Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

U-Sushi

Titolo del documento:

Report Finale

INDICE

Approcci all'Ingegneria del Software 2
- **BlueTensor** 2
- **Metodo Kanban** 2
- **IBM** 2
- **Meta** 3
- **U-Hopper** 3
- **RedHat** 3
- **Microsoft** 3
- **Molinari** 4
- **Marsiglia** 4
- **APSS & Trentino.ai** 4

Organizzazione del lavoro 5
Ruoli e attività 6
Carico e distribuzione del lavoro 6
Criticità 7
Autovalutazione 7

Approcci all'Ingegneria del Software

BlueTensor

L'azienda BlueTensor, che lavora in ambito AI e soluzioni personalizzate per le aziende, adotta metodi dell'ingegneria del software partendo da un'analisi di fattibilità, passando per un setup e design del prodotto, arrivando poi ad un Proof of Concept e ad un Minimum Viable Product, presentando dei prototipi utilizzabili ma ancora rudimentali. Viene poi applicato un miglioramento continuo al prodotto finale.
Questo è un processo a cascata (waterfall) e permette una comprensione chiara del progetto e della timeline fin dall'inizio ma allo stesso tempo rende più difficile tornare indietro e applicare modifiche in caso di ripensamenti. Questa è una problematica che abbiamo riscontrato anche nello sviluppo di U-Sushi, in quanto un paio di volte ci siamo ritrovati ad avere alcuni ripensamenti su alcune scelte fatte e ciò ci ha portato a dover investire del tempo nella revisione e modifica dei documenti precedentemente redatti.
La parte di sviluppo invece, viene portata avanti dall'azienda in modalità agile con CI/CD testando e revisionando in continuazione.
Altro punto in comune tra il nostro progetto e quelli dell'azienda sono i diagrammi UML.

Metodo Kanban

L'esperienza di simulazione Kanban ha evidenziato come il limitare il carico di lavoro in corso (ovvero avere un numero limitato di task attive all'interno di uno step del progetto) possa migliorare di molto il tempo di "chiusura" di una task, dal momento che obbliga i membri del team a collaborare sulle task che bloccano il flow del lavoro. Il metodo kanban in questo modo propone un approccio di pull delle task (prendere in carico una task solo quando è possibile) e non di push (riempirsi di task e non portarne a termine nemmeno una). Nel progetto U-Sushi, ci sono stati dei momenti in cui abbiamo gestito bene il carico di lavoro aiutandoci a vicenda, e altri in cui questo metodo non è stato applicato per mancanza di tempo o di competenze sufficienti per aiutare gli altri.

IBM

Il seminario di IBM non ha trattato metodi di ingegneria del software. È stato spiegato però cosa c'è dietro ad un provider di servizi cloud, come funziona il cloud e come si può mettere sul cloud un'applicazione o un qualsiasi servizio come databases o macchine virtuali. Da questa esperienza abbiamo potuto capire quanto sia difficile la progettazione di un servizio cloud anche senza vedere i metodi con cui viene progettato e sviluppato. Niente deve essere lasciato al caso, per evitare falle nella sicurezza, intesa sia come protezione da minacce che come ridondanza di informazioni e servizi, in caso di guasti.
È stato anche interessante vedere la semplicità con cui si può creare e fare il deploy di un'applicazione in cloud IBM.

Meta

In Meta non c'è un vero e proprio metodo di sviluppo come quelli trattati durante il corso. Meta non impone nessun tipo di metodologia di progettazione e realizzazione agli ingegneri del software dell'azienda e non vengono insegnate, tanto che l'oratore non le saprebbe spiegare. In generale però, anche non sapendo cosa sia o tutte le sue piccolezze, il metodo più usato è l'agile o meglio, molte delle azioni/decisioni che l'oratore fa/prende durante lo sviluppo sono tipiche dell'agile. Noi come gruppo concordiamo: non è obbligatorio usare una sola forma di sviluppo ma concentrandosi su segmenti più ristretti del processo oppure su macro-segmenti la visione cambia e cambia anche l'approccio al problema.

U-Hopper

U-Hopper si occupa di big data analytics e di soluzioni AI. Il processo di sviluppo dei progetti dell'azienda si identifica come agile, alternando ciclicamente fasi di coding, passando al testing e poi alla review, ricominciando poi dalla fase di coding. Un ciclo analogo è nella fase di deploying: deploy in staging → testing → release → deploy in production e da capo. Ciò permette di focalizzarsi su una piccola parte del progetto alla volta, massimizzando la qualità del prodotto/servizio, cosa che in gran parte abbiamo fatto anche noi, focalizzandoci su aspetti diversi solo alla risoluzione di quelli precedenti. Altro punto in comune tra U-Sushi e i progetti di U-Hopper è il version control e l'uso di git: prima di mandarle in produzione, le modifiche sono state fatte su un branch secondario ("dev"). È un metodo meno complesso di quello che usa l'azienda, ma ha aiutato ad avere una visione migliore dell'implementazione.

RedHat

Durante il seminario di RedHat non sono stati trattati metodi di ingegneria del software. L'incontro è stato incentrato sul mondo dell'Open source e sui vantaggi e opportunità che esso offre. Scrivere codice Open source può essere utile sia in ambito di carriera personale, in quanto aiuta a farsi conoscere, sia a scopo lucrativo, ad esempio fornendo supporto tecnico o consulenze a pagamento a chi lo utilizza. Esistono licenze copyleft, le più restrittive, e non-copyleft, per tutelare il nostro codice. Per le aziende, l'Open source può diventare anche uno strumento per guadagnare la fiducia dei clienti e/o smentire possibili diffamazioni riguardo il funzionamento del proprio codice (si veda Huawei per esempio). Come gruppo concordiamo che è stato molto interessante esplorare un mondo come questo, che a prima vista non offre alcun vantaggio, ma in realtà ne nasconde moltissimi.

Microsoft

Con l'esperto di Microsoft si è discusso sull'importanza di fare testing e su come farlo nel modo corretto, al fine di ridurre al minimo i problemi che possono verificarsi una volta terminato lo sviluppo del codice. Esistono varie tipologie di test e i medesimi test devono essere svolti in diversi ambienti. Esistono vari tools per svolgere testing tra cui molto

importanti i logs. Altri framework sono Wallaby e NCrunch, con cui è stato svolto un laboratorio. Noi come gruppo abbiamo fatto solo test di copertura, per verificare le porzioni di codice effettivamente utilizzate, individuare quelle inutilizzate o irraggiungibili e poter ragionare sul motivo di ciò (se perchè effettivamente inutili in seguito ad aggiornamenti o per un errore).

Molinari

Durante il seminario di Molinari non sono stati trattati direttamente metodi di ingegneria del software. Si è discusso dei sistemi Legacy, i quali garantiscono un'enorme potenza di calcolo e di come continuino ad essere usati per la loro robustezza, affidabilità, transazionalità, potenza di calcolo e business continuity. Per tali motivi questi sistemi continuano ad essere utilizzati anche se “obsoleti” e abbiamo visto i vari metodi nati per “ammodernarli”. Infine ci è stato spiegato come gestire un progetto legacy e il suo sviluppo e l’importanza di una corretta ed accurata documentazione tramite diagrammi UML, diagrammi di distribuzione, ecc… Durante il progetto U-sushi abbiamo sperimentato, anche se in piccolo, cosa significa gestire lo sviluppo di un progetto e l’importanza di creare un’adeguata documentazione.

Marsiglia

L’incontro era incentrato sulla gestione del ciclo di vita delle applicazioni. Per sviluppare un progetto è necessario: strutturare il progetto e stabilire degli standard architetturali; scegliere i linguaggi di programmazione e individuare le varie figure professionali necessarie per lo sviluppo, talvolta aventi una distribuzione geografica molto elevata; gestire la comunicazione e collaborazione tra i vari gruppi, ecc.... Durante lo sviluppo di U-Sushi ne abbiamo capito l’importanza, dato che una strutturazione iniziale frettolosa e imprecisa e alcuni errori di comunicazione e coordinamento tra le varie parti hanno portato a dei leggeri rallentamenti nello sviluppo dell’applicazione.
Vista la necessità di un livello di servizio tendente al 100%, si sono anche evoluti anche i processi di sviluppo del software: attualmente la tecnica più utilizzata è la dev/ops, che oltre a prevedere una fase di continuous integration (come nell’agile), prevede successivamente che la release venga messa in una repo (spesso diversa da quella del software), dove viene sottoposta ad una fase di continuous delivery. Inoltre, per quanto riguarda il cloud ha acquisito un ruolo fondamentale nello sviluppo di applicazioni, fornendo servizi infrastrutturali e di piattaforma, oltre talvolta a sistemi in grado di rilevare un eventuale carico eccessivo dell’applicazione e di scalarla in automatico.

APSS & Trentino.ai

APSS si occupa di gestire il bilancio e la distribuzione delle risorse di tutta la sanità in Trentino, al fine di garantire la continuità e l’umanizzazione dei percorsi di cura, equità nell’offerta dei servizi e il potenziamento e innovazione di questi, massimizzandone i

benefici. Per fare ciò è stato necessario progettare una struttura (un sistema “legacy”) in grado di raccogliere e gestire i dati di qualsiasi operazione e spesa fatta legata alla sanità, in modo da poterli analizzare al fine di ridistribuire meglio le risorse, il tutto tenendo sempre conto di rispettare le leggi sulla privacy dei dati.. Per calcolare il personale necessario vengono usati gli FTE (corrispondente a circa 1800h lavorative), al fine di avere una misurazione oggettiva delle risorse umane necessarie, indipendentemente dal tipo di contratto. Per esempio il progetto U-Sushi ha richiesto 0.2 FTE per lo sviluppo di una bozza e abbiamo stimato che sarebbero necessari altri 0.15-0.20 FTE per lo sviluppo completo.
Per la gestione dei dati si è scelto un processo di sviluppo software di tipo waterfall anziché agile vista la richiesta di un MVP elevatissimo. In tutto questo gioca un ruolo fondamentale l’AI, che permette di migliorare la qualità delle cure, l’efficienza operativa, la gestione dei pazienti, la precisione delle diagnosi, ecc…, oltre a potenziare la ricerca e sviluppo di nuovi farmaci.
Non abbiamo trovato molti punti in comune con il nostro progetto, ma è stato interessante capire tutte le considerazioni e le analisi necessarie per gestire le risorse e il bilancio di una realtà molto grande come quella della APSS.

Organizzazione del lavoro

Abbiamo cercato di spartire il lavoro in modo equo e in base alle competenze di ognuno di noi. All’inizio, nel primo deliverable abbiamo cercato di fare tutti tutto, ma si è rivelato un metodo lento ed inefficace. Quindi abbiamo deciso che ognuno si sarebbe occupato di una sottoparte di ogni deliverable.

Abbiamo preferito lavorare da soli sulle nostre parti e successivamente farle revisionare dagli altri due membri, confrontandoci di tanto in tanto sugli aspetti più implementativi (aggiungere o meno una funzionalità, nomenclature delle classi/tabelle/variabili, ecc.).
I meeting sono stati per la maggior parte in presenza durante le lezioni del corso; un paio di volte abbiamo dovuto fare una chiamata per fare il punto della situazione, mentre una volta sola ci siamo presi uno slot di un paio d’ore libere per decidere alcune delle cose più importanti.

Per la gestione dei documenti ci siamo affidati a Google Docs dal momento che permette il lavoro in contemporanea sullo stesso documento. La stessa cosa vale per i diagrammi UML, fatti con Lucidchart, che permette una condivisione e collaborazione semplici ed efficaci.
Per i mockup invece, ci siamo affidati al sito Moqups, che permette di creare mockup più minimali e meno dettagliati; abbiamo preferito usare quello per non perdere tempo sui dettagli, ma concentrarci sugli aspetti più essenziali della UI.
Per quanto riguarda il version control abbiamo utilizzato Git e GitHub per gestire il codice in maniera efficace.

Ruoli e attività

Componente del team	Ruolo	Principali attività
Di Cesare Daniele	Project Leader, analista requisiti non funzionali, sviluppatore backend, documentazione e testing delle API	Il ruolo principale è stato la gestione del progetto e delle attività. Ha contribuito attivamente a tutti i deliverable, in particolare al D4 con lo sviluppo, la documentazione e il testing del backend. Ha contribuito alla revisione dei punti sviluppati dagli altri
Tonini Isaia	Progettista mockup, sviluppatore frontend, analista dei componenti	Ha contribuito a tutti i deliverable, in particolare con i mockup e con il frontend. Ha aiutato nei diagrammi e nei requisiti funzionali revisionando i punti fatti dagli altri.
Zendri Matteo	Analista requisiti funzionali, analista del contesto, analista delle classi, progettista diagrammi	Ha contribuito a tutti i deliverable e in particolare stilato tutto il D3. Ha realizzato la maggior parte dei diagrammi e revisionato il lavoro fatto dagli altri, dando soprattutto consigli su frontend e backend

Carico e distribuzione del lavoro

Nella seguente tabella indichiamo le ore di lavoro impiegate per ogni deliverable da ognuno di noi (le ore sono state calcolate con una precisione di mezz'ora):

	D1	D2	D3	D4	D5	TOTALE
Di Cesare Daniele	15.5	21.0	1.5	80.0	2.5	120.5
Tonini Isaia	17.5	20.0	2.0	75.5	5.0	120.0
Zendri Matteo	21.0	22.5	37.5	39.0	4.0	124.0
TOTALE	54.0	63.5	41.0	194.5	11.5	364.5

Il lavoro è stato quasi sempre diviso in modo equo, a parte (come si può vedere) nei D3 e D4. Lo squilibrio è voluto e concordato con tutti i membri: mentre Zendri si dedicava alla creazione del diagramma delle classi, Tonini e Di Cesare si sono occupati di portare avanti rispettivamente frontend e backend. Il gruppo ha concordato su questa divisione per due motivi principali:

1. Lavorare in tre ad un diagramma unico sarebbe stato controproducente
2. Data la quantità di funzioni che sono state implementate, si è deciso di portarsi avanti con il lavoro per evitare di fare tutto in poco tempo

Criticità

Durante lo sviluppo di U-Sushi non ci sono state particolari criticità. Ci sono state incomprensioni, risolte tranquillamente con il dialogo.
Inizialmente il lavoro era distribuito male: tutti all'inizio facevamo tutto rendendo tutto più lento. Da metà del primo deliverable in poi, ognuno ha trovato il suo spazio e il metodo per andare avanti in modo abbastanza efficiente.
Nel D2 per impegni lavorativi Daniele è riuscito a dare meno il suo contributo ma non si è mai tirato indietro del tutto.
Il D3 lo abbiamo voluto assegnare completamente a Matteo per 2 ragioni:

1. sviluppando il diagramma delle classi in più persone le visioni potevano essere contrastanti, rallentando la stesura delle descrizioni e la costruzione del diagramma; per questo Daniele e Isaia hanno fatto una revisione a schema ultimato;
2. Isaia e Daniele nel mentre hanno potuto iniziare lo sviluppo di frontend e backend, portandosi avanti con un lavoro molto complesso e dispendioso in fatto di tempo.

Per il D4 il problema maggiore era l'iniziale assenza di documentazione e la non completezza dell'implementazione: ha reso difficile la stesura delle descrizioni delle APIs e della struttura del progetto.
Un'ulteriore difficoltà avuta verso la fine è stata quella di trovare un modo per hostare frontend e backend, ma siamo riusciti a superare anche questa.
Per quanto riguarda le scadenze soft siamo riusciti a rispettare le prime 3 ma per il D4 e il D5, abbiamo avuto rallentamenti causati da altri corsi ed esami. Nel complesso però siamo riusciti a stare al passo con i tempi.

Autovalutazione

Nel complesso abbiamo tutti lavorato con impegno e costanza, raggiungendo un "prodotto" di buona qualità. Ci siamo resi conto che potevamo dedicare un po' più tempo a definire i requisiti della nostra app. Ci siamo dedicati più o meno allo stesso modo al progetto, dando tutti un contributo significativo. Sulla base di queste considerazioni, questa è la nostra autovalutazione:

	VOTO
Di Cesare Daniele	30
Tonini Isaia	30
Zendri Matteo	30