document ID : deliverable_5

UNIVERSITY OF TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

REVHUB

Titolo del documento:

Report Finale

1.Approcci all'Ingegneria del software

Di seguito sono riportate le nostre esperienze e i nostri pensieri riguardanti i seminari a cui abbiamo potuto assistere durante il corso.

BlueTensor

Abbiamo usato il loro approccio per quanto riguarda la rapidità di sviluppo, considerandolo un punto di forza in unione agli sprint di sviluppo. In aggiunta è stata adottata la loro filosofia di organizzare un gruppo compatto e coeso. La caratteristica dell'ascolto di cosa desidera il cliente è stata adottata come punto chiave per una corretta creazione del progetto.
Inoltre è stato ampiamente usato in modo simile il loro sistema di sviluppo attraverso la loro roadmap, ovvero una analisi di fattibilità preliminare, per poi sviluppare un MVP, e iterare per un continuo miglioramento. Quindi una visione simile al metodo AGILE.
Come punti di forza, oltre a quello sopra descritto, sono stati usati strumenti in comune come diagrammi UML, Mockup e prototipi. Come il loro modello a cascata, anche nel gruppo si è fatta una analisi dei limiti e definizione degli obiettivi, per poi nel secondo passaggio l'analisi dei requisiti funzionali e non funzionali in quanto è stato ritenuto una caratteristica fondamentale per creare le fondamenta del progetto, e infine continuare con lo sviluppo.
E' stato riscontrato l'uso poco efficiente del loro metodo a cascata per la scarsa possibilità di ritornare ai punti precedenti del lavoro.

Meta

Con a Meta, abbiamo appreso il loro punto di vista per quanto riguarda le aspettative di un Software Engineer e quindi di come prima si parta dalla programmazione per poi acquisire esperienza nel design dell'architettura. Considerando ciò e in accordo con le lezioni del corso, abbiamo ritenuto più efficiente fare il contrario e quindi focalizzarci prima sull'architettura.
Come da loro consigliato abbiamo fatto uso del buon senso per ogni parte inerente alla progettazione dell'applicazione. Abbiamo fatto uso della loro specificità nell'utilizzo di più linguaggi per rendere il progetto più flessibile.

U-Hopper

Abbiamo usato la loro politica di sfruttare il concetto di microservizi, implementandolo per lo sviluppo di tante API, ognuna delle quali può collegarsi alle altre. In aggiunta sono stati utilizzati come punti chiave le funzioni asincrone e l'idea degli eventi con loro conseguente scatenamento di trigger per le API. Una

caratteristica ritenuta specifica della presentazione è il concetto di separare la elaborazione della domanda da quella della risposta, quindi un disaccoppiamento.
Abbiamo cercato di implementare uno dei loro punti di forza riguardo il ciclo formato da Code-Test-Review, per ottenere maggiori risultati. Come spiegato abbiamo usato e approfondito l'utilizzo di Git.

RedHat

Con RedHat abbiamo esplorato il tema dell'open source, ossia della possibilità di rendere il proprio codice pubblico e di ciò che questa scelta comporta.
In particolare, ci sono stati innanzitutto illustrati quelli che possono essere i vantaggi di questa pratica:

- condividere la propria conoscenza in un certo ambito della programmazione;
- consentire ad altri (magari anche più esperti di te) di fornire il loro punto di vista e le loro idee al fine di migliorare il codice e in generale il prodotto finale;
- avere un vero e proprio biglietto da visita, poiché chiunque può vedere il tuo lavoro e valutarlo, compresi coloro che potrebbero volerti assumere;
- anche se il codice è in generale pubblico, possiamo scegliere di non lasciare la totale libertà di utilizzo, utilizzando una licenza che obbliga chi usa il nostro codice a rendere a sua volta pubblico il suo intero codice sorgente e con lo stesso tipo di licenza usata da noi.

Uno dei migliori esempi di piattaforme usate per condividere e consultare software (o in generale qualunque tipo di contenuti digitali) in modalità open source è GitHub, uno strumento che include nella sua architettura numerosi strumenti (come issue, merge e fork) che semplificano la realizzazione di tutte le pratiche elencate sopra.
Il nostro stesso gruppo ha usufruito di tale piattaforma, poiché le politiche dell'open source possono essere facilmente applicate in scala più piccola ad un team di poche persone.

Microsoft

Abbiamo usato le loro esperienze per sviluppare la parte inerente al testing per la verifica dei requisiti funzionali e non funzionali, con particolare riguardo per i requisiti funzionali. Una differenza rispetto al seminario è l'assenza di svariate forme di testing e quindi di un'elevata stratificazione, che non esclude però la supervisione del numero di linee di codice eseguite e delle funzioni testate.
La parte inerente al testing è stata sviluppata, in accordo alle caratteristiche del seminario e similmente a come abbiamo fatto, per prevenire regressioni, per la modellazione, per l'identificazione di bug. Abbiamo usato l'approccio per cui il testing è stato sviluppato successivamente al codice.

Marsiglia

Durante questo seminario ci è stato mostrato il tipico ciclo di vita di un software, riassumibile nelle seguenti fasi:

1. Pianificazione - si cerca di analizzare rischi e possibilità, creando un primo piano per il progetto.
2. Requisiti - si fa una lista di tutti i requisiti (funzionali e non funzionali) necessari alla realizzazione del progetto.
3. Analisi e Design - partendo dai requisiti, si cerca di definire quale potrebbe essere l'architettura dell'applicazione da realizzare.
4. Scrittura del codice - si prosegue con la scrittura del codice dell'applicazione.
5. Testing - si testa il codice scritto per verificarne la correttezza e solidità.
6. Rilascio - si pubblica l'applicazione permettendone l'installazione al pubblico.
7. Mantenimento - dopo la pubblicazione, occorre assicurarsi il continuo funzionamento dell'applicazione, oltre che considerare l'introduzione di nuove funzionalità.

Naturalmente esistono anche altri approcci oltre a quello tradizionale, come quello:

- Iterativo, in cui si ripetono più volte i passaggi dell'approccio tradizionale al fine di avere tante piccole releases.
- Agile, in cui ci si basa su continui test ed interazioni col committente.
- DevOps, basato sul team, in cui vi sono continuamente miglioramenti, integrazioni, rilasci di nuove versioni e feedback dagli operatori, al fine di migliorare il prodotto finale.

Per il nostro progetto abbiamo adottato un approccio più vicino a quello tradizionale, dedicando una grossa fetta del nostro tempo alle fasi di pianificazione e raccolta ed analisi dei requisiti, per poi dedicarci alla scrittura del codice di una parte dell'applicazione in un singolo sprint seguita da una fase di testing.

APSS & Trentino.ai

L'Azienda Provinciale per i Servizi Sanitari (APSS) è un ente della Provincia Autonoma di Trento con autonomia imprenditoriale. La sua missione è gestire in modo coordinato le attività sanitarie e sociosanitarie nel territorio provinciale, seguendo le direttive del piano provinciale per la salute. I principi fondamentali includono la presa in carico completa degli utenti, l'omogeneità dell'offerta dei servizi su tutto il territorio e la semplificazione dei processi attraverso l'utilizzo di nuove tecnologie.

L'adozione dell'Intelligenza Artificiale (AI) nel settore sanitario è in crescita, con un notevole interesse da parte delle Aziende Sanitarie per migliorare la qualità delle cure e l'efficienza operativa, andando in generare ad impattare positivamente in tutti gli aspetti del settore. Questa tendenza coinvolge sia aziende pubbliche che private, utilizzando l'AI per ottimizzare la gestione dei pazienti, la diagnosi e la ricerca medica.

Nonostante il seminario abbia condiviso con noi interessanti informazioni su quello che alla fine dei conti è il funzionamento di un ente la cui attività che riguarda tutti noi, abbiamo trovato più difficile (rispetto agli altri casi) mettere in relazione quanto ascoltato col nostro lavoro.

Kanban

Durante la conferenza del dottor York Rössler abbiamo assistito ad una iniziale spiegazione del funzionamento e storia del metodo delle Kanban Boards, utilizzate nella gestione delle varie attività durante un ciclo di sviluppo.
Successivamente abbiamo preso parte ad una simulazione dove, divisi in gruppi da 2-3 persone, abbiamo sperimentato con mano come si utilizza una Kanban board, utilizzando dei post-it come rappresentazione delle diverse attività e, simulando lo scorrere del tempo, tenevamo traccia di quanto una singola attività impiegasse prima di essere conclusa.
A circa metà simulazione abbiamo introdotto il limite di 2 attività per colonna, e in questo modo, i tempi di svolgimento delle singole attività si sono ridotti drasticamente, come si è potuto evincere dal confronto finale tra le attività svolte nella prima e nella seconda parte della simulazione.
Queste nozioni si sono rivelate molto utili nella pianificazione delle varie parti del progetto, permettendoci di velocizzare i tempi di lavoro e di essere in pari con le varie scadenze.

IBM

Con IBM abbiamo assistito ad una presentazione riguardante il cloud computing e tutta l'infrastruttura messa a disposizione dall'azienda.
È stato spiegato come utilizzare i vari servizi messi a disposizione e come metterli in comunicazione tra loro per poter costruire sistemi di complessità anche elevata. Inoltre sono stati fatti anche esempi su come costruire delle architetture che abbiano un alto livello di sicurezza, resistenti ad attacchi informatici, ad esempio DDoS, e con una connessione crittografata con l'esterno, per poter garantire la confidenzialità.
Infine è stato mostrato come poter effettivamente creare un servizio in cloud partendo da un caso simil-reale. Per fare questo si è usato il linguaggio visuale proprio della piattaforma, NodeRED.
Questa presentazione è stata utile per poter capire come approcciarsi al mondo dei servizi in cloud, e non necessariamente servizi IBM. Grazie a ciò siamo riusciti a gestire la configurazione del database MongoDB durante la creazione del progetto.

Molinari

Durante questa presentazione, da parte del professor Andrea Molinari, sono stati presi in esame i cosiddetti sistemi legacy, ovvero quei sistemi obsoleti in una o più delle sue componenti.

Si è parlato di come i sistemi legacy siano costituiti sia da parti hardware, come i computer mainframe ancora in uso oggi nel cloud, e sia da parti software, come tutta l'infrastruttura bancaria e commercialistica, ma non solo, scritta in COBOL.
Inoltre si è parlato anche dei fattori che fanno continuare ad utilizzare questi sistemi legacy, e di come le aziende accumulano sempre più debito tecnologico, utilizzando sempre più approcci aggressivi e tendenti al rilascio di software prima di quando necessario.
Si è posto un accento anche sull'uso corrente del linguaggio COBOL e di come anche varie aziende diverse ci stiano puntando, chi con compilatori propri, chi con ambienti di sviluppo adatti.
Per ultima cosa si è notato come il mondo agile non si adatta bene al lavoro con i sistemi legacy, dal momento che la portata di questi progetti è estremamente grande e difficilmente gestibile in vari sprint della durata di pochi giorni.
Questa presentazione è stata interessante, tuttavia non abbiamo potuto sfruttare le conoscenze acquisite dal momento che il progetto non prevedeva l'uso di sistemi legacy, siano questi hardware o software.

2.Organizzazione del lavoro

Il carico di lavoro è stato spartito secondo i punti di forza di ogni partecipante, cercando di sfruttare le competenze pregresse di ogni componente assegnando ad ognuno il ruolo più adatto.

Dal punto di vista degli strumenti, abbiamo utilizzato il materiale delle lezioni del corso, quindi ad esempio MongoDB e NextJS, in quanto React e l'ambiente web erano già familiari a parte del gruppo.

Per quanto concerne gli incontri con i membri del gruppo, abbiamo lavorato in parte in asincrono e in parte "in presenza", tenendoci costantemente in contatto (per lo più tramite un gruppo Telegram) e incontrandoci circa due volte a settimana per confrontarci meglio sul lavoro svolto da ognuno, sui progressi complessivi e sulle fasi che seguivano. Inoltre comunicazioni e incontri venivano sfruttati al massimo per confrontarsi su problemi e per condividere opinioni e conoscenze, sfruttando l'attività anche per ottenere nuove competenze (magari già familiari ad altri componenti del gruppo) oltre che per portare avanti il progetto.

3.Ruoli e attività

La seguente tabella, ordinata per ordine alfabetico, rappresenta una breve schematizzazione delle principali attività svolte nel Team.

Componente del Team	Ruolo Principale	Principali Attività
Bastri Velia	Project leader	Scrittura dei documenti. Implementazione di una parte di frontend, e aiuto nel completamento di api.
De Giorgi Ismaele	Coder	Scrittura dei vari deliverables. Implementazione di funzionalità di fuzzy search e bug fix.
Scotton Alessandro	Coder	Scrittura del codice dell'applicazione. Contribuzione alla scrittura e revisione dei vari deliverables. Scrittura della documentazione delle api. Scrittura della parte di testing.

4.Carico e distribuzione del lavoro

Dal nostro calcolo dell'ammontare del quantitativo di lavoro, espresso in ore per persona, segue la tabella riassuntiva.

	D1	D2	D3	D4	D5	Codice	TOT
Velia	8	8	6	16	9	15	62
Ismaele	7	8	6	14	7	21	63
Alessandro	6	9	5	20	7	38	85
TOT	21	25	17	50	23	74	210

Prima di qualsiasi considerazione, occorre sottolineare che la riga e colonna del tempo totale va "presa con le pinze", poiché se due o tre componenti lavorano assieme ad una certa parte del progetto (per mettere a confronto delle idee, per risolvere un problema, o altro ancora…) il tempo totale dedicato a quell'attività rapportato ai progressi fatti aumenta in modo diverso rispetto al caso in cui una sola persona si occupi di quell'attività.

Detto ciò, ognuno ha cercato di dedicare il tempo che riteneva adeguato per ogni attività.
Eventuali squilibri possono essere ricondotti a due fattori in particolare:

- ad una diversa velocità dei singoli membri nel completamento di una certa attività;
- alla suddivisione del lavoro che è stata effettuata, che può aver portato un membro a dedicarsi ad una parte del progetto di più rispetto ad un'altra;
- al tempo richiesto per l'apprendimento di codice o di come usare i software richiesti per la creazione dei vari grafici.

5.Criticità

Durante l'attività non abbiamo incontrato criticità particolarmente difficili da risolvere. Infatti quelle che abbiamo trovato hanno richiesto "solo" un'elevata quantità di tempo per trovare una soluzione. Inoltre abbiamo incontrato problematiche particolarmente rilevanti solo durante la scrittura del codice dell'applicazione, mentre durante le altre fasi non abbiamo mai avuto particolari difficoltà, se non qualche dubbio facilmente risolvibile attraverso qualche domanda ad un Prof o ad un Tutor.

Le criticità più grandi affrontate, sono state:

- decidere quali funzionalità implementare nello sprint;
- capire come interagire con mongoDB per alcuni tipi di query;
- creare un contesto nell'applicazione per usare variabili globali.

6.Autovalutazione

Per trarre l'autovalutazione riportata nella seguente tabella per ognuno dei componenti del team, abbiamo considerato che tutti si sono impegnati secondo le loro abilità e la loro competenza, con l'aggiunta di pronta comprensione e ascolto alle idee altrui.
Dopo questa considerazione, possiamo affermare che Alessandro Scotton e la sua esperienza nello sviluppo web hanno contribuito positivamente al lavoro comune e all'incremento delle competenze del gruppo.

	VOTO
Bastri Velia	28
De Giorgi Ismaele	28
Scotton Alessandro	29