Every line of code is worth the SWEat

Piano di Qualifica

27 marzo 2025

Uso	Esterno
Destinatari	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Sync Lab S.r.l.
	Gruppo SWE@
Responsabile	Riccardo Milan
Redattori	Andrea Precoma
	Davide Marin
	Davide Martinelli
	Davide Picello
	Klaudio Merja
Verificatori	Andrea Precoma
	Davide Martinelli
	Davide Marin
	Davide Picello
	Riccardo Milan

Registro delle modifiche

Ver.	Data	Redattori	Verificatori	Descrizione
2.0.0	27/03/2025	Davide Marin Andrea Precoma	Davide Martinelli Davide Picello	• Aggiornamento del cruscotto di monitoraggio della qualità • Rimosse metriche MPD-SC, MPD-FD, MPD-COC, MPD-SFI, MPD-SFO e MPD-RM
1.5.0	27/03/2025	Davide Martinelli Klaudio Merja	Andrea Precoma	• Aggiunta dei *test* di integrazione del *backend*
1.4.0	23/03/2025	Davide Martinelli	Riccardo Milan Andrea Precoma	• Aggiunta dei *test* di unità del *backend* • Aggiornamento stato dei *test*
1.3.0	21/03/2025	Davide Picello	Andrea Precoma	• Aggiunta dei *test* di unità del simulatore
1.2.0	10/03/2025	Davide Marin	Riccardo Milan	• Aggiunta dei *test* di accettazione
1.1.0	6/03/2025	Davide Martinelli	Davide Marin	• Stesura sezione sui *test* di sistema • Piccoli fix in accordo con le Norme di Progetto
1.0.1	05/02/2025	Klaudio Merja	Andrea Precoma	• Correzione dei riferimenti alla documentazione del gruppo
1.0.0	27/01/2025	Davide Picello	Andrea Precoma	Approvazione versione finale del documento per rilascio in RTB
0.3.0	26/01/2025	Davide Picello	Davide Martinelli Davide Marin	• Creazione grafici nel cruscotto di monitoraggio della qualità
0.2.0	10/01/2025	Davide Picello	Davide Martinelli, Davide Marin	• Definite le metriche di qualità di processo e di prodotto. Definita la struttura delle specifiche dei *test* e del cruscotto di monitoraggio della qualità.
0.1.0	07/12/2024	Davide Martinelli	Andrea Precoma, Riccardo Milan	• Struttura e introduzione del documento

Indice

Elenco delle tabelle

Elenco delle immagini

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire le strategie di verifica e validazione implementate dal gruppo al fine di garantire la qualità dei processi e del prodotto finale, guidando il *team* lungo tutta la durata del progetto secondo un'ottica di miglioramento continuo. Per tale motivo il documento ha natura mutevole ed evolutiva e verrà aggiornato periodicamente per riflettere le modifiche apportate ai processi al fine di migliorarne l'efficacia e l'efficienza. L'ultima sezione del documento (sez. 4) è dedicata all'analisi dell'andamento delle metriche presenti nel cruscotto di monitoraggio della qualità durante l'arco di svolgimento del progetto.

1.2 Scopo del prodotto

L'obiettivo principale del prodotto è quello di fornire un sistema che monitori la posizione in tempo reale di ciascun utente e in base a questa crei, sfruttando la GenAI[g], inserzioni pubblicitarie personalizzate sulla base dei suoi dati di profilazione. Il fine ultimo è quello di migliorare l'esperienza pubblicitaria degli utenti, massimizzando di conseguenza il ROI[g].

1.3 Glossario

Per chiarire il significato di alcuni termini tecnici, abbreviazioni e acronimi utilizzati all'interno della documentazione viene fornito un glossario. Nel documento i termini che, alla loro prima occorrenza, vengono contrassegnati da una sottolineatura e una «g» posta ad apice (ad esempio termine[g]) avranno una corrispettiva descrizione dettagliata all'interno del Glossario.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Norme di Progetto (v2.0.0)
  https://sweatunipd.github.io/docs/pb/norme_di_progetto_ver2.0.0.pdf
- ISO/IEC 12207:1995, §§ 6.3-6.7 (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf

1.4.2 Riferimenti informativi

- Glossario (v2.0.0)
  https://sweatunipd.github.io/docs/pb/glossario_ver2.0.0.pdf
- Capitolato C4 - Sync Lab (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C4.pdf
- Lezione T07 - Qualità di prodotto (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T07.pdf
- Lezione T08 - Qualità di processo (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T08.pdf

2 Obiettivi di qualità

In questa sezione vengono definiti gli obiettivi di qualità che il gruppo si prefigge di raggiungere nell'ambito del progetto, sia per i processi che per il prodotto, sulla base delle metriche definite nel documento Norme di Progetto.

2.1 Qualità di processo

La qualità di processo è nota essere un fattore di fondamentale importanza per qualsiasi produzione di *software* che punti all'eccellenza qualitativa. Essa, infatti, influenza con un evidente rapporto di causa-effetto la qualità del prodotto finale.

Di seguito elenchiamo gli obiettivi di qualità che il gruppo si prefigge di raggiungere nell'ambito della qualità di processo, suddivisi per le tre categorie di processi individuate dallo *standard* ISO/IEC 12207:1995 (primari, di supporto e organizzativi).

2.1.1 Processi primari

Fanno parte dei processi primari le attività di acquisizione, fornitura, sviluppo, gestione operativa e di manutenzione. Data la natura didattica del progetto ci occupiamo solo di fornitura e sviluppo.

2.1.1.1 Fornitura

Attività e compiti del fornitore, il quale dovrà accordarsi con il proponente per stabilire ufficialmente i vari vincoli e requisiti del progetto.

2.1.1.1.1 Planned value (MPC-PV)

Rappresenta il valore pianificato del lavoro da completare entro una certa data.

2.1.1.1.2 Earned value (MPC-EV)

Misura il valore del lavoro completato fino a un determinato momento rispetto al *budget* pianificato.

2.1.1.1.3 Actual cost (MPC-AC)

Indica il costo effettivamente sostenuto per il lavoro completato fino a un determinato momento.

2.1.1.1.4 Cost performance index (MPC-CPI)

Misura l'efficienza del costo per il lavoro svolto fino a un determinato momento, valutando quanto valore si ottiene per ogni unità monetaria spesa.

2.1.1.1.5 Estimated at completion (MPC-EAC)

Fornisce una stima del costo totale del progetto basata sulle condizioni attuali.

2.1.1.1.6 Schedule variance (MPC-SV)

Indica la differenza tra il valore guadagnato (EV) e il valore pianificato (PV). Valori negativi mostrano ritardi rispetto alla pianificazione.

2.1.1.1.7 Budget variance (MPC-BV)

Misura la differenza tra il valore pianificato (PV) e il costo effettivo (AC) alla data corrente. Valori negativi indicano un superamento del *budget* pianificato.

2.1.1.1.8 Estimate to completion (MPC-ETC)

Stima i costi aggiuntivi necessari per completare il progetto.

2.1.1.1.9 Tabella metriche fornitura

Ricordiamo che il **BAC** rappresenta il *Budget at Completion*, ovvero il costo totale del progetto stabilito in fase di candidatura.

Metrica	Nome	Valore accettabile	Valore desiderabile
MPC-PV	*Planned value*	≥ 0	$\leq$ BAC
MPC-EV	*Earned value*	≥ 0	$\leq$ EAC
MPC-AC	*Actual cost*	≥ 0	$\leq$ EAC
MPC-CPI	*Cost performance index*	tra 0.95 e 1.05	1
MPC-EAC	*Estimated at completion*	$\pm 5\%$ rispetto BAC	BAC
MPC-SV	*Schedule variance*	$\pm 5\%$ rispetto BAC	0%
MPC-BV	*Budget variance*	$\pm 5\%$ rispetto BAC	0%
MPC-ETC	*Estimated to completion*	≥ 0	$\leq$ EAC

Tabella 1: Valori accettabili e desiderabili per ogni metrica riguardante il processo di fornitura

2.1.1.2 Sviluppo

Composta da attività il cui scopo è di descrivere le attività e i compiti necessari per creare e mantenere un sistema *software*, garantendo che il prodotto finale soddisfi i requisiti specificati nel contratto. Elenchiamo di seguito le metriche relative.

2.1.1.2.1 Indice di Stabilità dei Requisiti (MPC-ISR)

Valuta la stabilità dei requisiti nel corso del tempo.

2.1.1.2.2 Tabella metriche sviluppo

Metrica	Nome	Valore accettabile	Valore desiderabile
MPC-ISR	Indice di Stabilità dei Requisiti	$\geq 75\%$	100%

Tabella 2: Valori accettabili e desiderabili per ogni metrica riguardante il processo di sviluppo

2.1.2 Processi di supporto

Forniscono servizi e attività che assistono i processi primari. Includono: documentazione, gestione della configurazione, controllo qualità, verifica, validazione e risoluzione dei problemi.

2.1.2.1 Documentazione

Processo necessario per il tracciamento di tutte le attività relative al progetto.

2.1.2.1.1 Indice Gulpease (MPC-IG)

L' indice gulpease[g] è una metrica utilizzata per valutare la leggibilità di un testo in lingua italiana. Tiene conto della lunghezza delle parole e delle frasi, fornendo un punteggio da 0 a 100. Punteggi più alti indicano una maggiore leggibilità.

2.1.2.1.2 Correttezza Ortografica (MPC-CO)

La metrica della Correttezza Ortografica misura il numero di errori grammaticali e ortografici presenti in un documento.

2.1.2.1.3 Tabella metriche documentazione

Metrica	Nome	Valore accettabile	Valore desiderabile
MPC-IG	Indice Gulpease	≥ 40	≥ 80
MPC-CO	Correttezza Ortografica	0	0

Tabella 3: Valori accettabili e desiderabili per ogni metrica riguardante il processo di documentazione

2.1.2.2 Gestione della qualità

2.1.2.2.1 Percentuale di Metriche Soddisfatte (MPC-PMS)

Indica la percentuale di metriche che risultano soddisfare gli obiettivi minimi di qualità.

2.1.2.2.2 Tabella metriche gestione della qualità

Metrica	Nome	Valore accettabile	Valore desiderabile
MPC-PMS	Percentuale di Metriche Soddisfatte	$\geq 80\%$	100%

Tabella 4: Valori accettabili e desiderabili per ogni metrica riguardante il processo di gestione della qualità

2.1.3 Processi organizzativi

Riguardano la gestione e l'organizzazione del progetto. Includono: gestione dei processi, miglioramento, e formazione.

2.1.3.0.1 Efficienza temporale (MPC-ET)

Questa metrica valuta l'efficienza con cui il tempo disponibile viene impiegato in attività produttive, ossia quelle che contribuiscono direttamente al raggiungimento degli obiettivi del progetto.

2.1.3.0.2 Tabella metriche gestione dei processi

Metrica	Nome	Valore accettabile	Valore desiderabile
MPC-ET	Efficienza temporale	≤ 3	≤ 1

Tabella 5: Valori accettabili e desiderabili per ogni metrica riguardante il processo di gestione dei processi

2.2 Qualità di prodotto

La qualità del prodotto si concentra sulla valutazione del *software* sviluppato, considerando aspetti come usabilità, funzionalità, affidabilità, manutenibilità e, più in generale, le prestazioni complessive del sistema. L'obiettivo principale è garantire che il *software* non solo soddisfi le richieste del cliente e funzioni correttamente, ma che lo faccia rispettando specifici *standard* di qualità. Di seguito vengono illustrate le metriche che il team si impegna a rispettare per assicurare un elevato livello di qualità del prodotto.

2.2.1 Funzionalità

La funzionalità misura la capacità del *software* di soddisfare i requisiti obbligatori, desiderabili e opzionali.

2.2.1.1 Requisiti obbligatori soddisfatti (MPD-ROS)

Indica la percentuale di requisiti obbligatori implementati nel prodotto. Deve essere sempre pari al 100% per garantire la conformità alle specifiche.

2.2.1.2 Requisiti desiderabili soddisfatti (MPD-RDS)

Misura la percentuale di requisiti desiderabili implementati nel prodotto. Valori più elevati migliorano la soddisfazione del cliente.

2.2.1.3 Requisiti opzionali soddisfatti (MPD-RFS)

Rappresenta la percentuale di requisiti opzionali implementati. Una copertura opzionale più alta può aggiungere valore al prodotto.

2.2.1.4 Tabella metriche funzionalità

Metrica	Nome	Valore accettabile	Valore desiderabile
MDP-ROS	Requisiti obbligatori soddisfatti	100%	100%
MDP-RDS	Requisiti desiderabili soddisfatti	$\geq$ 0%	100%
MDP-RFS	Requisiti opzionali soddisfatti	$\geq$ 0%	$\geq$ 50%

Tabella 6: Valori accettabili e desiderabili per ogni metrica riguardante la funzionalità del prodotto

2.2.2 Affidabilità

L'affidabilità valuta la capacità del *software* di funzionare correttamente sotto condizioni specifiche.

2.2.2.1 Code coverage (MPD-CC)

Misura la percentuale di codice eseguita durante i *test*. Valori più alti indicano una migliore copertura del codice. Per questo progetto è richiesta una copertura pari o superiore all'80%.

2.2.2.2 Branch coverage (MPD-BC)

Calcola la percentuale di rami decisionali (*branch*[g]) del codice eseguiti durante i *test*. Aiuta a identificare scenari non testati.

2.2.2.3 Passed test cases percentage (MPD-PTCP)

Misura la percentuale di casi di *test* superati rispetto al totale dei *test* eseguiti. Un alto valore indica che il *software* soddisfa i requisiti funzionali e non funzionali previsti.

2.2.2.4 Failure density (MPD-FD)

Indica il numero di fallimenti correttamente riscontrati per unità di dimensione del *software*, spesso misurata in linee di codice (LOC) o punti funzione. Valori più bassi denotano un *software* di qualità superiore, con meno difetti rispetto alla sua complessità o dimensione.

2.2.2.5 Tabella metriche affidabilità

Metrica	Nome	Valore accettabile	Valore desiderabile
MPD-CC	*Code coverage*[g]	$\geq$ 80%	100%
MPD-BC	*Branch coverage*	$\geq$ 60%	100%
MPD-PTCP	*Passed test cases percentage*	100%	100%
MPD-FD	*Failure density*	100%	100%

Tabella 7: Valori accettabili e desiderabili per ogni metrica riguardante l'affidabilità del prodotto

2.2.3 Efficienza

2.2.3.1 Utilizzo risorse (MDP-UR)

Misura l'efficienza del sistema in termini di utilizzo delle risorse hardware, come CPU, memoria e altre risorse di sistema. Un uso efficiente delle risorse garantisce che il sistema funzioni in modo ottimale senza sovraccaricare le risorse disponibili.

2.2.3.2 Tabella metriche efficienza

Metrica	Nome	Valore accettabile	Valore desiderabile
MPD-UR	Utilizzo risorse	$\geq$ 75%	100%

Tabella 8: Valori accettabili e desiderabili per ogni metrica riguardante l'efficienza del prodotto

2.2.4 Usabilità

L'usabilità si riferisce alla facilità con cui un utente può interagire con il software.

2.2.4.1 Facilità di utilizzo (MDP-FU)

Misura il numero di errori commessi dagli utenti durante l'interazione. Un valore minimo indica un'interfaccia intuitiva.

2.2.4.2 Tempo di apprendimento (MDP-TA)

Valuta il tempo necessario a un utente per imparare a utilizzare il *software*. Tempi più brevi migliorano l'esperienza utente.

2.2.4.3 Tabella metriche usabilità

Metrica	Nome	Valore accettabile	Valore desiderabile
MPD-FU	Facilità di utilizzo	≤ 3 errori	0 errori
MPD-TA	Tempo di apprendimento	≤ 15 minuti	≤ 5 minuti

Tabella 9: Valori accettabili e desiderabili per ogni metrica riguardante l'usabilità del prodotto

2.2.5 Manutenibilità

2.2.5.1 Complessità ciclomatica per metodo (MPD-CCM)

La complessità ciclomatica valuta la complessità del codice sorgente attraverso la misurazione del numero di cammini indipendenti attraverso il grafo di controllo del flusso. Una complessità ciclomatica più alta indica che il codice è più difficile da comprendere e manutenere.

2.2.5.2 Code smell (MPD-CS)

Rileva potenziali problemi di progettazione o codice che potrebbero richiedere manutenzione. Segnala parti del codice che potrebbero non essere ottimali e che potrebbero causare difficoltà nel futuro, come un'architettura poco chiara o sezioni di codice ripetitive.

2.2.5.3 Tabella metriche manutenibilità

Metrica	Nome	Valore accettabile	Valore desiderabile
MPD-CCM	Complessità ciclomatica per metodo	≤ 5	≤ 3
MPD-CS	*Code smell*	0	0

Tabella 10: Valori accettabili e desiderabili per ogni metrica riguardante la manutenibilità del prodotto

3 Specifiche dei test

Questa sezione descrive le attività di *testing* effettuate per garantire che i vincoli definiti nei requisiti siano pienamente soddisfatti. In linea con quanto specificato nel documento Norme di Progetto, il piano di *test* adotta un approccio strutturato e si articola nelle seguenti categorie: *test* di unità, *test* di integrazione, *test* di sistema e *test* di accettazione.

3.1 Nomenclatura

Per identificare in modo univoco ciascun *test* viene adottata una nomenclatura che segue il seguente schema:

T[tipologia]-[numero]

Dove tipologia può essere:

- **U**: *test* di unità
- **I**: *test* di integrazione

- **S**: *test* di sistema
- **A**: *test* di accettazione

E dove «numero» è un **numero progressivo** che identifica il *test* all'interno della tipologia.

Inoltre ogni test ha uno stato che ne indica l'esito:
- **Verificato**: il *test* è stato eseguito e ha dato esito positivo
- **Non verificato**: il *test* non è stato eseguito
- **Non implementato**: il *test* non è stato implementato

3.2 Test di unità

Mirano a verificare il funzionamento corretto dei componenti *software* più piccoli e indipendenti, sviluppati principalmente nella fase di progettazione.

3.2.1 Test di unità del simulatore

I *test* di unità del simulatore sono finalizzati a verificare il corretto funzionamento delle classi e dei metodi che compongono il simulatore, garantendo che ciascuna funzionalità sia implementata in modo corretto.

I *test* sono stati sviluppati utilizzando il *framework* Vitest, uno strumento moderno e performante per il *testing* in ambiente TypeScript. Questo *framework* offre funzionalità utili per la scrittura dei *test* come l'utilizzo e la gestione dei *mock*, fondamentali per isolare le singole unità di codice.

Codice test	Descrizione	Stato
TU-1	Verifica che il costruttore della classe `GeoPoint` inizializzi correttamente un oggetto `GeoPoint` e i suoi attributi `latitude` e `longitude`.	Verificato
TU-2	Verifica che i metodi `getLatitude` e `getLongitude` restituiscano correttamente i valori di latitudine e longitudine dell'oggetto `GeoPoint`.	Verificato
TU-3	Verifica che il metodo `radiusKmToGeoPoint` converta correttamente un raggio in chilometri in un oggetto `GeoPoint`.	Verificato
TU-4	Verifica che il metodo `radiusKmToGeoPoint` restituisca la relativa eccezione se il raggio usato è maggiore di 1000 chilometri.	Verificato
TU-5	Verifica che il metodo `radiusKmToGeoPoint` restituisca la relativa eccezione se il raggio usato è maggiore di 300 chilometri.	Verificato
TU-6	Verifica che il metodo `generateRandomPoint` generi correttamente un punto casuale all'interno di un raggio specificato da un oggetto `GeoPoint`.	Verificato
TU-7	Verifica che il metodo `startSimulation` avvii correttamente la simulazione con il numero iniziale di *rent* specificato.	Verificato
TU-8	Verifica che il metodo `startRent` avvii correttamente un *rent* utilizzando un *tracker* disponibile.	Verificato
TU-9	Verifica che il metodo `startRent` lanci un'eccezione se non ci sono *tracker* disponibili.	Verificato
TU-10	Verifica che il metodo `startRentsInRuntime` avvii correttamente i *rent* a *runtime* con intervalli casuali.	Verificato
TU-11	Verifica che il metodo `startSimulation` lanci un eccezione se si verifica un errore durante l'avvio di un nuovo noleggio.	Non implementato

Codice test	Descrizione	Stato
TU-12	Verifica che il metodo `startRentsInRuntime` lanci un eccezione se si verifica un errore durante l'avvio di un nuovo noleggio.	Non implementato
TU-13	Verifica che il metodo `activate` chiami correttamente i metodi `fetchTrack` e `move`.	Verificato
TU-14	Verifica che il metodo `activate` gestisca correttamente un errore quando `fetchTrack` genera un'eccezione.	Verificato
TU-15	Verifica che il metodo `listenToAdv` inizializzi e connetta correttamente il consumer.	Verificato
TU-16	Verifica che il metodo `listenToAdv` gestisca correttamente un errore quando `initAndConnectConsumer` genera un'eccezione.	Verificato
TU-17	Verifica che il metodo `getIsAvailable` restituisca correttamente lo stato di disponibilità del tracker.	Verificato
TU-18	Verifica che il messaggio venga correttamente formato e inviato tramite il `KafkaManager`.	Verificato
TU-19	Verifica che quando il `tracker` raggiunge l'ultimo punto del percorso il metodo `move` disconnetta correttamente il `producer` ed il `consumer` Kafka[g].	Non implementato
TU-20	Verifica che il metodo `move` lanci un eccezione se si verifica un generico errore all'interno del metodo.	Non implementato
TU-21	Verifica che il metodo `fetchTrack` gestisca correttamente gli errori di richiesta HTTP.	Verificato
TU-22	Verifica che il metodo `fetchTrack` decodifichi correttamente una polilinea e la converta in un *array* di `GeoPoint`.	Verificare
TU-23	Verifica che il metodo `fetchTrack` campioni correttamente i punti della traccia quando il numero di punti supera il limite massimo.	Verificato
TU-24	Verifica che il metodo privato `request` restituisca correttamente una risposta HTTP valida.	Verificato
TU-25	Verifica che il metodo `initAndConnectProducer` sia in grado di inizializzare e connettere correttamente un produttore Kafka, controllando che il metodo `connect` del produttore sia stato chiamato.	Verificato
TU-26	Verifica che il metodo `initAndConnectProducer` gestisca correttamente gli errori di connessione, lanciando un errore quando la connessione fallisce.	Verificato
TU-27	Verifica che il metodo `disconnectProducer` disconnetta correttamente un produttore Kafka, controllando che il metodo `disconnect` del produttore sia stato chiamato.	Verificato
TU-28	Verifica che il metodo `disconnectProducer` gestisca correttamente gli errori di disconnessione, lanciando un errore quando la disconnessione fallisce.	Verificato

Codice test	Descrizione	Stato
TU-29	Verifica che il metodo `sendMessage` invii correttamente un messaggio tramite un produttore Kafka, controllando che il metodo `send` del produttore sia stato chiamato con i parametri corretti.	Verificato
TU-30	Verifica che il metodo `sendMessage` gestisca correttamente gli errori durante l'invio di un messaggio, lanciando un errore quando l'invio fallisce.	Verificato
TU-31	Verifica che il metodo `initAndConnectConsumer` sia in grado di inizializzare e connettere correttamente un consumatore Kafka, controllando che i metodi `connect`, `subscribe` e `run` siano stati chiamati correttamente.	Verificato
TU-32	Verifica che il metodo `initAndConnectConsumer` gestisca correttamente gli errori di connessione, lanciando un errore quando la connessione fallisce.	Verificato
TU-33	Verifica che il metodo `disconnectConsumer` disconnetta correttamente un consumatore Kafka, controllando che il metodo `disconnect` del consumatore sia stato chiamato.	Verificato
TU-34	Verifica che il metodo `disconnectConsumer` gestisca correttamente gli errori di disconnessione, lanciando un errore quando la disconnessione fallisce.	Verificato

3.2.2 Test di unità del backend

I *test* di unità del *backend* sono finalizzati a verificare il corretto funzionamento delle classi e dei metodi che compongono il *backend*, garantendo che ciascuna funzionalità sia implementata in modo corretto.

I *test* sono stati sviluppati utilizzando il *framework* JUnit 5, un efficace e consolidato strumento per il *testing* in ambiente Java, coadiuvato da Mockito per la creazione di *mock* e *stub*.

Codice test	Descrizione	Stato
TU-35	Verificare che il metodo `getConnection` funzioni correttamente alla sua prima chiamata, creando e restituendo un'istanza di tipo `ConnectionFactory`.	Verificato
TU-36	Verificare che il metodo `getConnection`, se chiamato più di una volta, restituisca sempre la stessa istanza di tipo `ConnectionFactory` (correttezza del *singleton*).	Verificato
TU-37	Verificare che gli oggetti di tipo `GPSDataDto` vengano istanziati correttamente, constatando in particolare che l'*override* del metodo `equals` si comporti come previsto.	Verificato
TU-38	Verificare che i metodi *getters* e *setters* della classe `GPSDataDto` funzionino correttamente, rispettivamente restituendo e modificando correttamente i valori degli attributi dell'oggetto.	Verificato
TU-39	Verificare che l'*override* del metodo `hashCode` della classe `GPSDataDto` funzioni correttamente, restituendo quindi lo stesso valore per oggetti uguali.	Verificato

Codice test	Descrizione	Stato
TU-40	Verificare che gli oggetti di tipo `GPSData` vengano istanziati correttamente, constatando in particolare che l'*override* del metodo `equals` si comporti come previsto.	Verificato
TU-41	Verificare che i metodi *setters* della classe `GPSData` funzionino correttamente, modificando correttamente i valori degli attributi dell'oggetto.	Verificato
TU-42	Verificare che l'*override* del metodo `hashCode` della classe `GPSData` funzioni correttamente, restituendo quindi lo stesso valore solo per oggetti uguali.	Verificato
TU-43	Verificare che gli oggetti di tipo `PointOfInterest` vengano istanziati correttamente, constatando in particolare che l'*override* metodo `equals` si comporti come previsto.	Verificato
TU-44	Verificare che l'*override* del metodo `hashCode` della classe `PointOfInterest` funzioni correttamente, restituendo quindi lo stesso valore solo per oggetti uguali.	Verificato
TU-45	Verificare che la richiesta di generazione di un annuncio tramite il metodo asincrono `asyncInvoke` funzioni correttamente, completando l'oggetto `resultFuture` con la tupla contenente il testo generato dalla LLM[g].	Verificato
TU-46	Verificare che nel caso in cui la stringa contenente gli interessi dell'utente nel *database*[g] sia vuota, la richiesta di generazione dell'annuncio tramite il metodo `asyncInvoke` porti comunque alla generazione di un annuncio.	Verificato
TU-47	Verificare che se durante la preparazione della *query* tramite la chiamata a `createStatement` nel metodo `asyncInvoke` della classe `AdvertisementGenerationRequest` viene generata un'eccezione, questa venga correttamente gestita e riportata nel *log*[g].	Verificato
TU-48	Verificare che la richiesta di ricerca del punto di interesse più vicino all'utente tramite il metodo asincrono `asyncInvoke`, in caso di successo, completi l'oggetto `resultFuture` con la tupla contenente un oggetto del tipo `PointOfInterest`.	Verificato
TU-49	Verificare che nel caso in cui il punto di interesse più vicino all'utente sia più lontano di 100 m in linea d'aria la richiesta di ricerca del punto di interesse più vicino tramite il metodo `asyncInvoke` completi la tupla `resultFuture` con un *set* vuoto.	Verificato
TU-50	Verificare che se durante la preparazione della *query* tramite la chiamata a `createStatement` nel metodo `asyncInvoke` della classe `NearestPOIRequest` viene generata un'eccezione, questa venga correttamente gestita e riportata nel *log*.	Verificato
TU-51	Verificare che il metodo `serialize` della classe `AdvertisementSerializationSchema` funzioni correttamente, controllando che i campi "rent_id" e "adv" dell'oggetto JSON restituito coincidano con il rentId dell'oggetto `GPSData` e la stringa contenente l'annuncio generato ricevuti in input dal metodo.	Verificato
TU-52	Verificare che il metodo `serialize` della classe `AdvertisementSerializationSchema` generi un'eccezione di tipo `NullPointerException` se viene invocato da un oggetto con valore `null`.	Verificato

Codice test	Descrizione	Stato
TU-53	Verificare che il metodo `deserialize` della classe `GPSDataDeserializationSchema` funzioni correttamente, controllando che ricevuta in input una stringa in formato JSON contenente i campi "rent_id", "latitude" e "longitude" questo restituisca un oggetto `GPSData` contenente i valori dei campi corrispondenti.	Verificato
TU-54	Verificare che il metodo `deserialize` della classe `GPSDataDeserializationSchema`, quando riceve in input una stringa in formato JSON non valida, generi un'eccezione di tipo `IOException`, che viene correttamente gestita e riportata nel *log*	Verificato
TU-55	Verificare che il metodo `execute` della classe `DataStreamJob` funzioni correttamente.	Verificato
TU-56	Verificare che il metodo `createTopic` della classe `KafkaTopicTest` funzioni correttamente.	Verificato
TU-57	Verificare che il metodo `createTopics` della classe `KafkaTopicTest` funzioni correttamente.	Verificato
TU-58	Verificare che il metodo `createTopic` della classe `KafkaTopicTest` gestisca correttamente il tentativo di creare un *topic*[g] con lo stesso nome di un *topic* già esistente, riportando nel *log* un avviso di *topic* già esistente.	Verificato

3.3 Test di integrazione

Successivi ai *test* di unità, hanno lo scopo di verificare l'interazione tra diverse unità *software* per garantire che lavorino in sinergia per compiti specifici.

3.3.1 Test di integrazione del backend

I *test* di integrazione del *backend* sono finalizzati a verificare il corretto funzionamento delle diverse componenti coinvolte nello *stream processing*[g] dei dati provenienti dai sensori.

Per l'esecuzione dei *test* di integrazione viene utilizzata la libreria Testcontainers per Java, che permette di semplificare la creazione dei *container* Docker[g]. Questa libreria è stata scelta per la sua facilità d'uso e per la sua capacità di integrarsi con i *framework* di *testing* esistenti. Un altro ausilio che è stato utilizzato è la classe `MiniClusterWithClientResource` offerta da i *test utils* di Flink[g], che permette di creare un *cluster* Flink locale solo per l'esecuzione dei *test*.

Codice test	Descrizione	Stato
TI-1	Verificare che il *backend* si interfacci correttamente con il *database* testando l'esecuzione di una *query*.	Verificato
TI-2	Verificare che il *job* di Flink funzioni correttamente testando il *flow* di un dato del sensore. Questo comincia con la generazione da parte di un *producer* Kafka e si conclude, alla fine del *processing*, con il salvataggio in *database* del dato geospaziale e dell'annuncio di cui questo ha provocato la generazione.	Verificato

3.4 Test di sistema

Precedono i *test* di accettazione e si concentrano sul sistema nel suo complesso, assicurando che vengano soddisfatti tutti i requisiti *software* stabiliti e tracciati dal documento di Analisi dei Requisiti.

Codice test	Descrizione	Requisito	Stato
TS-1	Verificare che ciascun sensore invii in modo corretto i propri dati di identificazione e di localizzazione geospaziale, a intervalli di tempo regolari.	ROF-1	Verificato
TS-2	Verificare che la *dashboard*[g] sia accessibile solo previa autenticazione da parte dell'amministratore con le proprie credenziali.	ROF-2	Verificato
TS-3	Verificare che l'amministratore abbia fornito un indirizzo *e-mail* per procedere con l'autenticazione.	ROF-3	Verificato
TS-4	Verificare che l'amministratore abbia fornito una password per procedere con l'autenticazione.	ROF-4	Verificato
TS-5	Verificare che, se l'amministratore inserisce almeno una credenziale errata, l'autenticazione fallisca e venga ritornato un messaggio di errore.	ROF-5	Verificato
TS-6	Verificare che l'amministratore possa visualizzare sulla *dashboard* principale una mappa geografica.	ROF-6	Verificato
TS-7	Verificare che l'amministratore possa visualizzare, tramite dei *marker*[g] sulla mappa geografica, la posizione di tutti i punti di interesse.	ROF-7	Verificato
TS-8	Verificare che l'amministratore possa visualizzare sulla mappa il percorso eseguito da ciascun noleggio attivo in quel momento.	ROF-8	Verificato
TS-9	Verificare che l'amministratore possa visualizzare, lungo il percorso di ciascun noleggio attivo, un *marker* specifico se quella posizione ricevuta dal sensore ha provocato una richiesta di generazione annuncio e questa ha avuto successo, ovvero l'utente è stato ritenuto interessato e quindi ha ricevuto l'annuncio.	ROF-9	Verificato
TS-10	Verificare che l'amministratore possa visualizzare, lungo il percorso di ciascun noleggio attivo, un *marker* specifico se quella posizione ricevuta dal sensore ha provocato una richiesta di generazione annuncio e questa non ha avuto successo, ovvero l'utente non è stato ritenuto interessato e quindi non ha ricevuto l'annuncio.	ROF-10	Verificato
TS-11	Verificare che l'amministratore possa visualizzare le informazioni relative ad un punto di interesse interagendo con il *marker* che lo rappresenta all'interno della mappa.	ROF-11	Verificato
TS-12	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un punto di interesse, il nome dello stesso.	ROF-12	Verificato
TS-13	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *mar-*	ROF-13	Verificato

Codice test	Descrizione	Requisito	Stato
	ker di un punto di interesse, la categoria di esercizio commerciale a cui appartiene.		
TS-14	Verificare che l'amministratore, interagendo con un *marker* che segnala l'avvenuta generazione di un annuncio, possa visualizzare le informazioni relative all'annuncio generato.	ROF-14	Verificato
TS-15	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio generato, il nome del punto di interesse associato a quell'annuncio.	ROF-15	Verificato
TS-16	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio generato, l'*e-mail* dell'utente destinatario dell'annuncio.	ROF-16	Verificato
TS-17	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio generato, la data e l'ora di generazione dell'annuncio.	ROF-17	Verificato
TS-18	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio generato, il testo dell'annuncio stesso.	ROF-18	Verificato
TS-19	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio generato, la categoria del punto di interesse associato a quell'annuncio.	ROF-19	Verificato
TS-20	Verificare che l'amministratore, interagendo con un *marker* che segnala la mancata generazione di un annuncio per incompatibilità con gli interessi dell'utente, possa visualizzare le informazioni relative all'annuncio non generato.	ROF-20	Verificato
TS-21	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio non generato, il nome del punto di interesse associato.	ROF-21	Verificato
TS-22	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio non generato, l'*e-mail* dell'utente a cui era destinato l'annuncio non generato.	ROF-22	Verificato
TS-23	Verificare che l'amministratore possa visualizzare, dalle informazioni esposte interagendo con il *marker* di un annuncio non generato, la data e l'ora di tentata generazione dell'annuncio.	ROF-23	Verificato

Codice test	Descrizione	Requisito	Stato
TS-24	Verificare che l'amministratore, interagendo nuovamente con il *marker* di un annuncio generato, possa chiudere la vista che espone le informazioni sull'annuncio.	ROF-24	Verificato
TS-25	Verificare che l'amministratore, interagendo nuovamente con il *marker* di un annuncio non generato, possa chiudere il messaggio contenente le informazioni relative alla mancata generazione di quell'annuncio.	ROF-25	Verificato
TS-26	Verificare che l'amministratore possa spostare la visuale della mappa geografica visualizzata interagendo con la stessa.	ROF-26	Verificato
TS-27	Verificare che l'amministratore, interagendo con la mappa geografica visualizzata, possa fare *zoom in* e *zoom out* sulla visuale.	ROF-27	Verificato
TS-28	Verificare che esista almeno un generatore di dati GPS[g] che simuli il comportamento di un sensore che interagisce col sistema.	ROF-28	Verificato
TS-29	Verificare che il generatore di dati GPS generi dei percorsi realistici, ovvero che seguono vie o strade percorribili.	ROF-29	Verificato
TS-30	Verificare che l'amministratore possa accedere dalla *dashboard* alla sezione dedicata allo storico degli annunci.	RDF-1	Verificato
TS-31	Verificare che l'amministratore possa visualizzare, nella sezione dedicata allo storico degli annunci, l'elenco degli annunci generati dal sistema.	RDF-2	Verificato
TS-32	Verificare che l'amministratore possa visualizzare l'elenco degli annunci nello storico sotto forma di lista o di griglia a discrezione dell'utilizzatore.	RDF-3	Non implementato
TS-33	Verificare che l'amministratore possa visualizzare, per ogni singolo elemento nello storico, le principali informazioni relative all'annuncio.	RDF-4	Verificato
TS-34	Verificare che l'amministratore possa visualizzare, per ogni singolo elemento nello storico, il nome del punto di interesse associato all'annuncio.	RDF-5	Verificato
TS-35	Verificare che l'amministratore possa visualizzare, per ogni singolo elemento nello storico, l'*e-mail* dell'utente destinatario dell'annuncio.	RDF-6	Verificato
TS-36	Verificare che l'amministratore possa visualizzare, per ogni singolo elemento nello storico, la data e l'ora in cui il sistema ha tentato la generazione dell'annuncio.	RDF-7	Verificato

Codice test	Descrizione	Requisito	Stato
TS-37	Verificare che, per ogni singolo elemento nello storico, l'amministratore possa visualizzarne i dettagli completi interagendo con l'elemento stesso.	RDF-8	Verificato
TS-38	Verificare che l'amministratore possa visualizzare, attraverso i dettagli completi di un annuncio, il nome del punto di interesse associato a quell'annuncio.	RDF-9	Verificato
TS-39	Verificare che l'amministratore possa visualizzare, attraverso i dettagli completi di un annuncio, l'*e-mail* dell'utente destinatario di quell'annuncio.	RDF-10	Verificato
TS-40	Verificare che l'amministratore possa visualizzare, attraverso i dettagli completi di un annuncio, la data e l'ora in cui il sistema ha tentato la generazione dell'annuncio.	RDF-11	Verificato
TS-41	Verificare che l'amministratore possa visualizzare, attraverso i dettagli completi di un annuncio, il testo dell'annuncio stesso.	RDF-12	Verificato
TS-42	Verificare che l'amministratore possa visualizzare, attraverso i dettagli completi di un annuncio, la categoria del punto di interesse associato a quell'annuncio.	RDF-13	Verificato
TS-43	Verificare che l'amministratore possa chiudere la vista che espone i dettagli completi di un singolo annuncio.	RDF-14	Verificato
TS-44	Verificare che l'amministratore possa filtrare gli annunci visualizzati nello storico in base all'*e-mail* dell'utente destinatario.	RDF-15	Verificato
TS-45	Verificare che l'amministratore possa filtrare gli annunci visualizzati nello storico in base al nome del punto di interesse associato all'annuncio.	RDF-16	Verificato
TS-46	Verificare che l'amministratore possa filtrare gli annunci visualizzati nello storico in base alla data in cui sono stati generati, selezionando un intervallo di date.	RDF-17	Verificato
TS-47	Verificare che l'amministratore possa filtrare gli annunci visualizzati nello storico in base all'orario in cui sono stati generati, selezionando una fascia oraria.	RDF-18	Verificato
TS-48	Verificare che l'amministratore possa accedere dalla *dashboard* alla sezione dedicata ai grafici statistici.	RFF-1	Verificato
TS-49	Verificare che l'amministratore possa visualizzare, nella sezione dedicata ai grafici statistici, un grafico per ciascuna analisi dati.	RFF-2	Verificato

Codice test	Descrizione	Requisito	Stato
TS-50	Verificare che ciascun grafico visualizzato abbia un titolo che ne descriva efficacemente il contenuto.	RFF-3	Verificato
TS-51	Verificare che in ciascun grafico visualizzato l'asse delle ascisse e l'asse delle ordinate siano correttamente etichettati e completi di tutti i valori.	RFF-4	Verificato
TS-52	Verificare che ciascun grafico visualizzato stia rappresentando lo specifico *set* di dati previsto per quel grafico.	RFF-5	Verificato
TS-53	Verificare che l'amministratore possa visualizzare, nella sezione dedicata ai grafici statistici, un grafico che mostri il numero di annunci generati dal sistema nelle ultime 24 ore, con granularità oraria.	RFF-6	Verificato
TS-54	Verificare che l'amministratore possa visualizzare, nella sezione dedicata ai grafici statistici, un grafico che riporti il numero medio di noleggi che vengono effettuati in ciascun mese dell'anno.	RFF-7	Verificato
TS-55	Verificare che l'amministratore, dalla sezione dedicata ai grafici statistici, possa selezionare un punto di interesse e contestualmente visualizzarne i relativi grafici.	RFF-8	Verificato
TS-56	Verificare che l'amministratore possa visualizzare il grafico che, per un certo punto di interesse, mette a confronto il numero di annunci generati con il numero di annunci non generati nell'ultima settimana.	RFF-9	Verificato
TS-57	Verificare che l'interfaccia creata per la visualizzazione degli annunci in tempo reale lato utente fruitore del servizio di noleggio sia funzionante.	RFF-10	Non implementato

3.5 Test di accettazione

Condotti insieme all'azienda proponente, servono a garantire che il prodotto finale sia conforme alle aspettative e ai requisiti contrattuali, permettendone il rilascio definitivo.

Codice test	Descrizione	Stato
TA-01	Verificare che all'apertura il prodotto mostri una mappa che visualizza in tempo reale i percorsi compiuti dai mezzi in movimento.	Verificato
TA-02	Verificare che il prodotto supporti la generazione tramite LLM di annunci personalizzati per ogni utente in base ai suoi dati di profilazione.	Verificato
TA-03	Verificare che il prodotto supporti la visualizzazione dei *marker* corrispondenti ai mezzi con noleggio attivi all'interno della mappa.	Verificato
TA-04	Verificare che il prodotto supporti la visualizzazione dei *marker* corrispondenti ai punti di interesse all'interno della mappa.	Verificato
TA-05	Verificare che il prodotto supporti la visualizzazione dei *marker* corrispondenti agli annunci generati all'interno della mappa.	Verificato
TA-06	Verificare che il prodotto supporti la visualizzazione dei *marker* corrispondenti alle mancate generazioni di annunci all'interno della mappa.	Verificato
TA-07	Verificare che il prodotto supporti la visualizzazione dei dati dei punti di interesse tramite i rispettivi *marker* sulla mappa.	Verificato
TA-08	Verificare che il prodotto supporti la visualizzazione dei dati degli annunci generati per ogni rispettivo *marker* sulla mappa.	Verificato
TA-09	Verificare che il prodotto supporti la visualizzazione dei dati delle mancate generazioni di annunci per ogni rispettivo *marker* sulla mappa.	Verificato
TA-10	Verificare che il prodotto supporti una generazione realistica dei percorsi dei noleggi.	Verificato
TA-11	Verificare che il prodotto persista in un *database* i dati simulati e quelli generati dalla LLM.	Verificato
TA-12	Verificare che il prodotto sia fruibile con le ultime versioni dei *browser web* principali, nello specifico: Google Chrome v134.0.6998.88, Mozilla Firefox v136.0.2, Microsoft Edge v134.0.3124.72 e Safari v18.3.	Verificato
TA-13	Verificare che il prodotto consenta la visualizzazione dei dati relativi agli annunci generati e non generati, prodotti nel corso del tempo, all'interno di una sezione apposita detta «storico».	Verificato
TA-14	Verificare che il prodotto fornisca una sezione dedicata alla visualizzazione di grafici statistici relativi ai dati raccolti.	Verificato

Tabella 11: Descrizione e stato di ogni test di accettazione

4 Cruscotto di monitoraggio della qualità

4.1 Estimated at Completion (MPC-EAC)

L'**EAC** rappresenta una stima aggiornata del costo totale previsto per completare un progetto, basata sui costi sostenuti e sulle previsioni future.

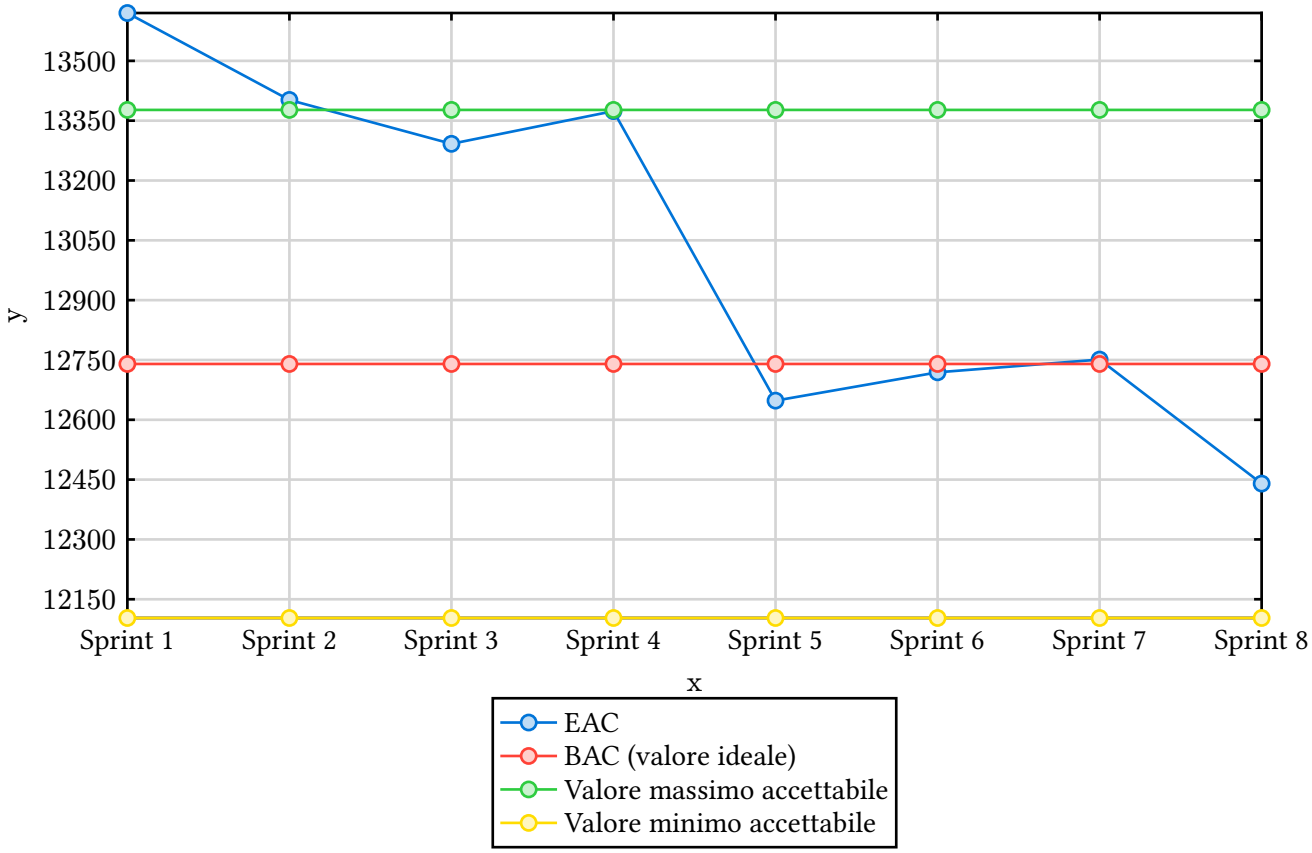

Figura 1: Tabella EAC

4.1.1 RTB

Dal grafico si può notare che il valore dell'**EAC** si è avvicinato, per 4 iterazioni su 5, al valore ideale del **BAC**, rimanendo sempre al di sopra del valore minimo accettabile e, per la maggior parte del tempo, al di sotto del valore massimo accettabile per arrivare, all'ultima iterazione, molto vicino al valore ideale.

Questi dati sottolineano il continuo automiglioramento che il team ha sempre cercato di perseguire.

4.1.2 PB

Il grafico mostra come, dopo la quinta iterazione, il valore dell'**EAC** è rimasto molto vicino al valore ideale abbassandosi leggermente solo nell'ultimo periodo, risultando quindi leggermente inferiore al **BAC**. Il valore si è leggermente alzato in seguito al colloquio RTB, in concomitanza del quinto *sprint*[g], a causa di rallentamenti nella produzione dovuti ad alcune correzioni da effettuare. Il gruppo è riuscito tuttavia a recuperare e abbassare nuovamente l'**EAC** durante l'ottavo *sprint*, portandolo al di sotto del **BAC**.

4.2 Earned Value (MPC-EV) e Planned Value (MPC-PV)

- **Earned Value** (**EV**): Indica il valore del lavoro effettivamente completato in termini di *budget* approvato.
- **Planned Value** (**PV**): Rappresenta il valore pianificato del lavoro che avrebbe dovuto essere completato entro un determinato momento.

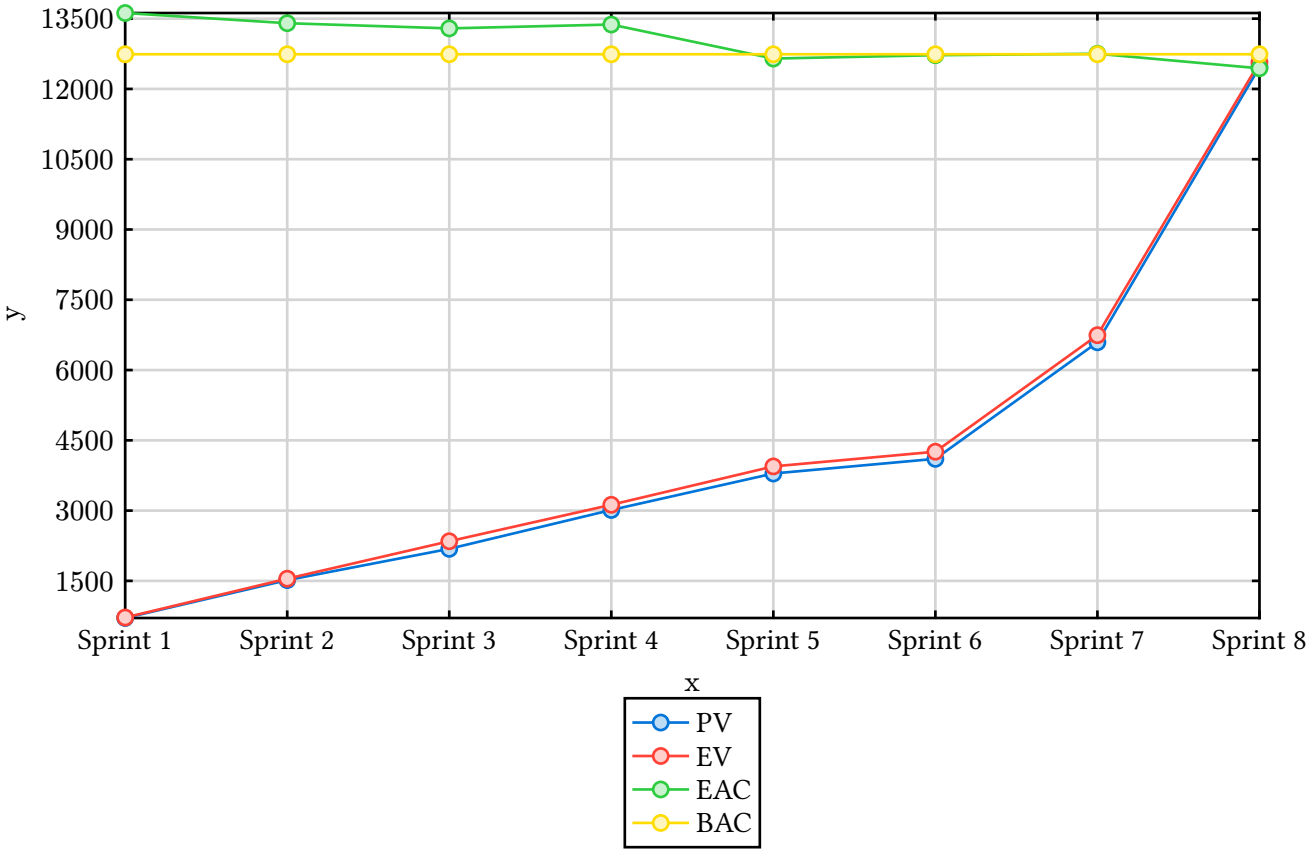

Figura 2: Tabella EV e PV

4.2.1 RTB

Dal grafico si può notare che il valore dell'**EV** è cresciuto pari pari al valore del **PV**, indicando che il lavoro svolto è stato effettivamente eseguito secondo la pianificazione.

4.2.2 PB

Il grafico mostra come il valore dell'**EV** abbia continuato a crescere in modo proporzionale al valore del **PV**, dimostrando che il gruppo è riuscito a pianificare correttamente le attività per tutta la durata del progetto. Dalla sovrapposizione delle due linee si evince inoltre che l'accuratezza della pianificazione è andata migliorandosi fino alla conclusione dell'ottavo *sprint*. I valori di **EV** e **PV** si sono avvicinati di molto a quelli di **EAC** e **BAC** durante l'ultima iterazione, dimostrando che il gruppo è riuscito anche ad essere coerente con il preventivo prodotto inizialmente.

4.3 Actual Cost (MPC-AC) e Estimate to Completion (MPC-ETC)

- **Actual Cost** (**AC**): Costo reale sostenuto per il lavoro eseguito fino a un determinato punto.
- **Estimate to Completion** (**ETC**): Stima dei costi rimanenti necessari per completare il progetto.

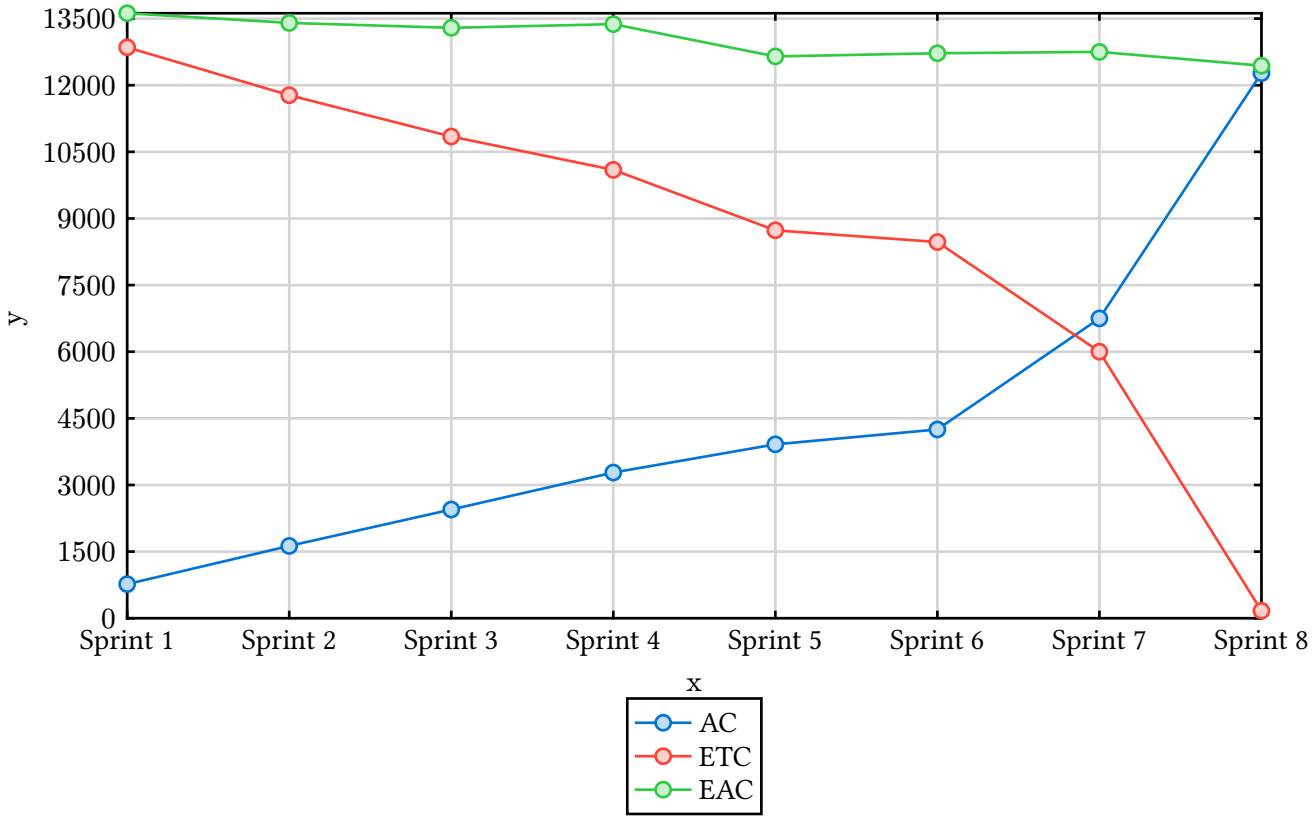

Figura 3: Tabella AC e ETC

4.3.1 RTB

Dal grafico si nota che il valore dell'**AC** è cresciuto in modo proporzionale alla decrescita dell'**ETC**, ed entrambi sono rimasti per tutto il tempo al di sotto del valore desiderato, cioè quello dell'**EAC**.

4.3.2 PB

Dal grafico si può notare come il valore dell'**AC** abbia continuato a crescere proporzionalmente alla decrescita dell'**ETC**. Si nota inoltre come nelle ultime iterazioni la crescita dell'**AC** sia stata maggiore rispetto al primo periodo; questo dimostra che il *team* ha avuto la possibilità di dedicare più tempo al progetto una volta terminati gli impegni universitari, portando ad un aumento repentino del costo reale che rimane comunque inferiore all'ideale dettato dall'**EAC**.

4.4 Budget Variance (MPC-BV) e Schedule Variance (MPC-SV)

- **Budget Variance** (**BV**): Differenza tra il valore guadagnato (EV) e il costo reale (AC), indica se il progetto è sotto o sopra il *budget*.
- **Schedule Variance** (**SV**): Differenza tra il valore guadagnato (EV) e il valore pianificato (PV), mostra se il progetto è in anticipo o in ritardo rispetto alla pianificazione.

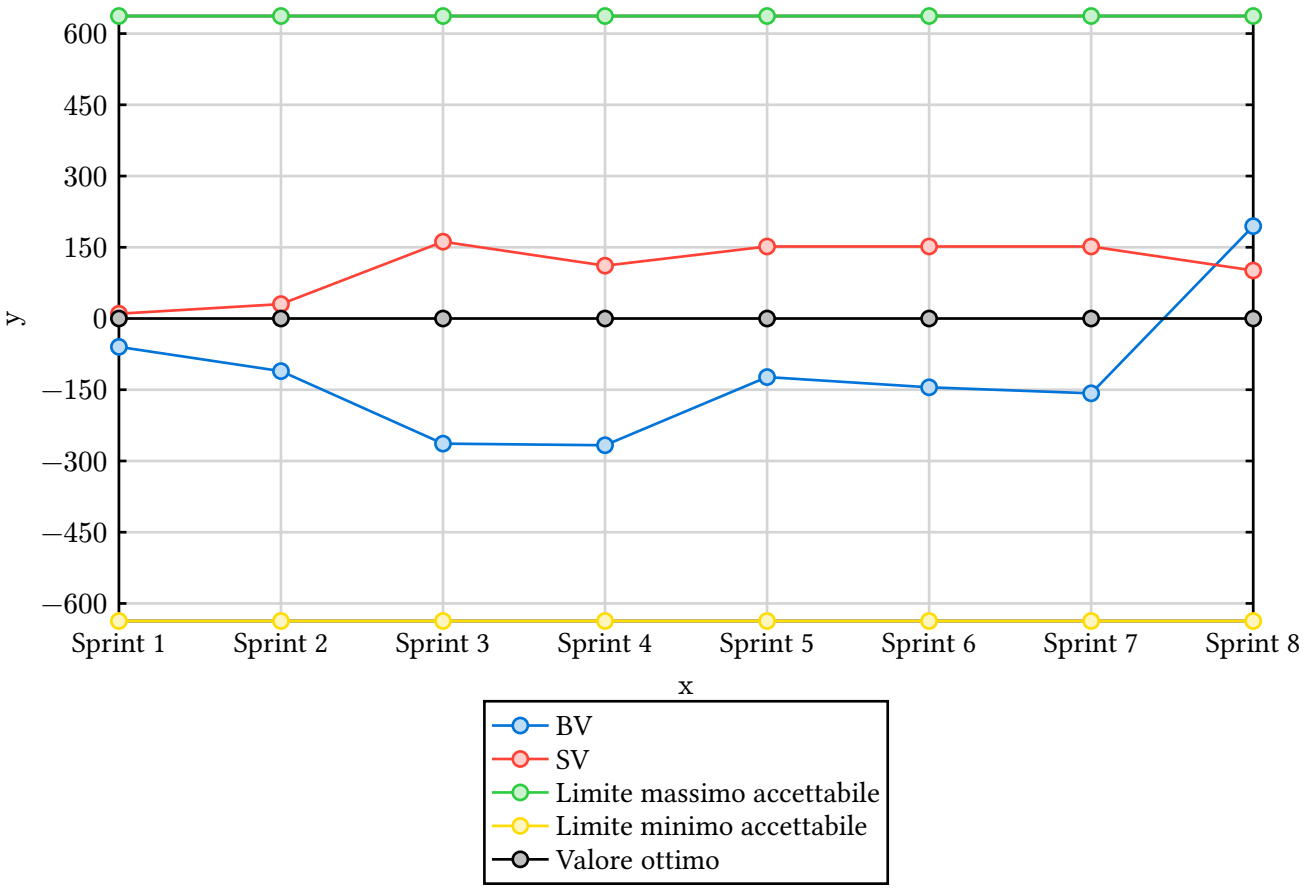

Figura 4: Tabella BV e SV

4.4.1 RTB

Dal grafico si può notare che le variazioni sono sempre state contenute: rientrano sempre ampiamente nei limiti accettabili e non si discostano molto dal valore ottimo. Il **BV** indica che abbiamo sempre speso leggermente oltre il *budget* pianificato, tuttavia abbiamo una prospettiva positiva come indica il **SV**.

4.4.2 PB

Il grafico mostra come dalla quinta iterazione l'**SV** sia sempre stato pressapoco invariato, segnalando di essere leggermente in anticipo rispetto alla pianificazione. Anche il **BV** è rimasto stabile fino al penultimo *sprint* dopo il quale ha raggiunto il semipiano positivo, indice che i costi sostenuti fino alla conclusione dell'ottavo periodo, e quindi del progetto, sono minori di quelli pianificati.

4.5 Indice di Stabilità dei Requisiti (MPC-ISR)

L'**ISR** misura la stabilità dei requisiti del progetto nel tempo, valutando quanto siano stati modificati o aggiornati durante il ciclo di vita.

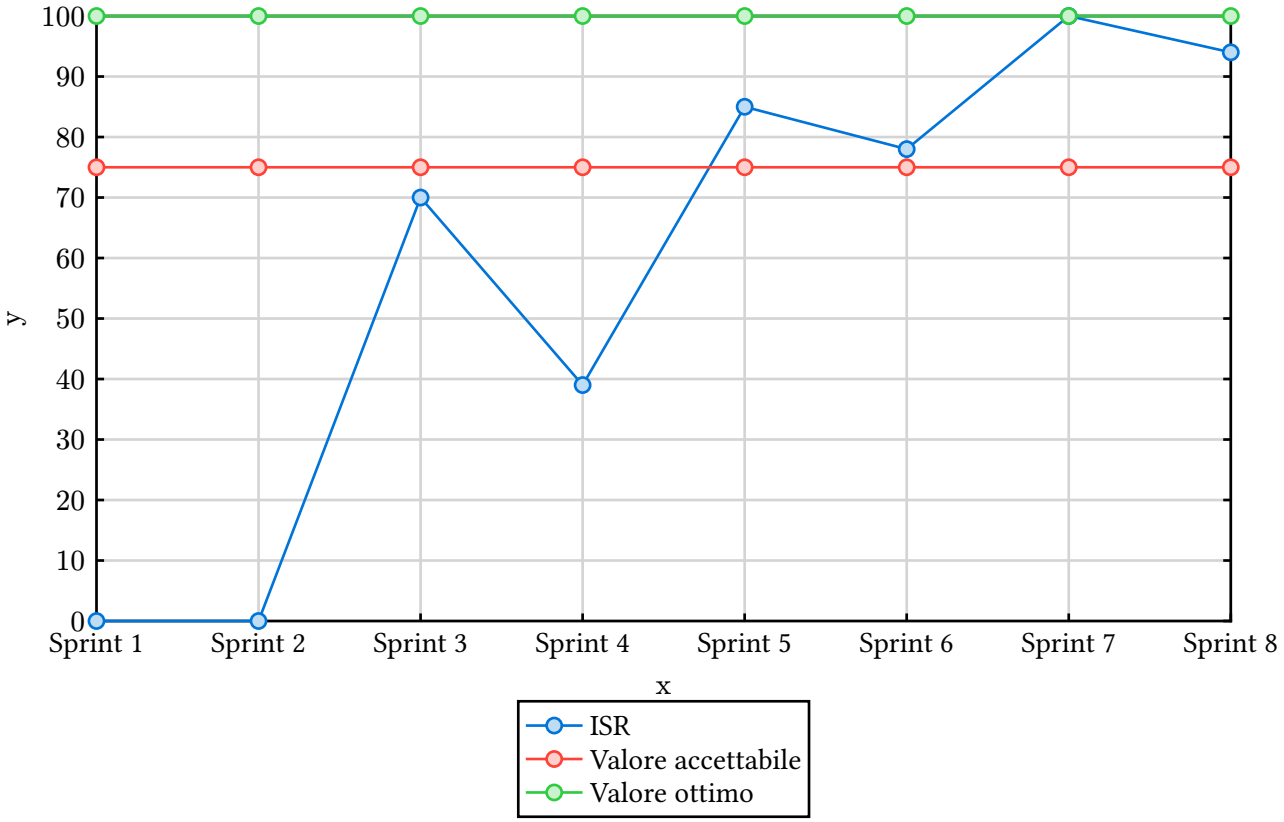

Figura 5: Tabella ISR

4.5.1 RTB

Il grafico è per la maggior parte ascendente tranne nella quarta iterazione, dove i requisiti hanno subito una forte modifica dovuta al colloquio con il professor Cardin, al quale sono seguite numerose correzioni.

Nell'ultimo periodo, quindi quello relativo alla consegna RTB, si è finalmente raggiunto il valore accettabile dell'ISR.

4.5.2 PB

Il grafico mostra come il valore dell'**ISR** sia sempre rimasto, dopo il quinto *sprint*, al di sopra del valore accettabile. I due andamenti discendenti del grafico, all'iterazione 6 e 8, coincidono con due colloqui con il prof. Cardin: nel primo caso sono state apportate le correzioni in seguito alla revisione RTB, nel secondo caso è stato concordato di rimuovere alcuni requisiti in quanto analizzati erroneamente nella prima fase del progetto.

4.6 Indice Gulpease (MPC-IG)

Indice che valuta la leggibilità dei documenti scritti in italiano.

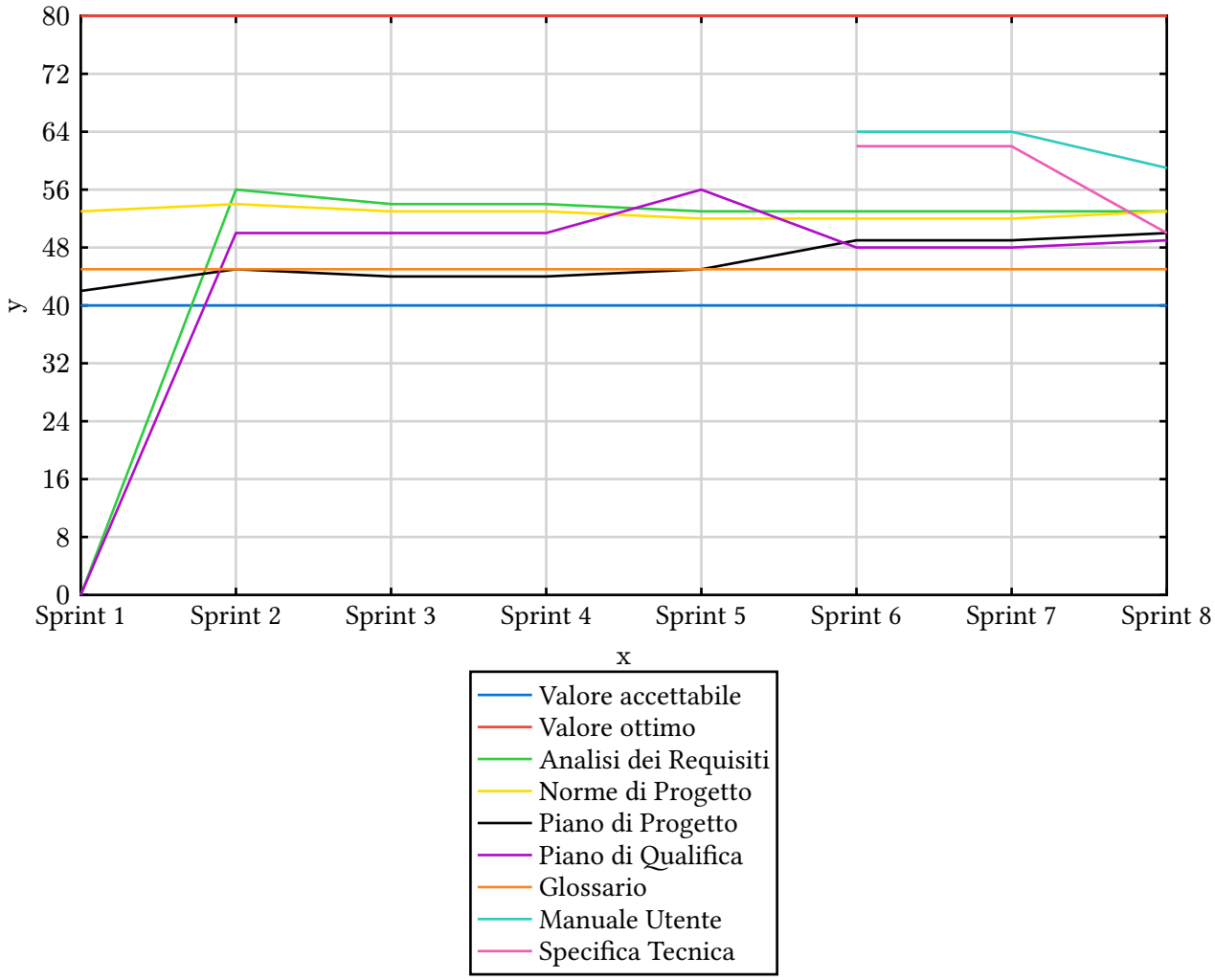

Figura 6: Tabella indice Gulpease

4.6.1 RTB

Si nota che i valori sono sempre sopra al valore accettabile, anche se ancora ben distanti dal valore ottimo. Tuttavia si nota per la maggior parte dei documenti una stabilizzazione del valore, con solo qualche piccola variazione.

Il valore nullo di alcuni documenti alla prima iterazione è dovuto alla stesura tardiva degli stessi. Lo sviluppo della documentazione si è infatti svolta parallelamente alle lezioni di teoria del corso.

4.6.2 PB

Il grafico mostra come i valori di tutti i documenti siano sempre stati al di sopra del valore accettabile e con poche variazioni per la maggior parte di essi, sebbene siano distanti dal valore ottimo. La stabilità dell'indice di Gulpease indica che il gruppo si è concentrato più sull'ampliamento e arricchimento dei documenti piuttosto che nel miglioramento della leggibilità, che rimane comunque al di sopra del valore accettabile per tutti gli elaborati.

Da notare come i valori dei nuovi documenti, cioè Manuale Utente e Specifica Tecnica, siano tra i più alti. Questo dimostra un'esperienza acquisita dal gruppo nella stesura della documentazione.

4.7 Correttezza Ortografica (MPC-CO)

Metriche che misurano la presenza di errori ortografici nei documenti, valutando la qualità formale del contenuto.

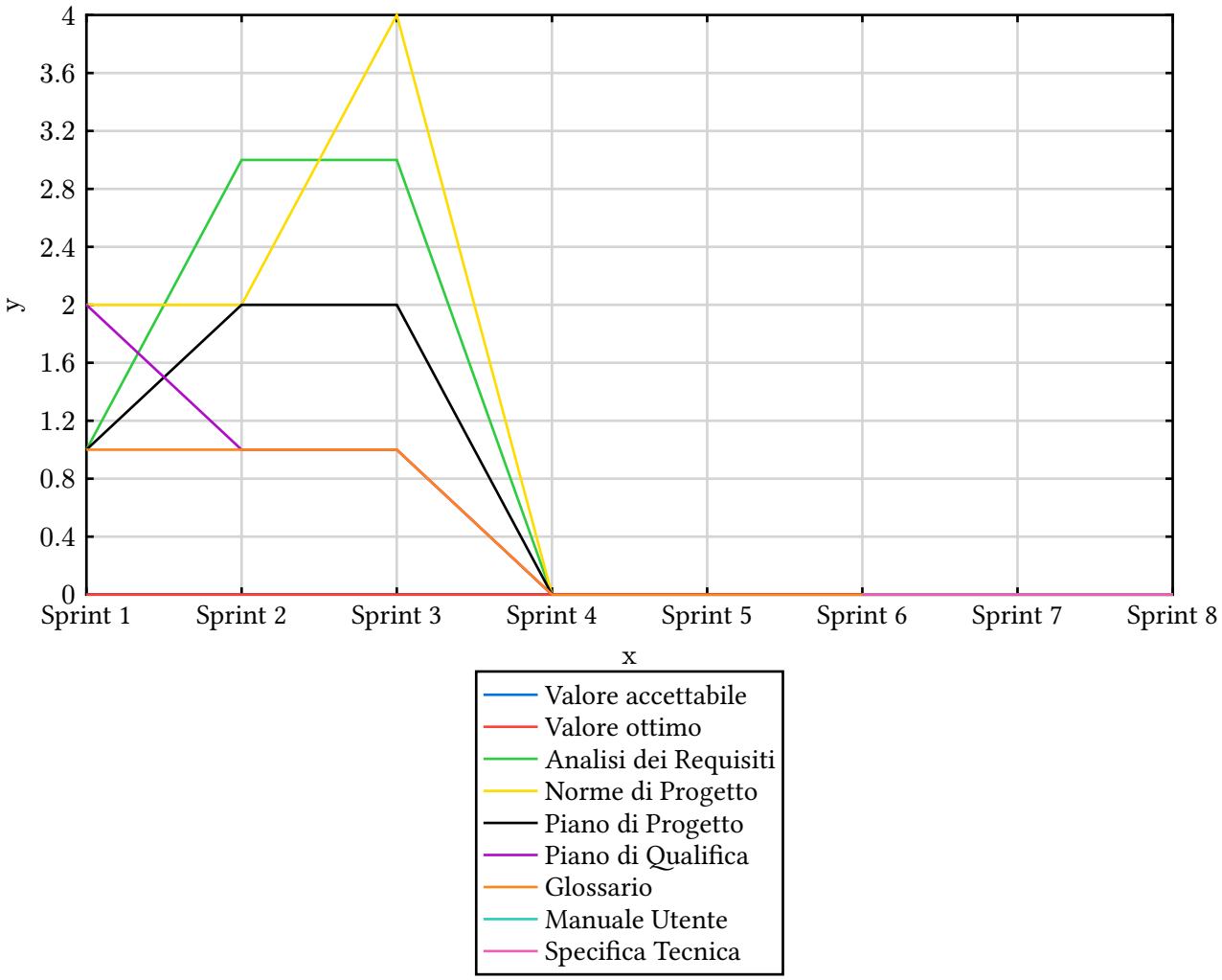

Figura 7: Tabella correttezza ortografica

4.7.1 RTB

Da questo grafico si nota che tutti i documenti hanno avuto alcuni difetti ortografici, quasi sempre di battitura. Tuttavia nel quarto *sprint*, con l'introduzione di uno strumento di *spell checking*, siamo riusciti a raggiungere il valore ottimo di zero errori per tutti i documenti.

4.7.2 PB

Dal grafico si può notare come tutti i documenti, compresi quelli nuovi, siano rimasti con il numero ottimo di zero errori per il resto del periodo di lavoro. Questo dimostra che lo strumento di *spell checking* scelto dal gruppo ha funzionato come desiderato e che il *team* è stato costante nel suo utilizzo.

4.8 Percentuale di Metriche Soddisfatte (MPC-PMS)

Percentuale di metriche di qualità definite per il progetto che sono state soddisfatte rispetto agli obiettivi prefissati.

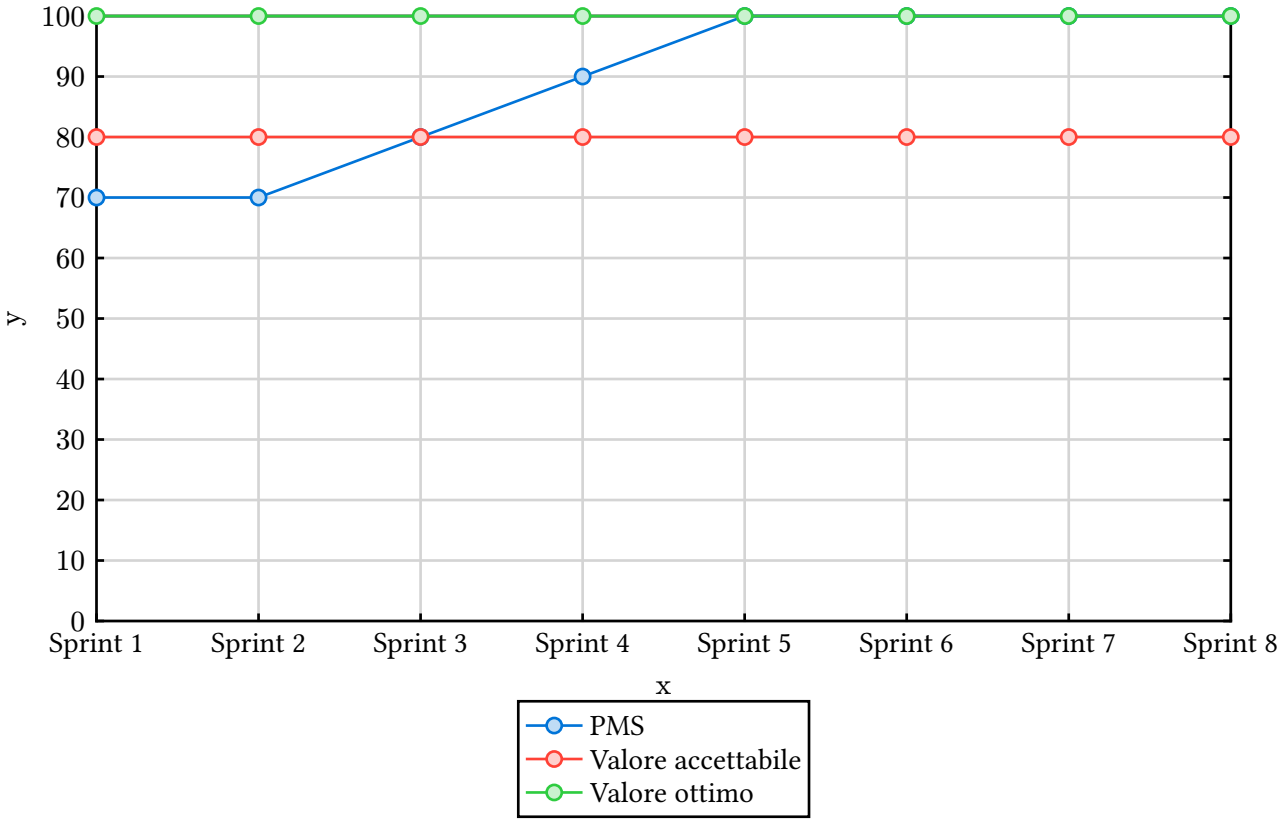

Figura 8: Tabella PMS

4.8.1 RTB

Possiamo notare come la percentuale di metriche soddisfatte sia quasi sempre cresciuta, ma comunque mai diminuita, raggiungendo addirittura il valore ottimo nell'ultimo *sprint*. Questo simboleggia una grande attenzione da parte del team nel perseguire la qualità del progetto, tramite autovalutazione ed automiglioramento.

4.8.2 PB

Il grafico mostra come la percentuale delle metriche soddisfatte sia sempre rimasta pari al valore ottimo dal quinto *sprint* in poi, dimostrando che il *team* è sempre riuscito a lavorare mantenendo una buona qualità del progetto.

4.9 Efficienza Temporale (MPC-ET)

Misura l'efficacia nell'utilizzo del tempo per completare le attività, confrontando il tempo effettivamente impiegato con quello previsto.

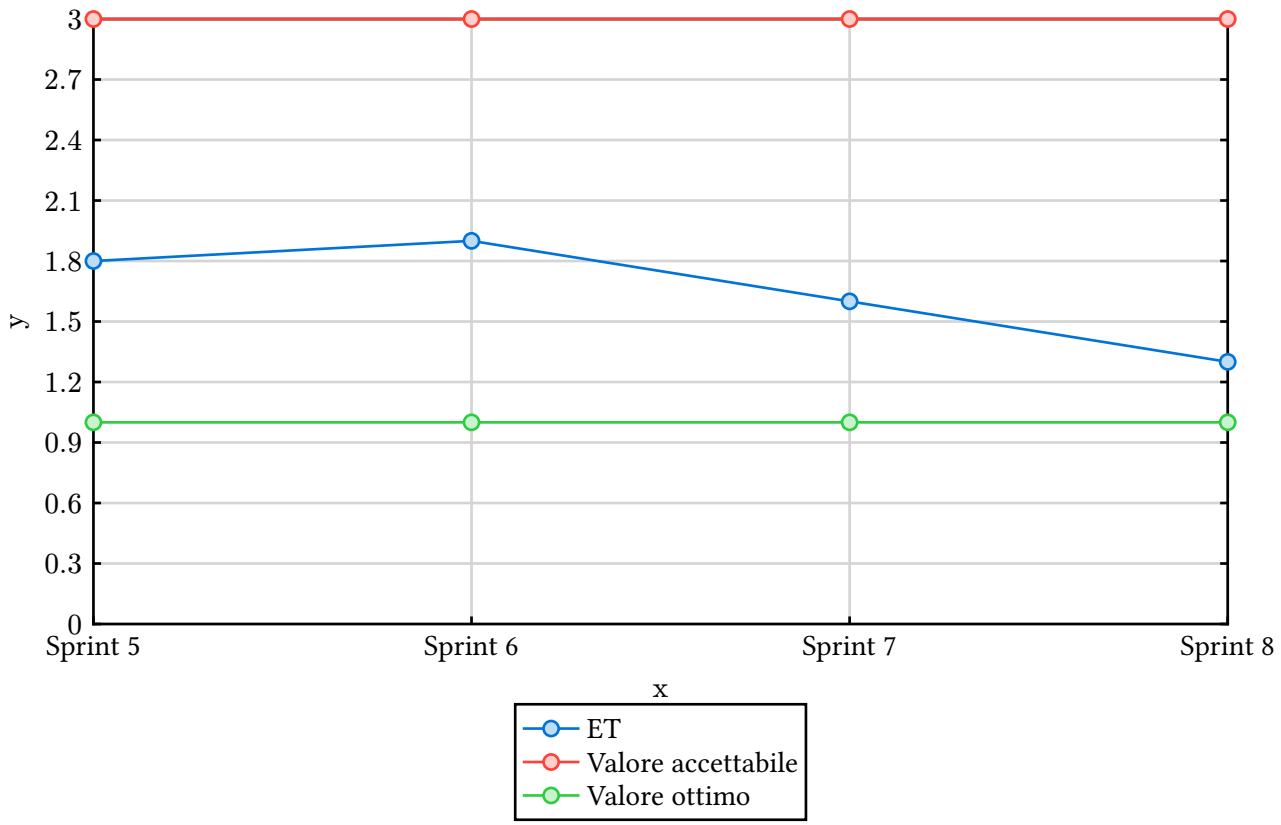

Figura 9: Tabella ET

4.9.1 RTB

La metriche in esame è stata confermata solo al termine del quarto *sprint*. Risulta quindi impossibile recuperare i valori passati e crearne un grafico, anche se siamo confidenti nel dire che il valore è progressivamente diminuito. Il *team* ha imparato sempre più a lavorare asincronamente massimizzando il tempo produttivo. Il gruppo confida sarà una metrica utile in fase di PB.

L'unico valore che possiamo dare di questa metrica è quindi quello dell'ultima iterazione, pari ad un rapporto di 1.8, che è ampiamente al di sotto del valore accettabile di 3 ma ancora al di sopra del valore desiderabile di 1.

4.9.2 PB

Il grafico dimostra come negli ultimi periodi, una volta raggiunta una certa conoscenza delle tecnologie da utilizzare, il *team* sia riuscito a lavorare in modo più efficiente, mantenendo il valore dell'**ET** al di sotto del valore accettabile di 3 e molto vicino al valore ottimo di 1. Il valore si è leggermente alzato durante il quinto *sprint* a causa di cambiamenti nelle tecnologie adottate dopo il colloquio RTB, ma il *team* è riuscito a recuperare velocemente.

4.10 Requisiti obbligatori soddisfatti (MPD-ROS)

Misura la percentuale di requisiti obbligatori soddisfatti dal prodotto.

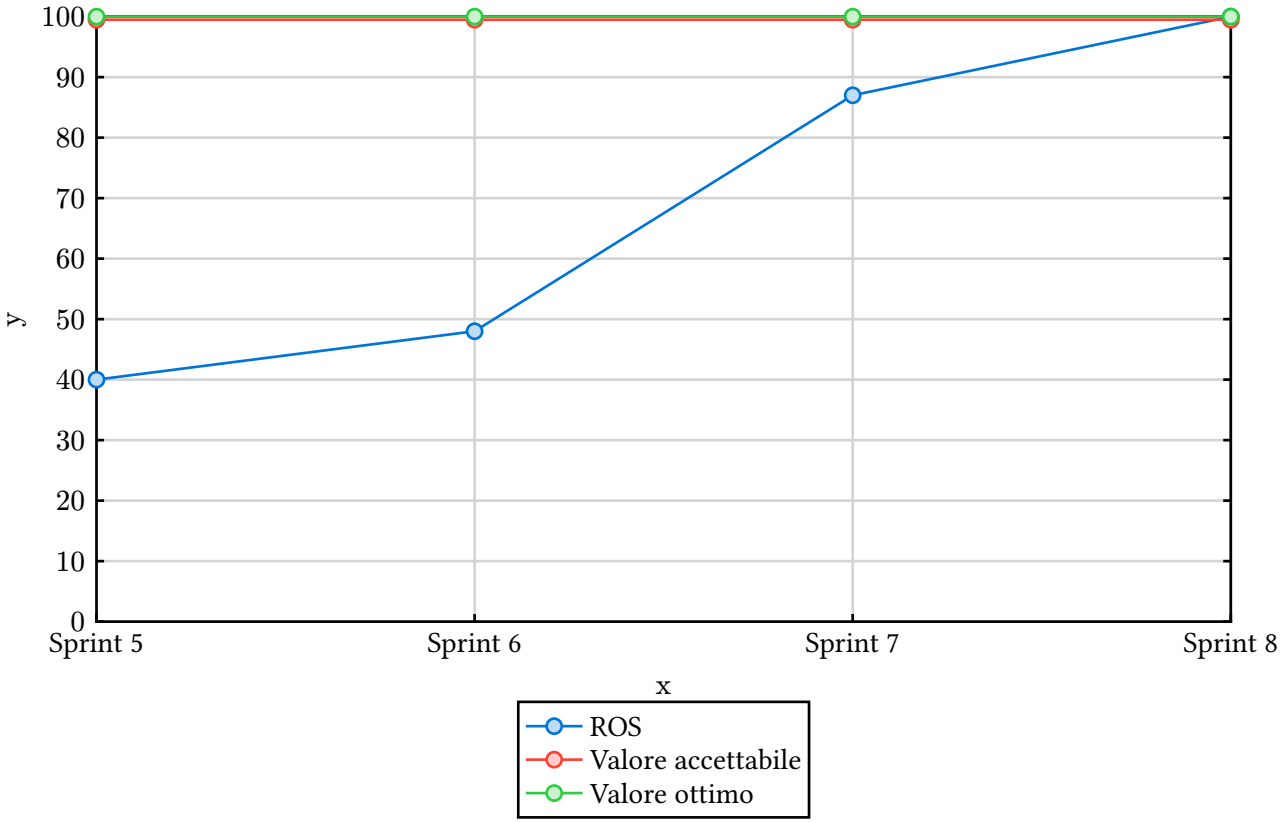

Figura 10: Tabella ROS

4.10.1 PB

Dal grafico si può vedere come il *team* abbia cominciato a lavorare al MVP[g] subito dopo la presentazione del RTB. Partendo da una buona base del PoC[g] il gruppo è andato a migliorare il prodotto andando a soddisfare la maggior parte dei requisiti obbligatori, ai quali era stata data maggiore priorità, entro la settima iterazione. Al termine dell'ottavo *sprint* sono stati soddisfatti tutti i requisiti obbligatori, anche quelli più onerosi.

4.11 Requisiti desiderabili soddisfatti (MPD-RDS)

Misura la percentuale di requisiti desiderabili soddisfatti dal prodotto.

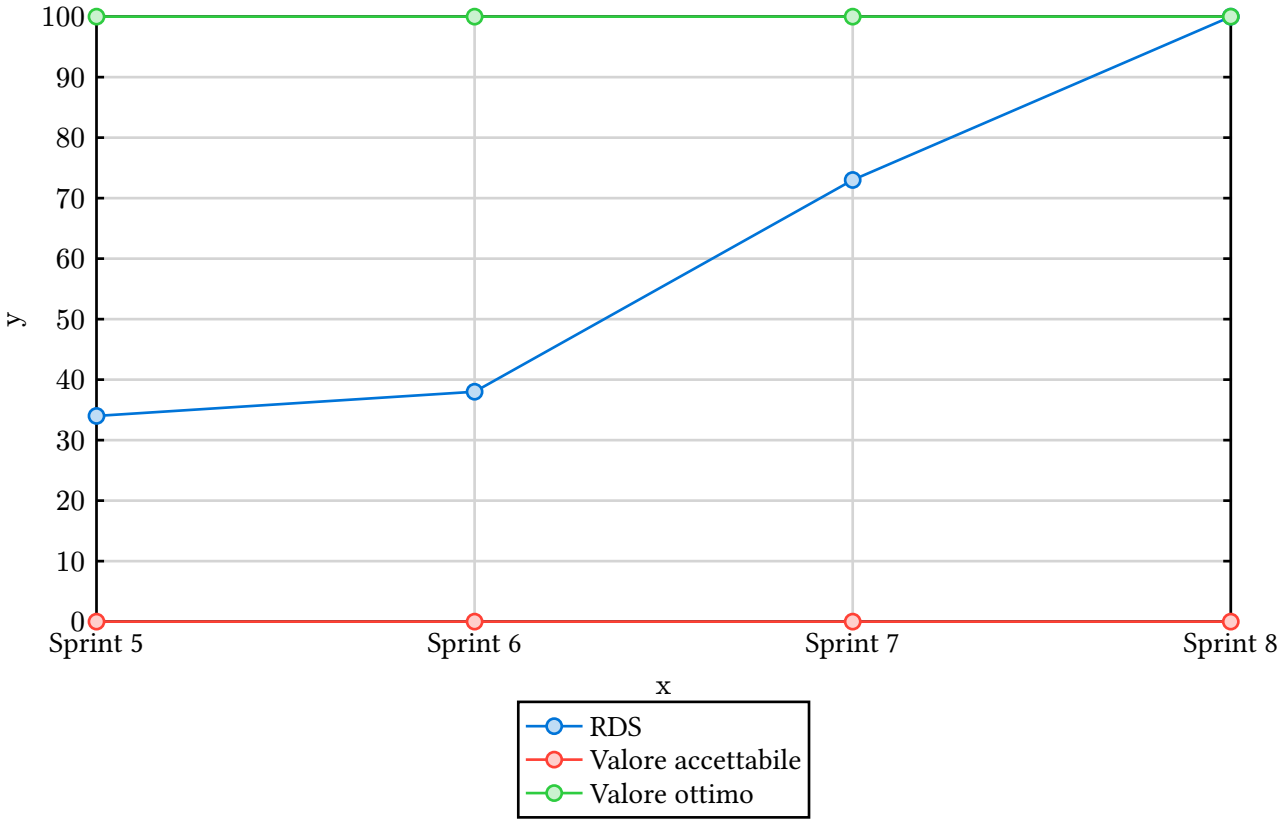

Figura 11: Tabella RDS

4.11.1 PB

Come si può vedere dal grafico diversi requisiti desiderabili, in particolare quelli legati allo storico degli annunci, erano già soddisfatti nel PoC. In parallelo ai requisiti obbligatori il gruppo ha soddisfatto anche quelli desiderabili in quanto aggiungevano molto valore al prodotto. I due andamenti sono infatti simili in quanto i secondi, pur non essendo obbligatori, erano fortemente desiderati dalla proponente.

4.12 Requisiti opzionali soddisfatti (MPD-RFS)

Misura la percentuale di requisiti opzionali soddisfatti dal prodotto.

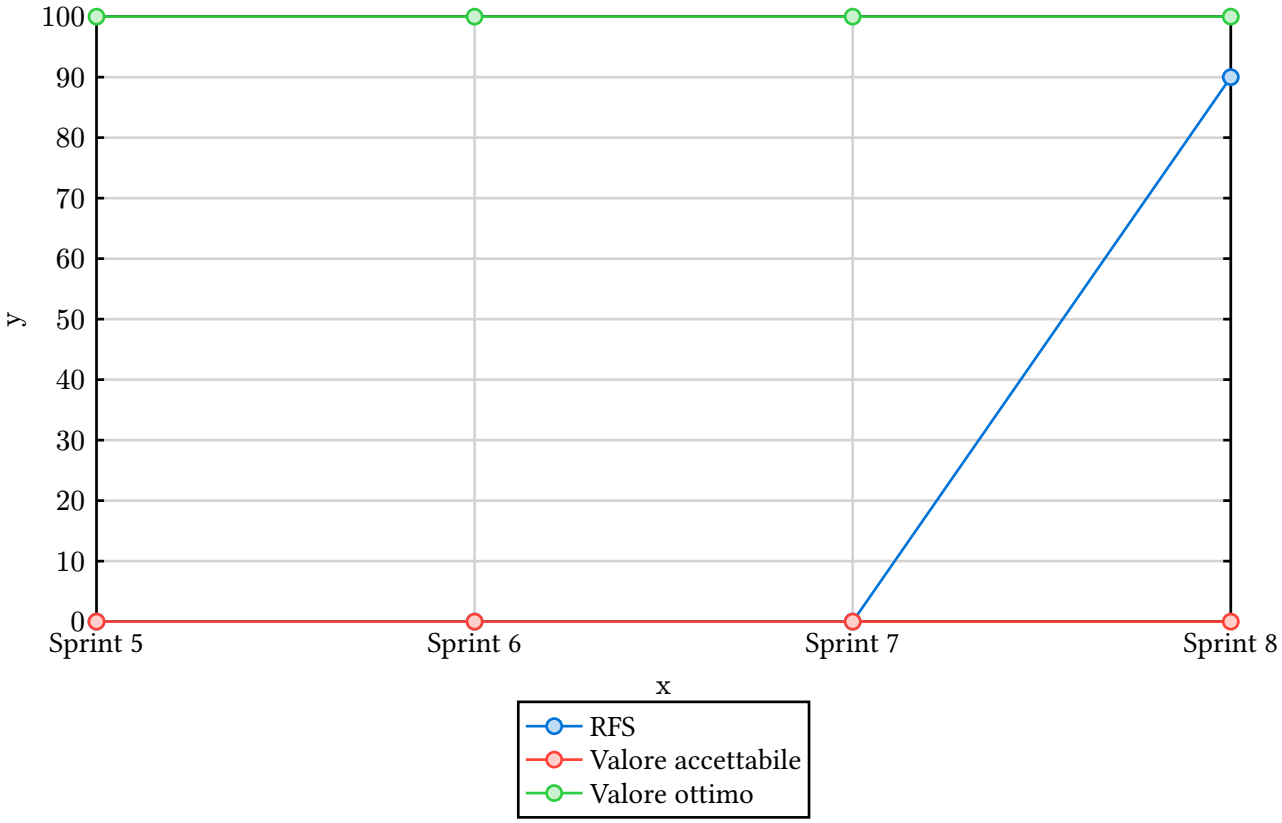

Figura 12: Tabella RFS

4.12.1 PB

Come si evince dal grafico i requisiti opzionali, avendo priorità più bassa degli altri, sono stati risolti nell'ultima iterazione. Avendo ricevuto riscontri positivi per quanto riguarda le funzionalità obbligatorie e desiderabili, il gruppo ha deciso di concentrare parte delle risorse ai requisiti opzionali in quanto potevano dare abbastanza valore al prodotto in rapporto al lavoro necessario per soddisfarli.

4.13 Code coverage (MPD-CC)

Misura la percentuale di codice eseguita durante i *test*.

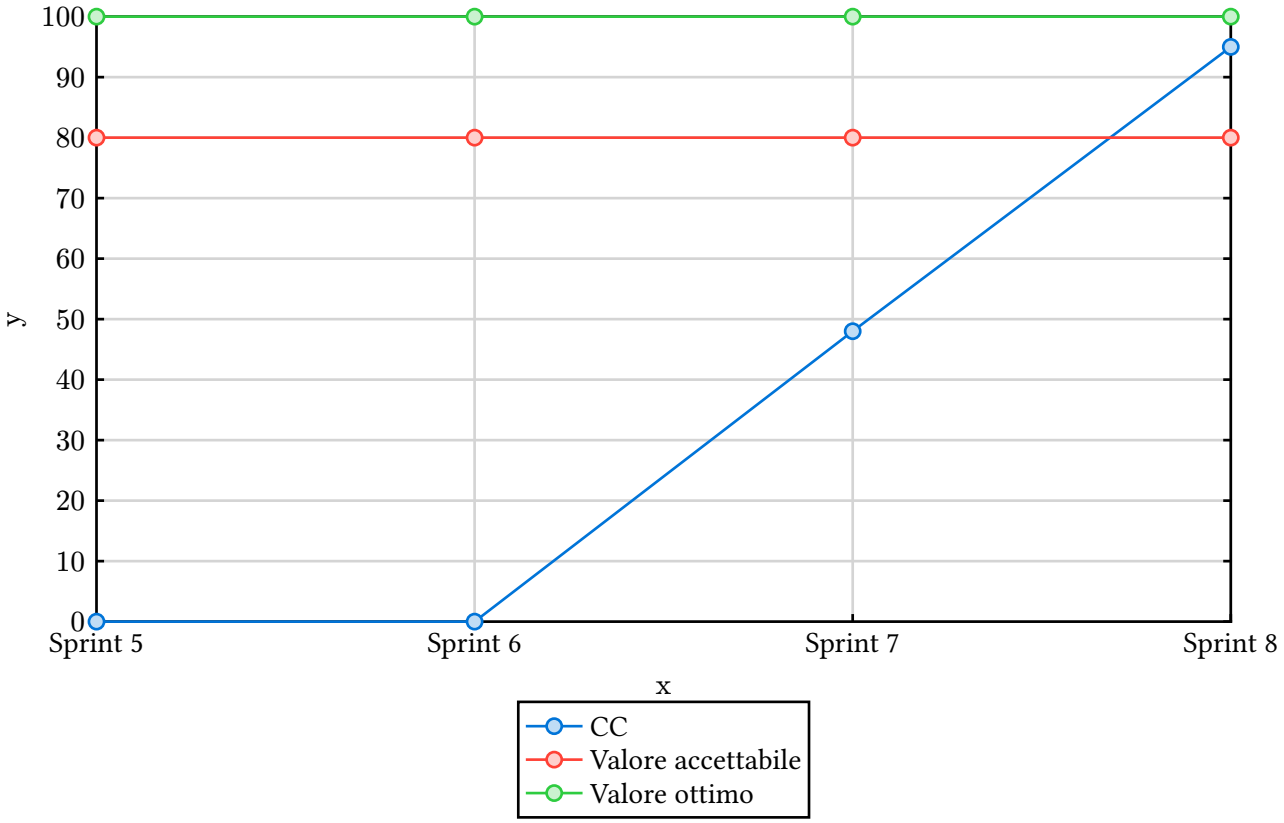

Figura 13: Tabella CC

4.13.1 PB

Come si può vedere dal grafico, il gruppo iniziato a lavorare sulla copertura del codice non appena è stata portata a termine la progettazione, parallelamente quindi alla stesura del codice. Ciò ha portato a soddisfare il valore accettabile del 80% solo durante l'ottava iterazione superandolo e avvicinandosi al valore ottimo con una copertura del 95%.

4.14 Branch coverage (MPD-BC)

Calcola la percentuale di rami decisionali (*branch*) del codice eseguiti durante i *test*.

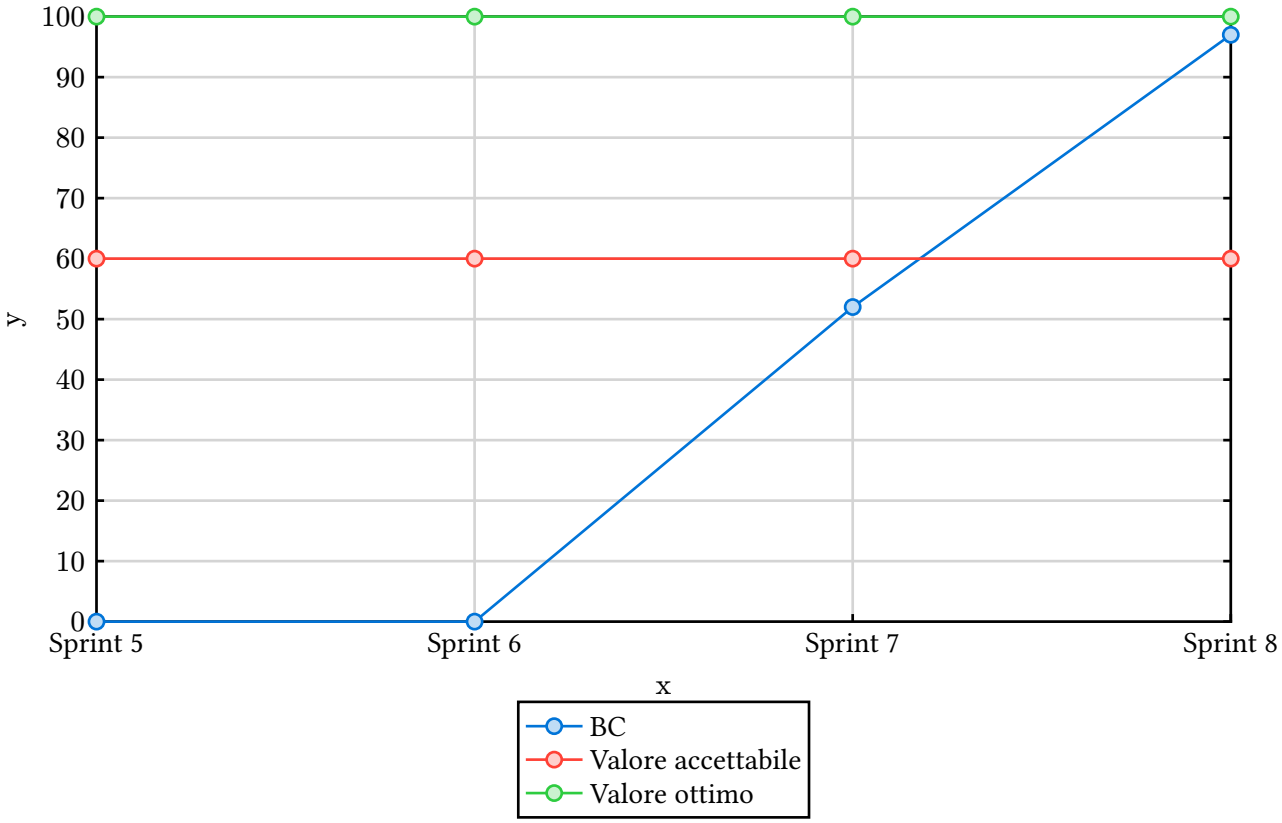

Figura 14: Tabella BC

4.14.1 PB

Seguendo naturalmente la copertura del codice, il valore di *branch coverage* ha iniziato a salire solo durante il settimo *sprint*, avvicinandosi di molto al valore accettabile, per poi salire fino al 97% nell'ultima iterazione. Questo dimostra come l'implementazione dei *test* prodotti dal *team* sia stata efficace e dimostra l'affidabilità del codice.

4.15 Passed test cases percentage (MPD-PTCP)

Misura la percentuale di casi di *test* superati rispetto al totale dei *test* eseguiti.

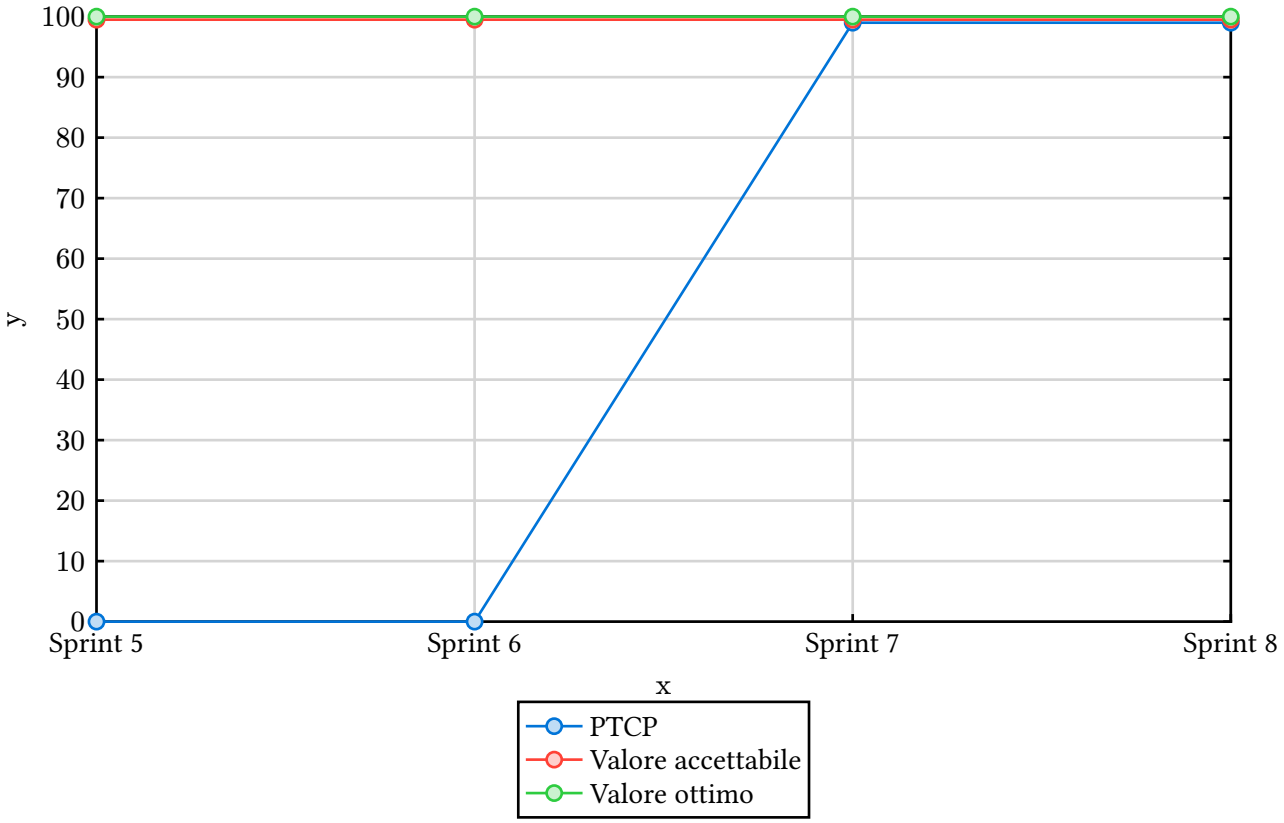

Figura 15: Tabella PTCP

4.15.1 PB

Come si evince dal grafico tutti i *test* sviluppati sono stati sempre superati. Si fa notare come questa condizione sia imperativa affinché si voglia distribuire un prodotto di qualità. Il valore accettabile e l'ottimo infatti coincidono al 100%. Si ricorda tuttavia che per quanto ci si impegni a sviluppare dei *test* esaustivi il fatto che questi vengano completamente superati non è garanzia di un *software* privo di errori o lacune.

4.16 Complessità ciclomatica per metodo (MPD-CCM)

Valuta la complessità per metodo del codice sorgente tramite la misurazione del numero di cammini indipendenti attraverso il grafo di controllo del flusso.

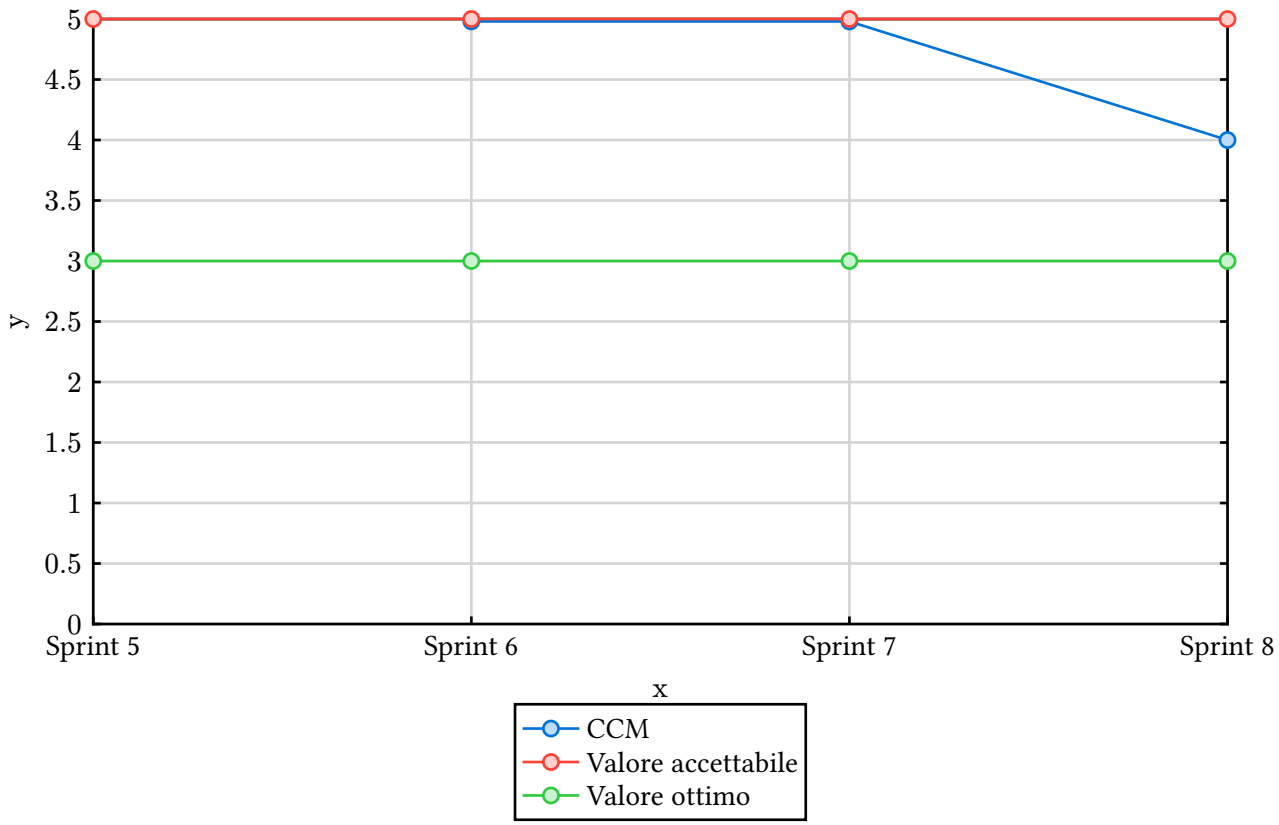

Figura 16: Tabella CCM

4.16.1 PB

Il gruppo ha deciso di appoggiarsi a uno strumento di analisi statica per il calcolo della complessità ciclomatica. Grazie all'analisi in supporto allo sviluppo del codice il gruppo è riuscito a garantire un valore per il **CCM** inferiore a cinque, il che lo fa rientrare nel valore accettabile per questa metrica.

4.17 Code smell (MPD-CS)

Rileva potenziali problemi di progettazione o codice che potrebbero richiedere manutenzione. Segnala parti del codice che potrebbero non essere ottimali e che potrebbero causare difficoltà nel futuro, come un'architettura poco chiara o sezioni di codice ripetitive.

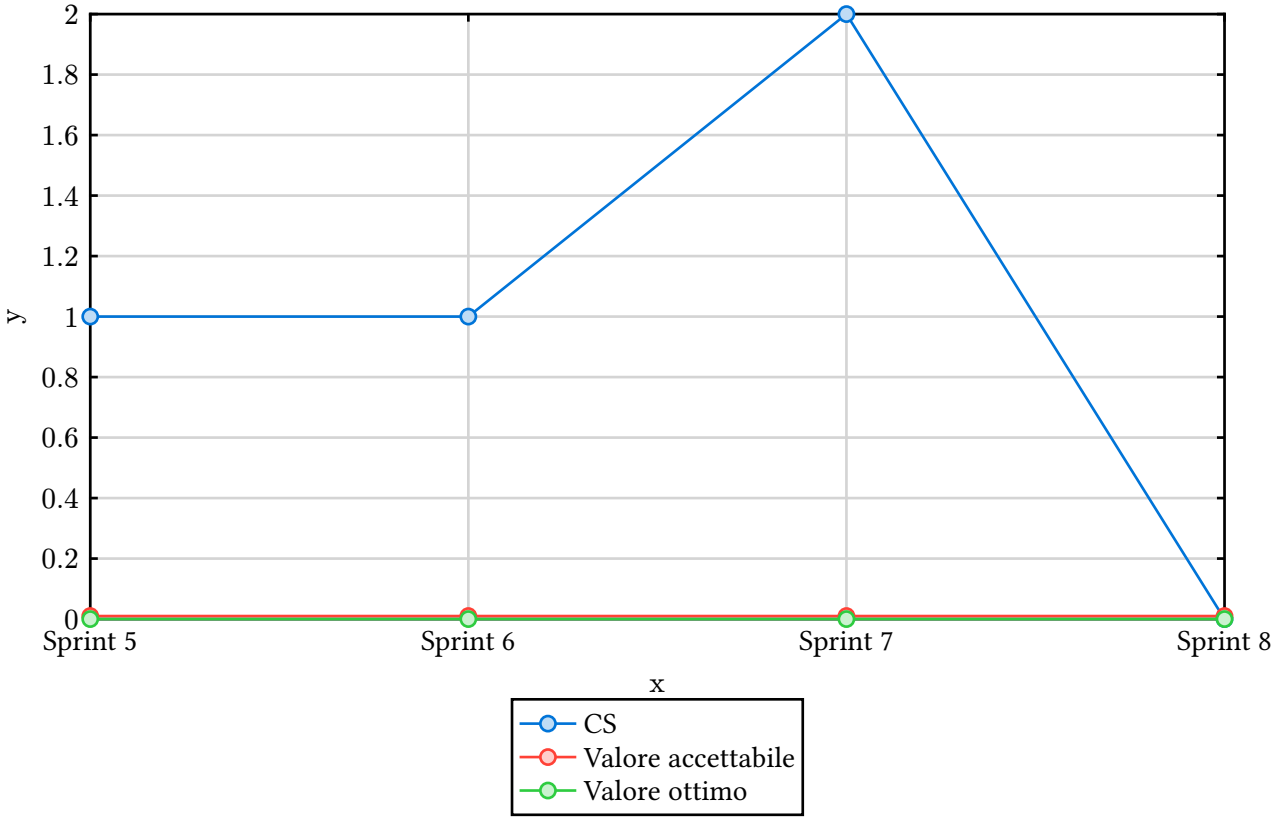

Figura 17: Tabella CS

4.17.1 PB

Il prodotto *software* finale è risultato dell'ulteriore sviluppo del PoC, sostenuto dalla progettazione. Questo processo si è svolto in maniera graduale durante gli *sprint* con alcune complicanze durante il penultimo a causa di un cambiamento di tecnologia nel *backend*. Al termine della *baseline* tuttavia si è riusciti a portare il valore di **CS** allo zero, in particolare a seguito delle analisi statiche del codice.