MatchPoint

Project Plan V. 1.0

Sommario

1. Introduzione

L'obiettivo del progetto "MatchPoint" è sviluppare un software per la prenotazione e la gestione dei campi da gioco nei centri sportivi in Italia. Ciò nasce dall'esigenza di gruppi di amici, squadre e appassionati di sport che sono in difficoltà a trovare campi e terreni di gioco vicini a dove si trovano e a prezzi accessibili.
I gestori dei centri sportivi possono registrarsi ed inserire i campi che hanno a disposizione (sport, dimensioni, disponibilità oraria, costi e servizi), mentre gli utenti base (giocatori) possono prenotare i campi a fasce orarie e gli eventuali servizi abbinati.
Il progetto sarà sviluppato da Filippo Bonfanti, Matteo Colombi e Luca Rossi nell'ambito del corso di Ingegneria del software (A.A. 2024/2025 - Proff. Gargantini e Bonfanti).

2. Modello di processo

Per questo progetto si è deciso di utilizzare un processo di tipo SCRUM; poiché non è possibile individuare un product owner e uno SCRUM master, questi due ruoli sono visti come "ipotetici" e le loro richieste vengono immaginate dal team di sviluppo.
Scrum è un metodo Agile che gestisce progetti in contesti difficili da pianificare in anticipo. Esso adotta un approccio incrementale e iterativo, focalizzandosi principalmente su ispezione, trasparenza e adattamento per migliorare la prevedibilità e la gestione del rischio.
Il progetto verrà suddiviso in brevi cicli di lavoro chiamati Sprint (dalla durata di 1-4 settimane), ciascuno dei quali produrrà un tassello in più per il miglioramento del software. Il team Scrum include il Product Owner (rappresentante degli stakeholder), il Development Team (nel nostro caso autonomo) e lo Scrum Master (facilita il team e rimuove ostacoli).
Il Product Owner crea e gestisce il product backlog, ovvero l'insieme degli obiettivi che ci si è prefissati per completare il prodotto. Gli Sprint sono pianificati all'inizio, fissando obiettivi e tempi. Durante uno Sprint gli obiettivi non cambiano, e ogni giorno il team si riunisce nel Daily Scrum, una riunione organizzativa che decide il da farsi della giornata. Alla fine di ogni Sprint si svolgono la Sprint Review (verifica di cosa si ha ottenuto in giornata) e la Sprint Retrospective. Il ciclo si ripete fino a completare il backlog, esaurire il budget o completare il prodotto.
Le milestones di questo progetto sono le seguenti:

- **Analisi dei requisiti**
  Definizione dettagliata delle funzionalità, con focus su esigenze degli utenti e requisiti dei gestori dei centri sportivi.
- **Design dell'architettura**
  Progettazione dell'architettura software, compresa la definizione del database e delle interfacce principali.
- **Sviluppo delle funzionalità complete**
  Implementazione delle funzionalità: login, registrazione, modifiche, prenotazioni, gestione degli orari, personalizzazione dei campi e pagamento.
- **Testing finale e miglioramento**
  Test completi per garantire che l'app soddisfi i requisiti di qualità e sia pronta per la distribuzione.
- **Rilascio della versione 1.0**
  Lancio ufficiale dell'applicazione.

I percorsi critici sono la gestione della base di dati e del login oltre che la parte grafica del progetto.

3. Organizzazione del progetto

Il team di sviluppo è composto da Filippo Bonfanti, Matteo Colombi e Luca Rossi, i membri collaboreranno attivamente a tutte le parti del progetto, in particolare ogni membro sarà responsabile di alcune parti di esso:

- **Filippo Bonfanti**: codice e backend
- **Matteo Colombi**: documentazione, UML e codice
- **Luca Rossi**: codice e frontend
  In ogni caso tutti i membri hanno partecipato attivamente ad ogni fase dello sviluppo del progetto.

4. Norme, linee guida e procedure

La scrittura della documentazione è un'attività che si svolge dividendo il lavoro in parti uguali e assegnando a ciascun sviluppatore del team una parte; una volta conclusa la fase di scrittura viene fatta una revisione completa da parte di tutti i membri, in modo da segnalare eventuali errori o imprecisioni.
Inoltre, la documentazione viene aggiornata ogni qual volta vi sia un cambiamento, segnando anche la creazione di un'eventuale nuova versione, in modo tale che tutto rimanga tracciato e in caso di errori sarà sempre possibile tornare a versioni precedenti.
In particolare, il rapporto tra la fase di modellazione e di design sarà governato dalla Model Driven Architecture (MDA): generalmente, negli approcci tradizionali si realizza un modello e questo viene tradotto in codice, qualsiasi evoluzione viene fatta cambiando il codice ma non il modello; così facendo, il modello diventa rapidamente obsoleto. MDA invece pensa ad aggiornare di volta in volta il modello e da questo si arriva al codice. Di conseguenza la manutenzione va eseguita prima sul modello.

5. Attività di gestione

Le attività di gestione del progetto sono guidate da obiettivi specifici e priorità stabilite per garantire il rispetto dei requisiti, dei tempi e dei costi pianificati. In primo luogo, bisogna gestire il monitoraggio e reporting: il team si impegna a presentare relazioni periodiche sullo stato di avanzamento del progetto. Questi rapporti, redatti tendenzialmente alla fine di ogni Sprint, forniranno un quadro complessivo dei progressi, delle sfide incontrate e dei risultati raggiunti rispetto agli obiettivi prefissati. La presenza regolare di questi report permetterà al team di tenere sotto controllo eventuali ritardi o scostamenti rispetto al piano originale, facilitando interventi correttivi tempestivi. Un'altra attività molto importante è il bilanciamento dei requisiti: durante lo sviluppo, sarà necessario bilanciare i requisiti funzionali e non funzionali del sistema. Ciò comporta la gestione delle priorità assegnate a ciascun requisito, in base al modello MoSCoW e la verifica continua che le funzionalità chiave siano sviluppate e testate secondo le aspettative degli utenti. Eventuali modifiche ai requisiti saranno valutate e inserite nei successivi Sprint solo se coerenti con gli obiettivi e le risorse disponibili. Il modello MoSCoW segue i seguenti criteri:

- **Must haves**: requisiti prioritari, si devono realizzare per rendere il sistema accettabile.
- **Should haves**: requisiti non obbligatori, ma altamente desiderabili.
- **Could haves**: requisiti che, se il tempo lo consente, verranno realizzati.
- **Won't haves**: non saranno realizzati, ma sono registrati. Essi potranno essere presi in considerazione in futuro.

La gestione dei tempi è sicuramente un aspetto fondamentale per la buona riuscita del progetto. Il rispetto delle scadenze è fondamentale per concludere il progetto nei limiti stabiliti. Durante le riunioni di Sprint Planning, il team definirà gli obiettivi da raggiungere entro il termine di ogni Sprint, pianificando dettagliatamente le attività necessarie e stimando accuratamente i tempi. In caso di ritardi imprevisti, i membri rivedranno il piano per riadattare le priorità e garantire che le funzionalità essenziali siano completate entro le scadenze previste.
Un altro aspetto su cui il team si focalizzerà molto è la gestione della qualità, che verrà meglio approfondita nel punto 9.
Infine è fondamentale mantenere un'ottima comunicazione interna. Una comunicazione efficace tra i membri del team è essenziale per una gestione corretta del progetto. Saranno organizzate riunioni regolari, come il Daily Scrum, per aggiornamenti giornalieri, e revisioni più estese per fare il punto della situazione al termine di ogni Sprint. In caso di problemi significativi o modifiche ai requisiti, si dovrà aprire un issue su GitHub, in modo da informare tutti gli altri membri del team, assicurando che ogni decisione sia condivisa e compresa.

6. Rischi

Durante il progetto "MatchPoint" potrebbero emergere diversi rischi che dovranno essere identificati il prima possibile per minimizzare gli impatti negativi che potrebbero apportare. I principali rischi includono:

- Mancanza di informazioni critiche: potrebbero verificarsi situazioni in cui informazioni essenziali per l'analisi dei requisiti o per il testing risultano incomplete o mancanti, aumentando il rischio di non soddisfare le esigenze degli utenti.
- Problemi tecnici imprevisti: bug critici, fallimenti di integrazione o incompatibilità con sistemi esterni possono rallentare il processo di sviluppo.

Per affrontare questi rischi, si pianificano azioni di mitigazione, come la creazione di un calendario con scadenze anticipate per le consegne, il monitoraggio continuo delle risorse disponibili e l'aggiornamento costante della documentazione. Inoltre, si terranno sessioni di revisione periodiche per identificare tempestivamente eventuali problematiche.

7. Personale

Il progetto richiederà personale con competenze differenti in base alla fase in corso. Le risorse sono distribuite come segue:

- Analisi dei requisiti e design: questa fase richiede competenze in ingegneria dei requisiti, analisi di sistema e progettazione software. Saranno impiegati membri del team con esperienza nell'analisi e design.
- Sviluppo delle funzionalità complete: necessità di sviluppatori con competenze avanzate in linguaggi di programmazione (Java) e di gestione di databases (DB Browser). Durante questa fase, tutti i membri saranno attivi, ciascuno con responsabilità specifiche su moduli e funzionalità.
- Testing e qualità: richiede competenze in testing manuale e automatizzato, e tecniche di quality assurance. Un membro del team sarà assegnato alla supervisione della qualità del software, mentre gli altri supporteranno i test periodici e le revisioni del codice.

- Rilascio e manutenzione: include competenze di deployment e gestione di sistema per il rilascio dell'applicazione. Saranno necessari membri del team per configurare e monitorare il lancio del software.

Ogni fase sarà eseguita garantendo una collaborazione fluida tra i membri del team e sfruttando competenze complementari per ottimizzare i tempi e la qualità complessiva del progetto.

8. Metodi e tecniche

Durante l'**ingegneria dei Requisiti**, verranno utilizzati metodi di raccolta e analisi dei requisiti come interviste, questionari e brainstorming. I diagrammi UML (Unified Modeling Language) saranno impiegati per descrivere funzionalità e flussi di lavoro del sistema. La documentazione prodotta in questa fase verrà utilizzata come base per le successive attività di progettazione.

Per la **progettazione**, verranno utilizzate tecniche di modellazione UML, per definire le interazioni tra i componenti. Verranno inoltre applicate tecniche di progettazione modulare per facilitare l'integrazione e la manutenzione del sistema.

Durante l'**implementazione**, saranno utilizzati metodi di sviluppo incrementale per consentire il rilascio di versioni intermedie, fornendo ai team di test componenti già parzialmente funzionanti. Ogni rilascio sarà accompagnato da documentazione di codice, incluse annotazioni dettagliate e informazioni sull'architettura adottata. Per la gestione della versione, verrà utilizzato GitHub, un sistema di controllo della versione distribuito, che consentirà di tracciare le modifiche al codice, gestire branch e versioni, e facilitare il lavoro collaborativo.

La fase di **test** seguirà una metodologia sia di testing unitario che di integrazione, usando strumenti automatizzati quando e dove possibile per garantire una verifica rapida e affidabile del codice. Verranno effettuati anche test a ogni aggiornamento per assicurarsi che le nuove funzionalità non compromettano quelle esistenti. Tutti i casi di test e i risultati saranno documentati per tenere traccia delle funzionalità testate e dei bug riscontrati.

Per garantire **coerenza** e **tracciabilità** durante lo sviluppo, verrà implementato un rigoroso sistema di controllo di versione tramite GitHub, con pratiche di commit regolari e messaggi descrittivi e issues.

La **documentazione** sarà prodotta e mantenuta durante tutto il ciclo di vita del progetto, in modo che rimanga aggiornata e rifletta le modifiche apportate in ogni fase. Verrà adottato uno standard di documentazione coerente, e il processo di revisione sarà pianificato per garantire che le specifiche tecniche siano accuratamente descritte e facilmente consultabili.

9. Garanzie di qualità

Per assicurare la qualità del software sviluppato in MatchPoint, verranno implementati controlli sistematici e test periodici, con l'obiettivo di mantenere il processo di sviluppo in linea con gli standard di qualità stabiliti.
Un membro del team si occuperà del Quality Assurance (QA) e sarà incaricato di definire e coordinare i test e le verifiche periodiche durante lo sviluppo. L'intero team sarà comunque responsabile di scrivere codice conforme alle linee guida e convenzioni stabilite, segnalando prontamente eventuali anomalie o incongruenze. Ogni membro del team contribuirà alle attività di testing manuale e automatizzato per verificare che il software rispetti i requisiti di qualità.
Inoltre, verranno rispettati gli standard di usabilità e accessibilità per garantire che l'interfaccia sia semplice e intuitiva per gli utenti finali (giocatori e gestori) e che i tempi di risposta dell'applicazione siano adeguati. Saranno definiti obiettivi specifici come i tassi di difetti accettabili e l'uso del software su diversi dispositivi.

Code review periodiche verranno eseguite per individuare e correggere errori di sintassi e altri bug, mentre test unitari e test di integrazione saranno eseguiti per ogni modulo sviluppato, in modo da verificare l'interazione corretta tra i vari componenti, sia individualmente che integrati nel sistema complessivo.
Alla conclusione dello sviluppo, verrà eseguita una fase finale di testing per assicurare il corretto funzionamento dell'applicazione. Infine, la documentazione sarà completata e consolidata, incorporando tutte le informazioni raccolte e aggiornate durante il processo di sviluppo.

10. Pacchetti di lavoro

Il progetto è stato suddiviso in alcuni pacchetti di lavoro come segue:

1. **Stesura del project plan e gestione del progetto**
2. **Analisi dei requisiti**
3. **Design del software**
4. **Implementazione**
   i. Sviluppo classi
   ii. Sviluppo DB
   iii. Sviluppo backend
   iv. Sviluppo GUI
5. **Testing**
6. **Manutenzione**

Per ogni pacchetto di lavoro saranno previsti uno o più sprint della durata di 1 o 2 settimane; prima di ogni sprint, la squadra decide lo sprint backlog (obiettivi da raggiungere al termine dello sprint) e al termine dello sprint effettua un'analisi dei risultati raggiunti (sprint retrospective). Al termine di ogni sprint i risultati vengono aggiunti al product backlog.

11. Risorse

Per quanto riguarda l'aspetto delle risorse umane al progetto lavorano i tre studenti precedentemente citati, che si sono distribuiti in maniera equa il lavoro da svolgere.

Le risorse tecnologiche utilizzate sono le seguenti:

- Eclipse per la parte di sviluppo del codice
- DB Browser (SQLite) per il DataBase
- Java Swing e SwingX per la parte di GUI
- Papyrus per la modellazione e i grafici UML
- GitHub per il versioning e la gestione collaborativa del repository
- Junit per il testing

Vengono inoltre impiegate delle risorse materiali, ovvero i portatili dei tre studenti ed i loro tablet, oltre che i testi del corso di ingegneria del software (software engineering e UML@Classroom)

Le risorse economiche non rientrano in questo project plan in quanto vengono utilizzate piattaforme e strumenti open source o comunque gratuite e il progetto è a scopo accademico.

12. Budget e programma

Stabilire un budget studiato (stima dei costi) è fondamentale per il corretto sviluppo del software e di conseguenza, dell'applicazione. Nel nostro caso, l'utilizzo di software open source ci permette di avere costi pari a zero. In particolare utilizzeremo: Eclipse (per la scrittura del codice e per la GUI), DB Browser (per il database), Papyrus (per la modellazione di grafici UML), Github per la comunicazione tra il team ed il versioning.

La stima dei tempi necessari per il completamento delle singole attività (e di conseguenza anche dello sviluppo dell'applicazione totale) varia in base alla metodologia applicata, in questo caso SCRUM. Essendo un metodo agile, la capacità di stimare i tempi delle attività è strettamente influenzata dalla stabilità del sistema, ovvero dalla capacità del gruppo di lavoro di garantire livelli di servizio stabili nel tempo. La stabilità del sistema è definita da un parametro, la stabilità metrica Ψ. Quest'ultima deriva dal rapporto tra il tasso di servizio λ (ovvero il numero di prodotti backlog correttamente creati dal team in un periodo di tempo T) e il tasso di arrivo (ovvero il numero di prodotti backlog correttamente aggiunti al sistema in un periodo di tempo T).

Per quanto riguarda la stima dei tempi delle attività da svolgere utilizziamo una Kanban Board condivisa con tutti i membri disponibile su Github; la board è divisa in To Do (attività da svolgere), in progress (attività in corso di svolgimento), in revision (in revisione) e done (attività terminata).

13. Cambiamenti

In Agile, i diversi cambiamenti che possono complicare lo sviluppo di un progetto sono previsti e accolti con favore. Pratiche agili come la collaborazione frequente, l'integrazione continua e le revisioni regolari si allineano bene con qualsiasi processo di gestione del cambiamento, facilitando transizioni fluide e riducendo la resistenza. Affinchè sia implementato un cambiamento sono fondamentali:

- Comunicazione efficace tra le parti interessate
- Coinvolgimento delle parti interessate anche nel processo decisionale
- Valutazione dell'impatto del cambiamento (con conseguenti stime di tempo e sforzo)

14. Consegna

Il team effettua la consegna tramite l'utilizzo dello strumento GITHub. Il professore sarà in grado di ottenere la versione più recente del progetto ma allo stesso tempo di risalire a versioni, issue, branch e commenti precedenti.