Document: FitCal_ReportFinale

Revision: 0.1

Progetto:

FitCal

Titolo del documento:

Report Finale

Document Info:

Document name	D5-FitCal_ReportFinale	Document number	RF5
Description	Report finale del progetto. Descriveremo la suddivisione del lavoro, i ruoli assegnati a ciascun membro, il tempo complessivo impiegato, autovalutazione e criticità riscontrate.		
Authors	Azemi Kevin Lollato Filippo Mattioli Michele		

INDICE

Contenuti del documento

Il presente documento rappresenta il report finale del progetto sviluppato dal gruppo G41.
Inizieremo con la descrizione dell'organizzazione del lavoro e dei ruoli assegnati a ciascun membro del team.
Successivamente, saranno esaminate le varie criticità incontrate e illustreremo le soluzioni adottate, accompagnate da un’autovalutazione basata sui fattori precedentemente descritti.
Concluderemo il documento con una panoramica dei seminari tenuti durante il corso.

Organizzazione del lavoro

In questo paragrafo facciamo un resoconto di come ci siamo gestiti la mole di lavoro, sia per la scrittura del sito web FitCal che per la produzione dei documenti.

Durante lo sviluppo del progetto abbiamo avuto l'opportunità di imparare a lavorare in gruppo, organizzandoci in modo tale da sfruttare al meglio le competenze di ogni membro del team, garantendo una comunicazione efficace.

Per ogni documento, abbiamo svolto meeting, sia virtuali che di persona, in modo tale da stendere un piano di lavoro concreto e fattibile; seguire tale piano è stato ovviamente più difficile dell'ideazione di esso, pero' siamo tutti e tre convinti che cio' ha avuto risvolti piu' che positivi, in quanto nella maggior parte dei casi riteniamo ci abbia risparmiato diversi problemi di logistica e organizzazione.

La nostra inesperienza e' venuta sicuramente fuori durante le prime fasi del D1, in quanto il corso costituiva per ognuno di noi una grande novita', intesa proprio come contenuti, e cio' ha portato a momenti di confusione e errori nell'assegnazione dei compiti; motivo per cui il primo documento presenta un carico di ore cosi' elevato, vista la sua relativa semplicita', se confrontato con gli altri.

Tuttavia, tali ore non sono andate certo sprecate, in quanto ci hanno sicuramente formato: cio', unito alle tecniche e metodologie di ingegneria del software apprese durante il corso, ci ha permesso di affrontare il resto dei documenti senza quasi mai riscontrare problemi dovuti ad un'errata organizzazione del lavoro.

Ruoli e Attività

Vediamo descritti nella seguente tabella i ruoli e attività svolte da ciascun membro del team.

Membro	Ruolo	Attività
Azemi Kevin	Software developer (frontend / backend), Technical Architect	Si e' occupato principalmente dello sviluppo del sito web, verificando che il design delle API, delle risorse proposte e delle interazioni fossero fedeli ai requisiti del progetto. - D1: ha contribuito all'ideazione dell'applicazione; ricerca di fattibilita' dei requisiti, funzionali e non funzionali; ha contribuito al design back-end. - D2: ha contribuito alla specifica dei requisiti non funzionali e di alcuni diagrammi. - D3: ha contribuito alla verifica della conformità dei diagrammi con quanto verrà implementato all'interno dell'applicazione. - D4: ha contribuito all'implementazione del sito, lato backend e frontend, alla descrizione e documentazione delle API, alla sezione di testing e deployment. - D5: ha contribuito alla presentazione delle criticità, all'organizzazione del lavoro e ruoli e attività assunte.
Lollato Filippo	Project manager, Technical Writer.	Si e' occupato in gran parte della stesura di documenti, oltre ad avere il ruolo di intermediario tra implementazione fisica dell'applicazione ed implementazione formale su documento. - D1: ha contribuito all'ideazione e stesura dei requisiti funzionali e non, ideazione e realizzazione dei mockup per il frontend e design del back-end. - D2: ha contribuito alla specifica dei requisiti funzionali ed alla verifica degli use case diagram. - D3: ha contribuito alla specifica e stesura dei diagrammi delle classi e codice OCL con relative verifiche d'implementazione lato applicazione. - D4: ha contribuito alla realizzazione dei resource ed API diagrams, stesura della struttura del progetto, dipendenze e forma dei modelli presenti all'interno dell'applicazione. - D5: ha contribuito alla descrizione dei ruoli e attività, alla descrizione delle criticità affrontate ed alla verifica formalità seminari.
Mattioli Michele	Technical Writer, Product Manager, Data Analyst.	Si e' occupato principalmente della stesura dei documenti e dei diagrammi, con attenzione al lato formale dello sviluppo dell'applicazione. - D1: ha contribuito all'ideazione e stesura dei requisiti funzionali e non funzionali. - D2: ha contribuito alla specifica dei requisiti funzionali e non, all'analisi del contesto, nonché all'analisi dei componenti, il tutto con i relativi diagrammi.

		- D3: ha contribuito alla verifica formale dei class diagram e OCL. - D4: ha contribuito alla creazione del diagramma user-flow e alla descrizione del front-end, con immagini. - D5: ha contribuito alla stesura dei vari seminari, alla descrizione dei ruoli e attivita', alla sezione organizzazione del lavoro.

Tab. 1: Riassunto dei ruoli e delle attivita' ricoperti da ciascun membro del team.

Carico e distribuzione del lavoro

Membro	D1	D2	D3	D4	D5	Totale
Azemi Kevin	17	15	15	102	4	153
Lollato Filippo	37	20	70	22	8	157
Mattioli Michele	20	89	15	25	13	162
Totale	74	124	100	149	25	472

Tab. 2: Distribuzione finale delle ore.

Criticità

Abbiamo deciso di suddividere le criticità in due sezioni:

- criticità sulla distribuzione del lavoro: verranno elencate le motivazioni e problemi riscontrati riguardo all'organizzazione e comunicazione tra i membri del team;
- criticità sullo sviluppo del progetto: verranno elencati problemi tecnici riscontrati durante la fase di sviluppo dell'applicazione.

Criticità sulla distribuzione del lavoro

Queste criticità hanno evidenziato l'importanza di una pianificazione accurata e di una gestione efficace delle risorse.

Inizialmente, la comunicazione tra i membri del team è stata frammentaria e piuttosto inefficace, in quanto non avevamo stabilito canali di comunicazione chiari e frequenti, il che ha portato a incomprensioni e ritardi nell'avanzamento del progetto. Ad esempio, le modifiche ai vari requisiti funzionali non avvenivano abbastanza tempestivamente, e non stavano quindi al passo con la fattibilita' e la ricerca del progetto.

Tuttavia, anche qua, molte delle difficoltà si sono susseguite durante la prima fase del progetto, per poi sparire del tutto durante il D2 ed il D3. Solo durante il D4 si sono presentate di nuovo piccole discussioni, riguardo soprattutto l'integrazione del codice, e se aggiungere o meno determinate funzionalita'.

Nel complesso, reputiamo che il nostro percorso sia stato veramente buono, e che abbiamo saputo superare ogni sfida che il corso ci ha presentato in maniera egregia, senza mai trascurare la componente sociale del progetto.

Criticità sullo sviluppo del progetto

Nello sviluppo del progetto abbiamo incontrato due principali criticità tecniche oltre alle iniziali incomprensioni:

- rate limiting e problemi con il fetch all'API di Open Food Fact;
- problemi con il deployment su Vercel.

Rate limiting e fetch di Open Food Facts

Una volta create le richieste all'apposita API di Open Food Facts avevamo notato che alcune di esse non avevano successo, mentre altre riuscivano a ritornare i dati, ma il tempo impiegato era oltre quello previsto dai requisiti non funzionali, in particolare RNF2 - Prestazioni.
Dopo esserci confrontati per possibili soluzioni ed aver cercato su forum online abbiamo deciso di implementare un sistema di caching iniziale tramite Redis. Questo ha incrementato notevolmente le performance, tramite un primo fetch dei dati e successivo caching,

permettendo un'esecuzione rapida per tutte le successive richieste di fetch allo stesso prodotto. Questo ci ha inoltre consentito di non raggiungere velocemente i limiti di richieste giornaliere imposte dall'API.

Deployment con Vercel

Un problema sorto durante la fase di deployment su Vercel era l'upgrade del piano dato che la repository di GitHub era sotto una organizzazione. Dopo un piccolo brief abbiamo deciso di aggirare il problema creando una fork locale, la cui unica funzionalità era il deployment su Vercel.
Le fase di development aggiornata si suddivide così in tre parti:

- commit sulla repository dell'organizzazione;
- pull degli aggiornamenti sulla repository locale;
- push su Vercel degli aggiornamenti.

Considerazioni sulle criticità

Nel complesso la fase di development è stata quasi priva di problemi, e la maggior parte di essi si sono presentati durante fase di organizzazione del lavoro e dei compiti assegnati a ciascun membro.

Autovalutazione

Nonostante le iniziali problematiche riscontrate, siamo riusciti a raggiungere tutti gli obiettivi da noi posti: sito funzionante, performante e flessibile per future aggiunte di funzionalità e fedele ai requisiti, sia funzionali che non funzionali.
Durante la fase di sviluppo l'organizzazione è migliorata notevolmente e ne abbiamo visto i risultati, considerati i pochi problemi riscontrati.

Membro	Autovalutazione
Azemi Kevin	30L
Lollato Filippo	30L
Mattioli Michele	30L

Approcci all'Ingegneria del Software

Blue Tensor

Durante questo seminario l'ing. Jonni Malacarne, CEO di Bluetensor Srl, ha descritto le attività della sua azienda, per poi passare alle metodologie e agli strumenti di ingegneria del software usati nello sviluppo del loro software.

Chi sono

Bluetensor è un'azienda che produce software di intelligenza artificiale, ad esempio un rilevamento automatico di errori nei componenti delle auto.

Sviluppo Software

Nello sviluppo di software AI, a differenza di un normale processo di creazione software, devono essere considerati anche gli aspetti di disponibilità dei dati e gli aspetti etici. Inoltre, seppur la metodologia agile sia stata confermata anche dal CEO di Bluetensor come la migliore a livello ideale, nelle aziende tale approccio non è applicabile per l'intero sviluppo. Questo perché ogni azienda necessita di fissare un budget, quindi non è possibile partire da carta bianca, bisogna avere gia' un'idea del tempo e delle risorse da investire nel progetto.

Fasi di sviluppo

L'intero processo di sviluppo viene suddiviso nelle seguenti 5 fasi:

1. *WP1*: Analisi e Fattibilità
    - Comprensione del contesto di riferimento;
    - Analisi di dettaglio delle funzionalità;
    - Prove di fattibilità;
    - Definizione obiettivi e limiti.

    Documenti prodotti: Documento di analisi preliminare, Report di prove fattibilità.
2. *WP2*: Setup e Design
    - Story mapping: definizione user stories;
    - Definizione requisiti funzionali;
    - Definizione requisiti non funzionali;
    - Design architettura hardware e software di sistema;

    Documenti prodotti: Documenti di analisi dei requisiti Funzionali e Non Funzionali, User Stories, Architettura Hardware, Architettura Software.
3. *WP3*: Preparazione dei dati
    - data selection;
    - feature selection;
    - data processing;

    Documenti prodotti: Dataset Training, Dataset Test.

4. *WP4*: Sviluppo & Training & Test & Deploy

Questa parte è fatta in modalità puramente agile con utilizzo della pipeline al fine di essere più produttivi, quindi vengono rilasciate costantemente porzioni di codice e le loro funzioni mostrate al cliente.

- moduli funzionalita' core;
- moduli 1-N.

Documenti prodotti: Ambiente Locale, Ambiente Test, Ambiente Staging, Ambiente Produzione.

5. *WP5*: Rilascio versione finale
    - raccolta feedback improvements;
    - formazione utenti;
    - continuous learning;

    Documenti prodotti: Documento con i vari miglioramenti futuri possibili, Versione Finale del software.

Considerazioni finali

E' stato molto interessante vedere come le tecniche studiate a lezione vengano usate effettivamente nelle aziende di software. Comparando il loro processo di sviluppo al nostro, ci sono state diverse analogie, ad esempio la suddivisione dell'intero lavoro in 5 fasi principali, e la metodologia agile, anche se nel loro caso viene sfruttata esclusivamente durante il WP4. Inoltre, molti degli strumenti da noi adoperati, o comunque visti in classe, sono essenziali per il team di Bluetensor, come i vari UML Diagrams (use-case, activity, state machine), i Mockups per l'interfaccia grafica, Docker e l'uso di GIT come software per il controllo distribuito di versione.
Ovviamente, viste le dimensioni dell'azienda Bluetensor, nel loro caso vengono usate anche altre tecnologie, quali ad esempio gli Architecture Diagrams e Amazon Web Services come cloud.

Kanban

In questo seminario il Dr. York Rossler ha introdotto le basi del metodo Kanban, per poi farci provare su pelle cosa significhi adottare tale approccio nella gestione delle attivita' di un progetto.
Ad ogni membro del gruppo e' rimasta impressa questa lezione in quanto abbiamo anche svolto un esperimento, col fine di dimostrare la validita' del metodo Kanban. Tale esperimento, riassumendo brevemente, prevede di formare dei gruppi di 4 persone; fatto cio' venivano assegnati 4 task: Options, Design, Implementation, Done, e dovevamo spostare quanti piu' ticket possibili dal campo Options al campo Done. Inizialmente abbiamo adoperato una normale strategia, per poi provare quella kanban e vederne effettivamente i benefici.
In particolare, la gestione dei processi con delle carte che rappresentano attivita' o compiti comporta i seguenti vantaggi sostanziali:

- visualizzazione del flusso di lavoro;
- gestione ottimizzata del lavoro in corso;
- adattabilita' e flessibilita'.

Considerazioni finali

Il metodo Kanban costituisce sicuramente un interessante metodo per gestire i vari processi, e si adatta bene allo sviluppo di un'applicazione software. Noi, come gruppo, abbiamo provato ad adoperarlo a partire dalla settimana dopo la presentazione, tuttavia abbiamo riscontrato alcune difficolta':

- gestione di alcuni processi complessi;
- limitata proattivita', che comporta spesso di riformulare il piano strategico da seguire.

Ovviamente siamo anche consapevoli che una mal suddivisione dei task, che sia con il metodo Kanban o meno porta inevitabilmente a risultati non positivi, dunque la nostra inesperienza potrebbe aver influito. Certamente tale metodo e' una valida alternativa, e riteniamo che se utilizzato bene possa portare notevoli benefici al processo di sviluppo software, come anche testimoniato dalle diverse compagnie che ne fanno, o hanno fatto, uso, quali ad esempio Microsoft, nello sviluppo della Xbox.

IBM

L'argomento di questo seminario, svolto da Ferdinando Gorga, e' stato il Cloud IBM e alcuni suoi casi d'uso.

Cosa è

Il Cloud IBM è una piattaforma distribuita che offre sia potenza computazionale che tecnologia software utilizzabile da chiunque, sotto forma di servizi. I dati vengono conservati su dei data center IBM, distribuiti sul pianeta.

Potenza computazionale

La potenza computazionale e' offerta in vari modi, in ordine di astrazione dal mezzo fisico: Bare metal, Virtual machines, Containers, Serverless Code Engine. Anche la tecnologia software e' offerta attraverso molteplici risorse, ad esempio: Containers, Machine learning, Blockchains, Databases ecc..

Attività svolte

Sono stati presentati dei progetti passati, realmente realizzati, insieme ad uno dei principali problemi di cui tener conto: la sicurezza verso gli attacchi informatici.
Dopo di che, abbiamo fatto un esperimento che prevedeva l'implementazione di una piattaforma per fare trading di Bitcoin, scegliendo accuratamente i servizi offerti dal catalogo IBM; alla fine, con le personalizzazioni effettuate, i primi 100.00 secondi avevano costo $0.00, mentre ciascun secondo successivo costava 0.0000319$.

Considerazioni finali

Come ogni tecnologia Cloud, il cloud IBM potrebbe fornire servizi aggiuntivi per la nostra applicazione, ad esempio noleggiando un servizio di Database potremmo avere notevole spazio dati aggiuntivo, utile, tra le altre cose, per effettuare dei backup.

META

Questo seminario ha visto la partecipazione di Heorhi Raik, software engineer presso META, che ha presentato alcune delle tecnologie utilizzate all'interno di una grande azienda di software quale essa e'.

Ruoli di un Software Engineer

Dopo avere esposto il suo ruolo in questi anni in META, ossia lavorare a stretto contatto con i dati al fine di produrre cataloghi per scopi pubblicitari e di e-commerce, insieme ad attivita' di web-scraping, abbiamo visto le responsabilita' di un swe:

- Per quanto riguarda quelle tecniche, le principali descritte da Heorhi sono: scrittura del codice, design di sistema, project management;
- Ad esse, poi, si aggiungono altri titoli, meno tecnici, se vogliamo, come ad esempio: occuparsi dei colloqui di lavoro e di organizzare eventi, ruoli che prevedono la scelta dei componenti per un team, mentoring/coaching.

Tecnologie usate

Molto interessante e' stata la confessione che Heorhi ha fatto sulle metodologie usate per lo sviluppo di software: sostanzialmente non viene seguito nessun processo specifico, che sia Agile, Kanban etc., ma ogni team sceglie la metodologia a suo piacimento. In particolare, Heorhi non ha mai studiato formalmente specifiche tecniche di sviluppo e, secondo lui, META non chiedera' mai tali argomenti durante un colloquio. Sempre riguardo all'ingegneria del software, spostandosi sui tools da lui usati abbiamo: Asana, Excel, Mercurial, VS Code, Daiquery/Presto.
Interessante e' stato scoprire i linguaggi di programmazione maggiormente utilizzati in META, osservando come quasi ogni tecnologia, quindi anche i linguaggi, venga modificata e adattata perfettamente alle esigenze dall'azienda: Hack (PHP), JS (React), Python, SQL, C/C++, Java, Haskell.

Quesiti interessanti

Il resto del seminario e' stato dedicato alle domande da parte di noi studenti; alcune delle piu' interessanti sono riportate in questo documento, con le relative risposte.

- Riguardo al "work-happiness balance" Heorhi sostiene che l'azienda ci tiene a mantenerlo sempre presente, quindi in ufficio in generale e' presente armonia e, la maggior parte dei dipendenti e' soddisfatta.
- Per comunicare idee spesso, oltre ai normali meeting di lavoro, viene usato Google Doc. Questo, in effetti, si riflette anche sul nostro progetto FitCal, in quanto per comunicare, nella maggior parte dei casi, abbiamo fatto riferimento ad esso.
- Interessante e significativo e' stato quando, dopo che gli e' stato chiesto cosa differenziasse un bravo swe da un normale swe, Heorhi ha risposto con una singola parola: *l'impatto*, inteso come l'abilita' di innovare all'interno dell'azienda, collegandosi poi alla profonda conoscenza degli algoritmi, e di come essi costituiscano un requisito fondamentale per chiunque voglia entrare a lavorare in META.

U-Hopper

Durante questo seminario, Daniele Miorandi, CEO, tra le altre cose, di U-Hopper, ha presentato gli obiettivi della sua ultima azienda, insieme allo stack di tecnologie e di processi dell'ingegneria del software utilizzati per gestire una tale realtà industriale.

Cosa è

U-Hopper e' un'azienda specializzata in AI e in Big Data, che aiuta altre aziende a ricavare valore dai dati posseduti, in quanto spesso mancano le competenze e le tecnologie per farlo autonomamente. Dispone di un team abbastanza ampio, 13 persone, ed ha sede a Trento.

Tech stack

Tra tutti i seminari affrontati, questo e' uno di quelli in cui i temi del corso, quindi tecnologie e patterns per lo sviluppo di buon software, sono stati affrontati in modo più esaustivo, con diversi approfondimenti.
Lo stack tecnologico di questa azienda e' molto ampio, troviamo, per la parte cloud: Google Cloud, AWS e Microsoft Azure. Per il controllo di versione e' impiegato Git, ed usufruiscono di Kafka e MQTT per gestire il flusso di dati come una coda. Per quanto riguarda l'elaborazione dei dati, abbiamo:

- Spark, per il batch processing;
- Spark streaming e Flink per lo stream processing.

Passando poi alla memorizzazione dei dati, anche qui ci sono diversi approcci, tutti utilizzati da U-Hopper:

- in-memory database: Redis;
- MongoDB, MySQL e Cassandra per database relazionali e non;
- InfluxDB per Time Series Analysis;
- Celery ed AirFlow per gestire i task ed orchestrarli;

Il linguaggio piu' usato e' Python (90%), impiegato anche per la parte OOP, seguito da Scala e Go. Un altra tecnologia usata è Docker, fatto girare su cloud, ed IaaS. Come IDE i consigliati sono PyCharm e Vs Code, insieme a Copilot.

Design patterns

Riguardo ai design pattern, spesso sono monolitici e le applicazioni asincrone , in quanto si lavora con molti dati e utilizzare microservizi produrrebbe overhead.
Il processo di sviluppo e' suddiviso in piccoli sprint, usando quindi le milestones di Git, per poi concludere stimando il tempo richiesto per tale processo. Ogni sprint sostanzialmente e' un ciclo composto da 3 fasi: Code → Test → Review. Per quanto riguarda i deployments, inizialmente vengono fatti in staging; seguono poi test, sempre in staging, la release dell'applicazione e, infine, il deployment in produzione, ciò che vedono i clienti.

RedHat - OpenSource

Questo seminario ha visto Mario Fusco, swe attivo su progetti liberi di Red Hat, raccontare la sua storia e presentare i principali aspetti del mondo open source.

Risvolti inaspettati

Nella prima parte del seminario Mario ha presentato il suo primo progetto open source: Lambdaj, una libreria di metodi statici che servivano ad emulare le lambda expression prima che fossero ufficialmente introdotte in Java. Questo attiro' l'attenzione di molti altri sviluppatori ed ha costituito sicuramente un punto di svolta nella sua carriera; cio' insegna come ogni nostro progetto possa effettivamente portare ad un cambiamento nell'industria informatica, spesso anche inaspettato.

Mondo open source

Interessante è stato quando abbiamo parlato in dettaglio dei progetti e, più in generale, del mondo open source. Alcuni vantaggi sono sicuramente:

- la possibilita' di condividere la conoscenza con gli altri;
- il fatto che il codice e' sotto gli occhi di tutti garantisce un maggiore controllo, diminuendo la probabilità di avere bug.
- la possibilita' di crearsi un portfolio, partecipando attraverso dei progetti veri;

Guadagnare nell'open source

Altro argomento trattato sono le licenze, che si suddividono in Copyleft (ogni lavoro derivato dal progetto iniziale deve avere sempre questa licenza) e Non-Copyleft (i progetti che derivano da quello iniziale possono non essere open source), insieme a come si guadagna tramite un progetto open source, tema questo su cui l'intera classe ha posto molte domande e su cui si e' aperto un notevole dibattito sulle varie possibilia'. In sostanza, sono:

- donazioni;
- contributi da un'azienda in particolare, per cui si lavora;
- funzionalita' in piu' sul sistema software, messe a disposizione del cliente solo a pagamento;
- fare delle conferenze ed organizzare corsi, ad esempio alle aziende, per spiegare come usare al meglio il software da noi sviluppato.

Nell'ultima parte del seminario abbiamo discusso di come entrare a far parte di un qualsiasi progetto open source, fatta eccezione per quelli in cui sono richieste particolari conoscenze specifiche, per esempio algoritmi avanzati in ambito di AI etc..

Considerazioni finali

Siamo rimasti molto colpiti dalla qualità dell'intervento di Mario Fusco che, a nostro parere, ha reso questo uno dei seminari più interessanti. Anche noi abbiamo pubblicato il codice su github, rendendolo a tutti gli effetti un progetto open source, con la speranza che in futuro sempre più persone possano contribuire al miglioramento dell'applicazione.

Microsoft

Durante questa presentazione il Dr. Diego Colombo ha descritto dettagliatamente come avviene il testing del codice in una grande azienda quale Microsoft. Il testing, come gia' sappiamo, e' la verifica, manuale o automatica, di parti del programma, in modo da rilevare qualsiasi tipo di comportamento indesiderato (bug) e verificare che esso rispetti i requisiti funzionali e non.
Diego ha mostrato i vari tipi di test usati da Microsoft, come ad esempio Unit test, Integration test, Regression test, e tanti altri; per quanto riguarda i tools usati, quindi lo stack tecnologico, per la fase di test, alcuni di essi sono: vari frameworks per i test (Mocha), UI test automation, Cucumber, file di log, Wallaby e NCrunch.
La regola da tenere sempre a mente, secondo Diego, e' di fare un setup per il testing che preveda che 'act' e 'assert' avvengano sempre allo stesso livello, sia che stiamo testando una classe o una libreria o un servizio.
Infine, molto interessante e' stata anche la restante parte del seminario: abbiamo visto in concreto diversi esempi di testing, in particolare unit test, scritti in typescript. Secondo noi questa seconda parte e' stata fondamentale per catturare la vera importanza dei test, perche' e' stato mostrato come fare un corretto setup per gli unit test, e anche l'inserimento di specifici errori che poi portavano a risultati errati. Ci e' servito molto questo seminario, in quanto circa una settimana dopo, quando dovevamo iniziare a testare il nostro codice, abbiamo preso spunto dal setup fatto da Colombo, ed in parte eravamo gia' preparati; grazie ai testing abbiamo inoltre individuato bug che, senza un metodo effettivo ed un corretto setup per gli unit test, difficilmente avremmo individuato per conto nostro.

Molinari

Argomento di questo seminario sono stati i sistemi legacy, presentati ed analizzati dal professore Andrea Molinari in dettaglio. Abbiamo poi discusso 4 casi di reali sistemi obsoleti.

Caratteristiche di un sistema legacy

Innanzitutto, per sistema legacy, o obsoleto, si intende un sistema informativo vecchio, sotto ogni punto di vista (sia hardware che software), ma che e' ancora in uso, in quanto svolge correttamente i task per cui era stato progettato. Abbiamo visto che, in generale, ogni applicazione di questo tipo ha le seguenti caratteristiche:

- l'hardware, oltre che datato, e' di tipo mainframe;
- tale hardware fornisce un'estrema potenza di calcolo (240 CPU su un singolo sistema, 40 TB di RAM, memoria fissa anche dell'ordine dei Peta, I/O velocissimo);
- interfaccia grafica minimale, quasi assente;
- occupazione di grandi spazi;
- sistema "chiuso";
- mancanza di qualsiasi forma di OOP e tecnologia multithreading.

Interessante è stato scoprire che il software di queste applicazioni e' strettamente legato all'hardware. Ciò si riflette anche sulla scelta del sistema operativo utilizzato, spesso z/OS, VMs o Linux, e sulla scelta del linguaggio con cui programmare tale sistema.

Perché sopravvivono

Fin dall'inizio della lezione, ci siamo chiesti questa domanda ed i motivi sono principalmente di natura economica:

- il motivo principale e' che il nuovo sistema richiede notevoli investimenti, insieme poi al periodo di passaggio ed ai costi di dismissione;
- a livello dei dipendenti, il cambiamento spesso crea disagio e resistenza interna;
- paura di migrare verso un sistema nuovo, a causa della scarsa documentazione e dell'assenza degli sviluppatori originali;
- TCO/TBO a vantaggio di tali sistemi rispetto a quelli moderni.

Ingegneria del software nei sistemi legacy

Per quanto riguarda gli strumenti e le tecniche di ingegneria del software, esse vengono usate in modo massivo durante il processo di modernizzazione di un sistema legacy, che puo' essere applicato in uno dei seguenti modi:

1. Dismissione: il sistema viene smantellato;
2. Migrazione: il software ed i dati vengono portati nel nuovo sistema;
3. Interazione: il vecchio sistema dovra' interagire con uno nuovo di supporto;
4. Inclusione: il nuovo sistema diventa una componente di quello vecchio.

Alcuni strumenti citati dal professor Molinari sono: UML, SysML (evoluzione di UML) ed Archimate, e l'AI generativa. Inoltre, vista la portata di questi progetti, in particolare dei costi, e la difficolta' nel coordinamento delle attivita', il metodo Agile non si presta bene, e viene quindi preferito lo schema Waterfall.

Marsiglia

Durante questo seminario, Gerardo Marsiglia, application architect, ci ha descritto il ciclo di vita tipico di un'applicazione software.

Organizzazione software

Un'organizzazione software si basa sulle seguenti architetture: tecnologica, applicativa, architettura dei dati, architettura del business. In generale, il ciclo di vita di un'applicazione software prevedere le seguenti fasi:

- Piano del progetto;
- Analisi requisiti;
- Scrittura del codice;
- Effettuare i test;
- Distribuzione del software;
- Mantenimento.

Durante il seminario sono state trattate le problematiche che normalmente vengono fuori durante il processo di sviluppo: gestione dei vari team, tenere traccia delle attività svolte, innumerevoli tecnologie da cui scegliere (che poi andranno necessariamente integrate tra loro) e comunicazione tra i membri del team. Alcune di queste problematiche, seppur in modo ridotto, in effetti si sono verificate anche nel nostro team, in quanto non sempre potevamo incontrarci tutti e tre assieme. Mentre per quanto riguarda la comunicazione e il controllo dei vari task eseguiti, sono andati sempre a buon fine. Risulta comunque necessario creare degli standard architetturali per la gestione di determinate situazioni che possono crearsi.

Abbiamo poi trattato i vari metodi, dai più tradizionali (waterfall) a quelli piu' recenti, quindi l'Agile, di cui il Kanban in particolare, per poi passare al DevOps. Quest'ultimo costituisce un'evoluzione dell'agile, rende piu' efficiente l'intero processo che va dalla raccolta dei requisiti alla distribuzione dell'applicazione.

Oltre a varie caratteristiche del metodo DevOps, quali la sicurezza, l'automazione dei processi e l'utilizzo di pipeline per ottenere una produzione continua, si e' parlato anche di Cloud e, soprattutto, della modernizzazione delle applicazioni, necessaria per ottenere i vantaggi delle nuove tecnologie software.

Considerazioni finali

Come gruppo ci siamo ritrovati molto nel piano descritto da Marsiglia, sia per l'analisi dei requisiti che per i testi e la distribuzione dell'applicazione. Per quanto riguarda la metodologia da noi usata esplicitamente, essa si riconduce all'Agile; in caso di miglioramenti futuri alla web app prenderemo senz'altro in considerazione la metodologia DevOps e il mantenimento, come descritto da Marsiglia.

APSS

Durante questo seminario gli ingegneri Alessandro Bazziga ed Alessandro Zorer hanno presentato una realta' pubblica quale l'APSS, ossia l'Azienda Provinciale per i Servizi Sanitari, che coordina tutte le attivita' sanitarie in Trentino.
Offre lavoro a centinaia di persone, e dispone di circa 24000 attrezzature sanitarie; e' stato interessante, ed inaspettato a nostro avviso, vedere la grande mole di dati con cui un'azienda del genere deve lavorare, infatti tabelle con elevati valori numerici, ad esempio riguardo al numero ed ai costi delle apparecchiature usate, hanno accompagnato l'intera presentazione.
Questo seminario e' stato l'ultimo dei dieci che abbiamo affrontato, ed anche qui e' stata ulteriormente ribadita l'importanza dell'ingegneria del software per la gestione e lo sviluppo di applicazioni, specialmente in una grande azienda come questa. Nello specifico, l'azienda APSS lavora continuamente per il mantenimento dei propri servizi software e per la migrazione dei servizi verso il cloud. Nonostante ciò, siamo rimasti abbastanza sorpresi quando nello schema logico e strutturale dell'azienda erano presenti ancora sistemi legacy, tema, anche questo, affrontato in uno dei seminari precedenti. Inoltre i due presentatori hanno rimarcato più volte l'importanza dell'intelligenza artificiale in una realta' cosi' grande quale e' APSS, che analizza una mole di dati enorme ogni giorno.
Concludiamo, osservando che in piu' seminari del corso e' stata citata l'intelligenza artificiale come strumento alla base delle attivita' dell'azienda, dunque anche un suo coinvolgimento in FitCal e' da tenere in considerazione, per esempio per implementare un chatBot per assistenza virtuale e info giornaliere per uno stile di vita sano.