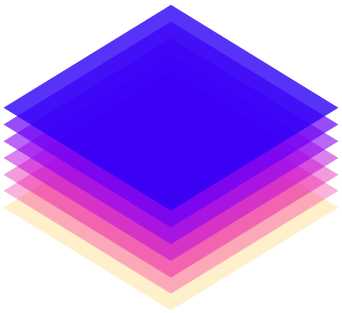

HEPTA CODE

Norme di progetto

Versione 1.0.0

Contenuto del Documento

Norme e linee guida operative del team *HeptaCode* nello sviluppo del progetto *Code Guardian*.

email: heptacode7@gmail.com

sito: heptacode-unipd.github.io

Registro delle modifiche

Versione	Data	Autore	Verificatore	Descrizione
1.0.0	2026/02/10	Riccardo Baldin	Angela Favaro	Completamento capitoli vuoti
0.5.0	2026/01/06	Angela Canazza	Nicola Simionato	Stesura Metriche per la Qualità
0.4.1	2026/01/04	Laura Venturini	Nicola Simionato	Stesura paragrafo Progettazione
0.4.0	2026/01/04	Angela Favaro	Nicola Simionato	Stesura Standard per la Qualità
0.3.0	2026/01/03	Angela Favaro	Nicola Simionato	Stesura Processi di Supporto
0.2.0	2025/12/28	Riccardo Baldin	Angela Favaro	Stesura Processi Primari e Processi di Supporto
0.1.0	2025/12/10	Amerigo Vegliante	Riccardo Baldin	Impostazione struttura del Documento

Indice dei contenuti

1. Introduzione

1.1. Scopo del documento

Questo documento serve a definire le norme e le linee guida del team *HeptaCode* per lo sviluppo del progetto *CodeGuardian*. Nello specifico, si descrivono i processi di lavoro, le modalità di collaborazione, standard di codifica e le pratiche di gestione della qualità che il team si impegna a seguire per garantire coerenza, qualità ed efficienza nel ciclo di vita del prodotto.

1.2. Scopo del prodotto

Code Guardian è un prodotto ideato dall'azienda *Var Group S.p.A*, si tratta di una piattaforma web basata su un sistema ad agenti per l'audit[G] e la remediation[G] dei repository[G] software[G]. Analizza repository GitHub[G] per valutarne qualità, sicurezza e manutenzione, generando report automatici e suggerendo miglioramenti su test, sicurezza (OWASP[G]) e documentazione.

1.3. Glossario

Termini tecnici e ambiguità sono chiariti nel documento *glossario.pdf*, ogni termine presente nel glossario presenta una lettera «G» come apice, in questo modo intendiamo rendere la lettura della documentazione coerente e chiara.

1.4. Riferimenti

1.4.1. Riferimenti Normativi

- Capitolato d'appalto C2 *Vargroup S.p.a.* - Code Guardian:

  https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C2.pdf

- Standard ISO/x 12207:1995 - Processi di Ciclo di Vita del Software:

  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf.
  A questo standard fanno riferimento: Sezione 2, Sezione 3 e Sezione 4.

- Standard ISO/IEC 9126 - Modello di Qualità:

  https://it.wikipedia.org/wiki/ISO/IEC_9126.
  A questo standard fanno riferimento: Sezione 5 e Sezione 6.

1.4.2. Riferimenti Informativi

- Documentazione Typst[G]:

  https://typst.app/docs[G]/

- Materiale didattico del corso di Ingegneria del Software 2025/2026:

  https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T06.pdf

2. Processi Primari

2.1. Processo di Fornitura

Secondo lo standard ISO/IEC 12207:1995, comprende le attività e le risorse necessarie al fornitore (*HeptaCode*) per realizzare il progetto.
Questo processo inizia dopo lo Studio di Fattibilità, quando le esigenze del committente sono chiare. Il fornitore stipula un contratto che definisce i requisiti e la data di consegna. Solo dopo l'accordo si passa alla fase esecutiva con la redazione del *Piano di Progetto*.

Il processo si articola nelle seguenti attività:

1. *Analisi della Richiesta*: valutazione dei requisiti espressi nel Capitolato per determinare la capacità di soddisfare le attese.
2. *Preparazione della Proposta*: definizione dell'offerta tecnica e preventivo economico.
3. *Pianificazione Tecnica*: redazione della documentazione di coordinamento e programmazione delle attività del progetto, analisi delle risorse e creazione delle milestone[G].
4. *Esecuzione*: sviluppo del software e monitoraggio costante tramite milestone, test, verifiche e documentazione.
5. *Controllo*: valutazione del lavoro svolto per garantire la conformità alle regole del gruppo a i requisiti contrattuali tramite esecuzione dei test di accettazione e revisione del codice scritto.
6. *Consegna*: trasferimento del prodotto finale al cliente.

2.1.1. Obiettivi

1. Ridurre al minimo le ambiguità nei requisiti.
2. Garantire un prodotto di qualità tramite verifiche sistematiche.
3. Mantenere una comunicazione fluida per gestire tempestivamente eventuali variazioni in corso d'opera.

2.1.2. Descrizione

In questo processo, il team delimita il perimetro entro cui verrà attuato tutto il lavoro tecnico. Stabilisce le regole con il committente, assicurando che lo sviluppo non proceda a tentoni, ma segua obiettivi precisi, tempi concordati e standard di qualità. In pratica, è una dichiarazione d'impegno per trasformare le idee del proponente in un software reale e verificato.

2.1.2.1. Rapporti con l'azienda Proponente

La comunicazione asincrona con il proponente avviene tramite *Slack*[G], mentre gli incontri telematici vengono fatti tramite la piattaforma *Microsoft Teams*. Durante gli incontri tra il team e l'azienda vengono redatti dei verbali nei quali si evidenziano gli argomenti discussi e le decisioni prese.

2.1.2.2. Documentazione Fornitura

La documentazione prodotta funge da prova tangibile dell'esecuzione dei processi:

- **Analisi dei requisiti:** documento redatto da chi copre il ruolo di Analista con lo scopo di definire le funzionalità necessarie del prodotto in relazione alle richieste della azienda proponente.
- **Piano di Progetto:** documento redatto da chi copre il ruolo di Responsabile[G] con lo scopo di definire una linea guida da seguire durante tutto lo svolgimento del progetto in maniera tale da avere sotto controllo i costi, i tempi e l'andamento generale dell'avanzamento.
- **Piano di Qualifica:** documento redatto da chi copre il ruolo di Amministratore con lo scopo di definire in dettaglio le strategie di verifica e validazione per garantire la qualità del prodotto finale e dei processi realizzativi.
- **Lettera di Presentazione:** documento formale tramite il quale il gruppo si è proposto al professor Vardanega come candidato per l'assegnazione del capitolato proposto da *Vargroup S.p.A.*.

2.1.2.3. Strumenti

Per la gestione del processo di fornitura, il gruppo utilizza:

Versionamento: Git[G] (GitHub) per il controllo dei file sorgente e della documentazione.

Project Management: GitHub Projects per il tracking delle task.

Comunicazione: Slack o Microsoft Teams per il coordinamento interno e con il committente, Telegram[G] per la comunicazione interna al team.

Documentazione: Typst per la realizzazione dei documenti e dei verbali.

2.2. Processo di Sviluppo

In conformità allo standard ISO/IEC 12207:1995, definisce le attività tecniche che il gruppo *HeptaCode* mette in atto per trasformare i requisiti contrattuali in un prodotto software finito. Lo sviluppo si occupa del come realizzare concretamente secondo le regole stabilite durante il processo di Fornitura.

2.2.1. Obiettivi

L'obiettivo principale è garantire che ogni riga di codice e ogni scelta siano tracciabili e rispondano direttamente alle esigenze della azienda proponente, assicurando al contempo che il prodotto sia robusto, manutenibile e privo di difetti critici.

2.2.2. Descrizione

2.2.2.1. Analisi dei Requisiti

Lo scopo principale dell'Analisi dei Requisiti è quello di formalizzare le specifiche funzionali, non funzionali e i vincoli del progetto «*Code Guardian*».
Agisce come una fonte di verità unica (*single source of truth*) per il team di sviluppo e per tutti gli stakeholder[G], con l'obiettivo primario di garantire una comprensione comune e ridurre al minimo le ambiguità che potrebbero emergere nelle fasi successive.

Tramite questo documento è possibile comprendere:
- le necessità degli utenti
- gli obiettivi del prodotto
- il contesto in cui il prodotto verrà utilizzato

Descrizione

L'Analisi dei Requisiti è un documento redatto da chi copre il ruolo di Analista.
Comprende i seguenti concetti:

- **Introduzione**: descrizione dell'obiettivo del documento e riferimenti usati per la stesura del documento.
- **Descrizione del prodotto**: illustra le funzionalità del prodotto e le caratteristiche del comportamento di esso in relazione ai[G] vari utenti che lo utilizzano. Analizza inoltre i vincoli del prodotto e del progetto.
- **Casi d'uso**: rappresentazione in modo formale e non ambiguo le interazioni tra gli utenti (attori) e il sistema.
- **Requisiti**: Una volta identificati i casi d'uso, il documento illustra i requisiti derivati dagli stessi e capitolato. I requisiti sono identificati da una nomenclatura:

  R-Numero-Categoria-Importanza

  dove:
  - **R** sta per *Requisito*;
  - **Numero** è l'identificativo univoco del requisito;
  - **Categoria** indica la classificazione del requisito, può essere:
    - **V** per *Vincolo*: Requisiti obbligatori e imprescindibili, imposti da fattori esterni o scelte progettuali vincolanti.

  - **A** per *Aspettativa*: Requisiti impliciti derivanti dai bisogni dell'utente o da standard di settore, spesso non espressi direttamente ma attesi.
  - **F** per *Funzionale*: Descrivono le funzionalità specifiche e i comportamenti che il sistema deve attuare per soddisfare le esigenze degli utenti.
  - **Q** per *Qualità*: Caratteristiche non funzionali che il sistema deve possedere per operare correttamente (es. prestazioni, affidabilità, sicurezza, manutenibilità).
- **Importanza**: indica l'urgenza di soddisfacimento del requisito, i possibili valori sono:
  - **O** per Obbligatorio;
  - **D** per Desiderabile;
  - **P** per Opzionale.

Notazione dei Casi d'uso

I Casi d'Uso rappresentano un elemento fondamentale all'interno del ciclo di vita dello sviluppo software in quanto definiscono in modo formale e non ambiguo le interazioni tra gli utenti (attori) e il sistema.

Ciascun caso d'uso[G] descrive una sequenza di azioni che un attore[G] compie per raggiungere un obiettivo, fornendo così la base per la progettazione del sistema, la sua implementazione e le successive fasi di test. Tutto ciò è volto a garantire che il prodotto finale sia allineato con le aspettative degli *stakeholder*.

I termini utilizzati nell'esposizione dei casi d'uso sono i seguenti:

- **Identificatore**: codice univoco, strutturato secondo la notazione gerarchica UCx.y (o UCx.y.z per casi specifici),, che viene assegnato a ogni singolo Caso d'Uso per individuarlo in modo inequivocabile all'interno dell'intera documentazione di progetto. La sua funzione primaria è quella di garantire una rapida rintracciabilità dei requisiti lungo tutto il ciclo di vita del software senza ambiguità;
- **Scenario principale**: descrive cosa succede quando l'interazione tra l'attore e il sistema procede linearmente verso il successo senza intoppi;
- **Scenario secondario**; Comprende tutti i percorsi alternativi o le gestioni delle eccezioni che si discostano dal flusso principale. Descrive come il sistema deve comportarsi quando si verificano errori o quando l'utente effettua scelte opzionali diverse da quelle standard;
- **Precondizioni**: Lo stato in cui devono trovarsi obbligatoriamente il sistema e l'ambiente circostante prima che il Caso d'Uso inizi. Rappresentano i vincoli e i requisiti indispensabili che si assumono come veri all'inizio dell'interazione;
- **Postcondizioni**: Lo stato finale del sistema una volta che il Caso d'Uso è stato completato con successo;
- **Trigger**: Evento scatenante specifico che avvia il Caso d'Uso;
- **Attori principali**: Coloro che interagiscono attivamente con il Sistema e svolgono l'azione indicata dal Caso d'Uso. Essi sono sempre gli iniziatori del processo;
- **Attori secondari**: Entità esterne al sistema. Hanno un ruolo reattivo e partecipano all'interazione solo in risposta ad una richiesta del sistema;
- **Inclusioni**: Relazione di dipendenza forte in cui un Caso d'Uso base incorpora il comportamento di un altro Caso d'Uso per poter portare a termine la propria funzione;
- **Estensioni**: Identifica una relazione in cui un Caso d'Uso può arricchire o modificare il comportamento di un Caso d'Uso base, ma solo qualora si verifichino specifiche condizioni o scelte discrezionali dell'utente. A differenza dell'inclusione, questa aggiunta funzionale è del tutto opzionale;
- **Generalizzazione**: Relazione di ereditarietà tra un elemento più generico, detto padre, e uno più specifico, detto figlio.

2.2.2.2. Progettazione

L'attività di progettazione serve per definire l'architettura logica del prodotto, cercando una soluzione tecnica che soddisfi tutti gli stakeholder. Inoltre, suddivide il sistema in singole componenti dalla complessità individuale minore possibile in modo da facilitare la successiva fase di codifica.
La progettazione avviene in contemporanea al rilevamento dei requisiti e si basa su di essi per definire le scelte architetturali più adatte; la realizzazione dell'architettura inizia con lo sviluppo del Proof of Concept, cioè della dimostrazione della Fattibilità della soluzione ipotizzata e della compatibilità tra le tecnologie adottate. Per effettuarla, i progettisti del team dovranno decidere di quali tecnologie avvalersi dopo aver analizzato le alternative possibili.

Aspettative
L'architettura progettata deve avere le seguenti proprietà:
- Sufficienza: deve essere in grado di soddisfare tutti i requisiti funzionali e non funzionali del sistema;
- Comprensibilità: deve essere capita da tutti gli stakeholder coinvolti nel progetto;
- Modularità: deve essere suddivisa in parti chiare e indipendenti tra loro;
- Robustezza: deve essere in grado di gestire diversi tipi di ingressi da parte dall'utente e dall'ambiente senza compromettere l'integrità del sistema;

Documentazione
La documentazione relativa alla progettazione verrà redatta dai progettisti. Questi membri del team dovranno assicurarsi di documentare il Proof of Concept, di conseguenza le scelte tecnologiche effettuate, le motivazioni di tali scelte e la definizione delle componenti architetturali del sistema. Inoltre, per la Product Baseline[G] dovranno essere documentati i test eseguiti per validare il Proof of Concept e i risultati ottenuti, i design pattern[G] che si pensano adatti all'implementazione del sistema e la definizione delle classi usate nel Proof of Concept.

Metriche Per garantire che l'architettura sia manutenibile e modulare, il team monitora le seguenti metriche di progettazione:

- Accoppiamento: misura il grado di dipendenza tra i moduli. Si deve tendere a un accoppiamento debole per facilitare la modifica dei componenti senza effetti a catena.
- Coesione: misura quanto le responsabilità di un singolo modulo siano correlate tra loro. Si persegue un'alta coesione (un modulo fa una sola cosa bene).
- Complessità Ciclomatica: utilizzata per valutare la complessità dei flussi logici progettati, con l'obiettivo di mantenere i componenti testabili (valore accettabile ≤ 15, valore ottimale ≤ 10).

Diagrammi UML Il team utilizza il linguaggio UML (Unified Modeling Language) per visualizzare e documentare l'architettura. I diagrammi richiesti per ogni componente sono:

- Diagrammi delle Classi: per descrivere la struttura statica del sistema, le gerarchie di ereditarietà e le relazioni tra oggetti.
- Diagrammi di Sequenza: per modellare le interazioni dinamiche tra gli oggetti nel tempo, particolarmente utili per descrivere il flusso dei dati durante l'audit dei repository.
- Diagrammi dei Componenti: per rappresentare l'organizzazione e le dipendenze tra i moduli software.

Design Pattern Per risolvere problemi di progettazione ricorrenti e garantire la scalabilità di Code Guardian, il team adotta i seguenti pattern:

- **Pattern Creazionali:**
  - Singleton: per la gestione di risorse condivise come la connessione al database.
  - Factory: per istanziare diversi tipi di agenti di analisi in base al linguaggio del repository.
- **Pattern Strutturali:**
  - Adapter: per interfacciare il sistema con le diverse API di GitHub o altri provider[G] di versionamento.
- **Pattern Comportamentali:**
  - Strategy: per cambiare dinamicamente l'algoritmo di analisi (es. sicurezza vs. qualità del codice) senza modificare il core dell'agente[G].
  - Observer: per notificare all'utente o ai sistemi di logging l'avanzamento dell'audit in tempo reale.

Il gruppo non esclude la possibilità di adottare *pattern* aggiuntivi qualora se sorgesse la necessità.

2.2.2.3. Codifica

Attività designata ai Programmatori, i quali devono rendere codice sorgente di qualità ciò che viene deciso durante la progettazione del prodotto.

Gli sviluppatori devono seguire in maniera scolastica gli standard e le linee guida definiti nel *Piano di Qualifica* al fine di garantire la produzione di codice affidabile, pulito e facilmente manutenibile anche da chi non l'avesse mai letto prima.

Aspettative

L'obiettivo è la produzione di codice di qualità in linea con i requisiti e gli obiettivi del progetto che soddisfi le seguenti caratteristiche di qualità:
- Sia semplice e leggibile;
- Sia efficiente nelle prestazioni;
- Abbia una adeguata copertura dei test;
- Rispetti i requisiti del *Piano di Qualifica*

Stile di Codifica

Al fine di garantire la manutenibilità del codice gli Sviluppatori dovranno attenersi alle seguenti linee guida:

- **Nomenclatura**: Le variabili, le funzioni, le classi, i metodi devono avere un nome chiaro, univoco e distintivo, al fine di rendere la lettura più semplice e la Manutenibilità del codice più rapida;
- **Indentazione**: Gli spazi e le tabulazioni devono essere utilizzate in modo corretto e uniforme;
- **Sicurezza**: Il codice deve gestire ogni potenziale attacco in maniera corretta, senza presentare vulnerabilità;
- **Efficienza**: Il codice deve poter eseguire ottimizzando il consumo di tempo e di risorse;
- **Testabilità**: Il codice deve essere facilmente testabile; Le funzioni devono avere un solo punto di ritorno e devono gestire un unico scenario;

Metriche

Per rendere oggettiva la qualità del software prodotto, il codice verrà monitorato tramite le seguenti metriche:
- **Complessità Ciclomatica (MPD07):** Monitorata per garantire che ogni funzione rimanga testabile e atomica (valore accettabile ≤ 15, valore ottimale ≤ 10).

- **Campi delle Classi:** Si limita il numero di attributi per classe per favorire la coesione e rispettare il principio di singola responsabilità.

3. Processi di Supporto

3.1. Processo di Documentazione

La scrittura di documentazione è un processo essenziale per lo sviluppo software in quanto aiuta sia chi produce sia chi utilizza riguardo alla comprensione del prodotto.

Tramite la documentazione infatti vengono monitorate le attività svolte e quelle da svolgere al fine di lavorare a regola d'arte e produrre del software di qualità.

3.1.1. Obiettivi

Ciò che si pretende dalla documentazione è che essa rispetti delle regole al momento della redazione per risultare coerente e uniforme in tutti i documenti.

3.1.2. Descrizione

3.1.2.1. Ciclo di Vita dei Documenti

I documenti prodotti dal gruppo *HeptaCode* attraverseranno le seguenti tre fasi:

- **Redazione**: scrittura del documento da parte di vari membri del gruppo a turno in relazione ai ruoli ricoperti.
- **Revisione**: controllo dei contenuti del documento da parte di una persona terza alla redazione dello stesso.
- **Distribuzione**: una volta che il documento viene revisionato da chi di dovere, esso viene pubblicato sul sito web del gruppo.

3.1.2.2. Template Typst

Per garantire la uniformità della documentazione il gruppo ha deciso di creare dei template in Typst sia per la stesura dei verbali (sia interni che esterni), sia per la documentazione; questa idea, oltre a velocizzare la stesura dei verbali, serve anche ad assicurare ai redattori che le loro produzioni siano coerenti con gli standard del gruppo.

3.1.2.3. Documenti del Progetto

Il gruppo produce sia documenti «interni», quindi destinati a rimanere in mano ad esso, sia «esterni», cioè da mostrare a terzi come l'azienda o i professori, i documenti di questi due gruppi sono:

- **Interni**:
  - Norme di Progetto
  - Verbali Interni
- **Esterni**:
  - Analisi dei Requisiti
  - Piano di Qualifica
  - Piano di Progetto
  - Glossario
  - Verbali Esterni

3.1.2.4. Struttura dei Documenti

Il template della documentazione garantisce che ogni documento pubblicato presenti la seguente struttura prima del corpo del documento, si noti che questa formattazione non è da applicare ai verbali, il quale template è descritto sotto:

- **Prima Pagina**:
  - Logo del gruppo;
  - Nome del documento;
  - Versione del documento;
  - Contenuto del documento;
  - Indirizzo Email e link al sito web del gruppo.
- **Registro delle Modifiche**: La seconda pagina mostrerà una tabella formata da 5 colonne che traccia chi ha redatto e chi ha verificato il documento, le colonne della tabella sono:
  - **Versione**: La versione del documento in formato SemVer x.y.z;
  - **Data**: La data di redazione del documento;
  - **Autore**: Il nome e il cognome di chi ha apportato modifiche al documento;
  - **Verificatore**[G]: Il nome e il cognome di chi ha approvato le modifiche al documento;
  - **Descrizione**: Una concisa descrizione delle modifiche apportate.
- **Indici**: In seguito al Registro dei Cambiamenti ogni documento presenta 3 indici, l'*Indice dei Contenuti* elenca i capitoli e i paragrafi, l'*Indice delle Tabelle* elenca le etichette delle tabelle presenti e l'*Indice delle Immagini* elenca le etichette delle immagini del documento.

3.1.2.5. Struttura dei Verbali

I template dei Verbali differiscono da quelli della documentazione e seguono il seguente standard:

- **Prima Pagina**:
  - Logo del gruppo;
  - Nome e data del verbale;
  - Ordine del Giorno.
- **Registro delle Modifiche**: Il registro delle modifiche risulta invariato.
- **Ubicazione e Partecipanti**: Viene dichiarato il luogo e l'orario dell'incontro e la partecipazione dei membri del gruppo e, nel caso di riunioni con terzi, di questi ultimi.
- **Indice dei Contenuti**: In seguito al Registro dei Cambiamenti ogni documento presenta esclusivamente l'Indice dei Contenuti, non essendo i verbali esageratamente corposi, sono facilmente navigabili anche senza l'ausilio degli altri due.

3.1.2.6. Convenzioni Stilistiche

Ogni file di documentazione deve essere chiamato come il suo titolo, ed ogni file di verbale deve avere nel nome la data in formato YY-MM-GG al fine di rendere più rapida la ricerca e la consultazione dei documenti.

All'interno degli indici non deve essere usato il grassetto, utilizzabile invece per enfatizzare termini importanti ad esempio in elenchi puntati.

3.1.2.7. Strumenti

Per produrre la documentazione il gruppo adopera il linguaggio di markup Typst sfruttandone il sistema di templating per garantire coerenza grafica e uniformità strutturale tra i diversi documenti.

Per condividere i documenti tra i membri essi vengono caricati nella repository GitHub tramite la quale vengono anche effettuati controlli grammaticali e sintattici automatici con l'ausilio delle GitHub Actions.

3.1.2.8. Metriche

- **Indice di Gulpease (MPD14):** Utilizzato per misurare la leggibilità dei documenti in lingua italiana. Il team mira a un valore $ge 50$ per garantire che la documentazione sia accessibile.

- **Correttezza ortografica:** Conteggio degli errori rilevati tramite tool di linting o revisione umana. Il valore accettabile è rigorosamente 0.

3.2. Processo di Verifica

Il processo di verifica costituisce un'attività di supporto fondamentale volta ad accertare, mediante analisi e prove oggettive, che i prodotti intermedi e finali del progetto soddisfino i requisiti specificati in ogni fase del ciclo di vita.

L'obiettivo primario è duplice:

- *per il software*: garantire che il codice e le componenti tecniche siano conformi alle specifiche di progettazione e ai vincoli tecnici, minimizzando il rischio[G] di difetti logici o di implementazione attraverso test funzionali e revisioni del codice.
- *per la documentazione*: assicurare la correttezza formale, la coerenza sintattica e la precisione semantica dei testi prodotti, verificando che le informazioni trasmesse siano complete, aggiornate e aderenti agli standard qualitativi prefissati (come l'utilizzo rigoroso dei template in Typst).

In sintesi la verifica fornisce una base di fiducia necessaria per il rilascio del sistema e del materiale di supporto.

3.2.1. Obiettivi

- Identificare alla radice errori nel software e nella documentazione, riducendo i costi di correzione.
- Dimostrare che il software e la documentazione siano coerenti con dei requisiti pregressi.
- Accettarsi che il codice rispetti le linee guida stilistiche e la documentazione sia coerente con il template Typst.
- Generare documenti che attestino lo stato di progresso del progetto.

3.2.2. Descrizione

3.2.2.1. Analisi Statica

L'analisi statica viene eseguita senza l'esecuzione diretta del codice sorgente o dei documenti, focalizzandosi sulla struttura e sulla conformità formale. Per quanto riguarda il software, l'attività si concentra sulla revisione del codice per identificare potenziali vulnerabilità o errori logici. Per la documentazione, la verifica statica è supportata dal compilatore Typst, che garantisce la correttezza sintattica e il rispetto dei vincoli definiti nel template di gruppo. In questa fase, la verifica mira ad assicurare che ogni componente sia strutturalmente pronto per le fasi successive.

3.2.2.2. Analisi Dinamica

L'analisi dinamica prevede la verifica del sistema attraverso la sua esecuzione. Grazie all'integrazione di pipeline di **Continuous Integration**[G] tramite **GitHub Actions**, gran parte di questi processi viene automatizzata per garantire feedback rapidi al team di sviluppo.

3.2.2.2.1. Test di Unità

Questi test sono mirati a verificare il corretto funzionamento delle singole unità logiche (funzioni, classi o moduli) in modo isolato. Verranno implementati utilizzando framework[G] specifici per gli stack tecnologici scelti per validare la logica interna prima dell'integrazione.

3.2.2.2.2. Test di Integrazione

Considerando l'architettura basata su Database, Backend[G] e Frontend[G], i test di integrazione verificano che la comunicazione tra questi tre livelli avvenga correttamente. L'obiettivo è assicurare

che il flusso dei dati tra le API del backend e le collezioni del database sia coerente e privo di errori di interfaccia.

3.2.2.2.3. Test di Sistema

I test di sistema verificano l'intero applicativo nel suo complesso rispetto ai requisiti definiti nel capitolato. L'analisi mira a confermare che l'integrazione totale tra interfaccia utente e servizi rispetti le performance e le funzionalità attese.

3.2.2.2.4. Test di Regressione

Per ogni nuova funzionalità introdotta o bug corretto, vengono eseguiti test di regressione automatizzati tramite GitHub Actions. Questo assicura che le modifiche al codice non compromettano le funzionalità precedentemente verificate e funzionanti.

3.2.2.2.5. Test di Accettazione

Rappresentano la fase finale della verifica e mirano a confermare che il prodotto sia pronto per la consegna all'azienda cliente. Questi test si basano direttamente sui criteri di accettazione definiti con l'azienda, validando che il software soddisfi pienamente i requisiti contrattuali.

3.3. Processo di Validazione

Il processo di validazione fornisce la conferma, attraverso la fornitura di prove oggettive, che i requisiti per l'utilizzo specifico previsto siano stati soddisfatti. L'obiettivo è assicurare che il sistema realizzato sia conforme alle aspettative dell'azienda cliente e degli utenti finali.

3.3.1. Obiettivi

Le principali aspettative di questo processo includono:

- **Soddisfazione del capitolato**: dimostrare che ogni funzionalità richiesta dal committente sia stata implementata e risulti operativa;
- **Idoneità all'uso**: accertare che l'interfaccia e le logiche di business siano intuitive e risolvano i problemi reali dell'utente;
- **Accettazione formale**: ottenere l'approvazione finale dell'azienda esterna attraverso la dimostrazione del prodotto completo.

3.3.2. Descrizione

3.3.2.1. Proof of Concept (PoC)

Il gruppo ha previsto la realizzazione di un **Proof of Concept** come milestone fondamentale del processo di validazione.

Lo scopo del PoC è:

- **Validazione tecnologica**: dimostrare che lo stack scelto sia idoneo a soddisfare i requisiti critici del capitolato;
- **Riduzione del rischio**: identificare tempestivamente eventuali limiti tecnici o architetturali prima della fase di sviluppo intensivo;
- **Feedback immediato**: fornire all'azienda e ai docenti una prova tangibile del funzionamento, permettendo di validare la direzione intrapresa.

3.4. Processo di Gestione della Configurazione

Il processo assicura l'integrità degli artefatti e la tracciabilità delle evoluzioni, permettendo al gruppo di lavorare in parallelo minimizzando i conflitti e garantendo la riproducibilità di ogni versione rilasciata.

3.4.1. Obiettivi

La gestione della configurazione ha l'obiettivo di identificare, organizzare e controllare le modifiche a tutti i prodotti del progetto (codice e documentazione).

3.4.2. Descrizione

3.4.2.1. Codice di Versionamento

Il gruppo adotta Git come sistema di controllo di versione. La strategia di branching prevede l'utilizzo di un ramo principale destinato alle versioni stabili.

- **Documentazione**: si utilizza un approccio «branch[G]-per-documento», dove ogni nuova sezione o revisione viene sviluppata in un ramo dedicato prima di essere integrata nel *main* tramite operazione di merge.
- **Software**: si prevede l'adozione di un modello simile, orientato alla gestione di funzionalità isolate per mantenere il codice sorgente sempre testabile.

3.4.2.2. Tecnologie Adottate

Il controllo di versione e la collaborazione sono centralizzati sulla piattaforma GitHub. Per garantire l'uniformità degli ambienti di sviluppo è previsto l'utilizzo di Docker, che permetterà di containerizzare i servizi di backend, frontend e database. Per il software, la gestione delle versioni seguirà lo standard Semantic Versioning, garantendo una comunicazione chiara sull'entità degli aggiornamenti rilasciati.

3.4.2.3. Repository

Il progetto è organizzato in una struttura con due repository per separare logicamente le responsabilità:

- **Repository Documentale**: ospitato su una repo[G] specifica, contiene esclusivamente i file sorgente Typst e le configurazioni per il sito web statico.
- **Repository Software**: organizzato come monorepo, contenente cartelle distinte per il frontend e il backend, facilitando la gestione delle dipendenze correlate e dei Dockerfile di orchestrazione.

3.4.2.4. Sincronizzazione

La sincronizzazione del lavoro tra i membri del gruppo è gestita attraverso:

- **GitHub Issues e Project Board[G]**: per la pianificazione e l'assegnazione dei task, permettendo di collegare ogni modifica a una specifica esigenza di progetto.
- **Continuous Integration**: per la documentazione, è attiva una GitHub Action che interviene ad ogni aggiornamento dei file `.typ`. Il runner compila automaticamente i sorgenti e genera i PDF aggiornati.
- **Gestione delle Pull[G] Request**: per garantire la stabilità del ramo `main`, il gruppo adotta il meccanismo delle Pull Request come filtro di qualità. Prima dell'integrazione definitiva, la CI[G] verifica che il codice (o il documento) sia privo di errori sintattici; il merge è consentito solo in caso di esito positivo della compilazione, impedendo che errori accidentali interrompano la disponibilità della documentazione online.

3.5. Processo di Gestione della Qualità

3.5.1. Obiettivi

L'obiettivo principale è garantire che il prodotto finale CodeGuardian e i processi utilizzati per realizzarlo soddisfino i requisiti di qualità prestabiliti.

Il team punta a:

- Prevenzione dei difetti: identificare potenziali problemi nelle fasi iniziali di pianificazione e progettazione.
- Miglioramento continuo: utilizzare i dati raccolti dalle metriche per ottimizzare le performance del team.
- Soddisfazione degli stakeholder: assicurare che il software sia affidabile, sicuro e conforme alle richieste di *Var Group*.

3.5.2. Descrizione

3.5.2.1. PDCA

Plan

Definizione degli obiettivi di qualità e dei processi necessari per ottenere i risultati attesi.

Do

Attuazione dei processi definiti e raccolta dei dati necessari per le misurazioni.

Check

Monitoraggio e misurazione dei processi e dei prodotti rispetto agli obiettivi e ai requisiti.

Act

Adozione di azioni correttive per migliorare continuamente le prestazioni dei processi.

3.5.2.2. Strumenti

Per la gestione della qualità, il gruppo utilizza:

- GitHub Actions: per l'esecuzione automatica dei test e il calcolo del Code Coverage[G] ad ogni Pull Request.
- Linter: per il monitoraggio statico della qualità del codice (complessità, densità commenti, aderenza agli standard).
- Fogli di calcolo / Dashboard[G]: per l'aggregazione delle metriche di processo e la generazione di grafici di andamento per le Revisioni di Avanzamento.

3.5.2.3. Struttura delle Metriche

Ogni metrica all'interno delle norme è identificata da un codice univoco:

- **MPC (Metrica di Processo):** valuta l'efficienza delle attività e l'uso delle risorse.
- **MPD (Metrica di Prodotto):** valuta le caratteristiche intere ed esterne del software e della documentazione.

Ogni metrica deve essere corredata da: Nome, Formula, Valore Accettabile (soglia minima di conformità) e Valore Ottimale (obiettivo di eccellenza).

3.5.2.4. Struttura degli Obiettivi

Gli obiettivi di qualità sono derivati dallo standard ISO/IEC 9126. Per ogni caratteristica di qualità, il team definisce degli obiettivi misurabili tramite una o più metriche. Il superamento dei «Valori Accettabili» attiva immediatamente una fase di *Act* nel ciclo PDCA per riportare il parametro sotto controllo.

3.5.2.5. Metriche

4. Processi Organizzativi

Insieme strutturato di attività correlate tra loro svolte in modo coordinato per raggiungere un obiettivo specifico e creare valore.
I processi organizzativi qui presenti sono:

1. Gestione dei Processi
2. Processo di Miglioramento
3. Processo di Formazione

4.1. Processo di Gestione

Il Responsabile è incaricato della gestione del prodotto, della gestione del progetto e della gestione delle attività dei processi applicabili.

4.1.1. Obiettivi

L'obiettivo principale del Processo di Gestione è assicurare che i processi siano pianificati, eseguiti, monitorati e completati in modo efficace ed efficiente, garantendo il raggiungimento degli obiettivi prefissati e la conformità ai requisiti stabiliti.

4.1.2. Descrizione

4.1.2.1. Definizione dello Scopo

Il processo di gestione ha inizio con la definizione degli obiettivi e dei requisiti del processo da svolgere.
Una volta chiariti tali requisiti, il responsabile (in accordo con il gruppo) valuta se il processo è realizzabile, verificando che le risorse necessarie siano disponibili e adeguate e che i tempi previsti per il completamento possano essere rispettati.
I requisiti possono essere adattati in questa fase per garantire il raggiungimento degli obiettivi finali.

4.1.2.2. Pianificazione delle Attività

Il Responsabile definisce le attività necessarie per realizzare il processo.
Le attività descrivono i compiti previsti e i prodotti software da consegnare. Inoltre specificano tutti gli elementi utili a garantire una corretta esecuzione del processo, tra cui:

- le tempistiche per completare le attività;
- la stima dell'impegno richiesto;
- le risorse necessarie;
- la distribuzione dei compiti;
- le responsabilità assegnate;
- i rischi potenziali legati alle attività o al processo;
- le modalità di controllo della qualità;
- i costi previsti;
- le risorse infrastrutturali e l'ambiente operativo necessari.

4.1.2.3. Controllo ed Esecuzione

Il responsabile avvia il piano per raggiungere gli obiettivi prefissati, mantenendo il controllo sull'intero processo.
Durante l'esecuzione, deve monitorare i progressi, fornendo aggiornamenti interni e (se necessario) ai professori o all'azienda.

> **Se sorgono problemi:**
> il Responsabile deve analizzarli e risolverli. Eventuali soluzioni possono comportare modifiche

ai piani, e spetta al Responsabile valutare e controllare l'impatto di questi cambiamenti.
Tutti i problemi e le relative soluzioni devono essere documentati.
Infine, il responsabile deve informare periodicamente sullo stato del processo, confermando l'aderenza ai piani e affrontando eventuali ritardi o ostacoli, tramite report interni ed esterni secondo le procedure.

4.1.2.4. Valutazione

Il Responsabile deve garantire che prodotti software e piani siano esaminati per verificare che soddisfino i requisiti previsti.
Deve inoltre analizzare i risultati delle valutazioni dei prodotti software, delle attività e dei compiti completati durante il processo, per *accertarsi che gli obiettivi siano stati raggiunti e che i piani siano stati portati a termine.*

4.1.2.5. Chiusura dell'Attività

Una volta completati tutti i prodotti software, le attività e i compiti, il Responsabile deve *verificare* se il processo può considerarsi terminato, seguendo i criteri definiti nel contratto o nelle procedure dell'organizzazione. Inoltre, deve controllare che i risultati e la documentazione relativi ai prodotti software, alle attività e ai compiti siano completi, archiviare tutto all'interno del repository del gruppo e pubblicare la documentazione nel sito.

4.2. Processo di Miglioramento

Processo per stabilire, valutare, misurare, controllare e migliorare un processo del ciclo di vita del software.

4.2.1. Obiettivi

L'obiettivo principale del Processo di Miglioramento è garantire che i processi del ciclo di vita del software siano continuamente valutati e migliorati per aumentare l'efficacia, l'efficienza e la qualità complessiva del prodotto software.

4.2.2. Descrizione

4.2.2.1. Stabilimento

Il gruppo definisce un insieme di processi per l'intero ciclo di vita del software.
Tali processi, insieme alle modalità di applicazione a casi specifici, devono essere documentati nella documentazione del gruppo.
Quando necessario, *deve essere implementato un meccanismo di controllo dei processi* per *garantirne lo sviluppo, il monitoraggio, la gestione e il miglioramento continuo.*

4.2.2.2. Valutazione

Deve essere creata, documentata e messa in pratica una procedura per la valutazione dei processi i relativi documenti sulla valutazione devono essere conservati e aggiornati.
Vanno effettuate periodicamente revisioni dei processi, al fine di assicurarne la continua adeguatezza ed efficacia sulla base dei risultati delle valutazioni.

4.2.2.3. Miglioramento

Devono essere implementati (da parte del gruppo) i miglioramenti ai processi ritenuti necessari a seguito delle valutazioni e delle revisioni, aggiornando la documentazione per riflettere le modifiche apportate.

4.3. Processo di Formazione

Il Processo di Formazione ha lo scopo di garantire una preparazione adeguata da parte del gruppo per lo svolgimento del progetto.

Per questo motivo, è essenziale pianificare e avviare la formazione fin dalle prime fasi, così che risorse qualificate siano pronte quando il prodotto software viene implementato o mantenuto.

4.3.1. Obiettivi

Ogni membro del gruppo deve acquisire le competenze necessarie per svolgere i propri compiti in modo efficace.
La formazione (o *palestra*) deve essere continua, adattandosi alle esigenze del progetto ed acquisendo una conoscenza approfondita delle tecnologie e metodologie utilizzate.
L'obiettivo è garantire che il team sia sempre aggiornato e in grado di affrontare i compiti richiesti dal progetto.

4.3.2. Descrizione

4.3.2.1. Formazione dei Membri del Gruppo

È necessario *esaminare i requisiti* del progetto per identificare e pianificare tempestivamente lo sviluppo delle risorse e delle competenze richieste al team.
Sulla base di queste informazioni, ogni membro deve organizzare autonomamente la propria formazione, utilizzando:

- risorse interne (università, docenti);
- risorse esterne (formazione aziendale, materiale online);
- eventualmente, il gruppo può decidere di organizzare sessioni di formazione interne per condividere conoscenze specifiche.

5. Standard per la Qualità

Il modello di qualità di riferimento per il progetto è definito dallo standard *ISO/IEC 9126-1*. Tale norma decompone la qualità del software in sei caratteristiche fondamentali:

1. Funzionalità
2. Affidabilità
3. Usabilità
4. Efficienza
5. Manutenibilità
6. Portabilità

5.1. Funzionalità

La Funzionalità definisce la capacità del software di erogare servizi e funzioni che rispondano alle esigenze (esplicite o implicite) dell'utente, quando utilizzato sotto specifiche condizioni. Essa rappresenta il nucleo dei requisiti funzionali e si articola nelle seguenti sotto caratteristiche:

5.1.1. Appropriatezza

Il software deve essere in grado di fornire una serie di funzioni appropriate per lo svolgimento di compiti e il raggiungimento degli obiettivi dell'utente senza complicazioni aggiuntive.

5.1.2. Accuratezza

I risultati e i calcoli prodotti dal software devono essere precisi e corrispondere sempre a quanto richiesto dagli standard del progetto.

5.1.3. Interoperabilità

La capacità del sistema di interagire e scambiare informazioni con uno o più sistemi specificati. Nel contesto del progetto *CodeGuardian*, questa caratteristica è fondamentale per garantire l'integrazione con la piattaforma esterna *GitHub* , permettendo l'accesso e la gestione delle repository.

5.1.4. Conformità
Il rispetto di tutte le regole e gli standard del settore in cui il software andrà a operare.

5.1.5. Sicurezza
La capacità del software di proteggere le informazioni e i dati, negando l'accesso a entità non autorizzate e garantendolo a quelle autorizzate. Nel contesto di *CodeGuardian*, non essendo prevista la memorizzazione diretta di dati sensibili o credenziali utente (delegate alla piattaforma GitHub), non è stato implementato un modulo di sicurezza proprietario complesso. Tuttavia, vengono adottate le **best practices** di sviluppo sicuro per prevenire vulnerabilità comuni nel trattamento dei dati analizzati.

5.2. Affidabilità
La capacità del software di mantenere un determinato livello di prestazioni quando utilizzato sotto condizioni specifiche per un determinato periodo di tempo.

5.2.1. Maturità
La capacità del software di evitare malfunzionamenti (failure) causati da difetti interni. Si ottiene attraverso test rigorosi che mirano a stabilizzare il prodotto, riducendo la frequenza di errori prima del rilascio.

5.2.2. Tolleranza agli errori (Fault Tolerance)
La capacità del sistema di mantenere un livello di prestazioni specificato anche in caso di guasti software o violazioni delle interfacce. Il sistema deve essere resiliente: gestire l'anomalia senza bloccarsi, garantendo il degrado controllato delle funzionalità o il ripristino di uno stato stabile.

5.2.3. Conformità
La capacità del prodotto software di aderire a standard, convenzioni o regolamentazioni relative all'affidabilità. I dettagli metrici sono specificati nel *Piano di Qualifica*.

5.3. Efficienza
La capacità del software di fornire prestazioni appropriate relative alla quantità di risorse utilizzate, sotto condizioni specifiche.

5.3.1. Comportamento rispetto al tempo
La capacità del software di fornire tempi di risposta, tempi di elaborazione e velocità di throughput (capacità di calcolo) che soddisfino i requisiti stabiliti durante l'esecuzione delle sue funzioni.

5.3.2. Utilizzo delle risorse
La capacità del software di utilizzare quantità e tipi di risorse in misura appropriata per svolgere i propri compiti, ottimizzando i consumi ed evitando sprechi ingiustificati.

5.3.3. Conformità
La capacità del prodotto di aderire a standard o convenzioni relative all'efficienza prestazionale, nel rispetto dei vincoli definiti nel *Piano di Qualifica*.

5.4. Usabilità
La capacità del prodotto software di essere semplice, intuitivo e di risultare piacevole per l'utente. Un prodotto usabile è un prodotto che non richiede sforzi eccessivi per essere capito e utilizzato.

5.4.1. Comprensibilità
La capacità del software di permettere all'utente di comprendere se il sistema è adatto ai propri compiti e di come utilizzarlo. L'utente deve poter identificare le funzionalità principali e la logica operativa fin dal primo impatto.

5.4.2. Apprendibilità
La capacità del software di essere intuitivo e permettere all'utente di imparare ad utilizzarlo con uno sforzo ridotto e in tempi minimi.

5.4.3. Operabilità
La capacità del software di mettere l'utente in condizione di operarlo e controllarlo interamente. Include la facilità di navigazione, la gestione degli errori e l'assenza di ostacoli nel completamento dei compiti.

5.4.4. Attrattiva
La capacità del software di essere piacevole per l'utente. L'interfaccia grafica deve essere curata, coerente e progettata per garantire un'esperienza d'uso soddisfacente.

5.4.5. Conformità
La capacità del software di aderire a standard, convenzioni o regolamentazioni relative alle linee guida consolidate di accessibilità.

5.5. Manutenibilità
La capacità del prodotto software di essere modificato. Le modifiche possono includere correzioni, miglioramenti o adattamenti a cambiamenti nell'ambiente senza compromettere ciò che già funziona.

5.5.1. Analizzabilità
La capacità del software di essere diagnosticato velocemente per cause di guasti o per identificare le parti da modificare. Il codice deve rispettare alcuni requisiti per ridurre il tempo di analisi e trovare con rapidità la natura di un problema: deve essere scritto in modo pulito, deve essere chiaro nella scelta dei nomi, deve essere modulare e avere una documentazione chiara.

5.5.2. Modificabilità
La capacità del software di permettere l'implementazione di una modifica. La struttura deve essere a basso accoppiamento o modulare affinché la modifica di un componente non richieda la riscrittura di moduli dipendenti.

5.5.3. Stabilità
La capacità del software di minimizzare gli effetti collaterali imprevisti derivanti da modifiche. Ogni modifica deve essere sicura, il sistema deve essere solido abbastanza da evitare che la correzione di un bug ne generi altri in parti diverse del programma (*effetti collaterali*); lo si farà tramite test di regressione.

5.5.4. Testabilità
La capacità del software di essere testato facilmente e in modo automatico. Il sistema deve essere progettato per facilitare la scrittura e l'esecuzione di test automatici che verifichino la correttezza delle nuove implementazioni.

5.6. Portabilità
La capacità del software di essere trasferito da un ambiente a un altro, sia che si tratti di hardware[G] sottostante o del sistema operativo.

5.6.1. Adattabilità
La capacità del software di adattarsi a diversi ambienti specificati senza dover applicare azioni pesanti o utilizzare versioni dedicare per ogni sistema. Nel contesto di *CodeGuardian*, l'adozione della containerizzazione (Docker) garantisce che l'applicativo possa essere eseguito su qualsiasi sistema host supportato senza conflitti di dipendenze.

5.6.2. Installabilità
La capacità del software di essere installato in un ambiente specificato. La procedura di configurazione e deploy deve essere semplice, rapida e documentata per l'ambiente di destinazione.

5.6.3. Conformità
La capacità del software di aderire a standard o convenzioni relative alla portabilità. Nel contesto di *CodeGuardian*, tale conformità è garantita tramite l'utilizzo di container Docker. Questo approccio evita il legame con tecnologie proprietarie o specifiche di sistema, assicurando che il software rispetti le specifiche universali di distribuzione.

6. Metriche per la Qualità
La qualità del prodotto software non è il risultato di un processo di valutazione che confronta il prodotto ottenuto con le attese prefissate. Secondo la norma ISO, la qualità è definita come l'insieme delle caratteristiche di un'entità che le conferiscono la capacità di soddisfare esigenze espresse (esplicite) ed implicite.

La valutazione della qualità dipende dal punto di vista dell'osservatore:

- **Prospettiva interna** (Sviluppatore[G]): Focalizzata sulla struttura, la manutenibilità e la correttezza tecnica del codice.
- **Prospettiva esterna** (Utente): Focalizzata sull'uso, sull'efficacia e sulla soddisfazione dei bisogni.
- **Prospettiva terza** (Committente): Focalizzata sulla conformità ai processi e sugli standard contrattuali.

6.1. Metriche Interne
Le metriche interne (*riferimento normativo: ISO/IEC 9126-3*) si applicano alla struttura statica del software (non eseguibile) durante le fasi di progettazione e codifica. Tali metriche valutano attributi architetturali quali: la modularità, la completezza della documentazione, la coesione e il livello di accoppiamento tra i componenti. L'obiettivo è garantire che il codice sorgente possieda elevate caratteristiche di: manutenibilità, modificabilità, analizzabilità e verificabilità.

Le metriche interne sono spesso invisibili all'utente finale, ma il loro monitoraggio è propedeutico al raggiungimento della qualità esterna e in uso, gli attributi interni influenzano direttamente il comportamento del software a runtime e l'adozione di queste metriche consente di:

- **Prevedere la qualità finale del software**, stimando il livello qualitativo del prodotto prima del rilascio.
- **Individuare anomalie strutturali e difetti potenziali** prima che venga prodotto l'eseguibile, riducendo i costi di correzione.
- Assicurare che **il progetto possa evolvere nel tempo** senza degradare in debito tecnico ingestibile.

6.2. Metriche Esterne

Le metriche esterne (riferimento normativo: *ISO/IEC 9126-2*) misurano il comportamento del sistema software in esecuzione (runtime). A differenza delle metriche interne, queste misure si basano sulla validazione dinamica, sull'operatività e sull'osservazione degli output in ambienti di test o in simulazione operativa.

L'obiettivo è verificare la conformità del software rispetto ai requisiti tecnici e di business stabiliti, garantendo che il prodotto soddisfi le esigenze dell'utente quando viene eseguito in un contesto reale.

Le metriche esterne permettono di quantificare attributi qualitativi quali:

- **Adeguatezza funzionale**: Il software fornisce le funzioni appropriate per i task previsti.
- **Affidabilità**: Il software mantiene le prestazioni nel tempo senza rompersi.
- **Efficienza**: I tempi di risposta e l'uso delle risorse sono adeguati.
- **Usabilità**: L'utente riesce a capire e operare il software con facilità.
- **Sicurezza**: I dati e gli accessi sono protetti durante l'esecuzione.

6.3. Metriche della Qualità in Uso

La Qualità in uso (riferimento normativo: *ISO/IEC 9126-4*) rappresenta il punto di vista dell'utente finale nell'interazione reale con il sistema. Essa può essere conseguita pienamente solo se sono stati precedentemente soddisfatti i requisiti di qualità interna e qualità esterna.

Mentre le metriche tecniche misurano il codice o il comportamento del sistema, la qualità in uso misura l'impatto del software sull'utente. L'obiettivo è validare se il sistema abilita specifici utenti a raggiungere specifici obiettivi in un determinato contesto d'uso.

Gli attributi misurati sono:

- **Efficacia**: La capacità del software di permettere agli utenti di raggiungere gli obiettivi prefissati con accuratezza e completezza.
- **Efficienza**: La relazione tra l'efficacia ottenuta e le risorse spese (tempo, sforzo cognitivo, materiali).
- **Sicurezza** (Safety): La capacità del software di mantenere i livelli di rischio per le persone, l'ambiente o le apparecchiature entro soglie accettabili.
  L'attributo della sicurezza si riferisce all'incolumità e ai danni fisici o economici, distinguendosi dalla protezione dei dati (Security). Nel contesto del progetto *CodeGuardian*, questa metrica avrà un peso minore rispetto all'efficacia e all'efficienza, in quanto il software non presenta rischi critici per l'incolumità delle persone o dell'ambiente.
- **Soddisfazione**: La risposta soggettiva dell'utente all'interazione con il sistema.

6.4. Metriche della Qualità di Processo

La qualità del prodotto software è una diretta conseguenza della qualità dei processi utilizzati per realizzarlo. Un processo di sviluppo può essere modellato come un sistema che trasforma input (bisogni, requisiti) in output (prodotti intermedi o finali), consumando risorse quali tempo, sforzo umano e strumenti. Per garantire che l'output sia quello desiderato, il processo necessita di un sistema di controllo (feedback loop), composto da:

- Metriche: Misurazioni oggettive per conoscere lo stato istantaneo del processo.

- Azioni correttive: Decisioni e regole atte a modificare l'andamento del processo per riportarlo nelle misure previste.

L'approccio adottato segue il ciclo di PDCA (Plan-Do-Check-Act): le misurazioni devono essere preventive e continue, non solo a posteriori, per permettere un miglioramento iterativo del processo stesso durante il ciclo di vita del progetto.

6.4.1. Miglioramento

6.4.1.1. MPC01 - Schedule Variance (SV)

- **Formula**:

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

- **Valore accettabile**: $\geq -10\%$ del PV
- **Valore ottimale**: 0
- **Descrizione**: L'indicatore Schedule Variance rappresenta il divario, espresso in termini monetari, tra il valore del lavoro effettivamente realizzato (Earned Value) e quello del lavoro che era stato pianificato (Planned Value) alla data corrente. Se il valore è superiore a zero, significa che il team sta procedendo più velocemente rispetto alla pianificazione iniziale. Un valore negativo indica invece un ritardo, quantificando economicamente il lavoro ancora da recuperare per rientrare nella tabella di marcia.

6.4.1.2. MPC02 - Cost Variance (CV)

- **Formula**:

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

- **Valore accettabile**: $\geq 0€$
- **Valore ottimale**: $\geq 0€$
- **Descrizione**: L'indicatore Cost Variance rappresenta la differenza tra il costo realmente raggiunto dal progetto e quello del lavoro effettivamente svolto. Se si raggiunge un valore positivo, allora lo svolgimento del lavoro risulta più efficiente, viceversa se negativo. Se invece risulta pari a 0, si sta procedendo come pianificato.

6.4.1.3. MPC03 - Budget Variance (BV)

- **Formula**:

$$\mathrm{BV} = \frac{\mathrm{PV} - \mathrm{AC}}{\mathrm{PV}} \cdot 100$$

- **Valore accettabile**: $-10\% \leq \mathrm{BV} \leq 10\%$
- **Valore ottimale**: 0%
- **Descrizione**: L'indice misura lo scostamento percentuale tra il costo pianificato (PV) e il costo effettivo sostenuto (AC) alla data corrente.

6.4.1.4. MPC04 - Cost Performance Index (CPI)

- **Formula**:

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

- **Valore accettabile**: $\geq 90\%$
- **Valore ottimale**: 100%

- **Descrizione**:L'indice di Cost Performance misura la resa del budget attraverso il rapporto tra: il valore del lavoro completato e il costo realmente sostenuto.

6.4.2. Fornitura

6.4.2.1. MPC05 - Planned Value (PV)

- **Formula**

$$\mathrm{PV} = \mathrm{BAC} \cdot \frac{\mathrm{PH}}{\mathrm{THP}}$$

- **Valore accettabile**: $\geq 0€$
- **Valore ottimale**: $\leq \mathrm{BAC}$
- **Descrizione**: L'indicatore Planned Value rappresenta il valore del lavoro pianificato rispetto al budget totale previsto. Tale valore permette di monitorare l'avanzamento del progetto e la conformità con la sua pianificazione iniziale.
- **Legenda**:
  - **BAC**: Budget at Completion (Budget totale preventivato);
  - **PH**: Planned Hours;
  - **THP**: Total Hours Planned.

6.4.2.2. MPC06 - Earned Value (EV)

- **Formula**:

$$\mathrm{EV} = \mathrm{BAC} \cdot \frac{\mathrm{AH}}{\mathrm{THP}}$$

- **Valore accettabile**: $\geq 0€$
- **Valore ottimale**: $\leq \mathrm{EAC}$
- **Descrizione**: L'indicatore Earned Value rappresenta il valore del lavoro completato rispetto al budget totale previsto. Tale valore permette di valutare se lo svolgimento del progetto è rimasto in linea con le aspettative.
- **Legenda**:
  - **BAC**: Budget at Completion (Budget totale preventivato);
  - **AH**: Actual Hours;
  - **THP**: Total Hours Planned.

6.4.2.3. MPC07 - Actual Cost (AC)

- **Formula**:

$$\mathrm{AC} = \sum_{r}^{R} (\mathrm{AHR}_r \cdot \mathrm{HCR}_r)$$

- **Valore accettabile**: $\geq 0€$
- **Valore ottimale**: $\leq \mathrm{EAC}$
- **Descrizione**: L'indice Actual Cost rappresenta il costo effettivamente sostenuto alla data corrente.
- **Legenda**:
  - **AHR**: Actual Hours by Role;
  - **HCR**: Hourly Cost per Role.

6.4.2.4. MPC08 - Estimate at Completion (EAC):

- **Formula**:

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

- **Valore accettabile**: $\geq \mathrm{BAC} - 5\%$
- **Valore ottimale**: $\leq \mathrm{BAC} + 5\%$
- **Descrizione**: L'indice Estimate at Completion rappresenta la stima del costo totale del progetto al momento del suo completamento.
- **Legenda**:
  - **BAC**: Budget at Completion (Budget totale preventivato);
  - **CPI**: Cost Performance Index.

6.4.2.5. MPC09 - Estimate to Complete (ETC)

- **Formula**:

$$\mathrm{ETC} = \mathrm{BAC} - \mathrm{EV}$$

- **Valore accettabile**: $\geq 0€$
- **Valore ottimale**: $\leq \mathrm{EAC}$
- **Descrizione**: L'indice Estimate to Complete rappresenta la stima dei costi necessari per completare tutte le attività previste rimanenti.
- **Legenda**:
  - **BAC**: Budget at Completion (Budget totale preventivato).

6.4.3. Verifica e validazione

6.4.3.1. MPC10 - Code Coverage (CC)

- **Formula**:

$$\mathrm{CC} = \frac{\text{Codice testato}}{\text{Codice totale}} * 100$$

- **Valore accettabile**: $\geq 80\%$
- **Valore ottimale**: $\geq 90\%$
- **Descrizione**: L'indice di Code Coverage misura la percentuale di codice sorgente che viene eseguita durante l'esecuzione dei test automatici. Indica quanto il codice è stato verificato dal processo di testing.

6.4.3.2. MPC11 - Test Success Rate (TSR)

- **Formula**:

$$\mathrm{TSR} = \frac{\text{Test passati}}{\text{Test totali}} * 100$$

- **Valore accettabile**: 85%
- **Valore ottimale**: 100%
- **Descrizione**: L'indice Test Success Rate misura la percentuale dei test superati rispetto a quelli totali.

6.4.3.3. MPC12 - Statement Coverage (SC)

- **Formula**:

$$\text{SC} = \frac{\text{Linee eseguite}}{\text{Linee totali}} * 100$$

- **Valore accettabile**: $\geq 90\%$
- **Valore ottimale**: 100%
- **Descrizione**: L'indice Statement Coverage misura la percentuale di istruzioni elementari eseguire dai test.

6.4.3.4. MPC13 - Branch Coverage (BC)

- **Formula**:

$$\text{BC} = \frac{\text{Branch eseguiti}}{\text{Branch totali}} * 100$$

- **Valore accettabile**: $\geq 70\%$
- **Valore ottimale**: $\geq 80\%$
- **Descrizione**: L'indice Branch Coverage misura la percentuale di rami decisionali (i percorsi true e false delle condizioni if, while, for) che sono stati percorsi dai test.

6.4.4. Documentazione

6.4.4.1. MPC14 - Correttezza ortografica

- **Formula**:

$$\text{Correttezza ortografica} = \text{numero di errori ortografici}$$

- **Valore accettabile**: 0
- **Valore ottimale**: 0
- **Descrizione**: La correttezza ortografica è un indicatore della qualità della documentazione. Tutti i documenti ufficiali devono essere privi di errori ortografici.

6.5. Metriche della Qualità di Prodotto

6.5.1. Funzionalità

6.5.1.1. MPD01 - Requisiti obbligatori soddisfatti (RS)

- **Formula**:

$$\text{RS} = \frac{\text{Requisiti obbligatori soddisfatti}}{\text{Requisiti obbligatori totali}} \cdot 100$$

- **Valore accettabile**: 100%
- **Valore ottimale**: 100%
- **Descrizione**: L'indice dei requisiti obbligatori soddisfatti rappresenta la percentuale dei requisiti obbligatori completati.

6.5.1.2. MPD02 - Requisiti desiderabili soddisfatti (RDS)

- **Formula**:

$$\text{RDS} = \frac{\text{Requisiti desiderabili soddisfatti}}{\text{Requisiti desiderabili totali}} \cdot 100$$

- **Valore accettabile**: $\geq 0\%$
- **Valore ottimale**: $\geq 60\%$

- **Descrizione**: L'indice dei requisiti desiderabili soddisfatti rappresenta la percentuale dei requisiti desiderabili completati.

6.5.1.3. MPD03 - Requisiti opzionali soddisfatti (ROS)

- **Formula**:

$$\text{ROS} = \frac{\text{Requisiti opzionali soddisfatti}}{\text{Requisiti opzionali totali}} \cdot 100$$

- **Valore accettabile**: $\geq 0\%$
- **Valore ottimale**: $\geq 30\%$
- **Descrizione**: L'indice dei requisiti opzionali soddisfatti rappresenta la percentuale dei requisiti opzionali completati.

6.5.2. Efficienza

6.5.2.1. MPD04 - Tempo di caricamento

- **Valore accettabile**: ≤ 10 secondi
- **Valore ottimale**: ≤ 5 secondi
- **Descrizione**:L'indice del tempo di caricamento misura il tempo medio che il programma impiega per avviarsi.

6.5.2.2. MPD05 -Tempo medio di risposta (Sistema)

- **Valore accettabile**: ≤ 5 secondi
- **Valore ottimale**: ≤ 2 secondi
- **Descrizione**: Misura il tempo medio che il sistema impiega per rispondere a operazioni tecniche standard che non coinvolgono l'elaborazione dell'IA. Garantisce la fluidità dell'interazione tecnica.

6.5.2.3. MPD06 -Tempo medio di risposta (Elaborazione AI)

- **Valore accettabile**: ≤ 15 minuti
- **Valore ottimale**: ≤ 10 minuti
- **Descrizione**: Misura il tempo medio che il sistema impiega per restituire un risultato quando viene sollecitata una richiesta utente che coinvolge il motore di Intelligenza Artificiale.

Nel contesto di *CodeGuardian*, l'uso di modelli LLM[G] per l'analisi e il refactoring del codice nelle repository GitHub richiede un tempo di ragionamento computazionale più alto rispetto alle operazioni standard.

6.5.3. Manutenibilità

6.5.3.1. MPD07 - Complessità Ciclomatica

- **Formula**:

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

- **Valore accettabile**: ≤ 15
- **Valore ottimale**: ≤ 10
- **Descrizione**: Misura la complessità logica di un metodo calcolando il numero di cammini linearmente indipendenti attraverso il grafo di controllo del flusso.

Un valore elevato indica che la funzione ha troppe logiche condizionali (if, loop) ed è quindi difficile da testare e soggetta a errori. Se il valore supera la soglia accettabile, è necessario suddividere il metodo in sotto-funzioni più semplici (Refactoring).

- **Legenda**:

- $v(G)$: Numero ciclomatico del grafo G;
- E: Numero di archi (collegamenti) nel grafo di controllo;
- N: Numero di nodi (blocchi di istruzioni) nel grafo di controllo;
- P: Numero di componenti connesse (generalmente $P = 1$ calcolando la metrica per singolo metodo).

6.5.3.2. MPD08 - Parametri per metodo

- **Valore accettabile**: ≤ 6
- **Valore ottimale**: ≤ 4
- **Descrizione**: L'indice rappresenta il numero massimo di parametri che un metodo può accettare nella sua firma all'interno del codice sorgente. Un numero eccessivo di parametri riduce la leggibilità e aumenta la complessità di test.

6.5.3.3. MPD09 - Linee di codice per metodo

- **Valore accettabile**: ≤ 35
- **Valore ottimale**: ≤ 20
- **Descrizione**: L'indice rappresenta il numero massimo di righe di codice da cui può essere composto un singolo metodo. Metodi troppo lunghi violano il principio di «Single Responsibility».

6.5.3.4. MPD10 - Linee di codice per file

- **Valore accettabile**: ≤ 120
- **Valore ottimale**: ≤ 80
- **Descrizione**: L'indice rappresenta il numero massimo di righe di codice da cui può essere composto un singolo file sorgente nel progetto.

6.5.3.5. MPD11 - Densità dei commenti (CD^G)

- **Formula**:

$$\mathrm{CD} = \left(\frac{\mathrm{CM}}{\mathrm{CL}}\right) \cdot 100$$

- **Valore accettabile**: $\geq 10\%$
- **Valore ottimale**: $\geq 15\%$
- **Descrizione**:L'indice rappresenta il rapporto percentuale tra le righe di commento e il totale delle righe di codice presenti in un modulo. Misura quanto il codice è documentato inline.
- **Legenda**:
  - **CM**: Numero di righe di commento (Comment Lines);
  - **CL**: Numero di righe di codice (Code Lines).

6.5.3.6. MPD12 - Coefficient of Coupling (CoC)

- **Formula**:

$$\mathrm{CoC} = \frac{\text{Numero di dipendenze}}{\text{Numero di componenti}}$$

- **Valore accettabile**: ≤ 0.4
- **Valore ottimale**: ≤ 0.2
- **Descrizione**: L'indice misura il grado di accoppiamento medio tra i moduli del sistema.
  Si calcola dividendo il numero totale di dipendenze (import, chiamate tra classi) per il numero totale di componenti (classi o moduli).
  Un valore basso indica un'architettura modulare e disaccoppiata (migliore manutenibilità), mentre

un valore superiore a 0.4 indica un accoppiamento eccessivo che rende difficile modificare una parte del sistema senza impattare le altre.

6.5.4. Usabilità

6.5.4.1. MPD13 - Tempo di apprendimento

- **Valore accettabile**: ≤ 10 minuti
- **Valore ottimale**: ≤ 5 minuti
- **Descrizione**: L'indice del tempo di apprendimento misura il tempo medio che un utente impiega per imparare l'utilizzo del programma in analisi, nel caso di questo progetto è *CodeGuardian*.

6.5.4.2. MPD14 - Indice di Gulpease

- **Formula**:

$$\text{Indice Gulpease} = 89 + \frac{300 \cdot \text{numero di frasi} - 10 \cdot \text{numero delle lettere}}{\text{numero di parole}}$$

- **Valore ≥80**: La complessità del testo è molto semplice e adatta a lettori che hanno completato la scuola primaria.
- **Valore tra 60 e 80**: La complessità del testo è di media difficoltà e adatta a lettori che hanno completato la scuola dell'obbligo.
- **Valore tra 40 e 60**: La complessità del testo è abbastanza complessa e adatta a lettori che hanno almeno un istruzione di livello superiore.
- **Valore <40**: La complessità del testo è complessa e adatta a lettori che hanno un livello di istruzione universitaria.
- **Valore accettabile**: ≥ 50
- **Valore ottimale**: ≥ 70
- **Descrizione**: L'Indice Gulpease è un indice di leggibilità del testo. Tale indice serve per classificare la difficoltà di lettura di un testo per un lettore medio. La formula tiene conto del numero di lettere, parole e frasi nel testo.

6.5.5. Affidabilità

6.5.5.1. MPD15 - Test Failure Rate

- **Formula**:

$$\text{TFR} = \frac{\text{Test falliti}}{\text{Test eseguiti}} \cdot 100$$

- **Valore accettabile**: $\leq 15\%$
- **Valore ottimale**: $\leq 0\%$
- **Descrizione**: L'indice di Error Rate indica la percentuale di errori durante l'esecuzione. Gli eventuali errori verranno riportati dai programmatori al fine di calcolare il valore della metrica.