2.1 Software Life Cycle

Per lo sviluppo del progetto che si poneva l'obiettivo di creare un'applicazione di messaggistica per la Comunicazione Aumentativa Alternativa (CAA), il team ha adottato un modello di processo **Iterativo e Incrementale** ispirato alla metodologia **Agile Scrum**. Questa scelta ci ha permesso di gestire la complessità dello sviluppo mobile e di adattare i requisiti in corso d'opera. Lo sviluppo è stato suddiviso in più fasi denominate **Sprint**, ognuno dei quali aveva finalità precise e coerenti con l'avanzamento del progetto.

Pianificazione Temporale degli Sprint Dato l'utilizzo di un framework (Flutter/Dart) alternativo a quello proposto nel corso, la pianificazione ha previsto una fase iniziale estesa di studio e prototipazione, seguita da sprint di sviluppo intensivo.
Il programma effettivo è stato il seguente:

- **Sprint 1: Formazione Tecnologica e Configurazione (dal 06/11/2025 al 10/12/2025)**
    - Focus: Acquisizione competenze su Dart/Flutter, configurazione IDE Android Studio, setup del repository e definizione dell'architettura a widget.
    - Nota: Questo sprint ha avuto una durata di circa 5 settimane per seguire la curva di apprendimento necessaria per padroneggiare la nuova tecnologia prima di scrivere codice di produzione.
- **Sprint 2: Funzionalità Core e Primo Prototipo (dal 11/12/2025 al 28/12/2025)**
    - Focus: Realizzazione del primo prodotto con funzionalità minime richieste. Implementazione della struttura di navigazione, creazione delle prime schermate statiche e impostazione del collegamento a Firebase.
    - Stato: Rilascio della prima build interna funzionante prima della pausa accademica.
- **Attività di sviluppo sospesa (Pausa Invernale)**
    - Periodo: Dal 29/12/2025 al 06/01/2026 - Attività di sviluppo sospesa.
- **Sprint 3: Sviluppo Intensivo e Integrazione (dal 07/01/2026 al 21/01/2026)**
    - Focus: Sviluppo della maggior parte delle logiche di business (es. gestione chat, logica Tutor/CCN]), perfezionamento della UI e integrazione completa dei servizi backend.
    - Note: È stato lo sprint più produttivo, in cui il team ha lavorato a tempo pieno applicando le competenze consolidate nello Sprint 1.
- **Sprint 4: Finalizzazione, Testing e Documentazione (dal 22/01/2026 al 09/02/2026)**
    - Focus: Code Refactoring, stesura dei commenti Javadoc/Dartdoc, redazione di questa documentazione tecnica (Requisiti, Design, Test) e completamento della presentazione finale.

Approccio alla Modellazione e Deviazioni Tecniche Rispetto alle indicazioni generali del corso che prevedevano l'uso di Java e la generazione automatica del codice tramite Papyrus, il progetto ha seguito un percorso alternativo concordato con la docenza:

- **Tecnologia:** L'applicazione è stata sviluppata interamente in **Dart** utilizzando il framework **Flutter** su IDE Android Studio.
- **Modellazione:** Poiché non esistono strumenti affidabili per trasformare automaticamente i grafici in codice Dart (come invece accade per Java), abbiamo utilizzato i diagrammi UML come guida di riferimento per scrivere il codice manualmente, invece di farlo generare in automatico dal computer.
    - I diagrammi UML sono stati realizzati con **PlantUML** e utilizzati come "progetto guida" per l'implementazione manuale.
    - Questo ha garantito che l'architettura del software rimanesse coerente con la progettazione, pur mantenendo la flessibilità necessaria per sfruttare le caratteristiche specifiche dei Widget di Flutter.

Gestione del Cambiamento (Variazioni rispetto al Piano) Durante il ciclo di vita, abbiamo dovuto gestire alcune deviazioni rispetto alle previsioni iniziali:

- Curva di apprendimento: L'adozione di Flutter ha richiesto un tempo di apprendimento iniziale maggiore del previsto nello Sprint 1.
- Refactoring UI: A seguito di test intermedi, abbiamo deciso di modificare la struttura di navigazione per renderla più intuitiva, un'attività non prevista inizialmente ma inserita nello Sprint 3.
- Cambio Metodologia Sviluppo: Si è deciso di utilizzare la metodologia di sviluppo Scrum piuttosto che RUP poiché date le dimensioni piccole del team e la vicinanza dei componenti, il formalismo dettato dal RUP(soprattutto sulla documentazione) non è più sembrato utile quindi si è deciso di procedere con una metodologia agile ma pur mantenendo un approccio comunque di sviluppo incrementale nei vari sprint.

2.2 Configuration Management

Il team ha utilizzato **Git** come sistema di controllo versione distribuito e **GitHub** come piattaforma di hosting remoto. Per garantire ordine e tracciabilità, abbiamo adottato le seguenti convenzioni:

- **Repository:** Unico repository centralizzato contenente sia il codice sorgente che la documentazione.
- **Ignore Files:** È stato configurato un file **.gitignore** specifico per Dart/Flutter e Android Studio per escludere file binari e configurazioni locali (**es. .idea, /build**), evitando di "inquinare" il repository.

Abbiamo adottato una strategia di **Feature Branching** semplificata, adatta alle dimensioni ridotte del team.

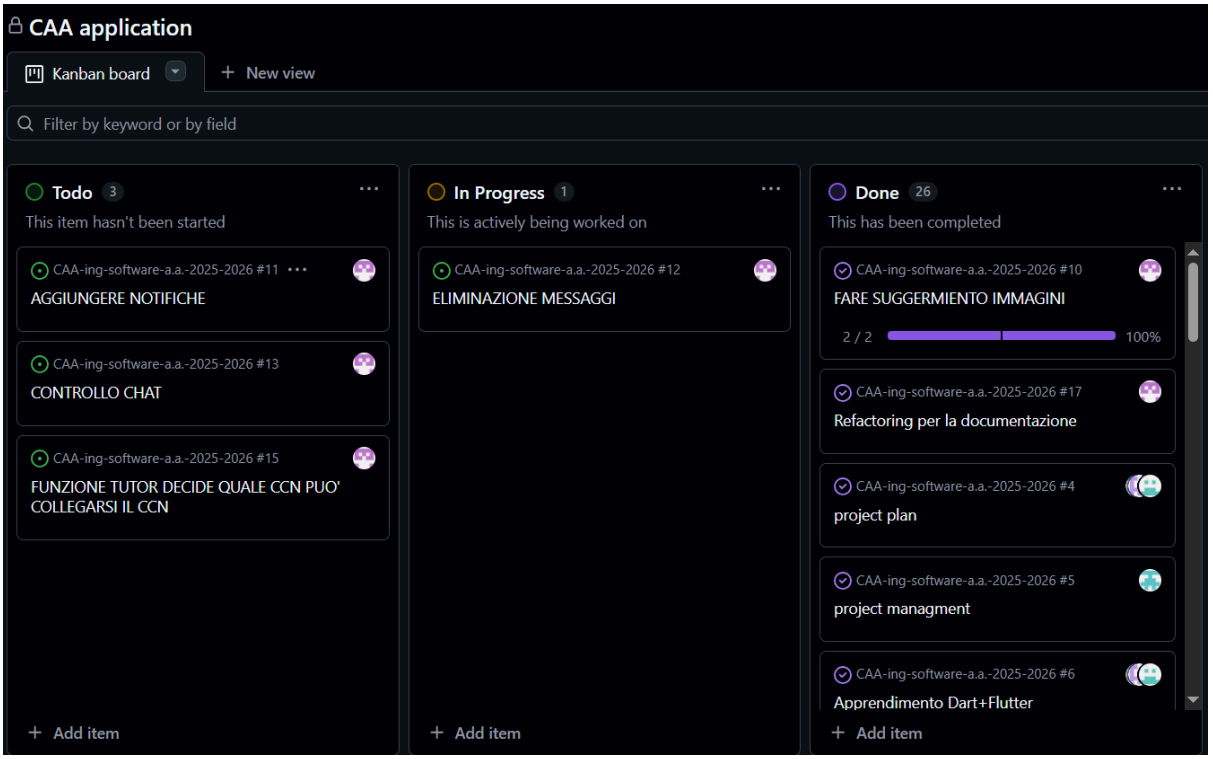
- **Main Branch:** Il ramo **main** rappresenta la **baseline** stabile del progetto. Su questo ramo viene caricato solo codice funzionante e testato.
- **Feature Branches:** Per lo sviluppo di nuove funzionalità o per la correzione di bug, sono stati creati branch temporanei derivati dal main (es. **ricerca_pittogrammi**). Questo ci ha permesso di lavorare in parallelo senza compromettere la stabilità della baseline.

Il processo di integrazione delle modifiche è stato gestito bilanciando formalità e agilità:

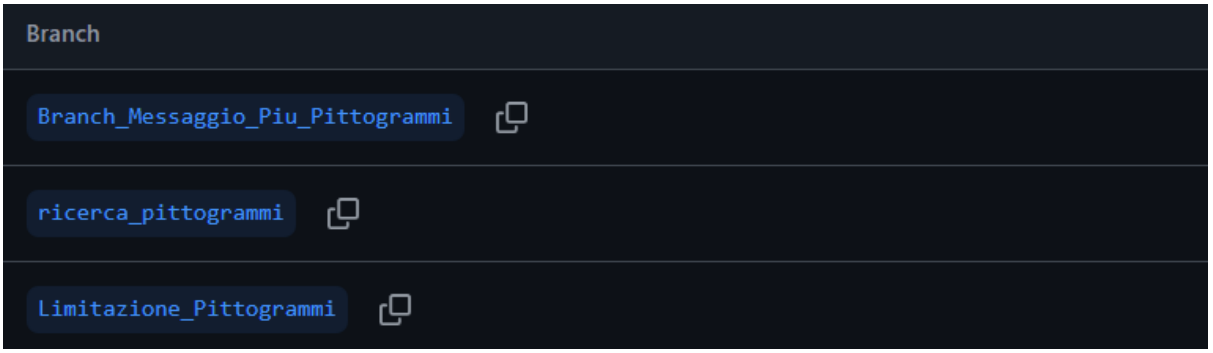
- **Sincronizzazione:** Gli sviluppatori hanno lavorato su branch separati, eseguendo commit per salvare gli stati intermedi.
- **Merge e Review:** A causa della natura strettamente collaborativa del team (spesso operante nella stessa stanza o in chiamata diretta), la fase di **Code Review** è stata svolta principalmente in modalità **sincrona**, piuttosto che attraverso il ciclo formale asincrono di apertura Issue -> Pull Request -> Commenti -> Approvazione su GitHub.
- **Giustificazione:** Sebbene GitHub offra strumenti avanzati per Issues e Pull Request, abbiamo ritenuto che per un team di queste dimensioni la comunicazione diretta fosse più efficiente. Le Pull Requests sono state utilizzate selettivamente per i merge più complessi, mentre le Issues sono state gestite principalmente tramite backlog esterno/verbale durante gli stand-up meeting.

Nota sulla tracciatura delle attività: Il team ha gestito l'assegnazione operativa dei micro-task principalmente tramite comunicazione diretta e strumenti di messaggistica istantanea (es. Whatsapp) durante gli sprint, per massimizzare la velocità di esecuzione. Le **Issue su GitHub** sono state utilizzate prevalentemente come strumento di **formalizzazione a consuntivo** o durante le fasi di Review, per garantire che il repository rappresentasse fedelmente lo stato finale del progetto e per mantenere uno storico pulito delle funzionalità implementate, senza il "rumore" della gestione giornaliera.

La Figura sottostante mostra la **Kanban Board** utilizzata dal team su GitHub.



la seguente figura invece mostra alcuni dei **branch** usati per isolare lo sviluppo delle nuove funzionalità e la risoluzione puntuale dei bug, senza compromettere la stabilità del codice principale (main).



2.3 People Management and Team Organization

Modello Organizzativo: Team Collaborativo Orizzontale
Per l'organizzazione del lavoro, il gruppo (di 4 membri) ha scelto di lavorare alla pari, senza gerarchie rigide. Abbiamo adottato una struttura collaborativa dove ogni componente ha lo stesso peso nelle decisioni e contribuisce attivamente a tutte le fasi del progetto, invece di avere un solo capo che comanda.

Questa scelta è stata motivata da:

1. **Obiettivi Didattici:** La volontà di permettere a tutti i membri di acquisire competenze sull'intero stack tecnologico (Flutter, Dart, gestione dati), evitando l'apprendimento di competenze esclusivamente verticali.
2. **Flessibilità:** In un team ristretto, la capacità di ogni membro di intervenire su qualsiasi parte del codice ha permesso di evitare colli di bottiglia nel caso di indisponibilità di un membro.

Divisione delle Responsabilità

Sebbene lo sviluppo del codice sia avvenuto in modalità "Full-Stack" con tutti i membri impegnati sia sulla UI che sulla logica, per gestire la complessità di un team di 4 persone e conformarsi ai requisiti di progetto, abbiamo assegnato delle responsabilità di coordinamento specifiche per garantire la qualità del risultato finale:

Membro del Team	Ruolo Tecnico (Primary)	Responsabilità Gestionale (Secondary)	Descrizione Attività
Fabio Mariani	Full Stack Dev	**Project Coordinator / Scrum Master**	Monitoraggio delle scadenze, organizzazione dei meeting, aggiornamento del backlog e coordinamento generale.
Luca Zucchetti	Full Stack Dev	**Repository Maintainer**	Gestione del repository GitHub, supervisione dei merge, pulizia dei branch e gestione dei conflitti, Testing.
Igli Daja	Full Stack Dev	**Quality Assurance (QA)**	Verifica della conformità ai requisiti e refactoring dei componenti software per allinearli ai modelli UML e

			garantire la qualità architetturale
Steven Paglialunga	Full Stack Dev	**Documentation Manager**	Responsabile della coerenza della documentazione, redazione dei verbali e mantenimento dei diagrammi UML aggiornati.

Gestione della Comunicazione e Decision Making

Coerentemente con il modello democratico, le decisioni architetturali sono state prese per **consenso** durante le riunioni di allineamento.

La comunicazione è avvenuta attraverso canali orizzontali aperti:

- Durante la fase di codifica, abbiamo fatto ampio uso di Pair Programming, una pratica che ha ridotto la necessità di code review formali e ha aumentato la qualità del codice alla fonte.
- Ogni membro ha avuto accesso completo a tutto il codice sorgente, garantendo che l'assenza di specializzazione verticale non si traducesse in disordine, ma in ridondanza positiva delle competenze.

Valutazione dell'Impegno

L'impegno è stato distribuito equamente. L'assenza di specializzazione tecnica rigida ha fatto sì che il carico di lavoro fosse bilanciato naturalmente: quando un task risultava più complesso del previsto, il team convergeva su di esso per completarlo, piuttosto che lasciarlo al singolo individuo.