Code7Crusaders

Software Development Team

Norme Di Progetto

Membri del Team:
Enrico Cotti Cottini, Gabriele Di Pietro, Tommaso Diviesti
Francesco Lapenna, Matthew Pan, Eddy Pinarello, Filippo Rizzolo

Versioni

Ver	Data	Redattore	Verificatore	descrizione
1.1	10/03/2025	Matthew Pan	Filippo Rizzolo	Aggiunta sezione UML classi
1.0	10/02/2025	Matthew Pan	Francesco Lapenna	Correzioni e approvazione documento
0.9	18/12/2024	Gabriele Di Pietro	Tommaso Diviesti	Stesura parte 6
0.8	17/12/2024	Matthew Pan	Gabriele Di Pietro	Fine stesura sezione 3
0.7	15/12/2024	Eddy Pinarello	Tommaso Diviesti	Fine stesura sezione 4
0.6	10/12/2024	Matthew Pan	Francesco Lapenna	Ultimata sezione 3
0.5	05/12/2024	Eddy Pinarello	Matthew Pan	Prima stesura sezione 4
0.4	25/11/2024	Eddy Pinarello	Mattehw Pan	Stesura sezione 3
0.3	21/11/2024	Matthew Pan	Eddy Pinarello	Completamento sezione 2
0.2	18/11/2024	Matthew Pan	Eddy Pinarello	Stesura sezione 2
0.1	12/11/2024	Matthew Pan	Enrico Cotti Cottini	Prima stesura delle sezioni 1 e 2

Indice

1 Introduzione **6**
- 1.1 Scopo del documento . . . 6
- 1.2 Scopo del progetto . . . 6
- 1.3 Glossario . . . 6
- 1.4 Riferimenti . . . 6
  - 1.4.1 Normativi . . . 6
  - 1.4.2 Informativi . . . 6

2 Processi primari **8**
- 2.1 Fornitura . . . 8
  - 2.1.1 Introduzione . . . 8
  - 2.1.2 Contatti con l'azienda proponente . . . 9
  - 2.1.3 Analisi dei Requisiti . . . 9
  - 2.1.4 Piano di Progetto . . . 9
  - 2.1.5 Piano di Qualifica . . . 10
  - 2.1.6 Glossario . . . 11
  - 2.1.7 Strumenti . . . 11
  - 2.1.8 Metriche . . . 11
- 2.2 Sviluppo . . . 11
  - 2.2.1 Introduzione . . . 11
  - 2.2.2 Analisi dei Requisiti . . . 12
    - 2.2.2.1 Descrizione . . . 12
    - 2.2.2.2 Scopo . . . 12
    - 2.2.2.3 Codifica dei casi d'uso . . . 13
    - 2.2.2.4 Diagrammi Casi D'uso . . . 13
    - 2.2.2.5 Requisiti . . . 17
    - 2.2.2.6 Fonti dei requisiti . . . 18
    - 2.2.2.7 Codifica dei requisiti . . . 18
    - 2.2.2.8 Metriche . . . 18
  - 2.2.3 Progettazione . . . 19
    - 2.2.3.1 Introduzione . . . 19
    - 2.2.3.2 Specifica Tecnica . . . 19
    - 2.2.3.3 Qualità dell'architettura . . . 20
  - 2.2.4 Diagrammi UML . . . 21
    - 2.2.4.1 Diagrammi delle Classi . . . 21
      - 2.2.4.1.1 Convenzioni sui metodi . . . 22
      - 2.2.4.1.2 Relazioni tra le classi . . . 22
  - 2.2.5 Desing Pattern . . . 24
  - 2.2.6 Test . . . 24
  - 2.2.7 Metriche . . . 24
  - 2.2.8 Codifica . . . 25
    - 2.2.8.1 Descrizione e Scopo . . . 25
    - 2.2.8.2 Aspettative . . . 25
    - 2.2.8.3 Norme di codifica . . . 25
    - 2.2.8.4 Metriche . . . 26
    - 2.2.8.5 Strumenti . . . 26

3 Processi di Supporto **27**
3.1 Documentazione . . . 27
3.1.1 Introduzione . . . 27
3.1.2 Ciclo di Vita del Documento . . . 27
3.1.3 Template . . . 27
3.1.4 Documenti Prodotti . . . 27
3.1.5 Struttura del Documento . . . 28
3.1.6 Verbali . . . 29
3.1.7 Nome del File . . . 29
3.1.8 Stile del Testo . . . 29
3.1.9 Glossario . . . 29
3.1.10 Tabelle . . . 29
3.1.11 Immagini . . . 29
3.1.12 Metriche . . . 29
3.2 Gestione della Configurazione . . . 30
3.2.1 Versionamento . . . 30
3.2.2 Repository . . . 30
3.3 Gestione della Qualità . . . 30
3.3.1 Descrizione . . . 30
3.3.2 Piano di Qualifica . . . 30
3.3.3 Metriche . . . 31
3.4 Verifica . . . 31
3.4.1 Introduzione . . . 31
3.4.2 Tipi di Verifica . . . 31
3.4.3 Metriche . . . 32
3.5 Validazione . . . 32
3.5.1 Introduzione . . . 32

4 Processi Organizzativi **33**
4.1 Gestione dei Processi . . . 33
4.1.1 Introduzione . . . 33
4.1.2 Pianificazione . . . 33
4.1.2.1 Descrizione . . . 33
4.1.2.2 Obiettivi . . . 33
4.1.3 Metriche . . . 34
4.1.4 Assegnazione dei Ruoli . . . 34
4.1.5 Ticketing . . . 35
4.1.6 Gestione dei rischi . . . 35
4.1.6.1 Struttura dei rischi . . . 35
4.1.7 Metriche . . . 35
4.2 Procedure Comunicative . . . 36
4.2.1 Comunicazioni Asincrone . . . 36
4.2.2 Comunicazioni Sincrone . . . 36
4.2.3 Riunioni Interne . . . 36
4.2.4 Riunioni Esterne . . . 36
4.3 Formazione . . . 37
4.3.1 Introduzione . . . 37
4.3.2 Metodo di formazione . . . 37

5 Standard per la qualità **37**
5.1 Caratteristiche del sistema ISO/IEC 25010:2023 . . . 37
5.1.1 Appropriatezza funzionale . . . 37
5.1.2 Performance . . . 37
5.1.3 Compatibilità . . . 38
5.1.4 Usabilità . . . 38
5.1.5 Affidabilità . . . 38
5.1.6 Sicurezza . . . 38
5.1.7 Manutenibilità . . . 38
5.1.8 Portabilità . . . 39
5.2 Suddivisione secondo standard ISO/IEC 12207:1995 . . . 39
5.2.1 Processi primari . . . 39
5.2.2 Processi di supporto . . . 39
5.2.3 Processi organizzativi . . . 40

6 Metriche di qualità **41**
6.1 Processi di base e/o primari . . . 41
6.1.1 Fornitura . . . 41
6.1.2 Sviluppo . . . 45
6.2 Processi di Supporto . . . 48
6.2.1 Documentazione . . . 48
6.2.2 Gestione della Qualità . . . 49
6.2.3 Verifica . . . 50
6.2.4 Risoluzione dei problemi . . . 52
6.3 Processi organizzativi . . . 53
6.3.1 Pianificazione . . . 53

Elenco delle tabelle

1 Metriche relative al processo di fornitura. . . . 11
2 Percentuali requisiti . . . 18
3 Metriche inerenti la Progettazione . . . 24
4 Metriche di Codifica . . . 26
5 Metriche relative al processo di Documentazione. . . . 29
6 Metriche relative al processo di Gestione della Qualità. . . . 31
7 Metriche relative al processo di Verifica. . . . 32
8 Metriche relative alla Pianificazione. . . . 34
9 Metriche relative ai Rischi. . . . 35

Elenco delle figure

1 Esempio di attore . . . 13
2 Esempio di sistema . . . 14
3 Esempio di caso d'uso . . . 14
4 Esempio di sottocaso d'uso . . . 15
5 Esempio di associazione . . . 15
6 Esempio di Generalizzaione casi d'uso . . . 16
7 Esempio di inclusione . . . 16
8 Esempio di estensione . . . 17

9 Esempio di Generalizzaione attore 17
10 Esempio di dipendenza 22
11 Esempio di associazione 23
12 Esempio di aggregazione 23
13 Esempio di composizione 23
14 Esempio di generalizzazione 23
15 Esempio di realizzazione 24

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di definire le regole e le procedure che ogni membro del team deve seguire durante lo sviluppo del progetto. In particolare, mira a stabilire il *Way of Working* del gruppo. La sua redazione inizia nelle prime fasi del progetto e continua anche durante le fasi successive, per essere costantemente aggiornato e adattato alle esigenze del team. Il processo seguirà le linee guida dello standard ISO/IEC 12207:1995, suddivise in:

- Processi primari
- Processi di supporto
- Processi organizzativi

1.2 Scopo del progetto

Il progetto si propone di sviluppare un Assistente Virtuale intelligente per aziende che operano nel settore della vendita multiprodotto. Questo assistente avrà il compito di semplificare l'accesso alle informazioni sui prodotti disponibili, rispondendo alle domande più frequenti poste dai clienti in modo rapido ed efficace. Grazie all'uso di tecnologie avanzate come il Machine Learning[G] e il Natural Language Processing[G], il sistema sarà in grado di analizzare i dati contenuti nei cataloghi aziendali e negli archivi digitali, fornendo risposte precise e personalizzate. L'obiettivo principale è ridurre la dipendenza dagli specialisti aziendali, che attualmente rappresentano l'unico canale di accesso per ottenere dettagli approfonditi sui prodotti. Questo migliorerà l'efficienza operativa, ottimizzerà le risorse e offrirà una migliore esperienza ai clienti, che potranno interagire con il sistema in modo intuitivo e diretto attraverso piattaforme digitali come siti web o chatbot. In sintesi, il progetto intende rendere l'accesso alle informazioni aziendali più semplice, veloce e scalabile, migliorando al contempo la qualità del servizio offerto ai clienti.

1.3 Glossario

Per evitare ambiguità e facilitare la comprensione del documento, si farà uso di un glossario, contenente la definizione dei termini tecnici e degli acronimi utilizzati, che sarà incluso all'interno del file *glossario*.

1.4 Riferimenti

1.4.1 Normativi

- **Capitolato C7:**
  `https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C7.pdf`
- **ISO/IEC 12207:1995**
  `https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf`

1.4.2 Informativi

- **Glossario RTB**
  `https://code7crusaders.github.io/docs/RTB/documentazione_interna/glossario.html`
- **Documentazione Git**
  `https://git-scm.com/docs`

- **Documentazione Latex**
  `https://www.latex-project.org/help/documentation/`

2 Processi primari

2.1 Fornitura

2.1.1 Introduzione

Il processo di fornitura rappresenta un percorso ben definito che stabilisce un contratto tra fornitore e cliente, accompagnando la creazione e la consegna del software. Fondamentale per garantire che il software risponda ai requisiti del cliente, rispetti i tempi e i costi, e soddisfi gli standard di qualità, il processo include anche un continuo dialogo tra le parti per chiarire le necessità, risolvere eventuali difficoltà tecniche e stabilire le basi per il corretto sviluppo del prodotto, attraverso un'accurata definizione dei requisiti e dei vincoli tecnologici. Il processo di fornitura si articola nelle seguenti fasi principali:

1. **Preparazione della proposta** Questa fase iniziale si concentra sulla raccolta delle informazioni necessarie e sulla stesura di una proposta formale per il cliente. Include:
   - Analisi delle esigenze del cliente.
   - Studio di fattibilità.
   - Elaborazione della proposta di candidatura.
2. **Pianificazione** Qui si stabilisce l'organizzazione e la programmazione delle attività del progetto, con particolare attenzione a:
   - Definizione delle milestone.
   - Creazione del piano di progetto.
   - Assegnazione di compiti e risorse.
3. **Esecuzione** Durante questa fase si procede con la realizzazione pratica del progetto, che comprende:
   - Sviluppo del software.
   - Test e verifiche.
   - Preparazione della documentazione.
4. **Revisione** Questa fase consiste nel valutare approfonditamente il lavoro svolto per verificarne la conformità agli standard di qualità e ai requisiti contrattuali. Le attività principali sono:
   - Revisione del codice.
   - Esecuzione dei test di accettazione.
   - Risoluzione di eventuali discrepanze.
5. **Consegna** Infine, il prodotto finale viene consegnato al cliente. Questa fase comprende:
   - Consegna del software.
   - Formazione del personale.

2.1.2 Contatti con l'azienda proponente

Code7Crusaders dispone di un indirizzo email(code7crusaders@gmail.com) e un canale Discord per le riunioni telematiche. Gli incontri online si svolgeranno settimanalmente, con la possibilità di pianificare riunioni aggiuntive su richiesta del team. Ad ogni incontro settimanale verrà redatto un verbale che riporterà gli argomenti discussi e le scelte intraprese. Per ogni meeting con l'azienda proponente sarà preparato un verbale che riepilogherà i punti principali discussi. Tutti i verbali interni e esterni per discussioni durante lo svolgimento dell'RTB[G] saranno accessibili al seguente link: `https://code7crusaders.github.io/docs/RTB/index.html`. Inoltre per una comunicazione più rapida e informale, il team e l'azienda utilizzeranno Telegram.

2.1.3 Analisi dei Requisiti

L'**Analisi dei Requisiti[G]** v1.0, redatto dagli Analisti, rappresenta un documento fondamentale per lo sviluppo del sistema software. Il suo obiettivo principale è definire in dettaglio le funzionalità necessarie affinché il prodotto soddisfi pienamente le richieste della Proponente. Il documento comprende i seguenti elementi essenziali:

- **Definizione degli attori**: entità o persone che interagiscono con il sistema.
- **Definizione dei casi d'uso**: rappresentazione narrativa di scenari specifici che descrivono come gli attori interagiscono con il sistema. I casi d'uso offrono una visione chiara delle azioni eseguibili all'interno del sistema e delle interazioni degli utenti con esso. All'interno di ciascun caso d'uso, viene fornito:
  - un elenco preciso delle azioni intraprese dall'attore per attivare il caso d'uso;
  - una base per facilitare l'estrazione dei requisiti corrispondenti.
- **Definizione di requisiti**: individuazione e categorizzazione dei requisiti in:
  - **Requisiti funzionali**: specificano le operazioni che il sistema deve essere in grado di eseguire;
  - **Requisiti di qualità**: si concentrano sulla definizione degli standard e degli attributi che il software deve possedere per garantire prestazioni, affidabilità, usabilità e sicurezza ottimali;
  - **Requisiti di vincolo**: delineano vincoli e limitazioni che il sistema deve rispettare, includendo restrizioni tecnologiche, normative o di risorse.

2.1.4 Piano di Progetto

Il **Piano di Progetto[G]** v1.0, redatto dal Responsabile, descrive in dettaglio il processo di sviluppo del progetto. Esso funge da guida fondamentale per il team al fine di:

- mantenere l'allineamento con gli obiettivi;
- gestire le risorse in modo efficace;
- affrontare e mitigare eventuali criticità durante le varie fasi.

Si articola nelle seguenti sezioni:

- **Analisi dei rischi**: identifica, valuta e gestisce i rischi potenziali che possono influenzare il successo del progetto. I rischi sono classificati in:
  - tecnologici;
  - comunicativi;

  - individuali;
  - organizzativi.

  Per ciascun rischio vengono definiti segnali di manifestazione, probabilità, impatto e strategie di mitigazione.

- **Modello di sviluppo**: descrive l'approccio metodologico scelto, nel nostro caso il framework *agile Scrum*. Include:
  - gli eventi principali del framework;
  - le pratiche adottate dal team.
- **Pianificazione**: include una roadmap dettagliata che descrive:
  - le attività necessarie per raggiungere gli obiettivi di ogni sprint;
  - la distribuzione temporale delle risorse.
- **Preventivo**: fornisce una stima delle ore produttive disponibili, distribuite tra:
  - i ruoli assegnati ai membri del team;
  - ogni sprint pianificato.

  Inoltre, include il costo stimato di ogni sprint.
- **Consuntivo**: analizza a posteriori la ripartizione effettiva delle ore e dei costi. Contiene:
  - una retrospettiva sulle discrepanze rispetto al preventivo;
  - eventuali miglioramenti nella pianificazione futura.

2.1.5 Piano di Qualifica

Il **Piano di Qualifica**[G] v1.0, redatto dall'Amministratore, descrive le strategie e gli approcci adottati per garantire la qualità del prodotto o servizio sviluppato. Si compone delle seguenti sezioni:

- **Qualità di processo**: specifica gli standard e le procedure seguite per garantire la qualità dei processi di sviluppo. Include:
  - metodologie utilizzate;
  - criteri per la misurazione e il miglioramento dei processi.
- **Qualità di prodotto**: descrive gli standard e le specifiche che il prodotto deve soddisfare per essere considerato di qualità. Include:
  - metriche e criteri di valutazione;
  - specifiche tecniche richieste.
- **Specifiche dei test**: fornisce una descrizione dettagliata dei test pianificati durante lo sviluppo per verificare che i requisiti siano soddisfatti.
- **Cruscotto delle metriche**: presenta un resoconto delle attività di valutazione svolte, utile per:
  - monitorare l'andamento del progetto rispetto agli obiettivi prefissati;
  - identificare azioni correttive necessarie.

2.1.6 Glossario

Il Glossario[G] rappresenta un riferimento completo che raccoglie e definisce i termini tecnici utilizzati nel progetto. Questo documento garantisce una comprensione uniforme della terminologia specifica del settore, riducendo il rischio di equivoci e favorendo una comunicazione chiara. Inoltre, contribuisce a migliorare la coerenza e la qualità della documentazione prodotta dal team.

2.1.7 Strumenti

Di seguito sono elencati gli strumenti software utilizzati nel processo di fornitura:

- **Discord**: piattaforma utilizzata per le riunioni interne.
- **Google Meet**: utilizzato per le riunioni formali online con l'azienda proponente.
- **Telegram**: piattaforma utilizzata come metodo informale per comunicare con l'azienda proponente.
- **LaTeX**: sistema per la creazione di documenti e slide di presentazione.
- **GitHubProject**: sistema di ticketing e roadmap integrato nella piattaforma di GitHub.

2.1.8 Metriche

Metrica	**Nome**
1PBM-PV	Planned Value
2PBM-ETC	Estimated to Complete
3PBM-EAC	Estimate at Completion
4PBM-EV	Earned Value
5PBM-AC	Actual Cost
6PBM-SV	Scheduled Variance
7PBM-CV	Cost Variance
8PBM-CPI	Cost Performance Index
9PBM-SPI	Scheduled Performance Index
10PBM-OTDR	On-Time Delivery Rate

Metriche relative al processo di fornitura.

2.2 Sviluppo

2.2.1 Introduzione

Il processo di sviluppo ha l'obiettivo fondamentale di identificare e pianificare con precisione i compiti e le attività che il team deve eseguire per realizzare il prodotto software richiesto. Questo processo non si limita alla semplice suddivisione delle mansioni, ma prevede anche l'assegnazione di ruoli specifici a ciascun membro del team, in modo da valorizzare al meglio le competenze individuali e garantire un flusso di lavoro armonioso e produttivo. Per assicurare che il software sviluppato soddisfi pienamente le aspettative e le necessità del committente, il gruppo **Code7Crusaders** definisce in maniera dettagliata gli obiettivi di sviluppo e design. Questo processo prevede la redazione di linee guida chiare e la realizzazione di un piano di lavoro che consenta di monitorare costantemente l'avanzamento delle attività e di

apportare eventuali correzioni. In particolare, il prodotto finale deve soddisfare le richieste del committente, come descritto nell'Analisi dei Requisiti[G]. Gli obiettivi di sviluppo definiti dal team devono essere rispettati, e il software deve superare con successo tutte le fasi di verifica e validazione[G]. Il processo include la pianificazione accurata delle attività, la definizione delle specifiche di design e lo sviluppo delle funzionalità richieste. Una volta implementato il prodotto, viene avviata una fase di test che garantisce la qualità complessiva e l'aderenza ai requisiti stabiliti. Parallelamente, viene prodotta un'adeguata documentazione che facilita la tracciabilità e il mantenimento futuro del software, contribuendo a preservare il valore del progetto nel tempo.

2.2.2 Analisi dei Requisiti

2.2.2.1 Descrizione

L'**Analisi dei Requisiti[G]** v1.0 è un documento redatto dagli Analisti che comprende i seguenti aspetti:

- **Introduzione**: descrive l'obiettivo del documento, il fine del prodotto e i riferimenti utilizzati per la sua stesura;
- **Descrizione del prodotto**: illustra le funzionalità attese del prodotto e le caratteristiche principali degli utenti;
- **Attori**: definisce i soggetti che utilizzeranno il sistema finale;
- **Casi d'uso**: identifica gli attori e descrive tutte le possibili interazioni con il sistema;
- **Requisiti**: raccoglie le caratteristiche essenziali da soddisfare e le fonti da cui queste sono state derivate.

2.2.2.2 Scopo

Lo scopo principale dell'**Analisi dei Requisiti[G]** v1.0 è quello di specificare in modo completo e preciso le funzionalità e le caratteristiche che il prodotto software deve offrire. Tale analisi consente di comprendere appieno:

- le necessità degli utenti;
- gli obiettivi principali del sistema;
- il contesto operativo in cui il sistema sarà utilizzato.

Gli obiettivi fondamentali di questa attività includono:

- Individuare e chiarire le finalità e le aspettative legate al prodotto da sviluppare;
- Fornire ai Progettisti una base dettagliata per definire l'architettura e il design del sistema;
- Offrire un supporto per la pianificazione del progetto utilizzando i requisiti identificati;
- Facilitare lo scambio di informazioni tra il team di sviluppo e la Proponente;
- Servire come riferimento per la fase di verifica del sistema.

2.2.2.3 Codifica dei casi d'uso

I casi d'uso sono codificati utilizzando la seguente notazione:

- **UC[ID-Principale][ID-Sottocaso]**: Identificativo univoco del caso d'uso, composto da un ID principale che identifica il caso principale e, se necessario, da un ID del sottocaso.
- **Titolo**: Breve descrizione del caso d'uso.
- **Attori**: Elenco degli attori coinvolti nel caso d'uso.
- **Precondizioni**: Condizioni che devono essere vere prima che il caso d'uso possa iniziare.
- **Postcondizioni**: Condizioni che devono essere vere dopo che il caso d'uso è stato completato con successo.
- **Scenario principale**: Descrizione dettagliata del flusso di eventi principale del caso d'uso.
- **Generalizzaioni**: Eventuali casi d'uso generalizzati.
- **Estensioni**: Eventuali casi d'uso estesi.

2.2.2.4 Diagrammi Casi D'uso

I diagrammi dei casi d'uso rappresentano visivamente le interazioni tra attori e sistema, illustrando i vari scenari di utilizzo. Ogni caso d'uso descrive una sequenza di azioni necessarie per raggiungere un obiettivo specifico, aiutando a identificare i requisiti funzionali e a chiarire le aspettative degli utenti. Questi diagrammi facilitano la comunicazione tra sviluppatori e stakeholder, garantendo che tutte le funzionalità richieste siano considerate e implementate correttamente. Di seguito sono elencati i principali componenti di un diagramma dei casi d'uso.

- **Attori**: I soggetti che interagiscono con il sistema, rappresentati come uomini stilizzati possono essere persone, altri applicativi o dispositivi che utilizzano le funzionalità del sistema. 1

Figura 1: Esempio di attore

- **Sistema**: Indica il contesto del sistema software, indicando funzionalità interne al contesto definito. 2

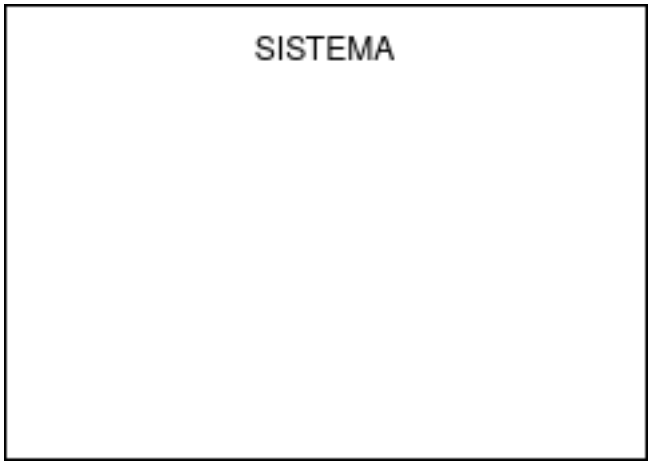

Figura 2: Esempio di sistema

- **Casi d'uso**: funzionalità offerte dal sistema che soddisfano le necessità di un Attore. Ogni caso d'uso descrive una sequenza specifica di interazioni tra gli attori e il sistema. 3

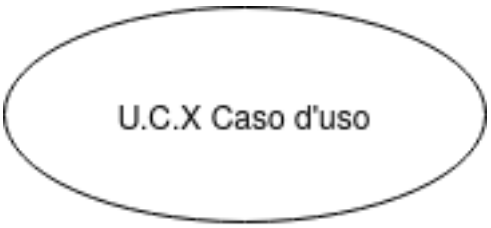

Figura 3: Esempio di caso d'uso

- **Sottocasi d'uso**: Scenari specifici che si verificano all'interno del caso d'uso principale. 4

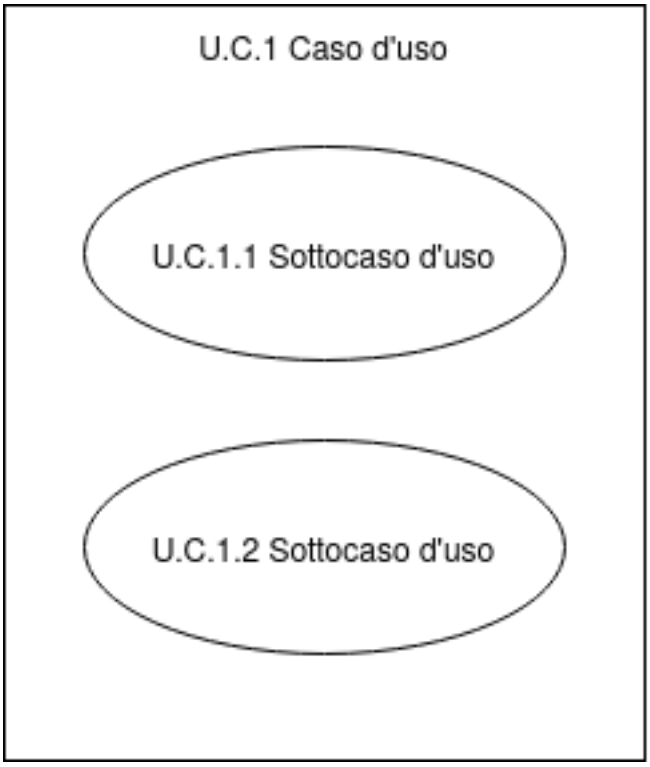

Figura 4: Esempio di sottocaso d'uso

- **Relazioni tra Attori e Casi d'Uso**:
  - **Associazione**: Collegamento tra un attore e un caso d'uso, indicando che l'attore è coinvolto nel caso d'uso. 5

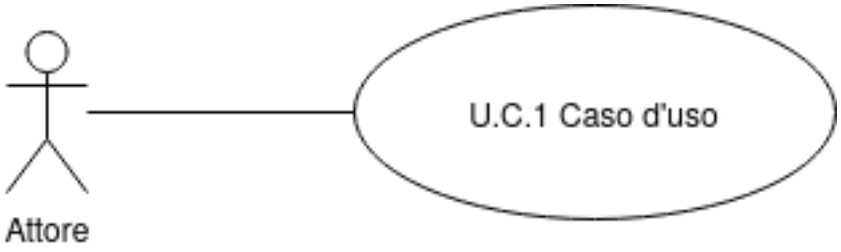

Figura 5: Esempio di associazione

- **Relazioni tra Attori**:
  - **Generalizzazione**: Un attore eredita le funzionalità di un altro attore. Una relazione padre figlio dove il figlio eredita almeno una funzionalità del padre. Utilizzata nel caso in cui due attori condividano funzionalità. 6

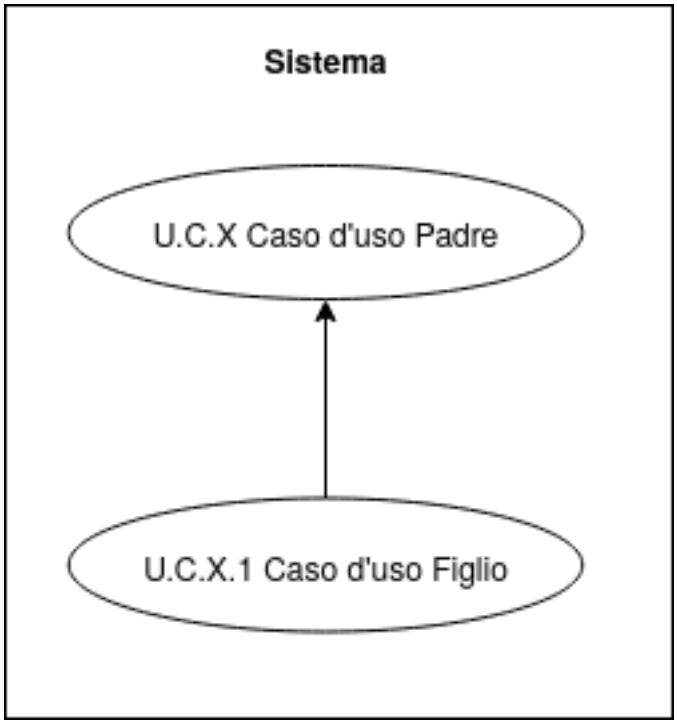

Figura 6: Esempio di Generalizzaione casi d'uso

- **Relazioni tra Casi d'Uso**:
  - **Inclusione**: Indica che un caso d'uso include un altro caso d'uso. Questo significa che durante l'esecuzione di un caso d'uso si eseguono anche i casi d'uso inclusi. Questo per evitare la ripetizione di funzionalità uguali in più casi d'uso. 7

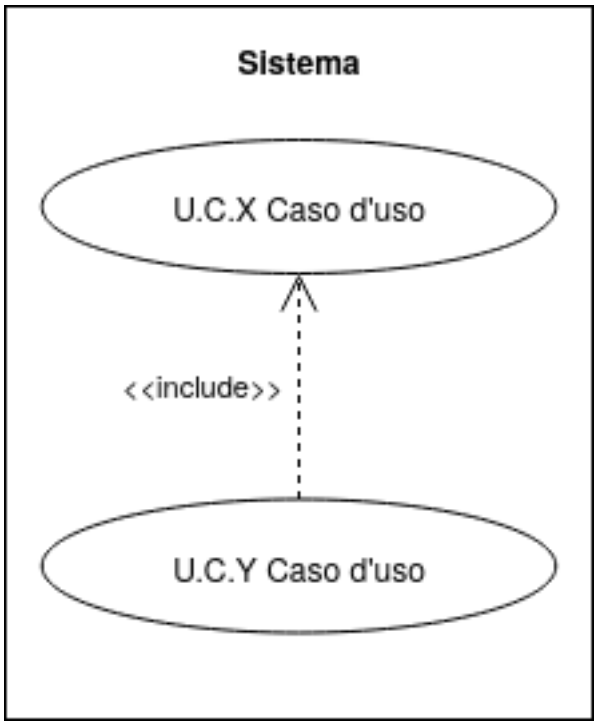

Figura 7: Esempio di inclusione

– **Estensione**: Un caso d'uso esteso aggiunge funzionalità al caso d'uso principale, ma viene attivato solo in specifiche circostanze. Quando ciò accade, il flusso del caso d'uso principale si interrompe temporaneamente per consentire l'esecuzione del caso d'uso esteso. 8

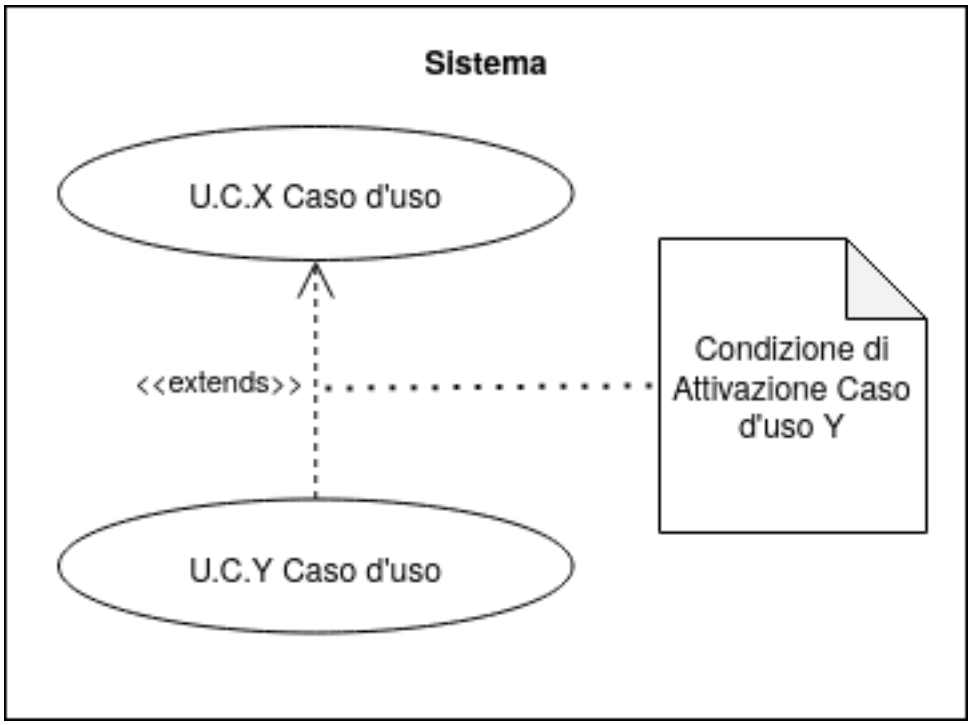

Figura 8: Esempio di estensione

– **Generalizzazione**: Un caso d'uso eredita le funzionalità di un altro caso d'uso. Una relazione padre figlio dove il figlio eredita almeno una funzionalità del padre. Utilizzata nel caso in cui due casi d'uso condividano funzionalità. 9

Figura 9: Esempio di Generalizzaione attore

2.2.2.5 Requisiti

I requisiti sono classificati in tre categorie principali:

- **Funzionali**: riguardano l'usabilità del prodotto finale;
- **Di qualità**: includono gli strumenti e la documentazione da fornire;
- **Di vincolo**: fanno riferimento alle tecnologie da utilizzare.

Ciascun requisito è indicato da:

- **Codice Identificativo**: codice univoco che identifica il requisito;

- **Descrizione**: breve spiegazione del requisito;
- **Fonte**: origine del requisito (es. capitolato, interno, ecc..);
- **Priorità**: importanza del requisito rispetto agli altri;

2.2.2.6 Fonti dei requisiti

I requisiti sono stati identificati a partire dalle seguenti fonti:

- **Capitolato**: requisiti individuati tramite analisi del capitolato;
- **interno**: requisiti individuati durante riunioni interne al gruppo di lavoro;
- **Esterno**: requisiti individuati durante incontri con il proponente;
- **Piano di Qualifica**[G]: Requisiti necessari per rispettare standard di qualità definiti nel documento Piano di Qualifica[G];
- **Norme di Progetto**[G]: Requisiti necessari per rispettare le norme di progetto definite nel documento Norme di Progetto[G];

2.2.2.7 Codifica dei requisiti

I requisiti sono codificati come segue: **R[Tipo][Importanza][Numero]**
Dove **Tipo** può essere:

- **F (funzionale)**
- **Q (di qualità)**
- **V (di vincolo)**

Importanza può essere:

- **O (obbligatorio)**
- **D (desiderabile)**
- **F (facoltativo)**

Numero è un numero identificativo univoco del requisito.

2.2.2.8 Metriche

Metrica	Nome
11PBM-PRO	Percentuale Requisiti Obbligatori
12PBM-PRD	Percentuale Requisiti Desiderabili
13PBM-PRF	Percentuale Requisiti Facoltativi

Percentuali requisiti

2.2.3 Progettazione

2.2.3.1 Introduzione

La fase di progettazione riveste un ruolo cruciale nel definire la struttura principale del progetto, basandosi sui requisiti individuati durante l'analisi e descritti nell'Analisi dei Requisiti^{G}. Questa attività è affidata ai progettisti, i quali elaborano un piano dettagliato per implementare tutti i requisiti specificati. Per questa fase del ciclo di vita del software, il nostro gruppo si pone i seguenti obiettivi:

- Trasformare i requisiti in specifiche tecniche dettagliate che coprano tutti gli aspetti del sistema.
- Garantire una struttura facilmente comprensibile per agevolare la manutenzione futura.
- Ottenere l'approvazione per il passaggio alla fase di sviluppo.

Il processo di progettazione si articola in tre livelli principali:

- **Design dell'interfaccia**: questa fase si concentra su un livello di astrazione elevato rispetto al funzionamento interno del sistema. Durante la progettazione dell'interfaccia, l'attenzione è rivolta alle tecnologie da utilizzare nella fase di sviluppo del software, portando alla creazione di un Proof of Concept^{GG}.
- **Progettazione architetturale**: si definisce la struttura generale del sistema a un alto livello, senza entrare nei dettagli interni dei componenti principali. In questa fase vengono anche definiti i test di integrazione.
- **Progettazione dettagliata**: si specificano gli elementi interni di ciascun componente principale, incluse le specifiche architetturali del prodotto. Si producono inoltre i diagrammi delle classi e si definiscono i test di unità per ogni componente. Questa fase culmina nella creazione della Product Baseline^{GG}.

2.2.3.2 Specifica Tecnica

La specifica tecnica è un documento che funge da fondamento per l'intero processo di sviluppo, guidando il team nelle scelte architetturali e tecnologiche, garantendo un approccio metodico e strutturato.

Elementi Chiave della Specifica Tecnica

- **Architettura del sistema**: questa sezione definisce la struttura complessiva del sistema software, identificando i componenti principali, i moduli e le relative interfacce. Include inoltre dettagli sull'organizzazione logica e fisica del sistema, come la suddivisione in livelli o strati e le relazioni tra le diverse parti.
- **Tecnologie adottate**: vengono specificate le tecnologie scelte per lo sviluppo, inclusi linguaggi di programmazione, framework, librerie e tool di supporto. Questa sezione comprende anche l'ambiente di sviluppo e gli strumenti di gestione del progetto.
- **Struttura dei dati**: si descrive l'organizzazione e la gestione dei dati all'interno del sistema, comprendendo database, file system, protocolli di accesso e tecniche di persistenza adottate.
- **Interfacce esterne**: vengono identificate e descritte le interfacce con altri sistemi o applicazioni esterne. Questo include i formati dei dati scambiati e i protocolli di comunicazione utilizzati per garantire un'integrazione efficace.
- **Design pattern**: si presentano i design pattern adottati durante lo sviluppo, fornendo soluzioni consolidate a problematiche comuni e ricorrenti.

- **Pianificazione e risorse**: questa sezione fornisce una pianificazione dettagliata delle attività necessarie per implementare la specifica tecnica. Vengono inclusi stime di tempi, costi e risorse richieste, sia umane che tecnologiche.
- **Procedure di testing e validazione**: vengono fornite indicazioni sulle procedure e sugli strumenti necessari per testare e validare il sistema, assicurandosi che soddisfi i requisiti funzionali e le aspettative del cliente.
- **Requisiti tecnici**: viene riportato un elenco completo e dettagliato dei requisiti tecnici che il sistema deve soddisfare, inclusi parametri relativi a funzionalità, prestazioni e caratteristiche richieste.

2.2.3.3 Qualità dell'architettura

La qualità dell'architettura è un aspetto fondamentale per garantire il successo di un sistema software. Essa si misura attraverso una serie di caratteristiche chiave che ne determinano l'efficacia, la robustezza e l'adattabilità. Di seguito vengono elencati i principali attributi:

- **Scalabilità**[G]: rappresenta la capacità del sistema di adattarsi a un incremento del carico di lavoro o delle risorse senza compromettere le prestazioni o la qualità del servizio. Un'architettura scalabile è in grado di rispondere dinamicamente alle variazioni delle richieste degli utenti e alle risorse disponibili.
- **Flessibilità**: indica l'abilità del sistema di adattarsi a modifiche nei requisiti o nell'ambiente operativo senza richiedere interventi significativi. Una buona flessibilità si ottiene tramite componenti ben separati e interfacce standardizzate, che agevolano l'aggiunta di nuove funzionalità o modifiche esistenti.
- **Manutenibilità**: definisce la facilità con cui il software può essere compreso, modificato e corretto durante il suo ciclo di vita. Un'architettura manutenibile è caratterizzata da un codice modulare, ben documentato e organizzato, facilitando l'identificazione e la risoluzione dei problemi.
- **Testabilità**: riguarda la facilità di verificare che il sistema soddisfi i requisiti funzionali e non funzionali. Un'architettura testabile prevede componenti isolati e interfacce ben definite, che consentono l'automazione dei test e garantiscono una verifica continua della qualità.
- **Affidabilità**: assicura il funzionamento corretto del sistema sia in condizioni normali che in situazioni anomale. Un'architettura affidabile integra meccanismi di gestione degli errori, recupero e ripristino, che permettono al sistema di continuare a operare anche in caso di guasti o interruzioni.
- **Performance**: misura la capacità del sistema di fornire risposte rapide e tempi di elaborazione ottimizzati anche sotto carichi di lavoro elevati. Le architetture performanti includono ottimizzazioni del codice, una gestione efficiente delle risorse e accessi ai dati ottimizzati.
- **Usabilità**: si riferisce alla facilità di utilizzo e comprensione del sistema da parte dell'utente finale. Un'architettura usabile prevede interfacce intuitive, una navigazione chiara e una presentazione delle informazioni coerente, garantendo così un'esperienza utente positiva.
- **Compatibilità**: indica la capacità del sistema di interagire senza difficoltà con altre piattaforme, sistemi o tecnologie. Una buona compatibilità si ottiene grazie all'adozione di standard aperti, interfacce ben definite e protocolli di comunicazione standardizzati.
- **Documentazione**: comprende la fornitura di documenti completi e dettagliati relativi all'architettura del sistema. Tra questi vi sono diagrammi, specifiche tecniche, manuali utente e guide

per gli sviluppatori. Una documentazione accurata facilita la comprensione, la manutenzione e l'evoluzione del sistema.

- **Safety**: rappresenta la capacità del sistema di garantire la sicurezza dei dati e delle informazioni sensibili, proteggendoli anche in caso di malfunzionamenti o situazioni impreviste.

2.2.4 Diagrammi UML

L'utilizzo dei diagrammi UML (Unified Modeling Language) nella progettazione del sistema software offre numerosi vantaggi. Di seguito sono elencati i principali aspetti positivi:

- **Chiarezza visiva**: i diagrammi UML forniscono una rappresentazione grafica chiara e intuitiva delle relazioni e delle interazioni tra i componenti del sistema software. Questo facilita la comprensione del sistema da parte degli stakeholder, sia tecnici che non tecnici.
- **Standardizzazione**: UML rappresenta uno standard internazionale ampiamente riconosciuto per la modellazione dei sistemi software. Grazie a questa standardizzazione, i diagrammi UML risultano facilmente comprensibili e interpretabili dai professionisti del settore a livello globale.
- **Comunicazione efficace**: i diagrammi UML forniscono un linguaggio visivo comune che facilita la comunicazione tra sviluppatori, stakeholder tecnici e non tecnici e altri membri del team di progetto. Questo consente una condivisione chiara delle informazioni e una comunicazione più efficace.
- **Supporto all'analisi e alla progettazione**: durante le fasi di analisi e progettazione del ciclo di sviluppo del software, i diagrammi UML possono essere utilizzati per visualizzare i requisiti, analizzare le relazioni tra i componenti e descrivere le interazioni all'interno del sistema.
- **Modellazione dei requisiti**: UML consente di modellare i requisiti funzionali e non funzionali in modo chiaro e strutturato. I diagrammi, come quelli dei casi d'uso e delle sequenze, permettono di identificare scenari, vincoli e comportamenti del sistema che guidano lo sviluppo.
- **Facilità di manutenzione**: grazie alla loro chiarezza, i diagrammi UML semplificano la comprensione dell'architettura del sistema, rendendo più agevole l'identificazione di problemi o aree di miglioramento. Questo facilita le attività di manutenzione ed evoluzione del software nel tempo.
- **Supporto ai principi di progettazione**: UML è un valido strumento per applicare e comunicare principi di progettazione consolidati, come l'incapsulamento, l'ereditarietà e il polimorfismo, che contribuiscono alla creazione di un'architettura software robusta e modulare.
- **Documentazione tecnica**: i diagrammi UML possono essere utilizzati per generare una documentazione tecnica dettagliata del sistema software. Questa documentazione rappresenta una guida completa sia per gli sviluppatori che per gli utenti finali e gli stakeholder coinvolti.

2.2.4.1 Diagrammi delle Classi

Un tipo di diagramma UML molto diffuso nella progettazione software è il **diagramma delle classi**, che rappresenta la struttura statica di un sistema evidenziando classi, attributi, metodi e le connessioni tra di esse.

Le classi vengono raffigurate tramite rettangoli suddivisi in tre sezioni:

1. **Nome della classe**: indica il nome della classe.
2. **Attributi**: elenco delle proprietà della classe con il rispettivo tipo di dato, seguendo il formato:

visibilità nome: tipo [molteplicità] = valore predefinito

Dove:

- **Visibilità**: livello di accesso agli attributi, con le seguenti convenzioni:
  - "+" = pubblico
  - "-" = privato
  - "#" = protetto
  - " " = package
- **Nome**: identificativo dell'attributo, chiaro e rappresentativo. Se l'attributo è una costante, il nome deve essere scritto in maiuscolo (es. `PIGRECO: double`).
- **Molteplicità**: usata per sequenze di elementi come array o liste, indica la quantità di elementi presenti. Se sconosciuta, si utilizza `*` (es. `tipoAttributo[*]`).
- **Valore predefinito**: valore assegnato di default all'attributo.

3. **Metodi**: definiscono il comportamento della classe, seguendo il formato:

   visibilità nome(parametri): tipo di ritorno

   Dove:

   - **Visibilità**: stesso criterio degli attributi.
   - **Nome**: rappresentativo e chiaro, segue la notazione `nomeMetodo(parametri): tipoRitorno`.
   - **Parametri**: elenco di parametri separati da virgola, con il formato `nomeParametro: tipo`.
   - **Tipo di ritorno**: specifica il valore restituito dal metodo.

2.2.4.1.1 Convenzioni sui metodi

- I metodi getter, setter e i costruttori sono generalmente omessi.
- I metodi statici sono sottolineati.
- I metodi astratti sono scritti in corsivo.
- Se una classe non possiede attributi o metodi, le relative sezioni nel diagramma vengono omesse.

2.2.4.1.2 Relazioni tra le classi

Le relazioni tra classi sono fondamentali per descrivere le interazioni e le dipendenze tra le diverse componenti del sistema, fornendo una visione chiara della sua architettura. Le principali relazioni sono:

- **Dipendenza**: mostra che una classe utilizza un'altra senza legami permanenti tra le loro istanze. Si indica con una linea tratteggiata con una freccia verso la classe dipendente.

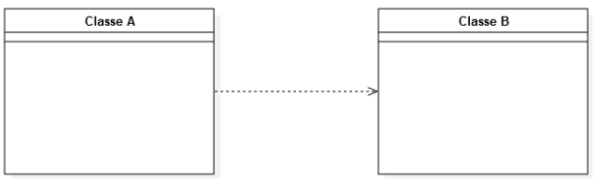

Figura 10: Esempio di dipendenza

- **Associazione**: lega due classi, indicando che un'istanza di una è correlata a un'istanza dell'altra. È rappresentata da una linea continua tra le classi con eventuali etichette che indicano nome, molteplicità e ruoli.

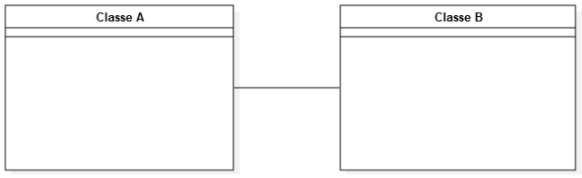

Figura 11: Esempio di associazione

- **Aggregazione**: rappresenta una relazione "tutto-parte", dove una classe è composta da una o più istanze di un'altra, ma queste possono esistere indipendentemente. Si rappresenta con una linea e un rombo vuoto sulla parte aggregata.

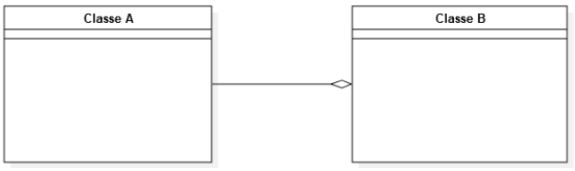

Figura 12: Esempio di aggregazione

- **Composizione**: simile all'aggregazione, ma più forte, indicando che le parti non possono esistere senza l'elemento principale. Rappresentata con una linea e un rombo pieno sulla parte composta.

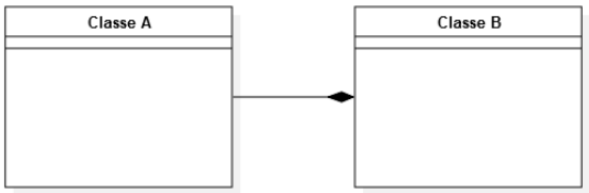

Figura 13: Esempio di composizione

- **Ereditarietà (Generalizzazione)**: descrive una relazione gerarchica tra una superclasse e una o più sottoclassi, che ne ereditano gli attributi e metodi. È raffigurata con una linea e una freccia vuota che punta dalla sottoclasse alla superclasse.

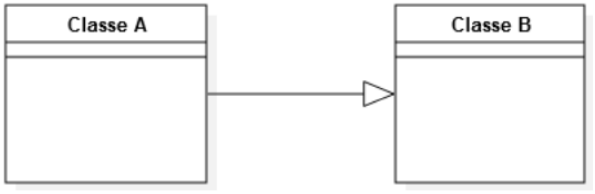

Figura 14: Esempio di generalizzazione

- **Realizzazione**: rappresenta l'implementazione di un'interfaccia da parte di una classe concreta. È indicata con una linea tratteggiata con una freccia vuota dalla classe implementatrice all'interfaccia.

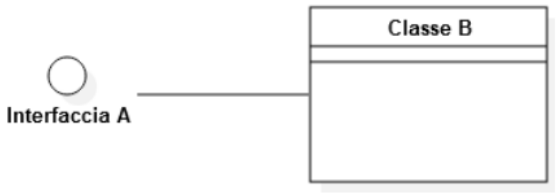

Figura 15: Esempio di realizzazione

2.2.5 Desing Pattern

I *design pattern* rappresentano soluzioni progettuali generiche e riutilizzabili per risolvere problemi comuni che emergono durante lo sviluppo del software. Si tratta di modelli architetturali, paradigmi o concetti consolidati, descritti spesso tramite codice o diagrammi, che forniscono una guida chiara per affrontare specifiche problematiche di progettazione in maniera efficiente ed efficace.

La documentazione relativa ai *design pattern* svolge un ruolo fondamentale come punto di riferimento per l'intero team di sviluppo. Essa permette agli sviluppatori di comprendere rapidamente come ciascun pattern è stato implementato all'interno del codice. Questo è particolarmente utile durante:

- le **fasi di progettazione**, per identificare soluzioni standard e ben consolidate;
- lo **sviluppo**, per garantire coerenza e chiarezza nell'implementazione;
- la **manutenzione del software**, per ottenere una visione chiara della struttura architetturale e dei pattern adottati.

L'utilizzo dei *design pattern* migliora la leggibilità del codice, ne facilita la manutenzione e promuove l'adozione di soluzioni standardizzate all'interno del sistema.

2.2.6 Test

L'attività di **testing** è una fase critica del processo di sviluppo software, finalizzata a garantire che il prodotto finale soddisfi i requisiti funzionali, prestazionali e qualitativi stabiliti. Tale fase include diverse attività fondamentali, tra cui:

- **Pianificazione dei test**: definizione degli obiettivi, della strategia e delle risorse necessarie per eseguire i test;
- **Progettazione dei casi di test**: sviluppo di scenari di test dettagliati basati sui requisiti del sistema;
- **Esecuzione dei test**: applicazione dei casi di test per verificare il comportamento del software;
- **Analisi dei risultati**: valutazione degli output dei test per individuare difetti, errori o anomalie.

2.2.7 Metriche

Metriche inerenti la Progettazione

Codice	Nome Esteso
14PBM-PG	Profondità delle Gerarchie

2.2.8 Codifica

2.2.8.1 Descrizione e Scopo

L'attività di sviluppo del codice è affidata ai Programmatori, responsabili della conversione delle scelte progettuali in codice sorgente funzionante. Gli sviluppatori operano in un contesto organizzato, seguendo scrupolosamente le linee guida e gli standard definiti durante la progettazione dell'architettura software. Tale approccio garantisce uniformità nell'implementazione e l'applicazione delle migliori pratiche, promuovendo la realizzazione di codice efficiente, affidabile e facilmente manutenibile. Per mantenere elevati standard qualitativi, i Programmatori devono attenersi alle metriche specificate nel Piano di Qualifica[G] v1.0.

2.2.8.2 Aspettative

La fase di implementazione è finalizzata alla realizzazione di un software che risponda completamente ai requisiti e agli obiettivi stabiliti dalla Committente. Il codice prodotto deve soddisfare le seguenti caratteristiche fondamentali:

- Rispetto delle specifiche fornite;
- Semplicità e leggibilità;
- Efficienza in termini di prestazioni;
- Copertura di test per assicurare il corretto funzionamento.

2.2.8.3 Norme di codifica

Le seguenti linee guida sono state definite per garantire la qualità e la manutenibilità del codice:

- **Nomi significativi**: I nomi di variabili, costanti, classi e metodi devono essere chiari e descrittivi, per facilitare la comprensione del codice.
- **Commenti**: Il codice deve essere scritto in modo chiaro e facilmente comprensibile. I commenti devono essere usati solo quando strettamente necessario, per spiegare logiche complesse o decisioni particolari.
- **Indentazione e formattazione consistente**: Il codice deve essere correttamente indentato per favorire la leggibilità. È importante utilizzare spazi o tabulazioni in modo uniforme.
- **Gestione delle eccezioni**: Le eccezioni e gli errori devono essere gestiti correttamente, fornendo messaggi utili e trattando adeguatamente le situazioni anomale.
- **Riutilizzo del codice**: Scrivere il codice in modo modulare, per favorire il riuso in altre parti del programma o in progetti futuri.
- **Testabilità**: Le funzioni devono essere piccole e focalizzate, per rendere il codice facilmente testabile e verificabile.
- **Sicurezza**: Il codice deve essere sicuro, prevenendo vulnerabilità e proteggendo da potenziali attacchi.
- **Performance**: Ottimizzare il codice per ottenere buone prestazioni, utilizzando in modo efficiente le risorse disponibili e minimizzando il tempo di esecuzione.
- **Compatibilità**: Garantire che il codice funzioni correttamente su diverse piattaforme, sistemi operativi e browser, per una fruizione omogenea da parte degli utenti.

2.2.8.4 Metriche

Metrica	Nome
15PBM-PPM	Parametri per Metodo
16PB-CPC	Campi per Classe
17PBM-LCPM	Linea Di Commento Per Metodo
18PBM-CCM	Complessità Ciclomatica Metrica

Metriche di Codifica

2.2.8.5 Strumenti

Visual Studio Code è un ambiente di sviluppo integrato (IDE) creato da Microsoft, scelto dal team come strumento principale per lo sviluppo software. Questo IDE offre funzionalità avanzate che facilitano la scrittura, la modifica e il debugging del codice, supportando una vasta gamma di linguaggi di programmazione e framework.

3 Processi di Supporto

3.1 Documentazione

3.1.1 Introduzione

La documentazione software si riferisce al testo che accompagna un programma, descrivendo il prodotto sia per gli sviluppatori che per gli utilizzatori. Essa ha l'obiettivo di supportare i membri del team durante lo sviluppo, monitorando i processi e documentando tutte le attività, per facilitare anche la manutenzione del software e migliorare la qualità del prodotto finale.

In base a quanto sopra, la documentazione svolge un ruolo cruciale nel ciclo di vita del software. Le aspettative nei suoi confronti includono:

- Definizione di regole chiare e concise per la redazione dei documenti.
- Adozione di una struttura uniforme e standard per tutti i documenti nel ciclo di vita del software, per garantire omogeneità.

3.1.2 Ciclo di Vita del Documento

Il ciclo di vita di un documento software si articola in tre fasi principali:

- **Redazione**: la fase di creazione del documento, che viene suddivisa tra i membri del gruppo e supportata dall'uso di un sistema di versionamento.
- **Verifica**: una volta completata la stesura, il documento passa alla fase di verifica, che può essere effettuata su parti del documento o su tutto il contenuto. Ogni sezione deve essere verificata da una persona distinta dal redattore della sezione stessa.
- **Approvazione**: il documento, una volta completato e verificato, viene approvato dal Responsabile di Progetto.

3.1.3 Template

Il gruppo ha deciso di utilizzare un template semplice, creato con Latex. Questo modello è stato standardizzato e viene utilizzato per la redazione di tutti i documenti ufficiali.

3.1.4 Documenti Prodotti

I documenti generati durante il ciclo di vita del software sono suddivisi in due categorie principali:

Formali

I documenti formali sono quelli con un nome univoco e utilizzati per regolare le attività interne al gruppo durante tutto il ciclo di vita del software. Sono versionati e approvati dal Responsabile di Progetto. Questi documenti si suddividono in:

- **Ad uso interno**: destinati esclusivamente ai membri del gruppo, come ad esempio:
  - Norme di progetto
  - Verbali interni
- **Ad uso esterno**: destinati a enti esterni come il committente o il proponente, e consegnati nell'ultima versione approvata. Tra questi:
  - Analisi dei Requisiti[G]

  - Piano di Progetto[G]
  - Piano di Qualifica[G]
  - Glossario[G]
  - Verbali esterni

Informali

I documenti informali comprendono:

- Documenti non ancora approvati dal Responsabile di Progetto.
- Bozze e appunti brevi.
- Documenti che non necessitano di essere versionati.

Questi documenti sono gestiti in una sezione separata, dove il gruppo ha creato un Google Drive condiviso per facilitarne la gestione.

3.1.5 Struttura del Documento

Tutti i documenti ufficiali seguono una struttura rigida che deve essere rispettata. La struttura include:

Prima Pagina

La prima pagina include:

- Il titolo del gruppo.
- Il nome del documento.
- Il logo del gruppo.
- I membri del team.

Registro dei Cambiamenti - Changelog

Il registro dei cambiamenti tiene traccia della storia del documento. In questa sezione sono inclusi:

- La versione del documento.
- La data di ogni modifica.
- L'autore che ha effettuato la modifica.
- Il verificatore delle modifiche.
- Una breve descrizione delle modifiche.

Indice

Ogni documento include un indice subito dopo il registro dei cambiamenti. Questo indice aiuta a navigare nel documento, rendendo più facile la ricerca di sezioni specifiche.

3.1.6 Verbali

I verbali sono documenti speciali con una struttura diversa rispetto agli altri. Non includono il registro dei cambiamenti né l'indice. La struttura dopo la prima pagina prevede:

- **Partecipanti**: orario di inizio e fine dell'incontro, seguito da una tabella con i nomi e le durate di presenza dei partecipanti.
- **Sintesi ed elaborazione incontro**: un riassunto degli argomenti trattati e una sezione per eventuali dubbi o indicazioni per i prossimi incontri.

I verbali sono suddivisi in interni (tra i membri del gruppo) ed esterni (con l'azienda o il committente).

3.1.7 Nome del File

I file devono avere nomi coerenti, con la lettera iniziale minuscola, Per quanto riguarda i verbali, sia interni che esterni, il nome del file deve essere "verbale_YY-MM-DD_vXX", dove:

- **YY-MM-DD**: data dell'incontro.
- **XX**: numero progressivo del verbale.

3.1.8 Stile del Testo

Lo stile del testo nei documenti ufficiali include:

- **Grassetto**: per titoli e parole di rilevanza.
- Sottolineato: solo per i link.

3.1.9 Glossario

Il glossario è un documento contenente termini e definizioni utili per comprendere meglio il linguaggio tecnico, evitando ambiguità. I termini sono registrati in ordine alfabetico.

3.1.10 Tabelle

Le tabelle nei documenti ufficiali devono avere un titolo che descriva il contenuto e devono essere centrate orizzontalmente nella pagina.

3.1.11 Immagini

Le immagini devono essere centrate orizzontalmente. Anche i diagrammi UML sono trattati come immagini.

3.1.12 Metriche

Metrica	**Nome**
1PSM-IG	Indice di Gulpease
2PSM-CO	Correttezza Ortografica

Metriche relative al processo di Documentazione.

3.2 Gestione della Configurazione

La gestione della configurazione è un processo fondamentale per mantenere il software in uno stato coerente, garantendo che il sistema continui a funzionare correttamente nonostante le modifiche apportate nel tempo. Problemi di configurazione possono causare incoerenze o non conformità, con un impatto negativo sulle operazioni del sistema. Il gruppo, attraverso la gestione della configurazione, mira a:

- Individuare e risolvere i problemi prima che diventino critici;
- Facilitare il tracciamento delle modifiche e l'identificazione degli errori.

3.2.1 Versionamento

Il versionamento è un processo che consente di tracciare le modifiche apportate a un documento. Inoltre, permette di ripristinare il documento a uno stadio precedente e di visualizzare i cambiamenti effettuati nel tempo, associandoli al relativo autore. Il nostro gruppo ha adottato il seguente formato per identificare la versione di un documento:

$$[x].[y]$$

Dove:

- **x**: numero intero che parte da 0 e viene incrementato dal Responsabile di Progetto (**RdP**) dopo l'approvazione del documento (versione di produzione);
- **y**: numero intero incrementato dal Verificatore (**Ve**) a ogni verifica del documento;

3.2.2 Repository

Il nostro gruppo ha deciso di utilizzare per la gestione della configurazione il servizio GitHub, basato sul sistema di controllo di versione distribuito Git.

3.3 Gestione della Qualità

3.3.1 Descrizione

La gestione della qualità di progetto comprende i processi e le attività svolte all'interno di un progetto per garantire che la qualità dei deliverable e delle performance siano in linea con gli obiettivi e i requisiti definiti. I membri del nostro gruppo si pongono i seguenti obiettivi:

- Comprendere, valutare e gestire le aspettative del committente, assicurandosi che i requisiti siano rispettati;
- Definire chiaramente i requisiti di qualità e documentare tutte le procedure necessarie per completare il progetto in conformità con le aspettative richieste;
- Consegnare il progetto in linea con il piano di qualità, garantendo che il prodotto finale sia consegnato nei tempi previsti, rispettando il budget e soddisfacendo i requisiti e le aspettative del committente.

3.3.2 Piano di Qualifica

Per garantire il rispetto di tutti gli aspetti del processo di gestione della qualità, utilizziamo il **Piano di Qualifica**^{G}, un documento che include un elenco strutturato dei dati necessari per assicurare un piano di alta qualità. In particolare, il Piano di Qualifica^{G} prevede:

- La definizione dei requisiti richiesti dal committente;

- L'identificazione di metriche e parametri per l'analisi dei dati;
- L'implementazione di un sistema per il controllo della qualità durante l'intero ciclo di vita del progetto;
- La pianificazione di un sistema di miglioramento che descriva le azioni necessarie per analizzare le prestazioni di qualità e individuare le attività utili a incrementare il valore del progetto.

3.3.3 Metriche

Metrica	**Nome**
3PSM-FU	Facilità di Utilizzo
4PSM-TA	Tempo di Apprendimento
5PSM-TR	Tempo di Risposta
6PSM-TE	Tempo di Elaborazione
7PSM-QMS	Metriche di Qualità Soddisfatte

Metriche relative al processo di Gestione della Qualità.

3.4 Verifica

3.4.1 Introduzione

La verifica del software è il processo di valutazione del prodotto per garantire che la fase di sviluppo sia eseguita correttamente, al fine di costruire il prodotto desiderato. Questo processo si svolge durante lo sviluppo del software, consentendo di rilevare difetti e guasti nelle fasi iniziali del ciclo di vita e di verificare che il prodotto soddisfi i requisiti del cliente.

Le aspettative per questo processo includono:

- Incrementare la fiducia del gruppo nel proseguire lo sviluppo del progetto in modo corretto;
- Garantire il raggiungimento del prodotto finale atteso;
- Identificare precocemente gli errori, riducendo così i costi e il tempo necessari per le correzioni.

3.4.2 Tipi di Verifica

Il processo di verifica si compone di due tipi principali, ciascuno focalizzato su diversi aspetti del software. Insieme, questi due tipi garantiscono che il software sia conforme ai requisiti specificati. Inoltre, viene considerato un terzo tipo per la verifica della documentazione.

Analisi Statica

L'analisi statica consiste nell'ispezione del codice prima della sua esecuzione, assicurando che il software soddisfi i requisiti e le specifiche definiti. Poiché non richiede l'esecuzione dell'oggetto in verifica, questo tipo di analisi può essere applicato non solo al codice, ma anche alla documentazione. Questo approccio analizza gli aspetti statici del sistema software, come le convenzioni del codice o il calcolo di metriche. Include sia tecniche di test manuali che automatizzate, come quelle orientate alla coerenza. L'analisi statica si divide in due metodi principali:

- **Walkthroughs**: una lettura di ampio spettro che consente di esaminare e discutere eventuali errori o difetti trovati. Questo metodo è utile quando non si ha certezza sulla posizione dei problemi.

- **Inspection**: un metodo mirato per identificare e rimuovere errori e difetti. Si utilizza un approccio più focalizzato, avendo già un'idea delle possibili problematiche.

Analisi Dinamica

L'analisi dinamica viene eseguita durante l'esecuzione del software e consiste nella fase di test. A differenza della verifica statica, comporta l'esecuzione del sistema e dei suoi componenti.

Il gruppo adotterà un insieme di test ripetibili e automatizzati. L'automatizzazione sarà possibile attraverso strumenti dedicati, che verranno definiti successivamente.

Verifica della Documentazione

La verifica della documentazione si compone delle seguenti attività:

- Controllo di ortografia e sintassi;
- Controllo dell'utilizzo corretto delle norme tipografiche o di altre norme di stile e formattazione concordate;
- Verifica della pertinenza dei contenuti scritti.

3.4.3 Metriche

Metrica	**Nome**
8PSM-CC	Code Coverage
9PSM-BC	Branch Coverage
10PSM-SC	Statement Coverage
11PSM-FD	Failure Density
12PSM-PTCP	Passed Test Case Percentage

Metriche relative al processo di Verifica.

3.5 Validazione

3.5.1 Introduzione

La validazione del software è un processo di valutazione del prodotto, finalizzato a garantire che il software soddisfi i requisiti predefiniti e specificati dal richiedente, nonché le richieste e le aspettative degli utenti finali/proponente. Un processo di validazione ha successo quando è stata effettuata una buona verifica durante tutta la fase di sviluppo. L'esito finale positivo della validazione assicura che il prodotto finale sia allineato con le aspettative.

- Rilevare possibili errori ignorati o trascurati durante la fase di verifica;
- Soddisfare i requisiti specificati nell'Analisi dei Requisiti^{G} per il prodotto finale;
- Contribuire a migliorare la qualità e il valore del prodotto software finale.

4 Processi Organizzativi

4.1 Gestione dei Processi

4.1.1 Introduzione

La gestione dei processi rappresenta una fase cruciale per il successo di un progetto, garantendo che venga completato in conformità agli obiettivi e ai requisiti predefiniti. Questa fase si concentra sulla pianificazione, organizzazione, monitoraggio e controllo delle attività coinvolte nel ciclo di vita del software, assicurando che il lavoro svolto rispetti gli standard di qualità e soddisfi le aspettative del cliente.

Le principali attività di gestione dei processi sono le seguenti:

- **Definizione dei processi:** Documentazione dei processi chiave adottati nel progetto, inclusi quelli relativi allo sviluppo del software, controllo di versione, gestione dei cambiamenti e assicurazione della qualità.
- **Pianificazione dei processi:** Definizione degli obiettivi del progetto, delle fasi, delle risorse necessarie e delle scadenze. In questa fase vengono stabiliti i criteri di successo e redatto un piano di lavoro dettagliato.
- **Assegnazione delle risorse:** Allocazione dei membri del team alle attività specifiche, tenendo conto delle loro competenze e disponibilità.
- **Monitoraggio e controllo:** Controllo continuo dei progressi rispetto al piano stabilito, comprendente tempi, costi e qualità, oltre a gestione dei rischi.
- **Gestione dei cambiamenti:** Valutazione e gestione delle modifiche ai requisiti, alla pianificazione o alla distribuzione delle risorse.
- **Assicurazione della qualità:** Implementazione di procedure per garantire che il software soddisfi i requisiti e le aspettative del cliente.
- **Comunicazione e coordinamento:** Facilitazione della comunicazione tra membri del team e stakeholder per mantenere tutte le parti informate sullo stato del progetto.
- **Miglioramento continuo:** Analisi dei processi per identificare aree di miglioramento e ottimizzare l'efficienza e la qualità.

4.1.2 Pianificazione

4.1.2.1 Descrizione

La pianificazione dei processi rappresenta un elemento fondamentale nell'ambito della gestione di un progetto, poiché implica l'analisi, la definizione, l'organizzazione e il controllo accurato delle varie attività che devono essere svolte per garantire il raggiungimento degli obiettivi prefissati. Si tratta di un'attività di natura strategica che non solo assicura una chiara direzione operativa, ma fornisce anche una struttura gestionale ben definita e solida, in grado di guidare il progetto attraverso tutte le fasi del suo ciclo di vita, dalla concezione iniziale fino al completamento.

4.1.2.2 Obiettivi

L'obiettivo principale della pianificazione è assicurare l'esecuzione efficiente ed efficace del progetto, rispettando gli obiettivi e i requisiti stabiliti. Inoltre:

- Ogni membro del team deve assumere almeno una volta ciascun ruolo, favorendo crescita e collaborazione.

- Ridurre e prevenire i rischi affrontando le sfide in modo anticipato, permettendo al team di superare eventuali difficoltà nei tempi previsti.

4.1.3 Metriche

Metrica	**Nome**
1POM-RSI	Requirements Stability Index

Metriche relative alla Pianificazione.

4.1.4 Assegnazione dei Ruoli

Durante l'implementazione del progetto, i membri del team di Code7Crusaders ricopriranno ruoli distinti. Ogni ruolo comporta specifiche responsabilità:

- **Responsabile:**
  - Coordina il gruppo di lavoro.
  - Pianifica e controlla le attività.
  - Gestisce le risorse e le comunicazioni esterne.
  - Redige il Piano di Progetto^{G}.
- **Amministratore:**
  - Gestisce l'ambiente di lavoro e le procedure.
  - Gestisce la configurazione del prodotto.
  - Redige le Norme di Progetto^{G}.
- **Analista:**
  - Analizza i requisiti del progetto e il dominio applicativo.
  - Redige l'Analisi dei Requisiti^{G}.
- **Progettista:**
  - Progetta l'architettura del prodotto.
  - Prende decisioni tecniche e tecnologiche.
  - Redige la Specifica Tecnica.
- **Programmatore:**
  - Scrive il codice e implementa le funzionalità richieste.
  - Redige il Manuale Utente.
- **Verificatore:**
  - Verifica che il lavoro svolto sia conforme alle norme e alle specifiche.
  - Redige il Piano di Qualifica^{G}.

4.1.5 Ticketing

Il gruppo code7crusaders utilizza GitHub Projects per il ticketing. La roadmap è organizzata in tre colonne: *To Do*, *In Progress*, e *Completed*. Ogni attività è classificata in base a:

- **Priorità:** Bassa, media o alta.
- **Dimensione:** XS, S, M, L, XL.
- **Stima ore:** Numero di ore necessarie per completare l'attività.

4.1.6 Gestione dei rischi

4.1.6.1 Struttura dei rischi

I rischi sono classificati in tre categorie principali:

- Rischi tecnologici;
- Rischi comunicativi;
- Rischi individuali;
- Rischi organizzativi;

Ogni rischio è identificato tramite un codice univoco con la seguente struttura:

R[Categoria][Indice] - [Nome]

Dove:

- **Categoria**: indica il tipo di rischio e può assumere i seguenti valori:
  - **T**: per i rischi tecnologici;
  - **C**: per i rischi comunicativi;
  - **I**: per i rischi individuali;
  - **O**: per i rischi organizzativi.
- **Indice**: un identificatore progressivo univoco all'interno della categoria di appartenenza;
- **Nome**: il nome descrittivo del rischio.

4.1.7 Metriche

Metrica	**Nome**
13PSM-RMR	Risk Mitigation Rate
14PSM-NCR	Rischi Non Calcolati

Metriche relative ai Rischi.

4.2 Procedure Comunicative

4.2.1 Comunicazioni Asincrone

Per assicurare una comunicazione efficace in modalità asincrona, sono stati identificati strumenti specifici per le interazioni interne al team e quelle con soggetti esterni. Di seguito si descrivono le piattaforme utilizzate in ciascun contesto.

- **Interne:** Il gruppo 7Crusaders detiene un gruppo Whatsapp come canale di comunicazione asincrono, consentendo comunicazione semplice e veloce.
- **Esterne:** Gestite tramite e-mail e tramite la piattaforma Telegram.

4.2.2 Comunicazioni Sincrone

Per garantire un'efficace gestione della comunicazione, è fondamentale distinguere tra le modalità di interazione sincrona utilizzate internamente al team e quelle adottate per le relazioni esterne con l'azienda Ergon. Di seguito si riportano le piattaforme selezionate per ciascun contesto.

- **Interne:** Viene adottata la piattaforma Discord per la velocità nell'effettuare riunioni in chiamata vocale.
- **Esterne:** L'azienda Ergon adotta Zoom come piattaforma di riunioni esterne.

4.2.3 Riunioni Interne

Le riunioni interne del gruppo Code7Crusaders si tengono ogni venerdì, utilizzando Discord come piattaforma di comunicazione. Questi incontri servono principalmente per monitorare i progressi delle attività in corso, discutere eventuali difficoltà riscontrate e pianificare i passi successivi. L'orario delle riunioni è fissato dalle 15:00 alle 16:00, salvo necessità particolari che richiedano un adattamento. Nel caso in cui un membro non possa partecipare, è tenuto a informare tempestivamente il resto del team. Se necessario, i membri assenti potranno recuperare le informazioni rilevanti consultando il verbale della riunione. Durante queste riunioni, il team lavora in modo collaborativo e dinamico: ogni membro condivide lo stato delle proprie attività, proponendo soluzioni a eventuali problemi emersi. L'approccio informale permette di discutere liberamente idee, priorità e obiettivi futuri, favorendo una comunicazione aperta ed efficace.

4.2.4 Riunioni Esterne

Durante lo sviluppo del progetto, è fondamentale organizzare incontri periodici con i committenti o con la proponente per valutare lo stato di avanzamento del lavoro, risolvere eventuali dubbi e discutere questioni rilevanti. La pianificazione e la gestione di questi incontri sono affidate al responsabile, che si occupa di convocarli e garantirne un'efficace organizzazione. Il responsabile ha anche il compito di presentare i punti principali della discussione alla proponente o ai committenti, coinvolgendo i membri del gruppo direttamente interessati in base agli argomenti trattati. Questo metodo garantisce una comunicazione chiara e mirata, evitando dispersioni di tempo e favorendo la comprensione reciproca. La partecipazione alle riunioni è considerata una priorità per tutti i membri del gruppo. Ogni membro si impegna a riorganizzare i propri impegni, quando possibile, per assicurare una presenza costante. Nel caso in cui un membro sia impossibilitato a partecipare per cause inderogabili, il responsabile si farà carico di informare prontamente i committenti o la proponente e, se necessario, proporrà il rinvio dell'incontro a una data più adeguata. Durante ogni incontro, sarà inoltre garantita una registrazione accurata delle informazioni discusse, per mantenerne traccia e facilitarne il recupero in futuro.

4.3 Formazione

4.3.1 Introduzione

La formazione è una componente essenziale per garantire che tutti i membri del team siano adeguatamente preparati per affrontare le sfide tecniche e gestionali del progetto. Questo processo si concentra sullo sviluppo delle competenze e delle conoscenze necessarie per utilizzare strumenti, tecnologie e metodologie specifiche del progetto, promuovendo così l'efficienza e la qualità del lavoro svolto.

4.3.2 Metodo di formazione

Individuale: Ogni individuo del team dovrà compiere un processo di autoformazione per riuscire a svolgere al meglio il ruolo assegnato. La rotazione dei ruoli permetterà al nuovo occupante di un ruolo di apprendere le competenze necessarie da chi lo ha precedentemente svolto, nel caso avesse delle lacune. Questo metodo permette di avere una formazione continua e di garantire che ogni membro del team sia in grado di svolgere ogni ruolo. **Di gruppo:** Oltre alla formazione individuale, il team parteciperà a sessioni di formazione collettiva per condividere conoscenze, affrontare problematiche comuni e rafforzare la collaborazione. Questi incontri includeranno workshop, presentazioni tecniche e sessioni di revisione del lavoro, mirate a uniformare le competenze tra i membri del gruppo. La formazione di gruppo consente inoltre di sviluppare una visione condivisa del progetto e di affrontare sfide complesse in modo coordinato, promuovendo il lavoro di squadra e la comunicazione efficace.

5 Standard per la qualità

Abbiamo scelto di utilizzare standard internazionali per valutare e migliorare la qualità dei processi e del software. In particolare, seguiremo lo standard ISO/IEC 12207:1995, che fornisce linee guida per organizzare i processi in categorie principali, di supporto e organizzative. Per quanto riguarda la qualità del software, adotteremo lo standard ISO/IEC 25010:2023, il quale offre una struttura chiara e completa per identificare e classificare le metriche di qualità. Abbiamo deciso di limitarci a questi due standard, poiché il precedente ISO/IEC 9126:2001 è stato ufficialmente ritirato e sostituito dallo standard ISO/IEC 25010:2023.

5.1 Caratteristiche del sistema ISO/IEC 25010:2023

5.1.1 Appropriatezza funzionale

- **Completezza:** Il software deve soddisfare tutti i requisiti definiti e le aspettative degli utenti.
- **Correttezza:** Il funzionamento del software deve essere accurato e coerente con le specifiche.
- **Appropriatezza:** Il prodotto deve risultare idoneo allo scopo previsto e al contesto operativo in cui viene utilizzato.

5.1.2 Performance

- **Tempo:** Il software deve rispettare le tempistiche di sviluppo e consegna stabilite.
- **Risorse:** L'utilizzo delle risorse di sistema deve essere ottimizzato ed efficiente.
- **Capacità:** Il software deve essere in grado di gestire carichi di lavoro attesi senza degradare le prestazioni.

5.1.3 Compatibilità

- **Coesistenza:** Il prodotto deve operare correttamente in presenza di altri software o sistemi.
- **Interoperabilità:** Deve essere possibile scambiare informazioni e collaborare con altri software o sistemi.

5.1.4 Usabilità

- **Riconoscibilità:** L'interfaccia utente deve essere intuitiva e facilmente comprensibile.
- **Apprendibilità:** Gli utenti devono poter imparare a utilizzare il software in modo rapido.
- **Operabilità:** Il software deve risultare semplice da utilizzare e da controllare.
- **Gestione degli errori:** Deve essere in grado di individuare e gestire gli errori in modo efficace.
- **Estetica:** L'interfaccia utente deve essere gradevole e visivamente accattivante.
- **Accessibilità:** Deve garantire l'accesso anche agli utenti con disabilità.

5.1.5 Affidabilità

- **Maturità:** Il software deve garantire stabilità, robustezza e affidabilità.
- **Disponibilità:** Deve essere accessibile e operativo ogni volta che è necessario.
- **Tolleranza ai guasti:** Il software deve saper gestire errori o condizioni inaspettate senza interruzioni gravi.
- **Recuperabilità:** Deve poter ripristinare i dati e le funzionalità in caso di errore o guasto.

5.1.6 Sicurezza

- **Riservatezza:** Il software deve proteggere i dati sensibili e garantire la privacy.
- **Integrità:** Deve assicurare che i dati siano accurati e completi.
- **Non ripudio:** Deve garantire che le transazioni non possano essere negate.
- **Autenticazione:** Deve verificare l'identità degli utenti e limitare l'accesso ai soli utenti autorizzati.
- **Autenticità:** Deve permettere di verificare la provenienza dei dati e delle informazioni.

5.1.7 Manutenibilità

- **Modularità:** Il software deve essere strutturato in moduli indipendenti per semplificarne la gestione.
- **Riutilizzabilità:** Parti del software devono poter essere riutilizzate in contesti o progetti differenti.
- **Analizzabilità:** Il codice e la documentazione devono consentire una facile identificazione dei problemi.
- **Modificabilità:** Il software deve poter essere aggiornato e migliorato senza difficoltà.
- **Testabilità:** Il prodotto deve consentire l'efficace esecuzione di test per validarne il funzionamento.

5.1.8 Portabilità

- **Adattabilità:** Il software deve essere in grado di funzionare correttamente in ambienti diversi, adattandosi a nuove tecnologie o requisiti specifici.
- **Installabilità:** Deve essere possibile installare e configurare il software in modo semplice e rapido.
- **Sostituibilità:** Il prodotto deve poter essere facilmente rimpiazzato da versioni aggiornate o soluzioni alternative senza impatti significativi.

5.2 Suddivisione secondo standard ISO/IEC 12207:1995

5.2.1 Processi primari

I processi primari sono fondamentali per lo sviluppo del software e includono:

- **Acquisizione:** Comprende tutte le attività necessarie per ottenere un prodotto software o servizio. Queste attività includono la stesura delle richieste, la scelta del fornitore e la gestione del contratto.
- **Fornitura:** Si riferisce alle attività svolte per consegnare il prodotto o servizio al cliente. Tra queste vi sono la preparazione delle offerte, la negoziazione contrattuale e la consegna finale.
- **Sviluppo:** Riguarda la progettazione e la creazione del software, partendo dall'analisi dei requisiti fino alla fase di implementazione, test e integrazione.
- **Operatività:** Insieme di attività necessarie per garantire il funzionamento del software in un ambiente reale, come la gestione delle operazioni quotidiane, il monitoraggio delle prestazioni e il backup dei dati.
- **Manutenzione:** Include le operazioni di aggiornamento o modifica del software dopo la sua consegna, sia per correggere difetti che per ottimizzarne le prestazioni.

5.2.2 Processi di supporto

I processi di supporto forniscono assistenza ai processi primari e comprendono:

- **Documentazione:** Creazione e gestione dei documenti necessari per supportare il ciclo di vita del software.
- **Gestione della configurazione:** Monitoraggio e controllo delle modifiche al software per garantire la coerenza e la tracciabilità delle versioni.
- **Assicurazione della qualità:** Attività di controllo e verifica per assicurare che processi e prodotti rispettino gli standard prefissati, includendo pianificazione della qualità, ispezioni e revisioni.
- **Verifica:** Controllo dei prodotti per accertarsi che soddisfino i requisiti stabiliti.
- **Validazione:** Garantisce che il software risponda alle reali esigenze degli utenti finali.
- **Revisioni congiunte con il cliente:** Sessioni di verifica periodica con tutte le parti coinvolte per monitorare i progressi del progetto.
- **Audit**$^{\mathbf{G}}$: Ispezioni formali per garantire il rispetto delle procedure, degli standard e dei requisiti.
- **Risoluzione dei problemi:** Attività dedicate all'identificazione e risoluzione tempestiva di eventuali malfunzionamenti o problematiche.

5.2.3 Processi organizzativi

Questi processi forniscono supporto all'intera organizzazione e comprendono:

- **Gestione:** Attività di pianificazione, monitoraggio e controllo delle operazioni del progetto.
- **Infrastruttura:** Gestione delle risorse e infrastrutture necessarie per supportare tutte le fasi del ciclo di vita del software.
- **Miglioramento:** Identificazione e implementazione di azioni mirate al miglioramento continuo dei processi organizzativi.
- **Formazione:** Programmi di formazione e aggiornamento delle competenze per il personale coinvolto nello sviluppo e nella gestione del software.

6 Metriche di qualità

La qualità di processo è un criterio fondamentale ed è alla base di ogni prodotto che rispecchi lo stato dell'arte. Per raggiungere tale obiettivo è necessario sfruttare delle pratiche rigorose che consentano lo svolgimento di ogni attività in maniera ottimale. Valutando nel miglior modo possibile la qualità del prodotto e l'efficacia dei processi, sono state definite delle metriche riportate di seguito. Lo scopo di questa sezione è quello di identificarne i parametri che le metriche devono rispettare per essere considerate accettabili o ottime. Esse sono state suddivise utilizzando lo standard ISO/IEC 12207:1995, il quale separa i processi di ciclo di vita del Software in tre categorie:

1. Processi di base e/o primari;
2. Processi di supporto;
3. Processi organizzativi;

6.1 Processi di base e/o primari

6.1.1 Fornitura

Nella fase di Fornitura si definiscono le procedure e le risorse necessarie per la consegna del prodotto. Definiamo quindi le seguenti metriche:

1PBM-PV Planed Value

- **Definizione:** il Planned Value[G] (Valore Pianificato) rappresenta il valore del lavoro programmato per essere completato fino a un determinato momento. Si tratta del budget preventivato per lo sprint in corso.
- **Come si calcola:**

$$PV = BAC \times LP$$

dove:

→ BAC: Budget At Completion

→ LP: Percentuale Lavoro Pianificato

- **Valore ammissibile:**

$$PV \geq 0$$

- **Valore ottimo:**

$$PV \leq BAC$$

2PBM-ETC Estimated to complete

- **Definizione:** l'Estimate to Complete[G] (o Stima al Completamento) rappresenta una previsione del costo necessario per completare le attività rimanenti del progetto basata sulle performance attuali.
- **Come si calcola:**

$$ETC = BAC - EV$$

dove:

→ BAC: Budget At Completion

→ EV: Earned Value[G]

- **Valore ammissibile:**

$$ETC \geq 0$$

- **Valore ottimo:**

$$ETC \leq EAC$$

3PBM-EAC Estimate at Completition

- **Definizione:** l'Estimate at Completion[G] (o Stima da Completare) rappresenta una previsione aggiornata del costo totale del progetto basata sulle performance attuali, calcolata in base ai costi effettivamente sostenuti e ai costi stimati per completare il lavoro rimanente.
- **Come si calcola:**

$$EAC = AC + (BAC - EV)$$

dove:

→ BAC: Budget At Completion

→ EV: Earned Value[G]

→ AC: Actual Cost[G]

- **Valore ammissibile:**

$$EAC \leq BAC + 10\%$$

- **Valore ottimo:**

$$EAC \leq BAC$$

4PBM-EV Earned Value

- **Definizione:** l'Earned Value[G] (Valore Guadagnato) rappresenta il valore del lavoro effettivamente completato fino al periodo in analisi.
- **Come si calcola:**

$$EV = AC \times LC$$

dove:

→ LC: Percentuale di lavoro completato

→ AC: Actual Cost[G]

- **Valore ammissibile:**

$$EV \geq 0$$

- **Valore ottimo:**

$$EV \leq EAC$$

5PBM-AC

- **Definizione:** l'Actual Cost[G] (Costo Effettivo) rappresenta il costo effettivamente sostenuto per completare il lavoro fino al periodo in analisi.
- **Come si calcola:** Si ottiene sommando tutti i costi effettivi sostenuti fino a quella data.
- **Valore ammissibile:**

$$AC \geq 0$$

- **Valore ottimo:**

$$AC \leq EAC$$

6PBM-SV Scheduled Variance

- **Definizione:** la Schedule Variance (o Variazione di Programma) rappresenta la differenza tra il valore del lavoro effettivamente completato e il valore del lavoro pianificato, calcolata in percentuale.
- **Come si calcola:**

$$SV = (EV - PV)/EV$$

dove:

→ EV: Earned Value[G]

→ PV: Planned Value[G]

- **Valore ammissibile:**

$$SV \geq -10\%$$

- **Valore ottimo:**

$$SV \geq 0\%$$

7PBM-CV Cost Variance

- **Definizione:** la Cost Variance[G] (o Variazione dei Costi) rappresenta la differenza tra il valore del lavoro effettivamente completato e il costo effettivamente sostenuto per completarlo, calcolata in percentuale
- **Come si calcola:**

$$CV = (EV - AC)/EV$$

dove:

→ EV: Earned Value[G]

→ AC: Actual Cost[G]

- **Valore ammissibile:**

$$CV \geq -10\%$$

- **Valore ottimo:**

$$CV \geq 0\%$$

8PBM-CPI Cost Performance Index

- **Definizione:** il Cost Performance Index rappresenta il rapporto tra il valore del lavoro effettivamente completato e i costi sostenuti per completarlo.
- **Come si calcola:**

$$CPI = EV/AC$$

dove:

→ EV: Earned Value[G]

→ AC: Actual Cost[G]

- **Valore ammissibile:**

$$CPI \geq 0.8$$

- **Valore ottimo:**

$$CPI \geq 1$$

9PBM-SPI Scheduled Performance Index

- **Definizione:** lo Schedule Performance Index rappresenta l'efficienza con cui il progetto sta rispettando il programma.
- **Come si calcola:**

$$SPI = EV/PV$$

dove:

→ EV: Earned Value[G]

→ PV: Planned Value[G]

- **Valore ammissibile:**

$$SPI \geq 0.8$$

- **Valore ottimo:**

$$SPI \geq 1$$

10PBM-OTDR On-Time Delivery Rate

- **Definizione:** l'On-Time Delivery Rate (o Tasso di Consegna nei Tempi) rappresenta la percentuale di attività completate entro la data di scadenza.
- **Come si calcola:**

$$OTDR = AP/AT$$

dove:

→ AP: Attività completate entro la data di scadenza

→ AT: Attività Totali

- **Valore ammissibile:**

$$OTDR \geq 90\%$$

- **Valore ottimo:**

$$OTDR \geq 95\%$$

6.1.2 Sviluppo

Nella fase di sviluppo si realizza il prodotto software, seguendo le specifiche definite in fase di progettazione.

6.1.2.1 Analisi dei Requisiti Questa fase consiste nell'esaminare le richieste del proponente e nel definire i requisiti che il prodotto dovrà soddisfare. Definiamo quindi le seguenti metriche:

11PBM-PRO Percentuale Requisiti Obbligatori

- **Definizione:** rappresenta la percentuale di requisiti obbligatori soddisfatti secondo quanto definito nel documento Analisi dei Requisiti[G].
- **Come si calcola:**

$$PRO = ROS/ROT$$

dove:

→ ROS: Requisiti Obbligatori Soddisfatti

→ ROT: Requisiti Obbligatori Totali

- **Valore ammissibile:**

$$PRO = 100\%$$

- **Valore ottimo:**

$$PRO = 100\%$$

12PBM-PRD Percentuale Requisiti Desiderabili

- **Definizione:** rappresenta la percentuale di requisiti desiderabili soddisfatti secondo quanto definito nel documento Analisi dei Requisiti[G].
- **Come si calcola:**

$$PRD = RDS/RDT$$

dove:

→ RDS: Requisiti Desiderabili Soddisfatti

→ RDT: Requisiti Desiderabili Totali

- **Valore ammissibile:**

$$PRD \geq 30\%$$

- **Valore ottimo:**

$$PRD = 100\%$$

13PBM-PRF Percentuale Requisiti Facoltativi

- **Definizione:** rappresenta la percentuale di requisiti facoltativi soddisfatti secondo quanto definito nel documento Analisi dei Requisiti[G].
- **Come si calcola:**

$$PRF = RFS/RFT$$

dove:

→ RFS: Requisiti Facoltativi Soddisfatti

→ RFT: Requisiti Facoltativi Totali

- **Valore ammissibile:**

$$PRF \geq 0\%$$

- **Valore ottimo:**

$$PRF = 100\%$$

6.1.2.2 Progettazione Questa fase consiste nel definire l'architettura del prodotto software, in modo da soddisfare i requisiti in fase di analisi. Definiamo quindi le seguenti metriche:

14PBM-PG Profondità delle gerarchie

- **Definizione:** la profondità delle gerarchie rappresenta il numero massimo di livelli di annidamento delle classi.
- **Come si calcola:** Si conta il numero massimo di livelli di annidamento delle classi
- **Valore ammissibile:**

$$PG \leq 7$$

- **Valore ottimo:**

$$PG \leq 5$$

6.1.2.3 Codifica Queste metriche aiutano a valutare la qualità del codice, la complessità e la manutenibilità

15PBM-PPM Parametri Per Metodo

- **Definizione:** numero medio di parametri passati ai metodi. Un numero elevato di parametri può indicare che un metodo è troppo complesso o che potrebbe essere suddiviso in metodi più piccoli.
- **Come si calcola:**

$$PPM = P/M$$

dove:

→ P: Numero totale di parametri

→ M: Numero totale di metodi

- **Valore ammissibile:**

$$PPM \leq 7$$

- **Valore ottimo:**

$$PPM \leq 5$$

16PBM-CPC Campi per classe

- **Definizione:** numero medio di campi (variabili di istanza) per classe. Un numero elevato di campi dati può indicare che una classe sta facendo troppo e che potrebbe essere suddivisa in classi più piccole.
- **Come si calcola:**

$$CPC = CD/CL$$

dove:

→ CD: Numero totale di campi dati

→ CL: Numero totale di classi

- **Valore ammissibile:**

$$CPC \leq 8$$

- **Valore ottimo:**

$$CPC \leq 5$$

17PBM-LCPC Linea di commento per Metodo

- **Definizione:** numero medio di linee di codice per metodo. Metodi troppo lunghi possono essere difficili da leggere, capire e mantenere.
- **Come si calcola:**

$$LCPM = LC/M$$

dove:

→ LC: Numero totale di linee di codice

→ M: Numero totale di metodi

- **Valore ammissibile:**

$$LCPM \geq 50$$

- **Valore ottimo:**

$$LCPM \geq 20$$

18PBM-CCM Complessità Ciclomatica Metrica

- **Definizione:** la Complessità CicloMatica rappresenta la complessità di un programma sulla base del numero di percorsi lineari indipendenti attraverso il codice sorgente. Un valore elevato indica un codice più complesso e Potenzialmente più difficile da mantenere.
- **Come si calcola:**

$$CCM = E - N + 2P$$

dove:

→ E: Numero di archi del grafo

→ N: Numero di nodi del grafo

→ P: Numero di componenti connesse

- **Valore ammissibile:**

$$CCM \leq 6$$

- **Valore ottimo:**

$$CCM \leq 3$$

6.2 Processi di Supporto

I processi di supporto si affiancano ai processi primari per garantire il corretto svolgimento delle attività.

6.2.1 Documentazione

La Documentazione è un aspetto fondamentale per la comprensione del prodotto e per la sua manutenibilità. A livello pratico consiste nella redazione di manuali e documenti tecnici che descrivano il funzionamento del prodotto e le scelte progettuali adottate. Per valutare la qualità di tale processo, sono state definite le seguenti metriche:

1PSM-IG Indice di Gulpease

- **Definizione:** l'Indice Gulpease è un indice di leggibilità di un testo tarato sulla lingua italiana. Misura la lunghezza delle parole e delle frasi rispetto al numero di lettere.
- **Come si calcola:**

$$IG = 89 + (300 \times F - 10 \times L)/P$$

dove:

→ F: Numero totale di frasi nel documento;

→ N: Numero totale di lettere nel documento;

→ P: Numero totale di parole nel documento;

- **Valore ammissibile:**

$$IG \geq 50$$

- **Valore ottimo:**

$$IG \geq 75$$

2PSM-CO Correttezza Ortografica

- **Definizione:** la correttezza ortografica indica la presenza di errori ortografici nei documenti.
- **Come si calcola:** si contano gli errori ortografici presenti nei documenti
- **Valore ammissibile:**

$$CO = 0errori$$

- **Valore ottimo:**

$$CO = 0errori$$

6.2.2 Gestione della Qualità

La gestione della qualità è un processo che si occupa di definire una metodologia per garantire la qualità del prodotto. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

3PSM-FU Facilità di Utilizzo

- **Definizione:** rappresenta il livello di usabilità del prodotto software mediante il numero di errori riscontrati durante l'utilizzo del prodotto da parte di un utente generico.
- **Come si calcola:** si contano gli errori riscontrati durante l'utilizzo del prodotto da parte di un utente che non ha conoscenze pregresse sul prodotto software.
- **Valore ammissibile:**

$$FU \leq 3errori$$

- **Valore ottimo:**

$$FU \leq 0errori$$

4PSM-TA Tempo di Apprendimento

- **Definizione:** indica il tempo massimo richiesto da parte di un utente generico per apprendere l'utilizzo del prodotto.
- **Come si calcola:** si misura il tempo necessario per apprendere l'utilizzo del prodotto da parte di un utente che non ha conoscenze pregresse sul prodotto software.
- **Valore ammissibile:**

$$TA \leq 12minuti$$

- **Valore ottimo:**

$$TA \leq 8min$$

5PSM-TR Tempo di Risposta

- **Definizione:** indica il tempo massimo di risposta del sistema sotto carico rilevato.
- **Come si calcola:** si misura il tempo massimo necessario per ottenere una risposta dal sistema.
- **Valore ammissibile:**

$$TR \leq 8secondi$$

- **Valore ottimo:**

$$TR \leq 4secondi$$

6PSM-TE Tempo di Elaborazione

- **Definizione:**indica il tempo massimo di elaborazione di un dato grezzo fino alla sua presentazione rilevato
- **Come si calcola:**si misura il tempo massimo di elaborazione di un dato grezzo dal momento della sua comparsa nel sistema fino alla sua presentazione all'utente.
- **Valore ammissibile:**

$$TE \leq 10 secondi$$

- **Valore ottimo:**

$$TE \leq 5 secondi$$

7PSM-QMS Metriche di Qualità Soddisfatte

- **Definizione:**indica il numero di metriche implementate e soddisfatte, tra quelle definite.
- **Come si calcola:**

$$QMS^{\mathrm{G}} = MS/MT$$

dove:

→ MS: Metriche soddisfatte;

→ MT: Metriche Totali;

- **Valore ammissibile:**

$$QMS^{\mathrm{G}} \geq 90\%$$

- **Valore ottimo:**

$$QMS^{\mathrm{G}} \geq 90\%$$

6.2.3 Verifica

La verifica è un processo che si occupa di controllare che il prodotto soddisfi i requisiti stabiliti e sia pienamente funzionante. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

8PSM-CC Code Coverage

- **Definizione:** la Code Coverage indica quale percentuale del codice sorgente è stata eseguita durante i test. Serve per capire quanto del codice è stato verificato dai test automatizzati.
- **Come si calcola:**

$$CC = LE/LT$$

dove:

→ LE: Linee di codice eseguite;

→ LT: Linee di codice totali;

- **Valore ammissibile:**

$$CC \geq 80\%$$

- **Valore ottimo:**

$$CC \geq 100\%$$

9PSM-BC Branch Coverage

- **Definizione:** la Branch Coverage indica quale percentuale dei rami decisionali (percorsi derivanti da istruzioni condizionali come if, for, while) del codice è stata eseguita durante i test.
- **Come si calcola:**

$$BC = BE/BT$$

dove:

→ BE: Rami eseguiti;

→ BT: Rami totali;

- **Valore ammissibile:**

$$BC \geq 80\%$$

- **Valore ottimo:**

$$BC \geq 100\%$$

10PSM-SC Statement Coverage

- **Definizione:** la Statement Coverage indica quale percentuale di istruzioni del codice è stata eseguita durante i test.
- **Come si calcola:**

$$SC = IE/IT$$

dove:

→ IE: istruzioni eseguite;

→ IT: istruzioni totali;

- **Valore ammissibile:**

$$SC \geq 80\%$$

- **Valore ottimo:**

$$SC \geq 100\%$$

11PSM-FD Failure Density

- **Definizione:** la Failure Density indica il numero di difetti trovati in un software o in una parte di esso durante il ciclo di sviluppo rispetto alla dimensione del software stesso.
- **Come si calcola:**

$$FD = DF/LT$$

dove:

→ DF: Difetti Trovati;

→ LT: linee di codice totali;

- **Valore ammissibile:**

$$FD \leq 15\%$$

- **Valore ottimo:**

$$FD = 0\%$$

12PSM-PTCP Passed Test Case Percentage

- **Definizione:** la Passed Test Case Percentage indica la percentuale di test che sono stati eseguiti con successo su una base di test.
- **Come si calcola:**

$$PTCP = TS/TT$$

dove:

→ TS: Test superati;

→ TT: Test totali;

- **Valore ammissibile:**

$$PTCP \geq 90\%$$

- **Valore ottimo:**

$$PTCP \geq 100\%$$

6.2.4 Risoluzione dei problemi

La risoluzione dei problemi è un processo che mira a identificare, analizzare e risolvere le varie problematiche che possono emergere durante lo sviluppo. La gestione dei rischi, in particolare, si occupa di identificare, analizzare e gestire i rischi che possono insorgere durante lo svolgimento del progetto. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

13PSM-RMR Risk Mitigation Rate

- **Definizione:** la Risk Mitigation Rate indica la percentuale di rischi identificati che sono stati mitigati con successo.
- **Come si calcola:**

$$RMR = RM/RT$$

dove:

→ RM: Rischi mitigati;

→ RT: Rischi totali identificati;

- **Valore ammissibile:**

$$RMR \geq 80\%$$

- **Valore ottimo:**

$$RMR \geq 100\%$$

14PSM-NCR Rischi Non Calcolati

- **Definizione:** indica il numero di rischi occorsi che non sono stati preventivati durante l'analisi dei rischi.
- **Come si calcola:** si contano i rischi occorsi e non preventivati.
- **Valore ammissibile:**

$$NCR \leq 3$$

- **Valore ottimo:**

$$NCR = 3$$

6.3 Processi organizzativi

I Processi organizzativi sono processi che si occupano di definire le linee guida e le procedure da seguire per garantire un'efficace gestione e coordinazione del progetto.

6.3.1 Pianificazione

La Pianificazione è un processo che si occupa di definire le attività da svolgere e le risorse temporali e umane necessarie per il loro svolgimento. Per valutare la qualità di tale processo sono state definite le seguenti metriche:

1POM-RSI Requirements Stability Index

- **Definizione:** il Requirements Stability Index^{G} (RSI) indica la percentuale di requisiti che sono rimasti invariati rispetto al totale dei requisiti inizialmente definiti. Si tratta di una metrica utilizzata per misurare quanto i requisiti di un progetto rimangono stabili durante il ciclo di vita del progetto stesso, è particolarmente utile per comprendere l'impatto delle modifiche ai requisiti sul progetto.
- **Come si calcola:**

$$RSI = \frac{(RI-(RA+RR+RC))}{RI}$$

dove:

→ RI: Requisiti iniziali;

→ RA: Requisiti aggiunti;

→ RR: Requisiti rimossi;

→ RC: Requisiti cambiati;

- **Valore ammissibile:**

$$RSI \geq 75\%$$

- **Valore ottimo:**

$$RSI = 100\%$$