durante il corso del progetto per finire poi con una breve autovalutazione del nostro lavoro.

# 1. Approcci all’Ingegneria del Software

## 1.1 BlueTensor

Durante il seminario BlueTensor ci è stato mostrato un approccio allo sviluppo di programmi IA composto da un misto di sistema a cascata e agile. In questo approccio il progetto viene diviso in cinque parti: studio di fattibilità preliminare, una parte di progettazione software e hardware con identificazione dei requisiti funzionali e non, una parte di sviluppo e test, il rilascio finale. Le prima tre parti, per via della loro natura intrinseca, prevedono un approccio a cascata che ha come risultato la realizzazione di un progetto per l’architettura del programma ed anche due set di dati completi, uno di test ed uno di addestramento. Successivamente, i processi di sviluppo, addestramento e test seguono un modello agile in cui ogni parte del programma viene divisa in moduli che implementano funzionalità base. Progressivamente ogni modulo viene completato, integrato con gli altri e rilasciato per mostrare l’andamento dello sviluppo al cliente ed interfacciarsi con lo stesso. Dopo questo sviluppo continuo, infine, si ha il rilascio vero e proprio del software che non sempre rappresenta la fine dello sviluppo, poiché possono essere necessari aggiornamenti o integrazioni in futuro. A mio avviso questo approccio è molto pragmatico e permette di sfruttare le migliori qualità sia del metodo agile sia del metodo a cascata. Non noto criticità poiché reputo che la parte di sviluppo a cascata sia necessaria, siccome si tratta di processi che sono sequenziali uno all’altro, mentre la parte di sviluppo agile è il modo migliore per sviluppare il software interfacciandosi allo stesso tempo col cliente.

## 1.2 Metodo Kanban

La simulazione del metodo Kanban è stata altamente interessante e formativa poiché è stata capace di un metodo molto pratico per visualizzare l’andamento del lavoro, evitando di “sovrapprodurre”. Il seminario si è svolto mediante una simulazione (avvenuta a seguito di una breve spiegazione sull’origine del metodo) in cui abbiamo simulato di “lavorare” usando il metodo Kanban. Durante la simulazione abbiamo simulato eventi reali con delle astrazioni. Suddetto metodo si dimostra non solo comprovato, ma anche utile poiché, oltre a visualizzare mediante i bigliettini il work-flow, permette di tener conto dei tempi di sviluppo di ogni singolo “bigliettino”, ovvero quanto tempo ci vuole a realizzare ogni singola idea “messa in ballo”. Tuttavia potrebbero esserci dei problemi nel caso in cui la nostra “lavagna” non sia ben aggiornata o se essa diventa troppo complicata. In aggiunta le varie fasi non sono basate sul tempo, quindi potrebbero esserci dei problemi con lo sviluppo. Personalmente abbiamo applicato il concetto del metodo Kanban allo sviluppo del nostro progetto, ottenendo un risultato abbastanza soddisfacente poiché ci ha fatto comprendere come stesse realmente andando lo sviluppo dell’applicazione.

## 1.3 IBM

Il seminario IBM si incentra sul Cloud IBM che è una piattaforma distribuita che offre potenza computazionale e tecnologia software sotto forma di servizi pagati a consumo. Mediante la piattaforma Cloud IBM è possibile creare un server, una macchina virtuale, database e molto altro in poco tempo e con costi accessibili, senza doversi preoccupare di alcun dettaglio tecnico. Un’altra funzione, a parer mio utile, è la possibilità di rendere un applicativo disponibile su internet a tutti in maniera semplice, veloce e sicura. Un altro servizio importante che ci è stato illustro è IBM Cloud Satellite. Esso permette di gestire automaticamente tutti i servizi IBM all'interno di macchine locali definite satellite e collegate al CloudIBM mediante un satellite link. Questo seminario ha evidenziato bene l'importanza di una buona architettura in confronto ad una poco pensata, andando a dare particolare importanza alla sicurezza della stessa mostrando degli esempi pratici. In aggiunta ha mostrato l'importanza dell'adattabilità dei servizi che si offrono poiché che devono sempre andare di pari passo con le nuove tecnologie.

## 1.4 Meta

Il seminario di META ha lo scopo di mostrare gli strumenti e le figure necessari alla realizzazione di un progetto software. Questa esperienza ha evidenziato come la corretta comunicazione tra team diversi e individui all’interno dello stesso gruppo di lavoro permettano di giungere a grandi risultati in breve tempo. Per mantenere la produttività elevata all’interno dei vari gruppi di lavoro, le persone sono libere di esprimersi e cambiare team qualora ne avessero la necessità. META non usa metodologie di ingegneria del software particolari, i team leader di ogni gruppo seguono principalmente una procedura di tipo agile. L’obiettivo finale viene quindi suddiviso in sotto processi che vengono assegnati alle persone a seconda delle loro capacità. I progressi vengono monitorati attraverso vari strumenti, il più importante dei quali all’interno di META è Asana. Sono presenti, anche, strategie di tipo “Spread Sheets”, dove i vari compiti vengono divisi in formato tabellare all’interno di fogli (“Sheets”) che sono un importante strumento di monitoraggio. Questo seminario ci ha fatto riflettere su come la semplicità nella comunicazione tra colleghi e la capacità di adattamento dei singoli siano più importanti di conoscere numeri linguaggi di programmazione o avere un qualche background particolare.

## 1.5 U-Hopper

Il seminario di U-Hopper ci ha fatto notare quanto sia importante riuscire a gestire in maniera soddisfacente i big data. Uno degli ultimi stadi dei grandi servizi cloud è l'ingestion, che con un sistema a code e, successivamente, l’elaborazione mediante batch processing o stream processing elabora i dati in batch o in tempo reale. I dati vengono memorizzati in database nella memoria centrale per garantire rapidi tempi di accesso. Gli strumenti utilizzati devono effettuare il logging per valutare le prestazioni e facilitare il debugging.

Nel processo di sviluppo, si adotta la metodologia agile con sprint di circa 2 settimane. Le milestone sono suddivise in issue, e per ogni issue si crea un branch su Git per lo sviluppo, una code review avviata al momento del merge. Il codice scritto viene testato con una copertura del 70%. Questo seminario ha fatto capire come per la gestione dei big data sia necessaria una architettura robusta e con bassi livelli di accoppiamento per rendere il tutto estremamente scalabile e inoltre come l’utilizzo di git sia necessario per garantire uno sviluppo isolato e parallelo garantendo al tempo stesso elevati livelli di qualità.

## 1.6 RedHat

Il Seminario RedHat è stato molto interessante per via anche se non si è parlato propriamente di metodi di ingegneria del software e di gestione (come nel caso Kanban per esempio), ma ci si è concentrati su come il codice viene distribuito ed in particolare sull’Open Source. Dopo aver parlato un po’ di sé, il relatore (Mario Fusco) ha spiegato i lati positivi dell’Open Source, ovvero: la condivisione di informazioni (scrivere e vedere codice sono i metodi migliori per apprendere), la supervisione da parte di altri sviluppatori (poiché il codice è visibile da tutti e tutti possono contribuire e mostrare criticità), la meritocrazia e la visibilità (le buone idee vanno avanti e loro, insieme al codice, parlano per il creatore), il senso di comunità che si crea. Sono state poi spiegate le principali categoria di licenze Open Source e come si partecipa ad un progetto Open Source. Nel far ciò sono state anche evidenziate alcune azioni da fare per mantenere un progetto Open Source “sano”. Il seminario ha quindi esposto il funzionamento dell’Open Source ed il perché partecipare/iniziare un progetto simile. Personalmente, è stato un seminario interessante che abbiamo sentito più vicino a noi rispetto agli altri per via del fatto che per realizzare il progetto ci siamo serviti di esempi di progetti degli anni passati che sono stati per noi come una bussola. Abbiamo sperimentato sulla nostra pelle parte dei benefici dell’Open Source.

1.7 Microsoft

## 1.8 Molinari

Il seminario di Molinari si incentra sui sistemy legacy, sistemi informativi obsoleti in tutte le loro componenti o nelle singole parti. I sistemi legacy sono principalmente caratterizzati da mainframe e hardware datati, personale non aggiornato, grandi computer con grandi consumi energetici, scarsa UI che continuano a sopravvivere principalmente a causa di problemi di investimento, robustezza dei sistemi legacy e paura delle tecnologie moderne.

Un problema noto nel mondo del software è il "tech debt", un problema che viene a galla quando un software datato viene aggiornano al posto di venir sostituito. I problemi principali derivati dal tech debt è l'inibizione verso nuovi sviluppi, l'insoddisfazione del cliente, l'aumento dei costi e rischi per la sicurezza. I sistemi legacy si possono portare nel mondo moderno attraverso quattro metodi.

* Dismissione: Il sistema viene smantellato, le procedure sostituite da nuovo software e i dati migrati.
* Migrazione: I dati e il software vengono modernizzati e il sistema legacy viene dismesso in futuro
* Interazione: Il sistema legacy continua ad esistere interagendo con il nuovo sistema.
* Inclusione: Il sistema legacy diventa parte attiva del nuovo sistema mediante emulazione.

Gestire un progetto legacy è complicato poichè possono essere progetti di grandi dimensioni dove i ruoli assegnati ad ogni personale sono difficili da capire. Per questo motivo un approccio di tipo Agile diventa meno efficiente e vengono usate metodologie di gestione più predittive. Queste metodologie sono meno concentrate con l'approccio al cliente finale più concentrate sui processi, software funzionante, contratti e seguire un piano preciso

## 1.9 Marsiglia

1.10 APSS

# 2. Organizzazione del Lavoro

Il lavoro è stato organizzato mediante una suddivisione per ruoli, assegnati in base alle competenze individuali. L’idea di partenza è stata quella di ripartire equamente in base alle competenze di ciascun individuo il carico di lavoro senza sovraccaricare alcun membro. Sin dall’inizio abbiamo diviso il lavoro in parti a cui ognuno di noi avrebbe singolarmente lavorato per poi unirle. Nel far ciò abbiamo stabilito dei ruoli inizialmente informali che poi si sono solidificati all’interno dell’organizzazione.

Nel far ciò abbiamo sia lavorato in gruppo sia singolarmente, il tutto in base alle necessità del Deliverables da elaborale. E’ essenziale il fatto che ogni qual volta che si dovesse completare un Deliverable, unendo i vari pezzi ci siamo incontrati per discutere del lavoro fatto e fare una revisione generale del lavoro del gruppo. Ciò ha permesso di trovare numero problemi e di migliorare così il tutto.

Gli incontri che abbiamo fatto sono sempre stati in presenza e molto frequenti. Abbiamo sfruttato il fatto di vivere nel medesimo appartamente per incontrarci ogni due giorni circa per parlare o in ogni caso in qualunque momento fosse necessario. Ciò ha permesso di avere aggiornamente costanti e frequenti sullo stato del lavoro di ciascuno di noi,

Per la strumentazione ci siamo affidati a Microsoft Word per la stesura dei Deliverables, VisualStudio Code per lo sviluppo del codice, PhPMyAdmin e MySQLWorkbench per il lavoro sul database del sito e GitHub per salvare i vari file e mantenere uno storico. Per ogni i vari diagrammi abbiamo utilizzato Draw.io che abbiamo preferito a LucidChart poiché sapevamo già usarlo, mentre per i mockup del Deliverable 1 abbiamo deciso di realizzarli direttamente in HTML, accellerando così lo sviluppo del front-end del sito web. Per segnare le ore abbiamo utilizzato dei file salvati sul git in cui semplicemente scrivevamo la data come nome del file e le ore di lavoro e l’attività svolta all’interno degli stessi.

# 3. Ruoli e Attività

Abbiamo riassunto nella seguente tabella la suddivisione dei ruoli all’interno del team.

|  |  |  |
| --- | --- | --- |
| Componente del team | Ruolo | Principali attività |
| Davide Colosimo | Capo progetto, analista requisiti funzionali e non con relative specifiche, architetto di sistema. | Il ruolo principale è stato la gestione del progetto e la coordinazione delle attività. Si è occupato dell’impaginazione dei documenti e della loro rifinitura. Ha contirbuito attivamente a tutti i deliverables e in maniare principale al D2 e al D3. |
| Gabriele Lanaro | Analista requisiti funzionali e non, progettista mockup, full stack developer. | Ha attivamente contribuito a tutti i deliverables, specialmente nella stesura dei mockup durante il D1 e nello sviluppo del front-end e del back-end del sito web durante il D4. |
| Ismael Checkri Belghiti | Analista requisiti funzionali e non, sviluppatore del database, architetto di sistema. | Ha contribuito attivamente a ciascun deliverables, con particolare attenziale ai D2 e D3, grazie ai diagrammi realizzati. Ha contribuito attivamente anche al D4 mediante la reazione dei database necessari per il sistema |

# 4. Carico e distribuzione del lavoro

La seguente tabella riporta la distribuzione delle ore di lavoro.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nome** | **D1** | **D2** | **D3** | **D4** | **D5** | **Totale** |
| Davide Colosimo | 16.15 | 24 | 23.30 | 2 | 2.30 | 68.15 |
| Gabriele Lanaro | 3 | 7.30 | 3 | 13 | 1 | 27.30 |
| Ismael Checkri Belghiti | 8 | 15.30 | 10 | 6.30 | 1 | 41 |
| **Totale** | 27.15 | 47 | 36.30 | 21.30 | 4.30 | **137.15** |

# 5. Criticità

Durante il corso di tutto il progetto abbiamo incontrato un unico grande problema, gli esempi dei vari deliverables. Essendo incompleti e poco chiari, più che indicazione ci hanno sviato dal percorso corretto a nostro avviso portandoci a rifare parte del lavoro in un tempo successivo. Tranne questo increscioso problema (che ha portanto indietro il progresso del progetto di molti giorni poichè abbiamo rifatto completamente D2 e D3), non reputiamo ci siano stati problemi durante il corso del progetto. Ad esclusione di ciò che è stato riferito prima, potremmo annoverare tra i problemi alcune incomprensioni sullo sviluppo del back-end del progetto, ma non riteniamo esse problemi per via della loro superfluità.

# 6. Autovalutazione

Nel complesso riteniamo di aver lavorato tutti molte ore al progetto, dimostrando sia quantità che qualità nel nostro lavoro.

La nostra autovalutazione è la seguente:

|  |  |
| --- | --- |
|  | **Voto** |
| Davide Colosimo | 30 |
| Gabriele Lanaro | 30 |
| Ismael Checkri Belghiti | cotone |