Piano di Progetto AMDD (Release Plan)

Roadmap di Sviluppo e Dettaglio Iterazioni

Team SPS-F1

Andrea Birolini (mat. 1087070)
Ivan Caccamo (mat. 1085892)
Luca Rossi (mat. 1086223)

23 Novembre 2025

Indice

1 Ruoli del Team e Responsabilità

Per garantire un flusso di lavoro efficiente e specializzato, i membri del team hanno assunto ruoli verticali basati sulle proprie competenze tecniche, pur mantenendo una visione orizzontale sull'intero ciclo di vita del progetto.

- **Andrea Birolini (Architect & System Integrator)**
  - Progettazione dell'architettura a microservizi (Backend Java + Python Service).
  - Definizione dei contratti di interfaccia (API REST/JSON).
  - Gestione del ciclo di vita dei container Docker e integrazione dei sottosistemi.
  - Modellazione UML (Diagrammi di Deployment, Componenti e Sequenza).
- **Ivan Caccamo (Algorithm Engineer)**
  - Responsabile del "Design in Piccolo" e del core computazionale.
  - Analisi matematica del problema di ottimizzazione strategica.
  - Implementazione e tuning dell'algoritmo di Programmazione Dinamica (Java).
  - Analisi teorica della complessità temporale e spaziale degli algoritmi.
- **Luca Rossi (Data Scientist & ML Engineer)**
  - Gestione della pipeline dei dati: acquisizione, pulizia e normalizzazione del dataset Kaggle/FastF1.
  - Sviluppo, training e validazione dei modelli di Machine Learning (Random Forest).
  - Implementazione del microservizio predittivo in Python (Flask).
  - Analisi statistica delle performance del modello (MSE, R2 Score).

2 Iterazione 0: Envisioning & Setup

Periodo: 24 Novembre 2025 - 30 Novembre 2025
Obiettivo Principale: Definire il perimetro del progetto, concordare i requisiti con la committenza (Docente) e predisporre l'infrastruttura tecnica per lo sviluppo collaborativo.

2.1 Project Management e Infrastruttura

In questa fase vengono gettate le basi organizzative per il lavoro di squadra.

- **Setup Repository GitHub:** Inizializzazione del repository `SPS-F1` con struttura a monorepo (`/backend-java`, `/ml-python`, `/docs`). Configurazione del file `.gitignore` per escludere artefatti di build e dataset pesanti.
- **Configurazione Ambienti di Sviluppo:**

  - *Java:* Configurazione Maven Wrapper e JDK 17 su tutte le macchine di sviluppo.
  - *Python:* Creazione script di setup per `virtualenv` e file `requirements.txt` iniziale.

- **Acquisizione Dati:** Download preliminare del dataset "F1 World Championship" (fonti Kaggle e FastF1 API) relativo al triennio 2021-2024 per analisi di fattibilità.
- **Gestione Task:** Creazione della Kanban Board di progetto (es. GitHub Projects o Trello) con colonne: *Backlog, To Do, In Progress, Done.*

2.2 Analisi dei Requisiti e Design Architetturale

Attività focalizzate sulla formalizzazione di ciò che il sistema deve fare e come sarà strutturato.

- **Stesura Documento di Visione:** Definizione degli obiettivi ad alto livello, del target di utenza (Ingegneri di pista) e del valore aggiunto del sistema rispetto ai competitor statici.
- **Ingegneria dei Requisiti:**
  - Identificazione degli Attori (Stratega, Admin).
  - Elicitazione dei Requisiti Funzionali (es. "Il sistema deve predire il degrado gomma").
  - Definizione dei Requisiti Non Funzionali (Performance $<$ 3s, Affidabilità).
- **Modellazione UML Preliminare:**
  - *Use Case Diagram:* Per mappare le interazioni utente-sistema.
  - *Deployment Diagram (Bozza):* Per identificare i nodi fisici (Server Java, Server Python, Database).

2.3 Milestone di Rilascio (`v0.1-envisioning`)

Al termine dell'iterazione, vengono prodotti i seguenti artefatti:

- Documento "Analisi dei Requisiti" approvato.
- Repository configurato e accessibile a tutto il team.
- Dataset grezzo disponibile per l'esplorazione.

3 Iterazione 1: Data Engineering & Machine Learning

Periodo: 1 Dicembre 2025 - 14 Dicembre 2025
Obiettivo Principale: Costruire la base di conoscenza del sistema trasformando i dati grezzi in insight predittivi. Il focus è lo sviluppo del sottosistema Python, dal preprocessing dei dati fino all'esposizione del modello ML tramite API REST.

3.1 Analisi Dati e Design delle API

Prima della codifica, viene effettuata un'analisi approfondita per comprendere la natura fisica del problema.

- **Exploratory Data Analysis (EDA):** Analisi statistica delle distribuzioni dei tempi sul giro e del degrado pneumatici utilizzando notebook Jupyter e librerie di visualizzazione (Matplotlib/Seaborn). Identificazione di outlier e correlazioni tra temperatura asfalto e usura.
- **Definizione Contratto API (Interface Contract):** Progettazione dello schema JSON per la comunicazione tra Backend Java e Servizio ML.
  - *Input:* Specifica dei campi obbligatori (`circuit_id`, `air_temp`, `track_temp`, `compounds_list`).
  - *Output:* Struttura della risposta contenente, per ogni mescola, il tempo base (T_{base}) e il coefficiente di degrado (D_{rate}).

3.2 Sviluppo Modulo Python (Data Science)

Implementazione della pipeline di Machine Learning e del microservizio.

- **Setup Ambiente:** Configurazione isolata tramite `virtualenv` e installazione delle dipendenze scientifiche (`pandas`, `scikit-learn`, `flask`, `joblib`).
- **Data Cleaning e Feature Engineering:** Sviluppo di script per la pulizia dei dati (gestione valori nulli, normalizzazione) e trasformazione delle variabili categoriche (es. tipo mescola, nome circuito) tramite *One-Hot Encoding*.
- **Training del Modello:** Addestramento di algoritmi di regressione (con focus su *Random Forest Regressor*) per apprendere le curve di degrado. Tuning degli iperparametri per minimizzare l'errore.
- **Sviluppo Microservizio API:** Implementazione del server Flask/FastAPI. Creazione dell'endpoint `POST /predict` che riceve i dati di gara, deserializza il modello pre-addestrato (`.pkl`) ed esegue l'inferenza in tempo reale.

3.3 Testing e Controllo Qualità

- **Validazione Modello ML:** Calcolo delle metriche di performance (MSE - Mean Squared Error, R2 Score) su un set di test riservato (Hold-out validation) per garantire che le predizioni siano statisticamente affidabili.
- **Functional API Testing:** Verifica del comportamento dell'endpoint REST tramite **Postman**. Test di casi positivi (input corretto) e negativi (dati mancanti o fuori range) per assicurare la robustezza del servizio.

3.4 Milestone di Rilascio (`v0.2-ml-service`)

- Microservizio Python dockerizzabile e funzionante.
- Modello predittivo serializzato e validato.
- Documentazione tecnica delle API (formato OpenAPI/Swagger o PDF).

4 Iterazione 2: Core Algoritmico & Integrazione

Periodo: 15 Dicembre 2025 - 28 Dicembre 2025
Obiettivo Principale: Sviluppare il cuore computazionale del sistema (il "Design in Piccolo") e realizzare l'integrazione completa tra il backend Java e il servizio ML.

4.1 Design in Piccolo e Modellazione

Fase concettuale dedicata alla risoluzione del problema matematico.

- **Modellazione Matematica:** Formalizzazione del problema della strategia di gara come ricerca del cammino minimo su un Grafo Diretto Aiciclico (DAG) pesato.
- **Progettazione Algoritmo:** Stesura dello pseudocodice per l'algoritmo di *Programmazione Dinamica* (DP) con approccio *Memoization*, per esplorare efficientemente lo spazio delle soluzioni.
- **Analisi della Complessità:** Studio teorico della complessità temporale e spaziale ($O(N \cdot 2^K)$) per garantire il rispetto del requisito non funzionale di performance ($<$ 3 secondi).

4.2 Sviluppo Backend (Java Spring Boot)

Implementazione della logica di business e integrazione dei sistemi.

- **Domain Layer:** Implementazione delle classi POJO fondamentali: `Race`, `Strategy`, `Stint`, `Tyre`.
- **Integration Layer (Service Client):** Sviluppo della classe `PredictionService` responsabile della comunicazione HTTP con il modulo Python. Implementazione del pattern *Adapter* per convertire i JSON esterni in oggetti di dominio Java.
- **Logic Layer (Optimization Engine):** Traduzione dello pseudocodice DP in codice Java ottimizzato. Gestione dei vincoli di gara (es. obbligo cambio mescola).
- **API Layer (Controller):** Esposizione dell'endpoint principale `GET /api/strategy` che orchestra la chiamata al servizio ML e l'esecuzione dell'algoritmo.

4.3 Testing e Validazione

- **Unit Testing (JUnit):** Creazione di suite di test per verificare la correttezza logica dell'algoritmo (es. verificare che per una gara di 1 giro venga scelta la gomma più morbida).
- **Integration Testing:** Verifica end-to-end del flusso dati: Java invia richiesta $\rightarrow$ Python elabora $\rightarrow$ Java riceve risposta $\rightarrow$ Algoritmo calcola.

4.4 Milestone di Rilascio (`v0.5-backend-complete`)

- Backend Java completo e integrato con dati reali.
- Capitolo della documentazione "Algoritmi e Strutture Dati" completato.

5 Iterazione 3: Finale - Frontend & Documentazione

Periodo: 29 Dicembre 2025 - 11 Gennaio 2026
Obiettivo Principale: Completare l'esperienza utente (GUI), elevare la qualità del codice tramite analisi statica e finalizzare l'intero pacchetto documentale per la consegna.

5.1 Sviluppo Frontend e Visualizzazione

Creazione dell'interfaccia utente per rendere il sistema accessibile allo Stratega.

- **Implementazione GUI:** Sviluppo di pagine Web responsive (HTML5/Bootstrap) per la configurazione dei parametri di gara (slider temperature, selezione circuito).
- **Integrazione Logica Client-Side:** Utilizzo di JavaScript (Fetch API) per chiamare asincronamente il backend Java e gestire il rendering dinamico dei risultati senza ricaricare la pagina.
- **Data Visualization:** Integrazione della libreria *Chart.js* per generare grafici interattivi che confrontano la telemetria delle strategie proposte (es. curva del degrado giro per giro).

5.2 Quality Assurance (QA) e Refactoring

Attività mirate a garantire la manutenibilità e la pulizia del codice.

- **Analisi Statica:** Scansione del codice sorgente tramite tool come **STAN4J** o **SonarLint** per identificare "Code Smells", dipendenze cicliche e violazioni delle convenzioni di stile.
- **Refactoring:** Ottimizzazione del codice basata sulle metriche raccolte (es. riduzione della Complessità Ciclomatica, estrazione di costanti, miglioramento dei nomi delle variabili).

5.3 Chiusura Documentazione e Rilascio

Assemblaggio finale di tutti gli artefatti prodotti.

- **Master Document:** Unione dei vari capitoli (Analisi, Architettura, Algoritmi, Test) in un unico PDF coerente ("Relazione Tecnica di Progetto").
- **Manuale Utente:** Stesura di una guida rapida con screenshot per l'installazione (requisiti, comandi di avvio) e l'utilizzo dell'interfaccia.
- **Presentazione:** Preparazione delle slide di sintesi per la discussione orale dell'esame.

5.4 Milestone di Rilascio (`v1.0-release`)

- Codice sorgente completo e commentato.
- Documentazione PDF definitiva.
- Pacchetto pronto per la consegna (zip o link repo).

A Struttura della Documentazione Finale

Il documento PDF finale (Output di Progetto) sarà organizzato nei seguenti capitoli, coprendo l'intero ciclo di vita del software:

1. **Introduzione e Visione:** Contesto del problema (F1), obiettivi del progetto e metodologia adottata.
2. **Analisi dei Requisiti:** Dettaglio degli attori, requisiti funzionali/non funzionali e Diagramma dei Casi d'Uso.
3. **Architettura del Sistema:** Modello 4+1 viste (Deployment Diagram, Class Diagram, Sequence Diagram) e descrizione dei pattern utilizzati.
4. **Algoritmi e Strutture Dati:** Approfondimento tecnico sul "Design in Piccolo", pseudocodice dell'algoritmo DP e analisi della complessità.
5. **Quality Assurance:** Report delle attività di testing (Unit Test, Integration Test) e metriche di qualità del codice (Analisi Statica).
6. **Manuale Utente:** Guida all'installazione e all'uso con riferimenti visivi.