Piano di Qualifica

Versione: 1.3.1 23/11/2024

Redattori	Malik Giafar Mohamed Stefano Baso
Verifica	Ion Cainareanu Maria Fuensanta Trigueros Hernandez Marco Perazzolo Malik Giafar Mohamed
Approvazione	Luca Parise
Uso	Esterno

nextsoftpadova@gmail.com

Registro dei cambiamenti

Versione	Data	Autore	Descrizione	Verifica
1.3.1	09/05/2024	Malik Giafar Mohamed	correzione formato test di accettazione	
1.3.0	09/05/2024	Malik Giafar Mohamed	Miglioramento sezioni test di unità, di integrazione e di accettazione	
1.2.1	04/05/2025	Stefano Baso	Aggiornamento ultimi verbali per indice di glupease	Malik Giafar Mohamed
1.2.0	04/05/2025	Stefano Baso	Aggiunta test di unità e integrazione	Malik Giafar Mohamed
1.1.0	05/04/2025	Stefano Baso	Aggiunte metriche in qualità di processo	Malik Giafar Mohamed

Indice

Elenco delle immagini

Elenco delle tabelle

1 Scopo del documento

Il *Piano di Qualifica*[G] è un documento soggetto a modifiche incrementali, finalizzate principalmente alla definizione delle *metriche*[G] di valutazione del prodotto. Tali metriche saranno stabilite in conformità ai requisiti e alle aspettative del proponente, con l'obiettivo di determinare correttamente la qualità del prodotto attraverso un processo di miglioramento continuo. Questo approccio tende ad evolversi nel tempo, in particolare una volta stabilita una linea guida.

Il presente documento si propone di:

- Definire le metriche e le metodologie di controllo e misurazione.
- Stabilire quantità, qualità dei *test*[G] e relative metriche.
- Descrivere l'applicazione dei test e documentarne i risultati, valutando la conformità rispetto alle attese e alle metriche definite.

Il documento sarà soggetto a modifiche e integrazioni durante il corso del progetto, in particolare durante le fasi di analisi e progettazione, e quindi non può essere considerato come definitivo.

1.1 Scopo del prodotto

Il prodotto, un plug-in per Visual Studio Code chiamato "Requirement Tracker", è progettato per automatizzare il tracciamento dei *requisiti*[G] nei progetti software complessi, con un focus particolare sull'ambito embedded. L'obiettivo principale è migliorare la qualità e la chiarezza dei requisiti, fornendo suggerimenti basati sull'analisi di un'intelligenza artificiale, riducendo al contempo i tempi e gli errori legati alla verifica manuale dell'implementazione nel codice sorgente. Il plug-in adotta un'architettura modulare che consente un'estensibilità semplice, rendendolo facilmente adattabile a nuove funzionalità o esigenze future. Inoltre, supporta gli sviluppatori avendo la capacità di utilizzare documenti tecnici come knowledge, ad esempio datasheet e manuali, permette di garantire una corretta copertura dei requisiti.

1.2 Glossario

I termini ambigui che necessitano di una spiegazione sono contrassegnati da una [G] come apice alla loro prima occorrenza nei documenti. Tutti i termini da glossario sono riportati in ordine alfabetico nell'omonimo documento.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- Analisi dei Requisiti v2.0.0
- Norme di Progetto v2.0.0

1.3.2 Riferimenti informativi

Materiale didattico del corso

- Qualità di prodotto
  - https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T07.pdf
- Qualità di processo
  - https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T08.pdf
- Indice di Gulpease
  - https://www.ilc.cnr.it/dylanlab/apps/texttools/
- ISO/IEC 9126

2 Qualità di processo

2.1 Scopo ed obiettivi

La qualità di un sistema è determinata dai processi che lo costituiscono e viene misurata attraverso l'uso di metriche specifiche, atte a valutare tali processi e verificarne il raggiungimento degli obiettivi di qualità stabiliti. Il modello di riferimento è il *Ciclo di Deming*[G] o PDCA (Plan - Do - Check - Act), il quale consente di avere un miglioramento continuo tramite una gestione strutturata delle attività. Questo approccio si basa su una *pianificazione*[G] accurata, il monitoraggio mediante *metriche*[G] definite e l'integrazione dei risultati ottenuti nella fase di produzione operativa.

Di seguito, vengono presentati i processi identificati e i corrispondenti livelli di qualità prefissati. Per ciascuna metrica è fornita una descrizione che ne illustra le modalità di applicazione e definisce i valori considerati accettabili nel contesto delle verifiche di qualità.

2.2 Processi primari

2.2.1 Fornitura

In questa fase del processo vengono analizzate tutte le scelte effettuate durante lo sviluppo, verificandone la conformità con gli obiettivi stabiliti nelle diverse fasi del progetto. Vengono definite le misure da implementare, assicurando il rispetto dei termini e delle condizioni prestabiliti. L'obiettivo principale è garantire che la *fornitura*[G] sia allineata alle aspettative, sia in termini di risorse impiegate che di risultati ottenuti.

Un concetto chiave in questo contesto è l'MPC (Minimum Predictive Capability), una metrica fondamentale per valutare l'affidabilità di un modello di apprendimento automatico nel generare risultati accurati. L'MPC rappresenta il livello minimo di precisione che un modello deve raggiungere per essere considerato accettabile, contribuendo a garantire che le previsioni siano coerenti con gli standard richiesti.

Di seguito sono descritte le principali metriche e calcoli associati che verranno riportati nella tabella sottostante mettendo in relazione il valore plausibile e il valore ottimale:

- BAC (Budget At Completion): Costo totale preventivato per il completamento del progetto.

$$\mathrm{BAC} = \sum \text{costi previsti}$$

- EAC (Estimated At Completion): Valore stimato per i compiti rimanenti.

$$\mathrm{EAC} = \frac{\mathrm{BAC}}{\mathrm{CPI}}$$

- CPI (Cost Performance Index): Indice di prestazione dei costi, misura l'*efficienza*[G] con cui il *budget*[G] viene utilizzato. Un valore > 1 indica che il progetto sta spendendo meno del previsto, mentre un valore < 1 indica che sta spendendo di più del previsto.

$$\mathrm{CPI} = \frac{\mathrm{EV}}{\mathrm{AC}}$$

- ETC (Estimated To Completion): Stima del costo finale aggiornato alla data di misurazione.

$$\mathrm{ETC} = \mathrm{EAC} - \mathrm{AC}$$

- EV (Earned Value): Valore ottenuto fino al momento attuale.

$$\mathrm{EV} = \left(\frac{\%\ \text{lavoro svolto}}{100}\right) * \mathrm{EAC}$$

- PV (Planned Value): Valore pianificato fino al momento attuale.

$$\mathrm{PV} = \left(\frac{\%\ \text{lavoro pianificato}}{100}\right) * \mathrm{BAC}$$

- AC (Actual Cost): Budget effettivamente speso fino al momento attuale.

$$\mathrm{AC} = \sum \text{costi effettivi}$$

- CV (Cost Variance): Differenza tra il valore ottenuto (EV) e il costo effettivo (AC).

$$\mathrm{CV} = \mathrm{EV} - \mathrm{AC}$$

- SV (Schedule Variance): Differenza tra il valore ottenuto (EV) e quello pianificato (PV). Un valore negativo indica un ritardo rispetto alla pianificazione.

$$\mathrm{SV} = \mathrm{EV} - \mathrm{PV}$$

- BV (Budget Variance): Differenza rispetto al budget preventivato.

$$\mathrm{BV} = \mathrm{AC} - \mathrm{CV}$$

Codice	Descrizione	Soglia accettabile	Ottimo
MPC01	Estimated at Completion	± 5% rispetto al *preventivo*[G]	Corrispondente al preventivo
MPC02	Estimated to Completion	>= 0	≤ EAC
MPC03	Earned Value	>= 0	≤ EAC
MPC04	Planned Value	>= 0	≤ BAC
MPC05	Actual Cost	>= 0	≤ EAC
MPC06	Cost Variance	≥ −5%	≥ 0%
MPC07	Schedule Variance	≥ −10%	≥ 0%
MPC08	Budget Variance	± 10%	≤ 0%

Table 1: Metriche di fornitura

Questi indicatori consentono di monitorare l'andamento del progetto in termini di costi, tempi e precisione delle previsioni, supportando una gestione efficiente e mirata.

2.2.2 Sviluppo

2.2.2.1 Progettazione di dettaglio

Indice per la media del numero di metodi presenti in ogni *package*[G], un indice alto potrebbe comportare il *refactoring*[G].

Codice	Descrizione	Soglia accettabile	Ottimo
MPC09	Number of Methods	3-11	3-8

Table 2: Metriche di progettazione di dettaglio

2.2.2.2 Codifica

- **BLC** (Bugs for Line of Code) = indice per il numero di righe di codice che possono contenere *bug*[G] o errori.
- **VNUD** (Variabili Non Utilizzate o non Definite) = indice per il numero di variabili utilizzate o non definite, queste sono a tutti gli effetti errori di programmazione che possono comportare bug. Variabili non utilizzate occupano spazio inutilmente in memoria e creano confusione all'interno del codice.

Codice	Descrizione	Soglia accettabile	Ottimo
MPC10	Bugs for Line of Code	0-70	0-25
MPC11	Variabili non utilizzate e non definite	0	0

Table 3: Metriche di codifica

2.2.2.3 Copertura dei test

Percentuale di elementi del sistema come funzionalità o casi d'uso verificati tramite test automatici o manuali. Valutare la qualità della fase di validazione e per identificare eventuali aree del prodotto non ancora testate.

Codice	Descrizione	Soglia accettabile	Ottimo
MPC12	Copertura dei test	>= 70%	>= 90%

Table 4: Copertura dei test

2.2.3 Conformità ai requisiti

2.2.3.1 Percentuale di requisiti soddisfatti

Indica il rapporto tra il numero di requisiti implementati rispetto al totale dei requisiti previsti. Il valore dell'indice valuta quanto il processo di sviluppo sia stato in grado di coprire le esigenze richieste inizialmente.

2.2.3.2 Indice di variazione dei requisiti

Monitora il numero e l'entità delle modifiche apportate ai requisiti nel corso del progetto. Una variazione eccessiva può indicare problemi nelle fasi iniziali di analisi o una cattiva gestione delle aspettative.

2.2.3.3 Percentuale di attività completate nei tempi previsti

Rappresenta il rapporto tra il numero di attività concluse entro le scadenze pianificate e il totale delle attività previste. indica l'efficienza organizzativa e la capacità del team di rispettare i tempi definiti nella fase di pianificazione.

Codice	Descrizione	Soglia accettabile	Ottimo
MPC13	Percentuale di requisiti soddisfatti	>= 90%	100%
MPC14	Indice di variazione dei requisiti	<= 20%	<= 10%
MPC15	Attività completate nei tempi previsti	≥ 85%	>= 95%

Table 5: Conformità ai requisiti

2.3 Processi di supporto

2.3.1 Documentazione

La documentazione ha ruolo di supporto, in particolare definisce le norme da seguire durante lo sviluppo, la divisione delle risorse e responsabilità. E' necessario quindi definire una linea guida anche per la redazione dei documenti per evitare ambiguità e renderli chiari

Codice	Nome	Descrizione	Metriche associate
OPC01	Leggibilità dei documenti	Per mantenere una buona comprensione, il documento deve essere leggibile	MPC12
OPC02	Correttezza ortografica	Numero di errori grammaticali o ortografici per documento	MPC13

Table 6: Obiettivo di qualità della documentazione

2.3.1.1 Indice di leggibilità di Gulpease

L'indice di leggibilità di Gulpease è una metrica che valuta la semplicità di un testo in italiano, ideata per stimare quanto sia comprensibile da lettori con livelli diversi di istruzione. L'indice si basa su tre parametri: il numero di lettere, il numero di parole e il numero di frasi. La formula è:

$$\text{Gulpease} = 89 - \frac{\text{N° lettere}}{\text{N° parole}}10 + \frac{\text{N° frasi}}{\text{N° parole}}30$$

Il punteggio varia da 0 a 100, dove valori alti indicano maggiore leggibilità. Tipicamente, un testo comprensibile per chi ha una licenza elementare ha un punteggio sopra 80, mentre per chi possiede una licenza media è sufficiente un punteggio superiore a 60. Il metodo è particolarmente utile per valutare documenti destinati a un pubblico ampio, come testi scolastici o burocratici.

Codice	Descrizione	Soglia accettabile	Ottimo
MPC12	Indice di leggibilità di Gulpease	GULP ≥ 40	GULP ≥ 60

Table 7: Obiettivo di leggibilità

2.3.1.2 Indice errori ortografici

Per raggiungere l'ottimo anche nella documentazione bisogna raggiungere la massima correttezza in termini di grammatica e ortografia.

Codice	Descrizione	Soglia accettabile	Ottimo
MPC13	Numero errori ortografici	0	0

Table 8: Obiettivo di leggibilità

3 Qualità del prodotto

Per mantenere ed assicurare la qualità del prodotto software il gruppo ha adottato il modello di qualità stabilito dallo standard ISO/IEC 9126, adattandolo alle esigenze e requisiti del progetto. Tale standard propone una serie di metriche e regole per migliorare l'organizzazione dei processi e di conseguenza la qualità del prodotto. Di seguito verranno elencate e descritte le metriche che verranno utilizzate.

3.1 Efficienza

L'efficienza indica il tempo di elaborazione della richiesta da parte del software per raggiungere il risultato.

Codice	Descrizione	Soglia accettabile	Ottimo
MPDS01	Tempo di risposta medio	3 secondi	2 secondi

Table 9: Metriche di tempo medio

3.2 Usabilità

L'*usabilità*[G] riguarda l'esperienza dell'utente nell'interagire con il nostro prodotto, capirne il suo funzionamento e apprezzarne le sue funzioni.

Codice	Nome	Descrizione	Metriche associate
OPDS01	Usabilità del prodotto	Il prodotto deve essere facilmente usabile dall'utente in modo da raggiungere il più velocemente possibile quello che cerca	MPDS01

Table 10: Obiettivo di funzionalità

3.2.1 Facilità di utilizzo

Questa rappresenta la velocità con cui l'utente trova quello che sta cercando, calcolata in base al numero di click minimo che si deve effettuare per arrivare all'obiettivo.

Codice	Descrizione	Soglia accettabile	Ottimo
MPDS01	Facilità di utilizzo	FU ≤ 3	FU ≤ 5

Table 11: Obiettivo di usabilità

3.3 Manutenibilità

La manutenibilità del software è la facilità con cui può essere modificato, corretto, adattato o aggiornato.

Codice	Nome	Descrizione	Metriche associate
OPDS02	Analizzabilità del prodotto	Una facile analisi del codice permette di localizzare in tempi minimi il blocco di codice che riguarda l'errore o l'aggiornamento	MPDS03 MPDS04 MPDS06
OPDS03	Modificabilità del prodotto	Permette una manutenzione più agevolata per la correzione	MPDS05 MPDS02

Table 12: Obiettivo di manutenibilità

3.3.1 Metriche

- **Complessità ciclomatica**: misura la complessità strutturale di un programma basandosi sul grafo di controllo del flusso del codice. In particolare, rappresenta il numero di cammini *linearmente indipendenti*[G] attraverso il codice sorgente. Viene calcolata con la seguente formula:

$$V(G) = E - N + 2P$$

in cui:

- E è il numero di archi (transizioni tra i nodi),
- N è il numero di nodi (blocchi di codice o decision points),
- P è il numero di componenti connesse (tipicamente P = 1 per un singolo metodo o funzione)

- **Profondità della gerarchia**: indica il numero massimo di livelli di ereditarietà in una *gerarchia*[G] di classi. Una gerarchia più profonda può favorire il riuso del codice ma aumenta la complessità e il rischio di propagazione degli errori. Per un *design*[G] più manutenibile, è preferibile mantenere la profondità entro limiti ragionevoli (3-4 livelli), favorendo la composizione rispetto a una struttura gerarchica troppo profonda.
- **Parametri per metodo**: indica il numero di parametri per metodo. Un indice basso rappresenta un numero basso di parametri richiesti dal metodo, di conseguenza risulta di più facile comprensione e utilizzo.
- **Code Smell**: indice che rappresenta indicatori qualitativi di potenziali problemi nel codice. E' utile per valutare la leggibilità, la modificabilità, e la testabilità del codice. Si dividono in:
  - Duplicated Code: frammenti di codice identici o simili in più punti, che aumentano i costi di manutenzione perché le modifiche devono essere replicate ovunque.
  - Long Methods: metodi eccessivamente lunghi, che riducono la leggibilità e la comprensione del codice.
  - God Class: una classe con troppe responsabilità (violazione del *principio di Single Responsibility*[G]), difficile da testare e modificare.
  - High Coupling: una forte *dipendenza*[G] tra componenti, che rende il sistema rigido e suscettibile a errori quando una parte viene modificata.
  - Low Cohesion: componenti con funzionalità eterogenee che non si relazionano strettamente, rendendo il codice più complesso da comprendere.
- **Facilità di comprensione**: rappresenta il rapporto tra commenti presenti e codice totale per capirne il suo funzionamento.

Codice	Descrizione	Soglia accettabile	Ottimo
MPDS02	Profondità di gerarchia	PG <= 3	PG <= 2
MPDS03	Parametri per metodo	PPM <= 8	PPM <= 4
MPDS04	Complessità ciclomatica	CC <= 20	CC <= 10
MPDS05	Code smell	CS <= 50	CS <= 10
MPDS06	Facilità di comprensione	FC >= 0.10	FC >= 0.20

Table 13: Metriche di manutenibilità

3.4 Affidabilità

L'affidabilità riguarda il livello minimo di prestazioni da mantenere durante l'uso in determinate situazioni.

Codice	Nome	Descrizione	Metriche associate
OPDS04	Prodotto maturo	Evita errori o malfunzionamenti durante l'utilizzo	MPDS07 MPDS10
OPDS05	Tolleranza agli errori	Mantiene il livello di prestazioni anche durante un uso scorretto o in presenza di errori	MPDS11 MPDS08 MPDS09

Table 14: Obiettivo di affidabilità

3.4.1 Metriche

- **Code Coverage**: percentuale di codice eseguito nei test. Un indice di copertura del codice alto significa che è stato testato più codice, riducendo quindi la presenza di bug.
- **Branch Coverage**: percentuale di copertura di tutti i *branch*[G] all'esecuzione del codice. Il compito dei test è anche quello di verificare tutti i rami esistenti per verificarne la correttezza.
- **Presenza di vulnerabilità**: indice per il numero di vulnerabilità ancora presenti nel codice.
- **Presenza di bug**: indice per il numero di bug ancora presenti nel codice.
- **Successo dei test**: indice in percentuale relativo al successo dei test definiti dai *programmatori*[G].

Codice	Descrizione	Soglia accettabile	Ottimo
MPDS07	Code Coverage	CC >= 75%	100%
MPDS08	Presenza di vulnerabilità	VLN <= 2	0
MPDS09	Presenza di bug	BUG <= 20%	BUG <= 5%
MPDS10	Branch Coverage	BC >= 75%	100%
MPDS11	Successo dei test	>= 75%	100%

Table 15: Metriche di affidabilità

3.5 Funzionalità

La funzionalità è la capacità di fornire funzioni / azioni per ogni esigenza stabilita.

Codice	Nome	Descrizione	Metriche associate
OPDS06	Appropriatezza del prodotto	Fornire le funzioni richieste ed essere in linea con i	MPDS12 MPDS13

Codice	Nome	Descrizione	Metriche associate
		requisiti fissati nell'Analisi dei Requisiti	

Table 16: Obiettivo di funzionalità

3.5.1 Metriche

- **Requirement coverage**: indice della copertura dei requisiti descritti nell'*Analisi dei Requisiti*[G]. Viene calcolato con il rapporto percentuale tra numero di requisiti rispettati e numero di requisiti totali, con la formula:

$$\mathrm{RC} = \frac{R_{\mathrm{RISP}}}{R_{\mathrm{TOT}}} 100$$

- **Requisiti obbligatori soddisfatti**: indice della copertura dei *requisiti obbligatori*[G] descritti nell'Analisi dei Requisiti. Viene calcolato con il rapporto percentuale tra numero di requisiti rispettati e numero di requisiti totali, con la formula:

$$\mathrm{RC} = \frac{R_{\mathrm{ROS}}}{R_{\mathrm{ROT}}} 100$$

Codice	Descrizione	Soglia accettabile	Ottimo
MPDS12	*Requirement coverage*[G]	RC >= 75%	100%
MPDS13	Requisiti obbligatori soddisfatti	100%	100%

Table 17: Obiettivo di usabilità

4 Test e specifiche

Il seguente capitolo presenta in maniera dettagliata le strategie e scelte di testing, atte a garantire la correttezza del prodotto e facilitarne la *validazione*[G]. Viene adottato il Modello a V, in cui ad ogni fase di sviluppo corrisponde una fase di *verifica*[G] e validazione, garantendo quindi un controllo strutturato del processo.

Il modello è suddiviso in tre parti:

- **Fase di sviluppo** (lato sinistro): definisce e dettaglia i requisiti e la progettazione del sistema.
  - Requirements Gathering: Definizione dei requisiti.
  - System Analysis: Analisi funzionale e tecnica.
  - Software Design: Progettazione architetturale del sistema.

  - Module Design: Definizione dei singoli *moduli software*[G].
- **Coding**: avviene l'implementazione vera e propria del software.
- **Fase di testing e validazione** (lato destro): verifica che ogni fase di sviluppo soddisfi i requisiti stabiliti.
  - Unit Testing: Test sui singoli moduli.
  - Integration Testing: Verifica dell'integrazione tra i moduli.
  - System Testing: Validazione dell'intero sistema.
  - Acceptance Testing: Verifica finale rispetto ai requisiti del cliente.

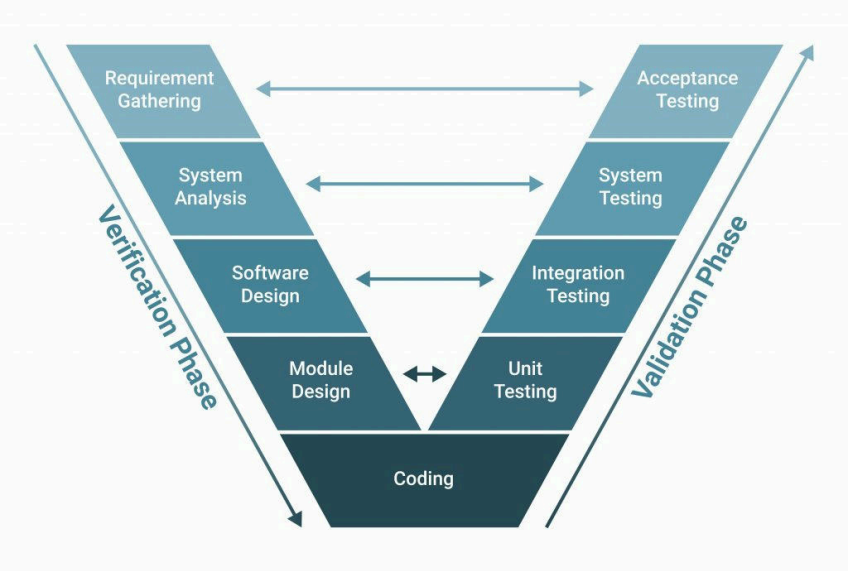

Figure 1: Modello a V

Questo modello prevede una stretta corrispondenza tra sviluppo e testing, assicurando che ogni fase sia verificata e validata, riducendo il rischio di errori.

4.1 Tipologie di test

4.1.1 Organizzazione dei test:

I test nel progetto sono suddivisi in due cartelle all'interno della principale `test`:

- `test/unit`: contiene i test di unità, incentrati sulla verifica dei singoli componenti (modelli, adattatori e servizi) del sistema.
- `test/integration`: contiene i test di integrazione, che verificano l'interazione tra diversi componenti o tra l'applicazione e servizi esterni.

I file di test seguono la convenzione di denominazione `*.spec.ts` per i test di unità e `*.int.spec.ts` per i test di integrazione.

4.1.2 Strumenti Utilizzati e Integrazione di Jest:

- **Jest**: È il framework di testing JavaScript principale utilizzato nel progetto. Jest fornisce:
  - Un ambiente di esecuzione per i test.
  - Funzioni globali come `describe()` per raggruppare i test in suite, e `it()` o `test()` per definire i singoli casi di test.
  - Un potente sistema di asserzioni tramite la funzione `expect()` combinata con vari matchers (es. `toBe()`, `toBeDefined()`, `toContain()`, `toHaveBeenCalledWith()`).
  - Funzionalità di mocking avanzate, tra cui `jest.mock()` per mockare interi moduli (come `axios`) e `jest.fn()` per creare funzioni mock flessibili che possono tracciare chiamate, definire valori di ritorno e implementazioni simulate.
  - Gestione di test asincroni tramite `async/await`.
- `@nestjs/testing`: questa libreria di NestJS facilita il testing dei componenti NestJS (moduli, controller, provider). La classe `Test` e il metodo `createTestingModule()` sono usati per creare un ambiente di test che rispecchia il sistema di dependency injection di NestJS, permettendo di istanziare e testare i componenti in modo isolato o integrato.
- **supertest**: utilizzato nei test di integrazione a livello applicativo (`application.int.spec.ts`) per effettuare richieste HTTP all'applicazione in esecuzione e verificare le risposte. Semplifica il testing degli endpoint API.
- **axios (mockato)**: nei test di integrazione per componenti che interagiscono con API esterne (come `OllamaApiAdapter`), `axios` viene mockato per controllare le risposte delle API e testare il comportamento dell'adapter in diverse condizioni senza effettuare chiamate di rete reali.

4.1.3 Test di Unità

I *test di unità*[G] valutano il corretto funzionamento delle singole unità di codice all'interno del software. Un'unità di codice è una funzione, una classe o qualsiasi componente che svolge un'attività specifica in modo indipendente rispetto al resto del sistema. I test di unità svolti sono stati i seguenti:

Codice	Descrizione	Esito
TU-001	`TraceabilityManager` deve restituire `true` per requisito con tracciabilità	Superato

Codice	Descrizione	Esito
TU-002	`TraceabilityManager` deve restituire `false` per requisito senza tracciabilità	Superato
TU-003	`TraceabilityManager` deve restituire `false` per requisito inesistente	Superato
TU-004	`TraceabilityManager` deve controllare prima i requisiti analizzati	Superato
TU-005	`TraceabilityManager` deve aggiornare il collegamento di tracciabilità nei requisiti di base	Superato
TU-006	`TraceabilityManager` deve aggiornare il collegamento di tracciabilità nei requisiti analizzati	Superato
TU-007	`TraceabilityManager` deve restituire `false` per requisito inesistente	Superato
TU-008	`TraceabilityManager` deve restituire `false` per indice di tracciabilità inesistente	Superato
TU-009	`TraceabilityManager` deve restituire `false` per requisito senza tracciabilità	Superato
TU-010	`TraceabilityManager` deve aggiungere nuovi collegamenti di tracciabilità	Superato
TU-011	`TraceabilityManager` non deve aggiungere collegamenti di tracciabilità duplicati	Superato
TU-012	`TraceabilityManager` deve gestire tracciabilità indefinita nel requisito analizzato	Superato
TU-013	`TraceabilityManager` deve restituire il requisito originale se non ci sono risultati di tracciabilità	Superato
TU-014	`TraceabilityManager` deve restituire il requisito originale se i risultati di tracciabilità sono indefiniti	Superato
TU-015	`TraceabilityManager` non deve modificare l'oggetto requisito originale	Superato
TU-016	`RequirementsFilterManager` deve impostare la modalità di ordinamento	Superato
TU-017	`RequirementsFilterManager` deve restituire la modalità di ordinamento predefinita inizialmente	Superato
TU-018	`RequirementsFilterManager` deve restituire la modalità di ordinamento corrente dopo l'impostazione	Superato
TU-019	`RequirementsFilterManager` deve impostare il termine di ricerca	Superato

Codice	Descrizione	Esito
TU-020	`RequirementsFilterManager` deve restituire tutti i requisiti quando non viene applicato alcun filtro	Superato
TU-021	`RequirementsFilterManager` deve filtrare i requisiti per termine di ricerca nel testo del requisito	Superato
TU-022	`RequirementsFilterManager` deve filtrare i requisiti per termine di ricerca nel percorso del file	Superato
TU-023	`RequirementsFilterManager` deve filtrare i requisiti per termine di ricerca nell'ID del requisito	Superato
TU-024	`RequirementsFilterManager` deve combinare i requisiti con le loro versioni analizzate	Superato
TU-025	`RequirementsFilterManager` deve ordinare i requisiti per ID quando è impostata la modalità ID_ASC	Superato
TU-026	`RequirementsFilterManager` deve ordinare prima i requisiti analizzati quando è impostata la modalità ANALYZED_FIRST	Superato
TU-027	`RequirementsFilterManager` deve ordinare prima i requisiti non analizzati quando è impostata la modalità UNANALYZED_FIRST	Superato
TU-028	`RequirementsFilterManager` non deve cambiare l'ordine quando è impostata la modalità DEFAULT	Superato
TU-029	`RequirementsFilterManager` deve gestire la ricerca senza distinzione tra maiuscole e minuscole	Superato
TU-030	`RequirementsFilterManager` deve gestire il termine di ricerca vuoto	Superato
TU-031	`RequirementsFilterManager` deve gestire i requisiti senza tracciabilità	Superato
TU-032	`RequirementsDataManager` deve impostare correttamente i requisiti di base	Superato
TU-033	`RequirementsDataManager` deve identificare e memorizzare i requisiti analizzati	Superato
TU-034	`RequirementsDataManager` deve gestire i requisiti con proprietà mancanti	Superato
TU-035	`RequirementsDataManager` deve identificare i requisiti analizzati con varie proprietà di analisi	Superato
TU-036	`RequirementsDataManager` deve restituire oggetto vuoto quando non esistono requisiti analizzati	Superato

Codice	Descrizione	Esito
TU-037	`RequirementsDataManager` deve restituire tutti i requisiti analizzati	Superato
TU-038	`RequirementsDataManager` deve restituire array vuoto quando non esistono requisiti	Superato
TU-039	`RequirementsDataManager` deve restituire tutti i requisiti	Superato
TU-040	`RequirementsDataManager` deve aggiornare un requisito analizzato esistente	Superato
TU-041	`RequirementsDataManager` deve aggiungere un nuovo requisito analizzato se non esiste	Superato
TU-042	`RequirementsDataManager` deve restituire `undefined` per requisito inesistente	Superato
TU-043	`RequirementsDataManager` deve restituire il requisito corretto per ID esistente	Superato
TU-044	`RequirementsDataManager` deve restituire `undefined` per requisito analizzato inesistente	Superato
TU-045	`RequirementsDataManager` deve restituire il requisito analizzato corretto per ID esistente	Superato
TU-046	`RequirementsDataManager` deve restituire `false` per requisito inesistente nei risultati di analisi	Superato
TU-047	`RequirementsDataManager` deve restituire `false` per requisito senza dati di analisi	Superato
TU-048	`RequirementsDataManager` deve restituire `true` per requisito con finalPassed	Superato
TU-049	`RequirementsDataManager` deve restituire `true` per requisito con punteggio di qualità	Superato
TU-050	`RequirementsDataManager` deve restituire `true` per requisito con punteggio di conformità	Superato
TU-051	`RequirementsDataManager` deve restituire `true` per requisito con problemi	Superato
TU-052	`RequirementsDataManager` deve restituire `true` per requisito con suggerimenti	Superato
TU-053	`RequirementsDataManager` deve restituire `false` per requisito con array vuoti di problemi e suggerimenti	Superato

Codice	Descrizione	Esito
TU-054	`RequirementsAnalysisManager` deve chiamare il servizio API con i parametri corretti	Superato
TU-055	`RequirementsAnalysisManager` deve gestire errori API	Superato
TU-056	`RequirementsAnalysisManager` deve gestire errori HTTP con codice di stato	Superato
TU-057	`RequirementsAnalysisManager` deve chiamare il servizio di parsing del codice con le informazioni di tracciabilità	Superato
TU-058	`RequirementsAnalysisManager` deve analizzare un singolo testo di requisito	Superato
TU-059	`RequirementsAnalysisManager` deve analizzare più testi di requisito in batch	Superato
TU-060	`RequirementsAnalysisManager` deve gestire nessun file C trovato	Superato
TU-061	`RequirementsAnalysisManager` deve gestire il caso senza frammenti di codice estratti	Superato
TU-062	`RequirementsAnalysisManager` non deve processare alcun requisito se l'array è vuoto	Superato
TU-063	`RequirementsAnalysisManager` deve processare ogni requisito e chiamare la callback	Superato
TU-064	`RequirementsAnalysisManager` deve saltare i requisiti senza tracciabilità	Superato
TU-065	`RequirementsAnalysisManager` deve rispettare il token di cancellazione	Superato
TU-066	`RequirementsAnalysisManager` deve gestire errori durante l'analisi	Superato
TU-067	`ApprovalManager` deve impostare lo stato di approvazione per requisito esistente	Superato
TU-068	`ApprovalManager` deve aggiornare lo stato di approvazione esistente	Superato
TU-069	`ApprovalManager` deve restituire `false` per requisito inesistente	Superato
TU-070	`ApprovalManager` deve restituire `undefined` per requisito senza stato di approvazione	Superato
TU-071	`ApprovalManager` deve restituire lo stato di approvazione per requisito con stato	Superato
TU-072	`ApprovalManager` deve restituire `undefined` per requisito inesistente	Superato
TU-073	`ApprovalManager` deve restituire `true` se il requisito ha tracciabilità	Superato

Codice	Descrizione	Esito
TU-074	`ApprovalManager` deve restituire `false` se il requisito non ha tracciabilità	Superato
TU-075	`RequirementsService` deve restituire i requisiti analizzati dal data manager	Superato
TU-076	`RequirementsService` deve restituire tutti i requisiti dal data manager	Superato
TU-077	`RequirementsService` deve restituire i requisiti filtrati dal filter manager	Superato
TU-078	`RequirementsService` deve impostare la modalità di ordinamento e attivare l'evento di cambiamento	Superato
TU-079	`RequirementsService` deve restituire la modalità di ordinamento dal filter manager	Superato
TU-080	`RequirementsService` deve impostare il termine di ricerca e attivare l'evento di cambiamento	Superato
TU-081	`RequirementsService` non deve fare nulla se non ci sono requisiti	Superato
TU-082	`RequirementsService` deve uscire subito se nessun requisito ha tracciabilità	Superato
TU-083	`RequirementsService` deve uscire subito se tutti i requisiti con tracciabilità hanno già risultati di analisi	Superato
TU-084	`RequirementsService` deve chiamare l'analysis manager con i requisiti filtrati e la callback	Superato
TU-085	`RequirementsService` deve gestire gli errori in modo sicuro	Superato
TU-086	`RequirementsService` deve lanciare errore se il requisito non viene trovato	Superato
TU-087	`RequirementsService` deve saltare l'analisi se il requisito non ha tracciabilità	Superato
TU-088	`RequirementsService` deve analizzare il requisito e aggiornare i risultati	Superato
TU-089	`RequirementsService` deve gestire gli errori di analisi	Superato
TU-090	`RequirementsService` deve chiamare il servizio CSV con i dati dei requisiti	Superato
TU-091	`RequirementsService` deve chiamare il servizio CSV e impostare i requisiti	Superato

Codice	Descrizione	Esito
TU-092	`RequirementsService` deve gestire gli errori di importazione	Superato
TU-093	`RequirementsService` non deve processare se non esistono requisiti	Superato
TU-094	`RequirementsService` deve processare i requisiti in batch e applicare i risultati	Superato
TU-095	`RequirementsService` deve gestire errori durante l'elaborazione batch	Superato
TU-096	`RequirementsService` deve gestire la cancellazione	Superato
TU-097	`RequirementsService` deve lanciare errore se il requisito non viene trovato (tracciabilità)	Superato
TU-098	`RequirementsService` deve analizzare la tracciabilità e applicare i risultati	Superato
TU-099	`RequirementsService` deve gestire errori di analisi della tracciabilità	Superato
TU-100	`RequirementsService` non deve fare nulla se il requisito non viene trovato (applyTraceabilityResult)	Superato
TU-101	`RequirementsService` deve creare un nuovo requisito analizzato se non esiste	Superato
TU-102	`RequirementsService` deve aggiornare il requisito analizzato esistente	Superato
TU-103	`RequirementsService` non deve aggiornare la tracciabilità se non fornita	Superato
TU-104	`RequirementsService` deve delegare al traceability manager	Superato
TU-105	`RequirementsService` deve delegare all'approval manager (canApproveRequirement)	Superato
TU-106	`RequirementsService` deve aggiornare il collegamento di tracciabilità e attivare l'evento di cambiamento quando ha successo	Superato
TU-107	`RequirementsService` non deve attivare l'evento di cambiamento quando l'aggiornamento fallisce	Superato
TU-108	`RequirementsService` deve impostare lo stato di approvazione e attivare l'evento di cambiamento quando ha successo	Superato
TU-109	`RequirementsService` non deve attivare l'evento di cambiamento quando l'aggiornamento fallisce (approvazione)	Superato
TU-110	`RequirementsService` deve delegare all'approval manager (getRequirementApproval)	Superato

Codice	Descrizione	Esito
TU-111	`VSCodeConfiguration` deve restituire il modello requisito configurato	Superato
TU-112	`VSCodeConfiguration` deve restituire il valore predefinito se non configurato (modello requisito)	Superato
TU-113	`VSCodeConfiguration` deve restituire il modello codice configurato	Superato
TU-114	`VSCodeConfiguration` deve restituire il valore predefinito se non configurato (modello codice)	Superato
TU-115	`VSCodeConfiguration` deve restituire il modello implementazione configurato	Superato
TU-116	`VSCodeConfiguration` deve restituire il valore predefinito se non configurato (modello implementazione)	Superato
TU-117	`VSCodeConfiguration` deve restituire la soglia di qualità configurata	Superato
TU-118	`VSCodeConfiguration` deve restituire il valore predefinito se non configurato (soglia qualità)	Superato
TU-119	`VSCodeConfiguration` deve restituire la soglia di conformità configurata	Superato
TU-120	`VSCodeConfiguration` deve restituire il valore predefinito se non configurato (soglia conformità)	Superato
TU-121	`VSCodeConfiguration` deve aggiornare il valore di configurazione	Superato
TU-122	`VSCodeConfiguration` deve registrare un listener per i cambiamenti di configurazione	Superato
TU-123	`VSCodeConfiguration` deve chiamare il callback quando la configurazione rilevante cambia	Superato
TU-124	`VSCodeConfiguration` non deve chiamare il callback quando la configurazione non è rilevante	Superato
TU-125	`RequirementsApiService` deve chiamare l'API con i parametri corretti e restituire l'embedding	Superato
TU-126	`RequirementsApiService` deve gestire errori API con risposta 500	Superato
TU-127	`RequirementsApiService` deve gestire errori API con risposta 400	Superato
TU-128	`RequirementsApiService` deve gestire errori di timeout	Superato
TU-129	`RequirementsApiService` deve gestire errori di server non disponibile	Superato
TU-130	`RequirementsApiService` deve gestire errori di rete senza risposta	Superato
TU-131	`RequirementsApiService` deve gestire errori generici	Superato

Codice	Descrizione	Esito
TU-132	`RequirementsApiService` deve chiamare l'API con i parametri corretti e restituire il risultato dell'analisi	Superato
TU-133	`RequirementsApiService` deve gestire errori API con risposta `400` (analisi requisito)	Superato
TU-134	`RequirementsApiService` deve gestire errori API con risposta `404` (analisi requisito)	Superato
TU-135	`RequirementsApiService` deve gestire errori di timeout (analisi requisito)	Superato
TU-136	`RequirementsApiService` deve gestire errori di rete senza risposta (analisi requisito)	Superato
TU-137	`RequirementsApiService` deve gestire errori generici (analisi requisito)	Superato
TU-138	`RequirementsApiService` deve usare la base URL fornita	Superato
TU-139	`RequirementsApiService` deve usare la base URL predefinita se non fornita	Superato
TU-140	`embeddingService` deve calcolare la similarità tra vettori identici come 1.0	Superato
TU-141	`embeddingService` deve calcolare la similarità tra vettori ortogonali come 0.0	Superato
TU-142	`embeddingService` deve calcolare correttamente la similarità tra vettori simili	Superato
TU-143	`embeddingService` deve lanciare errore per vettori di lunghezza diversa	Superato
TU-144	`embeddingService` deve restituire 0 per vettori nulli	Superato
TU-145	`embeddingService` deve restituire 0 per entrambi i vettori nulli	Superato
TU-146	`embeddingService` deve calcolare correttamente la similarità per valori negativi	Superato
TU-147	`embeddingService` deve gestire correttamente i valori floating point	Superato
TU-148	`embeddingService` deve gestire efficientemente vettori grandi	Superato
TU-149	`CodeExtractionService` deve estrarre dichiarazioni di funzione C	Superato
TU-150	`CodeExtractionService` deve estrarre direttive del preprocessore C	Superato
TU-151	`CodeExtractionService` deve estrarre dichiarazioni di struct C	Superato
TU-152	`CodeExtractionService` deve estrarre dichiarazioni di enum C	Superato

Codice	Descrizione	Esito
TU-153	`CodeExtractionService` deve gestire input vuoto	Superato
TU-154	`CodeExtractionService` deve gestire input con solo commenti e spazi	Superato
TU-155	`CodeExtractionService` deve estrarre dichiarazioni di funzione multilinea	Superato
TU-156	`CodeExtractionService` deve gestire prototipi di funzione	Superato
TU-157	`CodeExtractionService` deve gestire correttamente blocchi annidati	Superato
TU-158	`CodeExtractionService` deve gestire correttamente le direttive include	Superato
TU-159	`CodeExtractionService` deve gestire codice C complesso del mondo reale	Superato
TU-160	`ParseCodeService` deve impostare i percorsi ignorati	Superato
TU-161	`ParseCodeService` deve restituire una copia dei percorsi ignorati	Superato
TU-162	`ParseCodeService` deve analizzare il codice da un file in base alle informazioni di tracciabilità	Superato
TU-163	`ParseCodeService` deve usare i percorsi ignorati quando cerca i file	Superato
TU-164	`ParseCodeService` deve lanciare errore se il file non viene trovato	Superato
TU-165	`ParseCodeService` deve lanciare errore se l'intervallo di righe non è valido (start > end)	Superato
TU-166	`ParseCodeService` deve lanciare errore se l'intervallo di righe supera la lunghezza del documento	Superato
TU-167	`ParseCodeService` deve lanciare errore se la riga iniziale è minore di 1	Superato
TU-168	`FileReaderService` deve restituire i pattern predefiniti se nessuna cartella di workspace è disponibile	Superato
TU-169	`FileReaderService` deve leggere i pattern dal file .reqignore se esiste	Superato
TU-170	`FileReaderService` deve restituire i pattern predefiniti se il file .reqignore non esiste	Superato
TU-171	`FileReaderService` deve restituire i pattern predefiniti se .reqignore non è un file	Superato
TU-172	`FileReaderService` deve gestire errori di lettura dal file .reqignore	Superato
TU-173	`FileReaderService` deve gestire errori specifici di lettura file	Superato

Codice	Descrizione	Esito
TU-174	`FileReaderService` deve gestire errori di permesso nella lettura del file .reqignore	Superato
TU-175	`FileReaderService` deve gestire errori di stat nel controllo del file .reqignore	Superato
TU-176	`FileReaderService` deve gestire correttamente una lista di file vuota	Superato
TU-177	`FileReaderService` deve costruire correttamente l'esclusione quando esistono più pattern di ignore	Superato
TU-178	`FileReaderService` deve gestire errori nell'apertura dei file	Superato
TU-179	`FileReaderService` deve gestire errori specifici nell'apertura dei file	Superato
TU-180	`FileReaderService` deve gestire errori di codifica del testo durante la lettura dei file	Superato
TU-181	`FileReaderService` deve gestire errori quando findFiles fallisce	Superato
TU-182	`FileReaderService` deve usare il pattern di esclusione personalizzato se fornito	Superato
TU-183	`FileReaderService` deve gestire più pattern di ignore correttamente	Superato
TU-184	`FileReaderService` deve gestire una lista di file vuota restituita da findFiles	Superato
TU-185	`FileReaderService` deve chiamare getFilesContent con il pattern corretto	Superato
TU-186	`FileReaderService` deve propagare errori da getFilesContent	Superato
TU-187	`FileReaderService` deve restituire un oggetto vuoto quando non vengono trovati file C	Superato
TU-188	`FileReaderService` deve convertire correttamente vari pattern	Superato
TU-189	`FileReaderService` deve gestire correttamente i pattern che terminano con	Superato
TU-190	`FileReaderService` deve gestire i pattern con estensioni di file	Superato
TU-191	`FileReaderService` deve gestire pattern senza wildcard ma con estensioni	Superato
TU-192	`CsvService` deve restituire stringhe vuote per tracciabilità vuota	Superato
TU-193	`CsvService` deve formattare correttamente i dati di tracciabilità	Superato
TU-194	`CsvService` deve mostrare un messaggio di errore se non ci sono requisiti da esportare	Superato

Codice	Descrizione	Esito
TU-195	`CsvService` non deve fare nulla se l'utente annulla la finestra di salvataggio	Superato
TU-196	`CsvService` deve esportare i requisiti su file CSV	Superato
TU-197	`CsvService` deve gestire errori di scrittura file	Superato
TU-198	`CsvService` deve lanciare errore per dati CSV vuoti	Superato
TU-199	`CsvService` deve lanciare errore per intestazioni richieste mancanti	Superato
TU-200	`CsvService` deve analizzare dati di requisito di base	Superato
TU-201	`CsvService` deve analizzare dati di requisito con tracciabilità	Superato
TU-202	`CsvService` deve analizzare dati di requisito analizzato	Superato
TU-203	`CsvService` deve gestire più voci di tracciabilità	Superato
TU-204	`CsvService` deve lanciare errore per valori di range non validi	Superato
TU-205	`CsvService` deve lanciare errore se start > end nel range	Superato
TU-206	`CsvService` deve leggere e analizzare file CSV	Superato
TU-207	`CsvService` deve contare correttamente i requisiti analizzati	Superato
TU-208	`CsvService` deve gestire errori di lettura file	Superato
TU-209	`CsvService` deve gestire errori di parsing CSV	Superato

Table 18: Lista di test di unità

4.1.4 Test di Integrazione

I test di integrazione valutano il corretto funzionamento delle diverse componenti del software e il modo in cui vengono integrate tra loro, evidenziandone eventuali problemi. I test di integrazione individuati sono i seguenti:

Codice	Descrizione	Esito
TI-001	VS Code dovrebbe essere attualmente in esecuzione	Superato
TI-002	Il frontend deve essere in grado di accedere alle configurazioni	Superato
TI-003	Il frontend deve essere in grado di eseguire comandi inviati dall'utente	Superato
TI-004	Il frontend deve registrare ed eseguire il comando di analisi di tutti i requisiti	Superato
TI-005	Il frontend deve registrare ed eseguire il comando di filtro	Superato
TI-006	Il frontend deve registrare ed eseguire il comando di analisi del requisito	Superato

Codice	Descrizione	Esito
TI-007	Il frontend deve registrare ed eseguire il comando di caricamento	Superato
TI-008	Il frontend deve registrare ed eseguire il comando di analisi della tracciabilità	Superato
TI-009	Il frontend deve registrare ed eseguire il comando `setSortMode`	Superato
TI-010	Il frontend deve registrare ed eseguire il comando `setSortAnalyzedFirst`	Superato
TI-011	Il frontend deve registrare ed eseguire il comando setSortUnanalyzedFirst	Superato
TI-012	Il frontend deve registrare ed eseguire il comando `approveRequirement`	Superato
TI-013	Il frontend deve registrare ed eseguire il comando `refuseRequirement`	Superato
TI-014	Il frontend deve registrare ed eseguire il comando `editTraceability`	Superato
TI-015	Il frontend deve registrare ed eseguire il comando `exportCSV`	Superato
TI-016	L'entità `ParseCodeService` deve rispettare i percorsi ignorati	Superato
TI-017	L'entità `ParseCodeService` deve generare un errore per file inesistente	Superato
TI-018	L'entità `ParseCodeService` deve generare un errore per intervallo di linee non valido	Superato
TI-019	L'entità `FileReaderService` deve interagire con l'API dello spazio di lavoro di VS Code per ottenere i pattern di ignoranza	Superato
TI-020	L'entità `FileReaderService` deve utilizzare pattern glob compatibili con VS Code	Superato
TI-021	L'entità `FileReaderService` deve gestire la disponibilità delle cartelle dello spazio di lavoro di VS Code	Superato
TI-022	L'entità `CsvService` deve integrarsi con l'API del file system di VS Code per importare CSV	Superato
TI-023	L'entità `CsvService` deve gestire gli errori del file system di VS Code in modo appropriato	Superato
TI-024	L'entità `CsvService` deve gestire gli errori di parsing CSV	Superato
TI-025	L'entità `CsvService` deve analizzare correttamente i dati CSV con percorsi compatibili con VS Code	Superato

Codice	Descrizione	Esito
TI-026	L'entità `CsvService` deve essere in grado gestire più voci di tracciabilità	Superato
TI-027	L'entità `VSCodeConfiguration` restituirà valori di configurazione predefiniti quando questi non saranno impostati	Superato
TI-028	L'entità `VSCodeConfiguration` deve essere in grado di salvare e recuperare correttamente i suoi valori di configurazione	Superato
TI-029	Nell'entità `VSCodeConfiguration`, il listener di modifica della configurazione deve essere attivato quando le impostazioni cambiano	Superato
TI-030	L'entità `VSCodeConfiguration` deve gestire diversi tipi di dato numerici	Superato
TI-031	L'entità `RequirementsApiService` deve interfacciarsi correttamente con l'endpoint API delle embeddings	Superato
TI-032	L'entità `RequirementsApiService` deve interfacciarsi correttamente con l'endpoint API di analisi del requisito	Superato
TI-033	L'entità `RequirementsApiService` deve gestire correttamente gli errori dell'API	Superato
TI-034	L'entità `RequirementsApiService` deve gestire correttamente gli errori di timeout	Superato
TI-035	L'entità `RequirementsApiService` deve gestire correttamente gli errori di codice di stato HTTP	Superato
TI-036	L'entità `RequirementAnalysisService` deve integrarsi correttamente con tutte le dipendenze	Superato
TI-037	L'entità `RequirementAnalysisService` deve gestire gli errori di parsing JSON in modo appropriato	Superato
TI-038	L'entità `RequirementAnalysisService` deve utilizzare modelli personalizzati quando forniti	Superato
TI-039	L'entità `RequirementAnalysisService` deve integrarsi correttamente con le dipendenze per ottenere embeddings	Superato
TI-040	L'entità `RequirementAnalysisService` deve utilizzare un modello di embedding personalizzato quando fornito	Superato
TI-041	L'entità `OllamaApiAdapter` deve inviare correttamente un messaggio all'API di Ollama e restituire la risposta	Superato

Codice	Descrizione	Esito
TI-042	L'entità `OllamaApiAdapter` deve generare ExternalServiceError quando la richiesta all'API di Ollama fallisce	Superato
TI-043	L'entità `OllamaApiAdapter` deve ottenere correttamente embeddings dall'API di Ollama	Superato
TI-044	L'entità `OllamaApiAdapter` deve utilizzare il modello di embedding predefinito quando non fornito	Superato
TI-045	'entità `Application` Il modulo di analisi dei requisiti dovrebbe essere definito	Superato
TI-046	L'entità `Application` deve essere accessibile tramite una richiesta HTTP POST su `/requirement/analyze`	Superato
TI-047	L'entità `Application` deve essere accessibile tramite una richiesta HTTP POST su `/requirement/embedding`	Superato
TI-048	L'entità `Application` deve chiamare `analyzeRequirement` del servizio `RequirementAnalysisService` iniettato	Superato
TI-049	L'entità `Application` deve chiamare `getEmbedding` del servizio `RequirementAnalysisService` iniettato	Superato

Table 19: Lista di test di integrazione

4.1.5 Test di Sistema

I test di sistema verificano il sistema completo del prodotto software, prendendo in considerazione tutti i componenti e interfacce con altri sistemi. Questi test controllano che il software rispetti tutti i requisiti prestabiliti e che sia adatto all'uso in produzione.

4.1.6 Test di Accettazione

I test di accettazione assicurano che il software soddisfi i requisiti e parametri stabiliti dal *capitolato*[G]. Sono svolti in presenza del *committente*[G] e verificano che il prodotto possa essere consegnato al committente o messo in produzione.

Codice	Descrizione	Esito	Fonte
TA-001	Il plugin deve essere sviluppato come un'estensione installabile per Visual Studio Code	Superato	Capitolato
TA-002	Il server in ascolto delle richieste deve essere installabile e avviabile tramite un'immagine docker o manualmente	Superato	Proponente

Codice	Descrizione	Esito	Fonte
TA-003	Il sistema deve essere in grado di importare i requisiti e, se presenti, informazioni di tracciamento e risultati di analisi precedenti, da un file in formato .csv	Superato	RFO008, RFO010, RFO011, RFO015, RPF001
TA-004	Il sistema deve essere in grado di visualizzare i requisiti importati e le relative informazioni in una vista ad albero all'interno dell'interfaccia di Visual Studio Code	Superato	RFO002, RFO013, RFO027, RFO019
TA-005	L'utente deve essere in grado, in caso di assenza del tracciamento dei requisiti nel file .csv caricato, di tracciare l'implementazione dei requisiti all'interno del codice sorgente	Superato	RFO031, RFF004
TA-006	L'utente deve essere in grado di eseguire il tracciamento di un singolo requisito nel codice sorgente	Superato	RFO030, RFO031, RFO035
TA-007	L'utente deve essere in grado di condurre un'analisi dei requisiti importati ed ottenere una valutazione complessiva di ognuno di essi	Superato	RFO002,RFO017,RFO018, RFO030, RFO031, RFO032, RFO037
TA-008	L'utente deve essere in grado di eseguire l'analisi di un requisito specifico	Superato	RFO030,RFO035
TA-009	La valutazione di ogni requisito deve ritornare: risultato globale: passato/non passato; valutazione da 0 a 100 della semantica del requisito; valutazione a 0 a 100 dell'implementazione del requisito nel codice; una lista di suggerimenti (se presenti); una lista di problemi riscontrati (se presenti)	Superato	RFO002, RFO016, RFO017, RFO018, RFO027
TA-010	L'utente può ordinare i requisiti in ordine crescente, oppure ordinarli in base alla	Superato	RFO038

Codice	Descrizione	Esito	Fonte
	disponibiltà del risultato dell'analisi (analizzato/non analizzato)		
TA-011	Il sistema deve essere in grado, in qualsiasi momento dopo l'importazione di esportare i requisiti e le informazioni disponibili in un file in formato CSV	Superato	RFO024,RFO026
TA-012	L'utente deve essere in grado di segnare un requisito come "approvato" o "non approvato"	Superato	RFO016, RFO033, RFO036
TA-013	L'utente deve essere in grado di filtrare i requisiti per ID, per descrizione o per file sorgente tramite una barra di ricerca	Superato	RFO029
TA-014	L'utente deve essere in grado di impostare la soglia di conformità del codice	Superato	RFO036,RFO037
TA-015	L'utente deve essere in grado di impostare il modello LLM da utilizzare per l'analisi testuale del requisito e per l'implementazione di esso nel codice sorgente	Superato	RFF005
TA-016	L'utente deve essere in grado di impostare il modello di embedding per il tracciamento	Superato	RFF005
TA-017	L'utente deve essere in grado di aggiornare manualmente il tracciamento di un requisito nel codice (file, linea di inizio e linea di fine), modificando il relativo campo all'interno del singolo requisito	Superato	RFF014, RFO034
TA-018	Il sistema deve essere in grado di essere disinstallato correttamente	Superato	Propomente

Table 20: Lista di test di accettazione

4.1.7 Sviluppo

Le specifiche riguardanti i test descritti verranno definite nelle successive versioni del Piano di Qualifica

5 Resoconto delle attività di verifica

5.1 MPC05 - MPC02: Actual Cost e Estimated to Completion

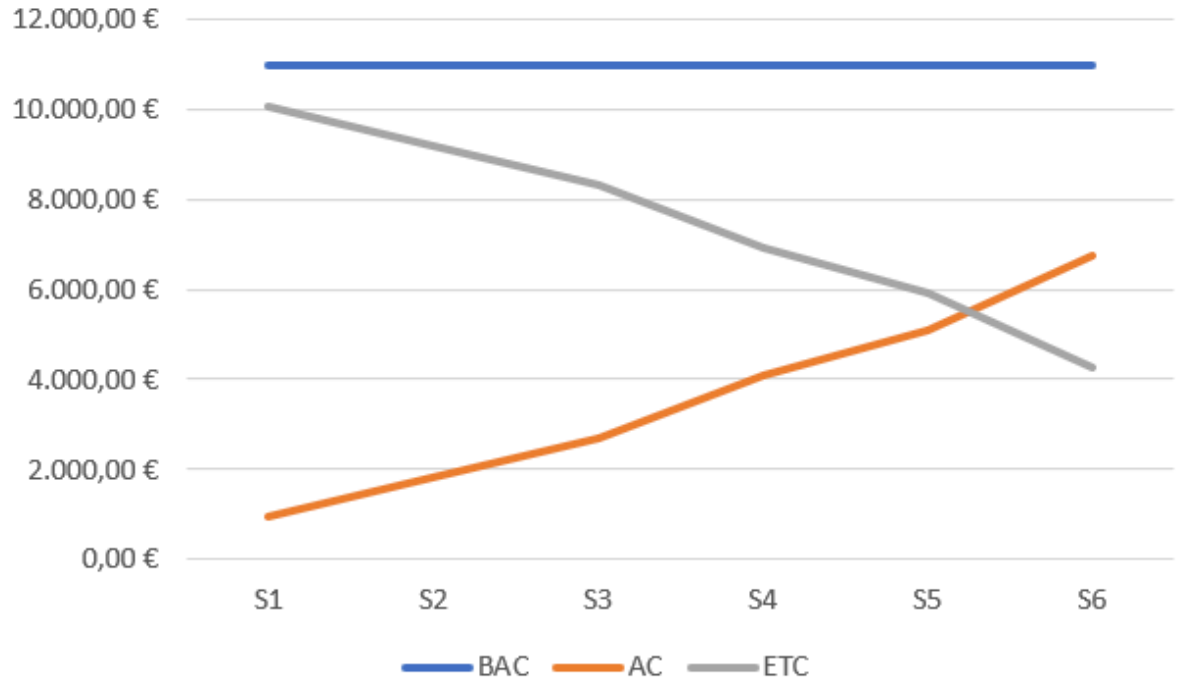

Figure 2: Grafico Actual Cost e Estimated to Completion

5.2 MPC03 - MPC04: Earned Value e Planned Value

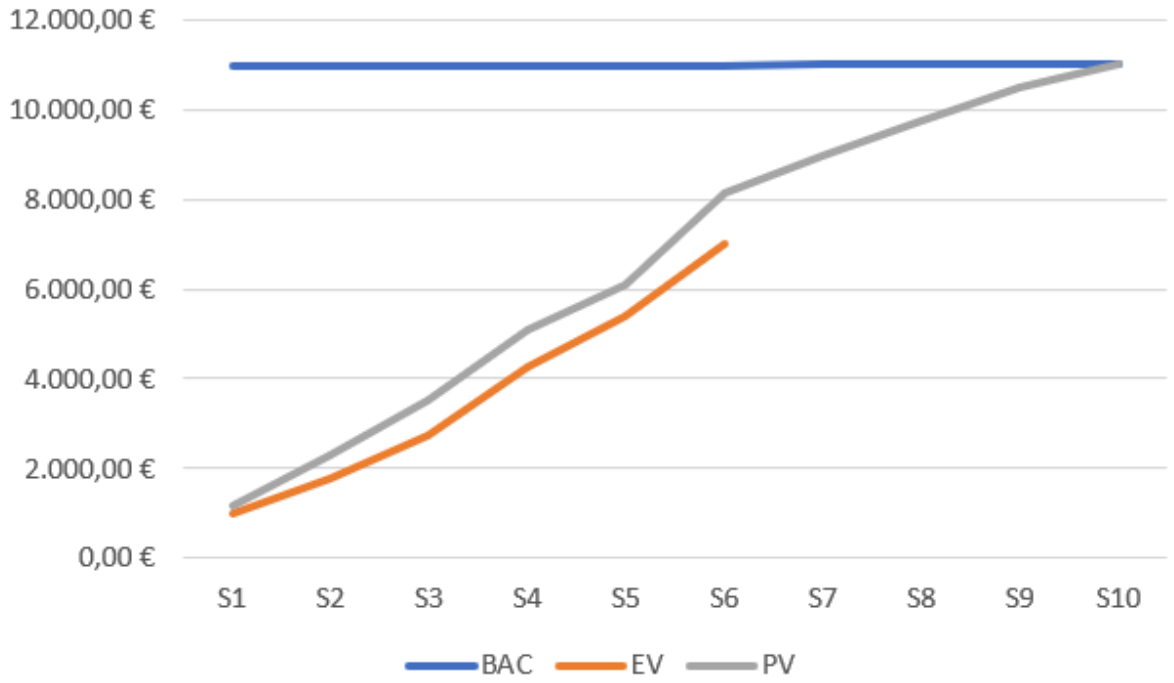

Figure 3: Grafico Earned Value e Planned Value

5.3 MPC07: Schedule Variance

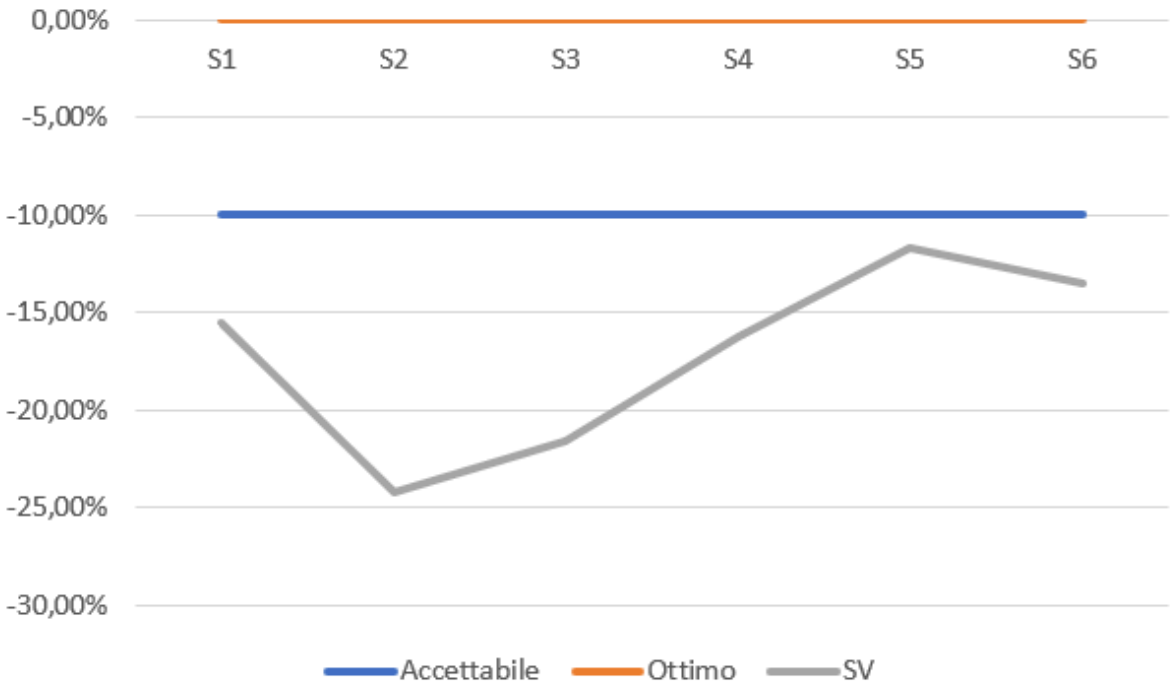

Figure 4: Grafico Schedule Variance

5.4 MPC06: Cost Variance

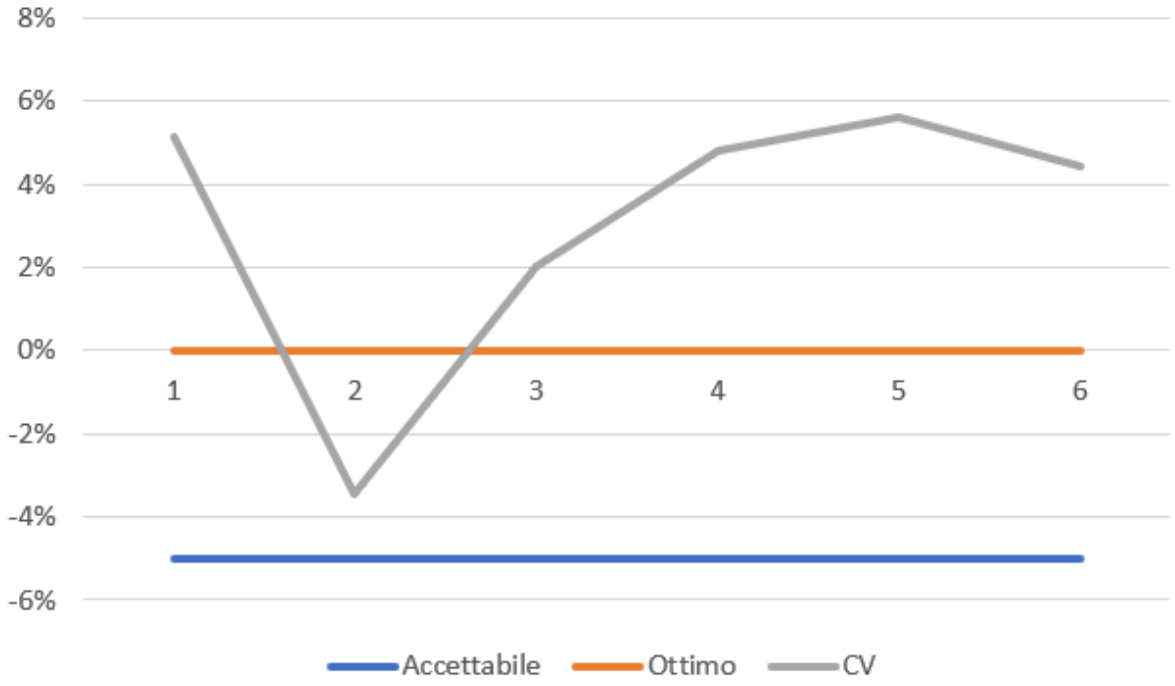

Figure 5: Grafico Cost Variance

5.5 MPC01: Estimated at Completion

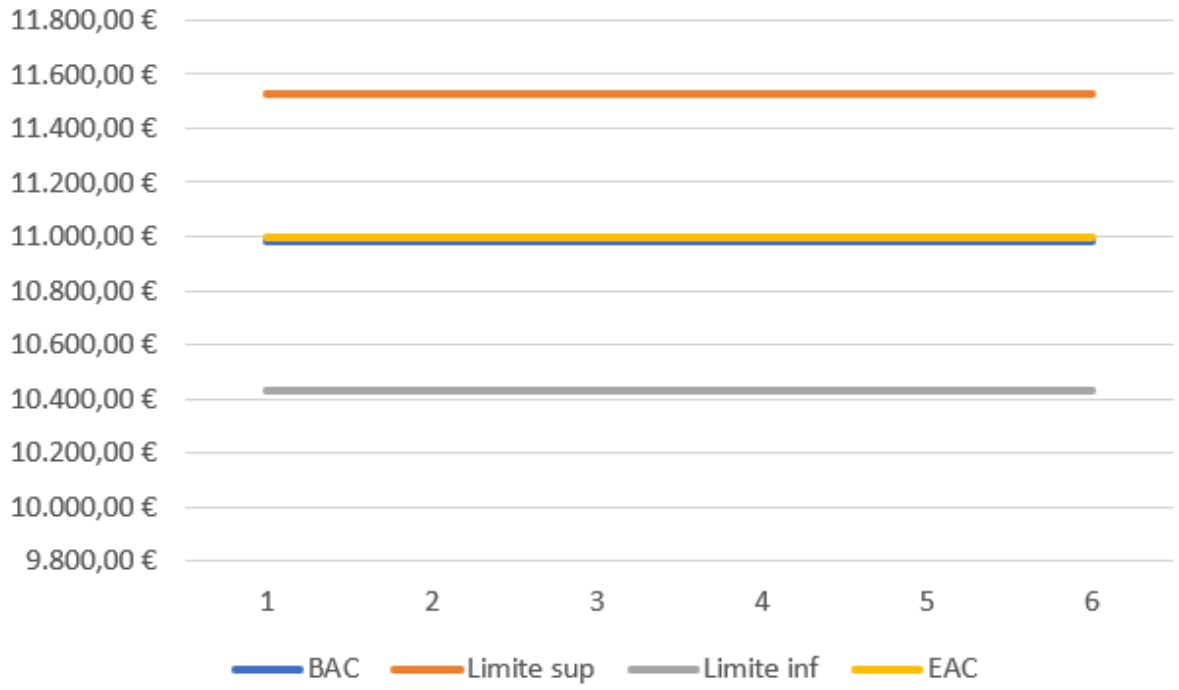

Figure 6: Grafico Estimated at Completion

5.6 MPC12: Indice di Gulpease

Di seguito la tabella con i risultati ottenuti dai documenti secondo l'indice di Gulpease. Come metro di valutazione del documento viene esclusa la prima pagina che, trattandosi dell'intestazione, potrebbe portare ad un risultato inesatto.

Documento	Risultato	Esito
Analisi dei Requisiti	83	Superato
Piano di qualifica	84	Superato
Piano di Progetto	80	Superato
Norme di Progetto	75	Superato
Glossario	83	Superato
2024-11-15	68	Superato
2024-11-24	66	Superato
2024-12-09	75	Superato
2024-12-18	66	Superato

Documento	Risultato	Esito
2025-01-03	73	Superato
2025-01-10	65	Superato
2025-01-19	63	Superato
2025-02-08	65	Superato
2025-02-20	67	Superato
2025-03-07	65	Superato
2025-03-30	65	Superato
2025-04-05	61	Superato
2025-04-19	64	Superato
2025-04-28	67	Superato
2024-11-25	67	Superato
2024-12-24	64	Superato
2025-02-25	65	Superato
2025-04-11	67	Superato
2025-05-06	64	Superato

Table 21: Valutazione documenti

6 Valutazioni per il miglioramento

Nel seguente capitolo vengono riportate delle osservazioni sulle criticità incontrate con lo scopo di individuare i problemi e adottare dei miglioramenti.

6.1 Valutazione sull'organizzazione

Problema	Descrizione	Gravità	Soluzione
Riunione di gruppo	Incontri settimanali di durata molto lunga con ripetizione di argomenti	Bassa	Preparazione di una presentazione con punti da discutere e su cui focalizzarsi

Table 22: Problemi organizzativi

6.2 Valutazione sui ruoli

Problema	Descrizione	Gravità	Soluzione
Rotazione dei ruoli	Durante le prime fasi del lavoro la rotazione dei ruoli non era definita e a tratti assente	Media	Nuova ripartizione del carico di lavoro e definizione dei ruoli alle riunioni settimanali

Table 23: Problemi rotazione ruoli

6.3 Valutazione degli strumenti di lavoro

Problema	Descrizione	Gravità	Soluzione
Poca conoscenza delle tecnologie richieste	Durante lo sviluppo sono state richieste l'uso di tecnologie non conosciute dal gruppo	Alta	Investito tempo nello studio e formazione dei membri con prove e test pratici, uso di Notion per condividere le ricerche fatte

Table 24: Problemi con strumenti di lavoro