Norme di Progetto

SnakeByte (Gruppo 1):
Valeria Baleanu, Leonardo Pellizzon, Filippo Venzo, Giuseppe De Fina,
Francesco Pasqual, Christian Libralato, Luca Granziero
(2109911, 2111006, 2113705, 2113187, 2103119, 2101047, 2075512)

Informazioni documento			
Versione	**Data**	**Stato**	**Destinatari**
0.9.0	10/2/2026	Verificato	SnakeByte, prof. Vardanega Tullio, prof. Cardin Riccardo

Contatti: snakebyteteam@gmail.com

Registro delle modifiche					
Versione	**Data**	**Autore**	**Verifica**	**Approvazione**	**Descrizione**
0.9.0	10/2/2026	G. De Fina	L. Granziero F. Pasqual	-	Aggiunto elenco delle tabelle.
0.8.0	9/2/2026	G. De Fina	L. Granziero F. Pasqual	-	Aggiunte "generalizzazioni" nella tabella per l'Analisi dei Requisiti, aggiunto comando *needspace*.
0.7.0	8/2/2026	G. De Fina	V. Baleanu	-	Modificata redazione verbali (responsabile), aggiunta sezione Gestione e tracciamento dei test, distinzione test funzionali/strutturali, aggiornamento strumento comunicazioni esterne a Microsoft Teams.
0.6.0	5/2/2026	G. De Fina	V. Baleanu	-	Completati Processi di Supporto con Garanzia della qualità, Revisioni congiunte, Audit e Risoluzione dei problemi.
0.5.0	5/2/2026	G. De Fina	V. Baleanu	-	Completati Processi Organizzativi con sezioni su Controllo e monitoraggio, Gestione sprint, Tracciamento attività e Gestione rischi.
0.4.0	11/1/2026	V. Baleanu	F. Venzo	-	Aggiunta processo di Verifica e Validazione.
0.3.0	1/1/2026	V. Baleanu	F. Venzo	-	Aggiunta metriche di qualità di processo e di prodotto.
0.2.0	08/12/2025	C. Libralato	L. Pellizzon	-	Aggiunte a processo di verifica e approvazione dei documenti (sez. 3.1.4, 3.1.5). Aggiunta Sviluppo con Analisi Requisiti ai processi primari (sez. 2.2)
0.1.6	29/11/2025	F. Pasqual	V. Baleanu	-	Aggiunto processo di verifica dei documenti tramite *pull request*
0.1.5	8/11/2025	F. Venzo	L. Granziero	-	Aggiunta tabella attività completate nella struttura dei verbali
0.1.4	30/10/2025	F. Venzo	L. Granziero	-	Modifica struttura tabella attività, modifiche tipografiche
0.1.3	26/10/2025	F. Venzo	L. Granziero	-	Aggiunta sezioni e termini glossario, modifica convenzione date nei nomi dei file

Versione	Data	Autore	Verifica	Approvazione	Descrizione
0.1.2	23/10/2025	F. Venzo	L. Granziero	-	Correzione errori ortografici e aggiunta link
0.1.1	22/10/2025	F. Venzo	L. Granziero	-	Aggiunta sezioni e sotto sezioni
0.1.0	17/10/2025	F. Venzo	L. Granziero	-	Prima stesura

Indice

Elenco delle tabelle

1 Introduzione

1.1 Finalità del documento

Il presente documento intende fissare le linee guida che il gruppo *SnakeByte* si impegna a rispettare ed attuare per perseguire la migliore efficienza ed efficacia nel processo di realizzazione del progetto didattico.

Il documento è strutturato secondo le norme dello Standard ISO/IEC 12207:1995 e segue quanto descritto nel *Regolamento del progetto didattico (A.a. 2025/2026)*. Presenta una descrizione dei *processi* del ciclo di vita del *software* e delle *attività* di cui sono composti. A sua volta, ogni attività, è composta da una serie di procedure metodiche dotate di obiettivi e strumenti ben definiti.

É importante notare che il documento in questione è in continua evoluzione fino al suo ritiro, poiché le norme contenute al suo interno vengono costantemente revisionate, ottimizzate ed aggiornate, seguendo un approccio incrementale.

Ogni attività svolta nell'interesse del progetto didattico e nei suoi materiali è regolamentata precedentemente all'esecuzione della stessa.

1.2 Glossario

Il documento cita alcuni termini la cui definizione può risultare ambigua. Per questo, è possibile consultare il *glossario*$_G$ il quale contiene le definizioni di tali espressioni, che saranno marcate da una lettera G a pedice.

1.3 Riferimenti Normativi

- **Standard ISO/IEC 12207:1995**:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
  (consultato il 30/10/2025);
- **Regolamento del progetto didattico**:
  https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/PD1.pdf
  (consultato il 23/10/2025).

1.4 Riferimenti Informativi

- **Sito dedicato alla documentazione**
  https://snakebyteteam.github.io
  (consultato il 23/10/2025)
- **Glossario**:
  https://snakebyteteam.github.io/glossary.html
  (consultato il 30/10/2025)

2 Processi primari

Le attività che compongono i processi primari considerate in questa sede sono un sottoinsieme proprio delle attività previste dallo Standard ISO/IEC 12207:1995.

2.1 Fornitura

Il processo di fornitura, come specificato nello Standard ISO/IEC 12207:1995, definisce le attività dell'organizzazione che fornisce il prodotto *software* all'acquirente, dalla concezione fino alla consegna del prodotto. Viene istanziato conseguentemente alla redazione della *Valutazione dei capitolati*$_G$.

2.1.1 Attività

Il processo di fornitura si compone delle seguenti attività

- **Avviamento**: revisione delle proposte dei richiedenti. Per i capitolati di maggiore interesse vengono mandate delle comunicazioni via mail per eventuali approfondimenti;
- **Preparazione della risposta**: viene scelto il capitolato per cui ci vuole candidare sulla base delle considerazioni fatte nella fase precedente e viene preparato il documento di **candidatura**.

2.1.2 Documentazione risultante

La documentazione prodotta durante le attività di fornitura, la quale verrà consegnata ai committenti$_G$ quali prof. Tullio Vardanega, prof. Riccardo Cardin e all'azienda proponente è la seguente

- **Valutazioni dei capitolati** contenente
  - Titolo del capitolato e nome dell'azienda proponente;
  - Una breve descrizione del capitolato e dei suoi obiettivi;
  - Considerazioni del gruppo;
- **Dichiarazione degli impegni** contenente
  - Impegni orari e suddivisione dei ruoli;
  - Preventivo dei costi totali del progetto (calcolato secondo il *Regolamento del progetto didattico*);
  - Data prevista di consegna.
- **Lettera di presentazione** contenente
  - Scelta del capitolato;
  - Motivazione della scelta;
  - Riassunto costi complessivi e data prevista di consegna.

2.2 Sviluppo

Il processo di Sviluppo consiste in un'insieme di attività necessarie per la realizzazione del prodotto *software*, queste sono particolarmente orientate verso analisi dei requisiti, design, codifica, integrazione e testing; si tratta infatti del processo primario principale che guida la realizzazione del prodotto *software* per la quasi totalità del ciclo di vita.

2.2.1 Attività

Il processo di sviluppo si compone delle seguenti attività:

- **System requirements analysis**;
- **Software requirements analysis**;
- **Software architectural design**;
- **Software detailed design**;
- **Software coding and testing**;
- **Software qualification testing**;
- **Software installation**;

2.2.2 Documentazione risultante

La documentazione prodotta durante il processo di Sviluppo consiste in:

- ***Diagrammi UML***$_G$: diagrammi realizzati secondo lo standard *UML2.5*$_G$ utili alla definizione dei *casi d'uso*$_G$ all'interno dell'Analisi dei Requisiti.
- **Analisi dei Requisiti**: documento che racchiude i risultati della *System requirements analysis* (requisiti funzionali) e *Software requirements analysis* (requisiti non funzionali).

2.2.3 Analisi dei Requisiti

Il suddetto documento, la cui redazione è affidata al ruolo dell'Analista, è necessario al fine di determinare e tracciare l'insieme di requisiti che l'applicativo deve soddisfare.

L'analisi è strutturata in maniera tale da ricavare i requisiti a partire dai *casi d'uso*, che consistono in una serie di scenari che condividono uno scopo per un utente che interagisce con il sistema. La ricerca dei casi d'uso avviene tramite *brainstorming*$_G$ interno al gruppo e si avvale di *feedback*$_G$ da parte della Proponente.
Il documento è strutturato nel seguente modo:

2.2.3.1 Introduzione

Una breve introduzione descrive le finalità del documento, i Riferimenti Informativi e Normativi.

2.2.3.2 Descrizione del prodotto

In questa sezione viene descritto lo scopo e l'obiettivo principale del prodotto, oltre che le principali funzionalità che deve possedere e l'insieme di utenti a cui è destinato.

2.2.3.3 Elenco *casi d'uso*

I *casi d'uso* vengono elencati seguendo la numerazione associata, sono presentati attraverso un diagramma *UML* e una descrizione testuale in forma tabellare contenente sia le informazioni all'interno del diagramma sia quelle non rappresentabili da esso, tra cui Pre-condizioni e le Post-condizioni.
La struttura è la seguente:

Campo	Descrizione
Attori	Coloro che partecipano attivamente al caso d'uso per raggiungere un preciso obiettivo
Pre-condizioni	Condizioni che devono essere soddisfatte prima dello scenario descritto dal caso d'uso
Post-condizioni	Condizioni che risultano soddisfatte dopo il completamento dello scenario principale del caso d'uso. Se viene completato uno scenario alternativo, saranno soddisfatte le Post-condizioni di quest'ultimo
Scenario principale	Sequenza di passi che l'utente deve seguire per completare il caso d'uso
Scenari alternativi	Scenario divergente dal principale per il verificarsi di una particolare condizione
Estensioni	Casi d'uso ulteriori eseguiti al verificarsi di una particolare condizione nel caso d'uso primario. Modificano Scenario e Post-condizioni
Inclusioni	Casi d'uso ulteriori eseguiti al fine di completare il caso d'uso principale. Vengono eseguiti tutti incondizionatamente.
Generalizzazioni	Casi d'uso più specifici che ereditano comportamento e caratteristiche da un caso d'uso più generale, potendo aggiungere o specializzare ulteriori funzionalità.

Tabella 1: Struttura dei casi d'uso

La creazione della tabella è facilitata dall'utilizzo del template `templateAdR.sty`.
Non tutti i *casi d'uso* necessitano della tabella nella sua interezza, la presenza dei campi: Trigger, Scenari alternativi, Estensioni, Inclusioni e Generalizzazioni dipende dalla situazione.

2.2.3.4 Elenco requisiti
I requisiti sono identificati da un codice univoco e sono associati ai casi d'uso da cui sono stati generati.

3 Processi di supporto

3.1 Documentazione

3.1.1 Struttura generale dei documenti

Le seguenti sezioni illustrano le componenti che ogni documento creato deve avere. Ogni documento deve essere redatto utilizzato il linguaggio LaTeX$_G$, in particolare il file `template.sty` e template forniti nel repository interno.

3.1.1.1 Prima pagina
La prima pagina di ogni documento deve riportare, in ordine di posizionamento dall'alto verso il basso

- Logo del gruppo *SnakeByte*;
- Titolo del documento;
- Il nome e il numero del gruppo *SnakeByte*;
- Nome e cognome di ogni componente e relativo numero di matricola UniPD;
- Informazioni generali del documento quali
  - Versione attuale;
  - Data di creazione della versione;
  - Lo stato attuale;
  - I destinatari del documento.
- Contatto email del gruppo *SnakeByte*.

3.1.1.2 Intestazione
Ogni pagina di qualsiasi documento deve riportare come intestazione

- Nome del gruppo *SnakeByte*;
- Titolo del relativo documento.

3.1.1.3 Registro delle modifiche
Tutti i documenti, interni ed esterni, devono riportare a partire dalla seconda pagina il *registro delle modifiche* sottoforma di tabella, la quale deve riassumere

- Versione del documento;
- Data di creazione della versione;
- Autore della versione;
- Verificatore della versione;
- Approvatore della versione;
- Descrizione riassuntiva delle modifiche alla versione precedente.

Ogni modifica ad un documento scatena la creazione di una nuova versione di esso e quindi la compilazione di una nuova riga, verso il basso, della tabella.

3.1.1.4 Indice

Ogni documento deve riportare l'indice dove saranno elencati i titoli di tutte le sezioni e sottosezioni. Ogni titolo deve essere provvisto di link che porta alla sezione associata all'interno dello stesso documento.

Metodologie

- Questo deve essere fatto tramite il pacchetto `hyperref` fornito dal linguaggio LaTeX$_G$

3.1.2 Struttura dei verbali

I verbali sia interni che esterni, oltre alla struttura descritta nel capitolo *§3.1.1*, devono essere composti dalle seguenti sezioni

- **Informazioni**, contenente
  - Data di svolgimento;
  - Ora inizio;
  - Ora fine;
  - Modalità di svolgimento (Presenza, Online o tramite canali asincroni).
- **Presenze**, contenente, in forma tabellare, le seguenti informazioni
  - Nome e cognome di tutti i membri;
  - Ruolo (ND se non definito);
  - Presenza alla riunione.
- **Ordine del giorno**, con all'interno una lista degli argomenti che vengono trattati all'interno dell'incontro;
- **Approfondimento** degli argomenti ordine del giorno;
- **Decisioni** (Sezione *§3.1.2.1*);
- **Attività da svolgere** (Sezione *§3.1.2.2*);
- **Attività svolte** (Sezione *§3.1.2.3*)

3.1.2.1 Tabella delle decisioni

Per il tracciamento e l'organizzazione di ogni decisione presa collettivamente dal gruppo *SnakeByte*, al termine di ogni verbale, deve essere presente una tabella che riporta le decisioni prese in seguito alla riunione in questione. Per ogni decisione deve essere riportato

- Identificativo alfanumerico della decisione, così composto
  `v{i, e}_AAAA_MM_GG.d<numero_decisione>`;
- Descrizione testuale della decisione presa.

3.1.2.2 Tabella delle attività da svolgere

Per il tracciamento delle attività da eseguire, emerse durante l'incontro trattato dal verbale, deve essere presente una tabella che riporta una lista di queste, riassumendo le seguenti informazioni

- Identificativo alfanumerico dell'attività, così composto
  `v{i, e}_AAAA_MM_GG.a<numero_attività>`;
- Descrizione testuale dell'attività;
- Id GitHub Issue associata all'attività (carattere *"-"* se non presente);
- Assegnatario dell'attività;
- Scadenza di completamento.

3.1.2.3 Tabella delle attività svolte

Per il tracciamento delle attività svolte, deve essere presente una tabella che riporta una lista di queste, riassumendo le seguenti informazioni

- Identificativo alfanumerico dell'attività, come definito nella Sezione *§3.1.2.2*;
- Id GitHub Issue associata all'attività (carattere "-" se non presente);
- Data di completamento.

3.1.3 Redazione dei documenti

La stesura e l'aggiornamento dei documenti è affidata alla persona che ricopre, in quel periodo, il ruolo di *amministratore*$_G$. Le informazioni che verranno usate per redigere i documenti saranno ricavate dalle riunioni e comunicazioni con gli altri componenti del gruppo.

3.1.3.1 Redazione dei verbali

La stesura dei vari verbali (sia esterni che interni) è affidata alla persona che ricopre in quel periodo il ruolo di *responsabile*$_G$. Le informazioni che verranno usate per comporre i documenti saranno ricavate dalle riunioni interne al gruppo, per i verbali interni, e dalle riunioni esterne con l'azienda Proponente, per i verbali esterni.

3.1.3.2 Redazione dell'analisi dei requisiti

Differentemente dagli altri documenti, l'analisi dei requisiti viene redatta dalla persona incaricata in quel momento del ruolo di *analista* (Sezione *§4.1.1.5*).

3.1.4 Verifica dei documenti

Ogni documento dopo essere stato redatto, deve essere verificato, ovvero la correttezza delle informazioni in esso contenute deve essere confermata.

L'azione di verifica dei documenti viene effettuata dalla persona che ricopre, in quel periodo, il ruolo di Verificatore.

3.1.4.1 Processo di verifica tramite *pull request*$_G$

Il processo di verifica dei documenti adotta il meccanismo delle *pull request* offerto dalla piattaforma *GitHub*$_G$. Tale strumento garantisce che nessuna modifica venga integrata nel *branch* principale (`main`) o nel *branch* di sviluppo principale (`develop`) senza un controllo esplicito del verificatore, il quale può approvare il documento o richiedere ulteriori modifiche direttamente all'interno della sezione *pull request* di *GitHub* tramite i commenti.
L'intero ciclo di verifica, incluse eventuali correzioni successive, avviene all'interno di un'unica *pull request*, garantendo la tracciabilità delle discussioni e dei *commit* di modifica.

Di seguito viene definito il *workflow* operativo:

1. **Esecuzione modifiche**
   Il redattore crea un nuovo *branch* denominato `modifica-<nome_documento>` nel suo repository locale (la versione del documento non va inclusa all'interno del nome del *branch* di modifica).
   Per definizione, le modifiche su documenti diversi dovranno essere presentate attraverso *branch* di modifica e pull request diverse.
2. **Apertura della Richiesta**
   Il redattore, completato il lavoro in locale sul *branch* dedicato, effettua il *push* in remoto del *branch* di modifica e apre una nuova *pull request* verso il *branch* `develop`. In questa fase deve:
   - definire come assegnatario e reviewer il verificatore e impostare i *tag*;
   - collegare la richiesta alla relativa *Issue* di progetto.

3. **Verifica**
   Il verificatore riceve la notifica ed esamina le modifiche. A questo punto si presentano due scenari:

   (a) **Esito Positivo:** Se il lavoro è corretto e completo, il verificatore approva la *pull request* e procede al *merge* automatico del *branch* di modifica nel *branch* di `develop`. L'operazione porta all'eliminazione automatica del *branch* di modifica previa attivazione dell'impostazione dedicata di GitHub (Settings>General>Automatically delete head branches).
   Spetta all'autore eliminare manualmente il *branch* di modifica nel suo repository locale.

   (b) **Esito Negativo:** Se sono necessari cambiamenti, il verificatore inserisce commenti *inline* sui punti critici e imposta lo stato della revisione su *Request changes*.

4. **Risoluzione e Nuova Verifica**
   Nel caso siano state richieste modifiche, il redattore:

   (a) viene notificato;

   (b) applica le correzioni in locale ed effettua un nuovo *push* sullo stesso *branch* (aggiornando automaticamente la *pull request* esistente);

   (c) risponde ai commenti o li risolve e richiede una nuova revisione (*Re-request review*);

   (d) in caso di esito di verifica negativo itera il punto 4.

3.1.5 Approvazione dei documenti

Ogni documento, dopo essere stato verificato, deve essere approvato per poter essere rilasciato come documentazione ufficiale del progetto nel *branch* `main`. L'approvazione avviene in maniera simile alla verifica da chi, in quel periodo, ricopre la figura di *responsabile*$_G$.

3.1.5.1 Processo di approvazione tramite *pull request*

I documenti vengono approvati in maniera progressiva al fine di evitare l'accumulo di un carico di lavoro insostenibile per il responsabile quando si raggiunge una baseline. Una volta che tutti i documenti contenuti nel *branch* `develop` sono stati approvati si può procedere con il *merge* del *branch* `develop` verso il *branch* `main`.

Il momento opportuno per sottoporre un documento ad approvazione non è a seguito della verifica ma è definito in base alle tempistiche imposte dalle *baselines*.
L'approvazione di un singolo documento consiste nei seguenti passi:

1. **Aggiornamento versione**
   Il redattore crea un *branch* locale denominato `approvazione-<nome_documento>` (la versione del documento non va inclusa nel nome) in cui modifica **esclusivamente** la versione aumentando l'indice *Major* e azzerando gli altri (X.0.0).

2. **Apertura Richiesta**
   Viene effettuata la *push* e aperta una *pull request* verso il *branch* `develop`. Devono essere impostati i *tag*, l'assegnatario e il *reviewer* devono coincidere con il responsabile.

3. **Approvazione**
   Il responsabile viene notificato e controlla il documento:

   (a) **Esito Positivo:** il documento è approvato con successo, il responsabile approva la *pull request* e il *merge* sul *branch* `develop`.
   Spetta al redattore eliminare manualmente il *branch* di approvazione in locale.

   (b) **Esito Negativo:** il documento contiene errori o ambiguità e richiede variazioni che vengono comunicate come commenti dunque lo stato della revisione viene impostato a "*Request Changes*".

4. **Risoluzione e Nuova Approvazione**
   In caso di esito di approvazione negativo, il redattore:

(a) viene notificato;

(b) applica le modifiche e ripete il processo di verifica (Sezione *§3.1.4.1*);

(c) rinnova la *pull request* esistente per l'approvazione;

(d) in caso di esito negativo itera il punto 4.

3.1.6 Nomenclatura dei documenti

3.1.6.1 Acronimi

Nella nomenclatura e all'interno dei documenti sono utilizzati i seguenti acronimi:

Abbreviazione	Significato
VI	Verbale interno
VE	Verbale esterno
NdP	Norme di Progetto
PB	Product Baseline
RTB	Requirements and Technology Baseline

Tabella 2: Acronimi dei documenti

3.1.6.2 Convenzione

La convenzione in uso per la nomenclatura dei file è la seguente (dove X, Y e Z sono le versioni descritte nella sezione *§3.2.1*)

- **Verbali**: vi_AAAA_MM_GG_vX.Y.Z
- **Generale**: `<ACR.DOC.>`_vX.Y.Z (dove `<ACR.DOC.>` è l'acronimo del documento)

3.2 Gestione della configurazione

La gestione della configurazione è l'insieme delle attività volte a identificare, tracciare e controllare le modifiche apportate a qualsiasi elemento nel progetto.

Secondo lo Standard ISO/IEC 12207:1995, le azioni di gestione della configurazione devono controllare le modifiche e i rilasci degli elementi, registrare e segnalare lo stato degli elementi e le richieste di modifica, garantire la completezza, la coerenza e la correttezza degli elementi, e controllare l'immagazzinamento, la movimentazione e la consegna degli elementi.

3.2.1 Numeri di versione documenti

La convenzione per il numero di versione dei documenti è il formato X.Y.Z (*MAJOR.MINOR.PATCH*) ogni documento parte dalla prima versione 0.1.0

- Ogni modifica minore (*patch version*), come correzione di errori grammaticali o aggiunta d'informazioni meno significative, fa avanzare Z di una unità;
- Ogni modifica maggiore (*minor version*), come aggiunta di sezioni o modifiche sostanziali fa avanzare Y di una unità;
- La cifra X avanza solamente quando l'approvazione da parte del responsabile termina con successo (*major version*).

Ogni avanzamento delle cifre X e Y riportano le cifre alla loro destra a 0.

3.2.2 Repository

I repository creati per la gestione della configurazione sono

- **MVP**: repository contenente il Minimum Viable Product del progetto;
- **snakebyte.github.io**: repository contenente il codice sorgente del sito web dedicato alla documentazione del progetto e la documentazione stessa.

3.2.2.1 Strumenti

Gli strumenti adottati per la gestione della configurazione e del versionamento dei file di progetto sono:

- **Git**$_G$: Version Control System distribuito e OpenSource
- **Github**: servizio di *hosting* per progetti *software* e implementazione di Git.

3.3 Verifica

Il processo di verifica garantisce che i prodotti realizzati siano conformi ai requisiti e agli standard di qualità definiti nel *Piano di Qualifica* (sezione 4), assicurando la correttezza e la completezza di ciascun prodotto. Le attività di verifica sono assegnate al membro del gruppo che, nel periodo di riferimento, ricopre il ruolo di verificatore. Tale figura è responsabile della registrazione degli esiti delle verifiche e, qualora vengano riscontrate non conformità rispetto ai vincoli stabiliti, della richiesta di opportune modifiche e correzioni.
Il gruppo *SnakeByte* applica il processo di verifica ai seguenti prodotti:

- **Documenti**: ogni documento prodotto è sottoposto a verifica seguendo il procedimento descritto nella sezione **??** e prevede controlli di conformità formale, correttezza ortografica e contenutistica;
- **Codice**: il processo di verifica del codice sarà definito e implementato nella fase successiva al raggiungimento della *Requirements and Technology Baseline*, quando saranno disponibili i componenti *software* da verificare. Tale processo includerà sia tecniche di analisi statica che dinamica, come descritto nelle sezioni successive.

3.3.1 Strumenti

Lo strumento adottato per la verifica dei prodotti è *pull request* della piattaforma *GitHub*: come viene dettagliatamente spiegato nella sezione §**??**, questo servizio consente di effettuare attività di controllo e di richiedere eventuali correzioni o miglioramenti degli artefatti di progetto, velocizzando il processo di verifica.

3.3.2 Analisi statica

L'analisi statica è una tipologia di verifica che non richiede l'esecuzione del prodotto. L'attività di verifica consiste nel controllo degli artefatti di progetto mediante tecniche di ispezione e analisi, quali il controllo sintattico, il controllo semantico e la verifica della conformità agli standard adottati. Attraverso tali attività è possibile individuare errori formali, incongruenze, violazioni delle convenzioni e potenziali difetti, sia nel codice che nella documentazione.
Le principali tecniche di analisi statica sono:

- **Walkthrough**: l'autore del documento o del codice spiega la logica e le scelte implementative al verificatore, il quale fornisce un *feedback*. Questa tecnica richiede una discussione attiva e sincrona tra autore e verificatore;
- **Inspection**: il verificatore esamina il materiale utilizzando una lista di controllo (*checklist*), identificando anomalie e difetti in modo sistematico. Successivamente, durante una riunione, vengono discussi i difetti rilevati.

Tra le due possibili strategie, il gruppo *SnakeByte* ha preferito implementare *Inspection* poiché garantisce un approccio più strutturato e ripetibile, riducendo la dipendenza dalla presenza simultanea di autore e verificatore. L'utilizzo di *checklist* predefinite consente inoltre di standardizzare il processo di verifica, assicurando che tutti gli aspetti critici vengano sistematicamente esaminati.

3.3.3 Analisi dinamica

L'analisi dinamica è una tipologia di verifica che richiede l'esecuzione del prodotto software in un ambiente controllato. L'attività di verifica consiste nell'esecuzione di test specifici per valutare la conformità ai requisiti funzionali e non funzionali e individuare eventuali difetti che si manifestano durante l'esecuzione. L'analisi dinamica permette di rilevare problematiche legate al comportamento a *runtime*, impossibili dunque da individuare con l'analisi statica.
Per garantire l'efficacia del processo di verifica, i test devono possedere le seguenti caratteristiche:

- **Ripetibilità**: i test devono produrre risultati consistenti quando eseguiti più volte nelle medesime condizioni;
- **Automatizzazione**: l'esecuzione automatica dei test consente di ridurre i tempi di verifica, minimizzare gli errori umani e facilitare l'integrazione nel processo di sviluppo continuo;
- **Determinismo**: ogni test deve avere un risultato atteso chiaramente definito, permettendo una valutazione oggettiva del successo o del fallimento dell'esecuzione;
- **Indipendenza**: ciascun test deve essere autonomo e non dipendere da altri test o dall'ambiente di esecuzione.

Il gruppo *SnakeByte* implementerà le seguenti tipologie di test:

- Test di unità;
- Test di integrazione;
- Test di sistema;
- Test di regressione.

3.3.3.1 Test di unità

I test di unità verificano il corretto funzionamento delle singole unità di codice (funzioni, metodi o classi). Ogni unità viene testata indipendentemente dalle altre componenti per garantire che produca i risultati attesi per diversi input, inclusi casi limite ed eccezionali. I test di unità possono essere suddivisi in due categorie principali:

- **Test Funzionali (Black-box)**: verificano che l'unità produca i risultati attesi per diversi input senza analizzare la logica interna. Per ciascuna unità vengono considerati:
  - valori non ammissibili inferiori;
  - valori estremi inferiori ammissibili;
  - valori ammissibili;
  - valori estremi superiori ammissibili;
  - valori non ammissibili superiori.
- **Test Strutturali (White-box)**: valutano la corretta esecuzione della logica interna dell'unità, assicurandosi che tutti gli statement rilevanti del codice siano eseguiti almeno una volta.

3.3.3.2 Test di integrazione

I test di integrazione verificano la corretta interazione tra le diverse componenti del sistema. L'obiettivo è individuare difetti nelle comunicazioni tra moduli, garantendo che le componenti collaborino correttamente. L'integrazione può avvenire seguendo una strategia *top-down*, che prevede l'integrazione progressiva a partire dai moduli con maggiori dipendenze d'uso, oppure una strategia *bottom-up*, che parte dai moduli con minori dipendenze d'uso per poi risalire verso quelli superiori.
Per facilitare i test di integrazione e di unità, il gruppo *SnakeByte* utilizza tecniche di *test doubling* quali:

- ***Stub***: componenti semplificati che forniscono risposte predefinite alle chiamate effettuate durante i test, simulando il comportamento di dipendenze non ancora implementate o esterne;
- ***Mock***: oggetti che simulano il comportamento di componenti reali e permettono di verificare che le interazioni tra moduli avvengano correttamente, registrando le chiamate ricevute e validando il loro utilizzo.

L'uso di *stub* e *mock* consente di isolare le unità sotto test e di verificare il comportamento del codice indipendentemente dalle dipendenze esterne.

3.3.3.3 Test di sistema

I test di sistema verificano il comportamento dell'intero sistema integrato, valutandone la conformità ai requisiti funzionali e non funzionali. Analizzano dunque il sistema dal punto di vista dell'utente finale, senza considerare la struttura interna del codice. Attraverso tali test si verifica che il sistema risponda correttamente agli scenari d'uso previsti e soddisfi le aspettative definite in fase di analisi.

3.3.3.4 Test di regressione

I test di regressione hanno lo scopo di garantire che modifiche o correzioni apportati al sistema non introducano nuovi difetti o compromettano funzionalità precedentemente verificate. Essi consistono nella riesecuzione di test già validati, al fine di garantire la stabilità e l'affidabilità complessiva del sistema nel tempo.

3.3.4 Nomenclatura dei test

Ogni test da svolgere è identificato da un codice univoco nella forma:

T[Tipo-test][X]

dove:

- **[Tipo-test]** rappresenta la tipologia del test (**U** = Unità, **I** = Integrazione, **S** = Sistema o **R** = Regressione);
- **[X]** rappresenta un numero intero progressivo.

3.3.5 Stato dei test

Ogni test, in base all'esito ottenuto, può avere uno dei seguenti stati:

- **NI**: il test non è ancora stato implementato;
- **S**: il test ha avuto successo;
- **F**: il test ha fallito.

3.4 Gestione e tracciamento dei test

I test sono una fase obbligatoria dello sviluppo del software: servono a controllare, con procedura sistematica, che il sistema rispetti i requisiti funzionali e non funzionali stabiliti in sede di analisi.
Ogni test è collegato a uno o più requisiti e verifica che il codice li implementi correttamente. Un requisito si intende soddisfatto solo quando tutti i test a lui relativi terminano con esito positivo.
Durante l'intero ciclo di sviluppo, i test vengono eseguiti: mentre si scrivono le funzionalità, dopo ogni modifica o correzione tramite test di regressione e infine nella fase di validazione e collaudo.
Ripetere i test permette di rilevare subito i difetti introdotti mentre il software evolve e di mantenere il prodotto stabile.
Il verificatore definisce, esegue e valuta i test, esamina i risultati e segnala le anomalie. I programmatori aiutano creando i test di unità e correggendo i difetti trovati.

3.5 Validazione

Il processo di validazione ha lo scopo di confermare che il prodotto finale soddisfi le aspettative e i bisogni della Proponente, verificando che il sistema realizzato sia quello effettivamente richiesto.
La validazione viene effettuata attraverso test di accettazione condotti in presenza della Proponente, utilizzando *casi d'uso* e scenari rappresentativi delle esigenze reali. L'esito positivo della validazione costituisce il presupposto per il rilascio del prodotto.

3.5.1 Collaudo

Il collaudo rappresenta l'attività conclusiva del processo di validazione e consiste nella verifica formale del soddisfacimento dei requisiti concordati. Durante il collaudo vengono eseguiti test di accettazione che dimostrano la conformità del sistema alle specifiche funzionali e non funzionali definite.
Il gruppo *SnakeByte* pianificherà le attività di collaudo in modo da:

- Predisporre un ambiente di test rappresentativo dello scenario d'uso reale del sistema;
- documentare in modo completo e tracciabile l'esito di ogni test di accettazione svolto;
- raccogliere *feedback* per eventuali miglioramenti o correzioni.

3.6 Garanzia della qualità

Il processo di garanzia della qualità assicura che i prodotti e i processi del progetto siano conformi agli standard e ai requisiti definiti. Tale processo prevede attività sistematiche di controllo volte a garantire che i criteri di qualità stabiliti nel *Piano di Qualifica* vengano rispettati durante l'intero progetto.

3.6.1 Obiettivi

Le attività di garanzia della qualità hanno lo scopo di:

- verificare la conformità dei prodotti agli standard di qualità adottati;
- monitorare l'applicazione dei processi definiti nelle *Norme di Progetto*;
- identificare non conformità e proporre azioni correttive;
- garantire la tracciabilità delle verifiche effettuate.

3.6.2 Obiettivi del verificatore

Il verificatore è responsabile di:

- controllare la conformità dei prodotti rispetto ai criteri di qualità definiti;
- garantire completezza e sistematicità delle verifiche;
- registrare gli esiti delle attività di *Quality Assurance*;
- segnalare anomalie e richiedere interventi correttivi quando necessario.

3.6.3 Metriche di qualità

Le metriche utilizzate per valutare la qualità di processi e prodotti sono definite dettagliatamente nelle sezioni §?? e §??. Il monitoraggio costante di tali metriche consente di identificare tempestivamente eventuali criticità e di intraprendere azioni correttive.

3.7 Revisioni congiunte

Il processo di revisioni congiunte prevede incontri periodici con la Proponente per presentare lo stato di avanzamento del progetto, raccogliere *feedback* e validare le scelte implementative adottate.

3.7.1 SAL - Stato Avanzamento Lavori

I SAL (*Stato Avanzamento Lavori*) rappresentano momenti formali di revisione congiunta con la Proponente, durante i quali vengono presentati:

- progressi realizzati rispetto alla pianificazione;
- piccole dimostrazioni delle funzionalità implementate;
- problematiche emerse e soluzioni proposte;
- pianificazione delle attività successive.

Ogni SAL viene documentato attraverso un verbale esterno che traccia discussioni, decisioni e *feedback* ricevuti. I SAL sono organizzati con cadenza concordata con la Proponente e rappresentano un'opportunità per validare incrementalmente il lavoro svolto e allinearlo alle aspettative.

3.7.2 Strumenti

Gli strumenti utilizzati per le revisioni congiunte sono:

- **Microsoft Teams**: piattaforma per incontri *online* con la Proponente;
- **Verbali esterni**: documenti che tracciano contenuti e decisioni emerse durante i SAL;
- ***Demo***: eventualmente, presentazioni del prodotto in sviluppo per raccogliere *feedback* operativo.

3.8 *Audit*

Il processo di *audit* consiste in verifiche formali e sistematiche condotte per accertare che i prodotti e i processi siano conformi ai requisiti definiti e agli standard adottati. Gli *audit* vengono svolti in corrispondenza delle *milestone* di progetto per garantire la completezza e la qualità degli artefatti prima delle consegne ufficiali.

3.8.1 *Audit* di *baseline*

Prima del raggiungimento di ogni *baseline* (*Requirements and Technology Baseline* e *Product Baseline*), il responsabile conduce un *audit* formale per verificare:

- completezza della documentazione richiesta;
- conformità dei documenti rispetto ai requisiti del capitolato e alle aspettative della Proponente;
- tracciabilità tra requisiti, *casi d'uso*, test e implementazione;
- rispetto degli standard di qualità definiti nel *Piano di Qualifica*.

3.8.2 Responsabilità

Il responsabile è incaricato di:

- pianificare e condurre gli *audit* in corrispondenza delle *milestone*;
- verificare la conformità degli artefatti rispetto ai criteri stabiliti;
- documentare gli esiti degli *audit* e le eventuali non conformità rilevate;
- richiedere azioni correttive in caso di difetti o mancanze.

Gli *audit* garantiscono che ogni consegna sia conforme agli standard di progetto e rappresentino un controllo di qualità finale prima del rilascio ufficiale.

3.9 Risoluzione dei problemi

Il processo di risoluzione dei problemi definisce le modalità con cui il gruppo *SnakeByte* identifica, traccia e risolve anomalie, difetti e richieste di miglioramento emerse durante le attività progettuali.

3.9.1 Tracciamento tramite GitHub Issues

Il gruppo utilizza il sistema di *Issue* di $GitHub_G$ per la gestione centralizzata dei problemi. Ogni segnalazione viene tracciata attraverso una *Issue* che contiene:

- titolo e descrizione del problema;
- classificazione tramite *label*;
- assegnatario responsabile della risoluzione;
- *milestone* di riferimento e priorità;
- stato di avanzamento e commenti di aggiornamento.

3.9.2 Workflow di risoluzione

Il processo di risoluzione segue i seguenti passi:

1. **Segnalazione**: un membro del gruppo identifica un problema e crea una *Issue* su *GitHub*, fornendo una descrizione chiara e dettagliata;
2. **Analisi**: il responsabile o l'assegnatario analizza il problema, valuta priorità e impatto;
3. **Assegnazione**: il problema viene assegnato a un membro del gruppo con le competenze appropriate;
4. **Risoluzione**: l'assegnatario implementa la soluzione seguendo il processo di verifica tramite *pull request* descritto nella Sezione *§3.3*;
5. **Verifica**: il verificatore controlla la correttezza della soluzione;
6. **Chiusura**: una volta verificata la risoluzione, la *Issue* viene chiusa e collegata al *commit* o alla *pull request* risolutiva.

4 Processi organizzativi

4.1 Processo di gestione

Il processo di gestione contiene le attività e i compiti generici che possono essere impiegati da qualsiasi ruolo che debba gestire i rispettivi processi.

4.1.1 Pianificazione

4.1.1.1 Assegnazione delle responsabilità

Per tutta la durata del progetto, per ragioni formative, i membri del gruppo ricopriranno a rotazione 6 ruoli fondamentali nel campo dell'ingegneria del *software*. Ogni componente del gruppo *SnakeByte* dovrà ricoprire almeno una volta tutti i seguenti ruoli:

- Responsabile (Sezione *§4.1.1.2*)
- Amministratore (Sezione *§4.1.1.3*)
- Progettista (Sezione *§4.1.1.4*)
- Analista (Sezione *§4.1.1.5*)
- Verificatore (Sezione *§4.1.1.6*)
- Programmatore (Sezione *§4.1.1.7*)

La rotazione dei ruoli avviene all'inizio di ogni $sprint_G$.

4.1.1.2 Responsabile

Il responsabile (*Project Manager*) governa il *team* e rappresenta il progetto verso l'esterno. Deve avere conoscenze e capacità tecniche per valutare rischi, scelte e alternative.

I compiti principali del responsabile sono:

- prendere scelte nell'interesse del gruppo;
- approvare il lavoro realizzato dagli altri ruoli;
- pianificare le attività e gestire le risorse necessarie;
- coordinare e relazionarsi con l'esterno.

Le persone incaricate come responsabile devono essere una per periodo.

4.1.1.3 Amministratore

L'amministratore di sistema (*Sysadmin*) definisce, controlla e manutiene l'ambiente informatico di lavoro.

- Definisce, seleziona e mette in opera le risorse informatiche a supporto delle *Norme di Progetto*;
- Gestisce le segnalazioni (*ticket*) sul non funzionamento dell'infrastruttura.

Le persone incaricate come amministratore devono essere una per periodo.

4.1.1.4 Progettista

Il progettista ha competenze tecniche e tecnologiche aggiornate e per questo ha il ruolo di determinare le scelte realizzative del prodotto. Segue il progetto durante la fase di sviluppo *software* ma non durante la sua manutenzione.

I principali compiti del progettista sono:

- ideare l'architettura del prodotto in modo da ottenere la migliore efficienza ed efficacia possibile;
- supervisionare la fase di codifica supportando i programmatori.

La realizzazione del progetto a regola d'arte richiede la presenza di più progettisti, i quali devono aver la possibilità di confrontarsi tra di loro in modo da prendere le scelte attuative in maniera critica.

4.1.1.5 Analista

L'analista conosce il dominio del problema ed ha competenze avanzate sull'analisi dei requisiti. Ha molta influenza sul successo del prodotto. Non segue il progetto fino alla consegna, ma il suo intervento è richiesto solo nei momenti di bisogno.

I suoi compiti principali sono

- Analizzare i requisiti e il dominio applicativo;
- Definire fattibilità e obiettivi delle attività;
- Redigere il documento *Analisi dei requisiti*.

Più membri possono essere occupati in questo ruolo contemporaneamente.

4.1.1.6 Verificatore

Il compito principale del verificatore è analizzare la conformità agli obiettivi del lavoro svolto dagli altri ruoli presenti nel progetto. Ha competenze tecniche e profonda conoscenza delle *Norme di Progetto*.

I principali compiti del verificatore sono

- Verificare che gli artefatti *software* e di documentazione siano aderenti alle *Norme di Progetto* e agli obiettivi;
- Segnalare eventuali errori da correggere;
- Redigere test di verifica e di aderenza ai requisiti per i prodotti *software*.

Più membri possono essere occupati in questo ruolo contemporaneamente.

4.1.1.7 Programmatore

Il programmatore è la figura responsabile della realizzazione e manutenzione del prodotto *software*. Ha competenze tecniche ampie ma deleghe limitate.

I principali compiti del programmatore sono

riunione in prese

- Scrivere codice secondo le specifiche del *progettista* e che persegue gli obiettivi e requisiti definiti dall'*analista*;
- Redigere la documentazione tecnica (*Manuale*) del prodotto.

É il ruolo in cui vengono occupati più membri contemporaneamente.

4.1.2 Coordinamento

4.1.2.1 Riunione interna fissata

Il gruppo *SnakeByte* ha scelto di fissare un giorno all'interno della settimana lavorativa in cui si svolge una riunione in presenza o asincrona a cui prendono parte i componenti del gruppo. La partecipazione di tutti i membri è desiderabile ma non tassativa. Inoltre nel caso di impossibilità della maggior parte del gruppo la riunione verrà annullata. Il giorno attuale in cui si tiene la riunione è il Lunedì alle ore 12-15 circa in poi. Altre eventuali riunioni possono essere fissate tramite confronto e accordo tra i componenti del gruppo su giorno e ora.

4.1.2.2 Comunicazioni interne

Per le comunicazioni interne, il gruppo *SnakeByte* ha individuato come mezzo per la trasmissione di informazioni in formato sincrono e asincrono la piattaforma *Discord*$_G$.

4.1.2.3 Comunicazioni esterne

Per le comunicazioni esterne, il gruppo *SnakeByte* ha individuato come mezzo per la trasmissione di informazioni in formato asincrono i messaggi email, esclusivamente tramite l'indirizzo ufficiale del gruppo (*snakebyteteam@gmail.com*). Invece, per le comunicazioni sincrone, si è optato per l'applicazione di videoconferenze *Microsoft Teams*.

4.1.3 Controllo e monitoraggio

Il processo di controllo e monitoraggio ha lo scopo di garantire che le attività di progetto procedano secondo la pianificazione stabilita e che le risorse vengano utilizzate in modo efficiente ed efficace. Il gruppo *SnakeByte* adotta strumenti e pratiche per tracciare l'avanzamento del lavoro, gestire le attività e monitorare le ore impiegate da ciascun membro del gruppo.

4.1.3.1 Gestione degli *sprint*

Il gruppo *SnakeByte* adotta il modello di sviluppo *agile*$_G$ basato su *sprint*$_G$ di durata fissa pari a due settimane. Ogni *sprint* si articola nelle seguenti fasi:

- ***Sprint Planning***: all'inizio di ogni *sprint*, il gruppo si riunisce per definire gli obiettivi da raggiungere, selezionare le attività da svolgere e stimare l'impegno orario necessario. Durante questa fase vengono assegnati i ruoli ai membri del gruppo secondo il principio di rotazione e viene aggiornato il documento di pianificazione condiviso;
- **Sviluppo**: durante lo *sprint*, ciascun membro svolge le attività assegnate secondo il ruolo ricoperto. L'avanzamento viene monitorato attraverso gli strumenti di tracciamento descritti nelle sezioni successive;
- ***Sprint Review***: al termine dello *sprint*, il gruppo verifica il lavoro completato, valuta il raggiungimento degli obiettivi prefissati e raccoglie *feedback* dalla Proponente quando previsto;
- ***Sprint Retrospective***: conclusa la *review*, il gruppo analizza criticamente il processo seguito, identifica problematiche emerse e propone azioni di miglioramento per gli *sprint* successivi. Gli esiti della retrospettiva vengono documentati nel *Piano di Progetto*.

4.1.3.2 Tracciamento delle attività

Il gruppo *SnakeByte* utilizza *GitHub Projects*$_G$ come strumento principale per la gestione e il tracciamento delle attività di progetto. Ogni attività viene rappresentata come una *Issue*$_G$ all'interno del repository di progetto e viene associata a:

- Titolo descrittivo dell'attività;
- Assegnatario o assegnatari responsabili del completamento;
- Etichette (*label*$_G$) che classificano la tipologia di attività (documentazione, sviluppo, verifica, etc.);
- *Milestone*$_G$ di riferimento che identifica lo *sprint* o la fase di progetto;
- Scadenza prevista per il completamento.

Le *Issue* vengono organizzate all'interno di una *dashboard*$_G$ di progetto che visualizza lo stato di avanzamento attraverso le seguenti colonne:

- **Backlog**: attività identificate ma non ancora pianificate;
- **Ready**: attività pianificate per lo *sprint* corrente;
- **In Progress**: attività in corso di svolgimento;
- **In Review**: attività completate in attesa di verifica;
- **Done**: attività verificate e concluse.

Ogni membro del gruppo è tenuto ad aggiornare lo stato delle proprie *Issue* con regolarità, spostando le *Issue* tra le colonne in base all'avanzamento del lavoro e commentando eventuali difficoltà o variazioni rispetto alla pianificazione.

4.1.3.3 Pianificazione e monitoraggio delle ore

Per la pianificazione dei ruoli e il monitoraggio delle ore lavorate, il gruppo *SnakeByte* utilizza un documento condiviso *Google Sheets*$_G$ strutturato come segue:

- **Foglio di pianificazione degli *sprint***: contiene, per ogni *sprint*, l'assegnazione dei ruoli a ciascun membro del gruppo e il numero di ore preventivate per ogni ruolo;
- **Fogli di consuntivo**: per ogni *sprint* viene creato un foglio dedicato in cui ciascun membro registra le ore effettivamente lavorate, specificando il ruolo ricoperto e le attività svolte. I dati raccolti vengono utilizzati per calcolare le metriche di progetto definite nel *Piano di Qualifica* e per confrontare l'impegno preventivato con quello reale;

I dati contenuti nel documento condiviso vengono periodicamente sincronizzati con il *Piano di Progetto* per garantire la coerenza tra pianificazione, esecuzione e reportistica.

4.1.3.4 Strumenti

Gli strumenti adottati per il controllo e il monitoraggio delle attività di progetto sono:

- **GitHub Projects**: piattaforma integrata con *GitHub*$_G$ per la gestione delle *Issue* e la visualizzazione dell'avanzamento del progetto attraverso *board* personalizzabili;
- **Google Sheets**: strumento di fogli di calcolo condivisi per la pianificazione dei ruoli e il tracciamento delle ore lavorate da ciascun membro del gruppo.

4.1.4 Gestione dei rischi

Il processo di gestione dei rischi ha lo scopo di identificare, analizzare e mitigare i potenziali eventi che potrebbero compromettere il successo del progetto. Il gruppo *SnakeByte* adotta un approccio proattivo alla gestione dei rischi, svolgendo attività di identificazione continua durante l'intero ciclo di vita del progetto.

La gestione operativa dei rischi, comprensiva della classificazione, dell'analisi probabilistica e delle strategie di mitigazione, è documentata in modo esaustivo nel *Piano di Progetto*, dove vengono dettagliati:

- Le categorie di rischio considerate (tecnologici, di comunicazione, organizzativi, interpersonali);
- La valutazione di probabilità e impatto per ciascun rischio identificato;
- Le strategie di prevenzione e mitigazione adottate;
- Il monitoraggio dello stato dei rischi durante gli *sprint*.

Durante le riunioni di *Sprint Retrospective*, il gruppo analizza i rischi manifestatisi durante lo *sprint* concluso e valuta l'efficacia delle strategie di mitigazione applicate, proponendo eventuali azioni correttive per gli *sprint* successivi.

5 Metriche di qualità di processo

In questa sezione vengono definite le metriche di qualità adottate per monitorare e valutare in modo quantitativo l'efficacia e l'efficienza dei processi utilizzati durante lo sviluppo del progetto.
Ogni metrica di qualità di processo è identificata da un codice univoco nella forma *MPCX*, dove *MPC* è l'acronimo di *Metrica di ProCesso* e *X* rappresenta un numero intero progressivo.

5.1 Processi primari

5.1.1 Fornitura

5.1.1.1 MPC1 - Planned Value (PV)

Codice	MPC1
Formula	$PV_k = BAC \cdot$ Percentuale completamento pianificato allo sprint k Dove: • BAC = Budget totale previsto (*Budget At Completion*); • k = Numero dello sprint di riferimento.
Descrizione	Valore pianificato del lavoro che, secondo il piano, dovrebbe essere raggiunto al termine dello *sprint* k. Il *Planned Value* viene calcolato al termine di ciascuno *sprint* per monitorare l'avanzamento del progetto rispetto alla pianificazione.
Utilità	Fornisce un punto di riferimento temporale per valutare se il progetto sta procedendo secondo la schedulazione prevista, permettendo di identificare tempestivamente eventuali ritardi o anticipi rispetto al piano originale.

Tabella 3: MPC1 - Planned Value (PV)

5.1.1.2 MPC2 - Earned Value (EV)

Codice	MPC2
Formula	$EV_k = BAC \cdot$ Percentuale completamento effettivo allo sprint k Dove: • BAC = Budget totale previsto (*Budget At Completion*); • k = Numero dello sprint di riferimento.
Descrizione	Valore effettivo del lavoro che è stato raggiunto al termine dello *sprint* k. L'*Earned Value* viene calcolato al termine di ciascuno sprint per monitorare il progresso reale del progetto rispetto al piano.
Utilità	Confronta il lavoro completato sia con i costi sostenuti (tramite CPI) che con la pianificazione temporale (tramite SPI), costituendo quindi la base per tutte le analisi di performance.

Tabella 4: MPC2 - Earned Value (EV)

5.1.1.3 MPC3 - Actual Cost (AC)

Codice	MPC3
Formula	$AC_k = \sum_{i=1}^{k}$ Costo sostenuto nello sprint i Dove: • k = Numero dello sprint di riferimento.
Descrizione	Rappresenta il costo effettivo, cioè il costo reale sostenuto per il lavoro svolto fino ad un determinato *sprint*. Viene calcolato alla fine di ogni *sprint*.
Utilità	Permette di confrontare le spese effettive con il budget pianificato e con il valore prodotto, consentendo di rilevare tempestivamente sforamenti di budget o inefficienze nell'utilizzo delle risorse.

Tabella 5: MPC3 - Actual Cost (AC)

5.1.1.4 MPC4 - Cost Performance Index (CPI)

Codice	MPC4
Formula	$$CPI_k = \frac{EV_k}{AC_k}$$ Dove: • EV_k = *Earned Value* allo *sprint* k (si veda §**??**); • AC_k = *Actual Cost* allo *sprint* k (si veda §**??**); • k = Numero dello sprint di riferimento.
Descrizione	Rappresenta l'efficienza con cui il budget viene utilizzato e corrisponde al rapporto tra il valore del lavoro raggiunto e il costo reale sostenuto, al termine dello *sprint* k. Interpretazione dei valori: • CPI $>$ 1: il progetto è sotto il budget; • CPI $<$ 1: il progetto è sopra il budget; • CPI = 1: il progetto è in linea con il budget pianificato.
Utilità	Quantifica l'efficienza nell'uso delle risorse e prevede se il progetto terminerà entro il budget previsto. Un monitoraggio costante del CPI permette di identificare tempestivamente inefficienze e di adottare misure correttive per evitare sforamenti significativi.

Tabella 6: MPC4 - Cost Performance Index (CPI)

5.1.1.5 MPC5 - Schedule Performance Index (SPI)

Codice	MPC5
Formula	$$SPI_k = \frac{EV_k}{PV_k}$$ Dove: • EV_k = *Earned Value* allo *sprint* k (si veda §**??**); • PV_k = *Planned Value* allo *sprint* k (si veda §**??**); • k = Numero dello sprint di riferimento.
Descrizione	Rappresenta l'efficienza con cui il lavoro viene completato rispetto alla pianificazione e corrisponde al rapporto tra il valore del lavoro completato e il valore pianificato del lavoro fino ad un determinato *sprint*. Interpretazione dei valori: • SPI $>$ 1: il progetto è in anticipo rispetto alla pianificazione; • SPI $<$ 1: il progetto è in ritardo rispetto alla pianificazione; • SPI = 1: il progetto procede secondo i tempi previsti. Un SPI costantemente inferiore a 1 segnala la necessità di rivedere la pianificazione o di allocare risorse aggiuntive per recuperare i ritardi accumulati.
Utilità	Permette di valutare se il gruppo ha rispettato le scadenze previste e permette di stimare se il progetto terminerà nei tempi pianificati.

Tabella 7: MPC5 - Schedule Performance Index (SPI)

5.1.1.6 MPC6 - Estimate At Completion (EAC)

Codice	MPC6
Formula	$$EAC_k = AC_k + (BAC - EV_k)$$ Dove: • AC_k = *Actual Cost* allo *sprint* k (si veda §**??**); • BAC = Budget totale previsto (*Budget At Completion*); • EV_k = *Earned Value* allo *sprint* k (si veda §**??**); • k = Numero dello sprint di riferimento.
Descrizione	Stima del costo totale finale del progetto al completamento, calcolata come la somma tra i costi sostenuti fino al momento della misurazione e il budget rimanente.
Utilità	La metrica EAC fornisce una previsione realistica del budget necessario considerando il lavoro effettivo già eseguito. Consente inoltre di identificare tempestivamente potenziali sforamenti di budget, permettendo di applicare azioni correttive prima che gli scostamenti diventino critici.

Tabella 8: MPC6 - Estimate At Completion (EAC)

5.1.1.7 MPC7 - Estimate To Complete (ETC)

Codice	MPC7
Formula	$$ETC_k = EAC_k - AC_k$$ Dove: • EAC_k = *Estimate at Completion* allo *sprint* k (si veda §**??**); • AC_k = *Actual Cost* allo *sprint* k (si veda §**??**); • k = Numero dello sprint di riferimento.
Descrizione	Stima del costo necessario per completare il lavoro rimanente del progetto, calcolata al termine di un determinato *sprint*. Rappresenta la proiezione delle risorse economiche ancora necessarie per portare a termine tutte le attività non ancora completate e viene calcolata come la differenza tra il costo finale stimato e il costo sostenuto.
Utilità	Permette di valutare se le risorse economiche residue sono sufficienti per completare il progetto.

Tabella 9: MPC7 - Estimate To Complete (ETC)

5.1.2 Sviluppo

5.1.2.1 MPC8 - Requirements Stability Index (RSI)

Codice	MPC8
Formula	$$RSI = \frac{R_{iniziali} - (R_{modificati} + R_{cancellati})}{R_{iniziali} + R_{aggiunti}} \cdot 100$$ Dove: • $R_{iniziali}$ = Numero di requisiti definiti all'inizio del progetto; • $R_{modificati}$ = Numero di requisiti iniziali che hanno subito modifiche; • $R_{cancellati}$ = Numero di requisiti iniziali che sono stati rimossi; • $R_{aggiunti}$ = Numero di nuovi requisiti aggiunti durante il progetto.
Descrizione	Indice che misura la stabilità dei requisiti del progetto durante il suo ciclo di vita, quantificando la percentuale di requisiti che non hanno subito modifiche, aggiunte o cancellazioni rispetto al totale. Interpretazione dei valori: • RSI = 100%: requisiti stabili, eccellente analisi iniziale; • 80% ≤ RSI < 100%: stabilità accettabile, alcune modifiche sono normali; • 50% ≤ RSI < 80%: elevata volatilità, problemi nella definizione dei requisiti; • RSI < 50%: instabilità critica.
Utilità	Valuta la qualità dell'analisi iniziale dei requisiti e identifica problemi nella loro definizione.

Tabella 10: MPC8 - Requirements Stability Index (RSI)

5.2 Processi di supporto

5.2.1 Documentazione

5.2.1.1 MPC9 - Indice di Gulpease (IG)

Codice	MPC9
Formula	$$IG = 89 + \frac{300 \cdot N_{frasi} - 10 \cdot N_{lettere}}{N_{parole}}$$ Dove: • N_{frasi} = Numero di frasi nel testo; • $N_{lettere}$ = Numero di lettere nel testo; • N_{parole} = Numero totale di parole nel testo.
Descrizione	Indice che misura la leggibilità di un testo in lingua italiana, basato sulla lunghezza delle parole e delle frasi. Interpretazione dei valori: • IG $\geq$ 80: testo molto facile, comprensibile da studenti di scuola elementare; • 60 $\leq$ IG $<$ 80: testo facile, comprensibile da studenti di scuola media; • 40 $\leq$ IG $<$ 60: testo abbastanza difficile, comprensibile da studenti di scuola superiore; • IG $<$ 40: testo difficile, richiede istruzione universitaria.
Utilità	Garantisce la qualità della documentazione: assicura che i documenti siano comprensibili, riducendo ambiguità e incomprensioni.

Tabella 11: MPC9 - Indice di Gulpease (IG)

5.2.1.2 MPC10 - Indice di Frammentazione (IF)

Codice	MPC10
Formula	$$IF = \frac{N_{sezioni} + N_{paragrafi}}{N_{linee}} \cdot 100$$ Dove: • $N_{sezioni}$ = Numero di sezioni e sottosezioni del documento; • $N_{paragrafi}$ = Numero di paragrafi del documento; • N_{linee} = Numero totale di linee di testo del documento.
Descrizione	Misura il grado di suddivisione del testo in unità logiche (sezioni e paragrafi). Un valore bilanciato indica che il documento non presenta "muri di testo" eccessivi, facilitando la lettura veloce e la memorizzazione dei concetti. Interpretazione dei valori: • IF $<$ 5%: il testo è molto denso e difficile da leggere; • 5% $\leq$ IF $\leq$ 20%: bilanciamento ottimale tra contenuto e suddivisione spaziale; • IF $>$ 20%: il testo è eccessivamente frammentato, rischiando di perdere coesione.
Utilità	Assicura che la documentazione sia strutturata in modo da facilitare la consultazione e il reperimento rapido delle informazioni. Previene inoltre problemi di leggibilità dovuti a testo eccessivamente denso o frammentato.

Tabella 12: MPC10 - Indice di Frammentazione (IF)

5.2.2 Verifica

5.2.2.1 MPC11 - Test Success Rate (TSR)

Codice	MPC11
Formula	$$TSR = \frac{T_{successo}}{T_{totali}} \cdot 100$$ Dove: • $T_{successo}$ = Numero di test eseguiti con esito positivo; • T_{totali} = Numero di test eseguiti.
Descrizione	Percentuale di casi di test che sono stati eseguiti con esito positivo rispetto al numero totale di casi di test eseguiti. Interpretazione dei valori: • TSR = 100%: qualità eccellente, tutti i test sono stati superati; • 80% ≤ TSR < 100%: qualità accettabile; • 70% ≤ TSR < 80%: qualità discreta, sono necessarie correzioni; • TSR < 70%: qualità inaccettabile, necessaria revisione approfondita del codice.
Utilità	Fornisce una misura della correttezza del *software* e dell'efficacia dell'attività di *testing*.

Tabella 13: MPC11 - Test Success Rate (TSR)

5.2.3 Gestione della qualità

5.2.3.1 MPC12 - Percentuale Metriche Soddisfatte (PMS)

Codice	MPC12
Formula	$$PMS = \frac{M_{soddisfatte}}{M_{totali}} \cdot 100$$ Dove: • $M_{soddisfatte}$ = Numero di metriche soddisfatte; • M_{totali} = Numero di metriche definite.
Descrizione	Percentuale di metriche di qualità soddisfatte rispetto al numero di metriche totali. Interpretazione dei valori: • PMS ≥ 90%: eccellente, il progetto rispetta gli standard di qualità; • 80% ≤ PMS < 90%: buono, ma alcune aree richiedono miglioramento; • 70% ≤ PMS < 80%: sufficiente, necessarie azioni correttive; • PMS < 70%: insoddisfacente, il progetto presenta problemi significativi di qualità.
Utilità	Offre una valutazione della qualità complessiva del progetto. Facilita il controllo dell'aderenza agli standard di qualità definiti e l'identificazione tempestiva di criticità che richiedono azioni correttive.

Tabella 14: MPC12 - Percentuale Metriche Soddisfatte (PMS)

5.3 Processi organizzativi

5.3.1 Gestione dei processi

5.3.1.1 MPC13 - Percentuale Rischi Inattesi (PRI)

Codice	MPC13
Formula	$$PRI_k = \frac{R_{inattesi,k}}{R_{totali,k}} \cdot 100$$ Dove: • $R_{inattesi,k}$ = Numero di rischi non previsti che si sono verificati nello *sprint* k; • $R_{totali,k}$ = Numero di rischi totali che si sono verificati nello *sprint* k; • k = Numero dello *sprint* di riferimento.
Descrizione	Percentuale di rischi non previsti che si sono manifestati nel corso dello *sprint* k. Interpretazione dei valori: • PRI = 0: eccellente; • PRI $\leq$ 5%: buono, alcuni rischi non previsti sono normali; • 5% $<$ PRI $\leq$ 20%: accettabile ma migliorabile; • 20% $<$ PRI $\leq$ 40%: insufficiente, è necessario migliorare l'analisi dei rischi; • PRI $>$ 40%: gestione inadeguata, l'attività di gestione dei rischi è inefficace.
Utilità	Fornisce una misura del livello di accuratezza dell'attività di analisi e gestione dei rischi.

Tabella 15: MPC13 - Percentuale Rischi Inattesi (PRI)

5.3.1.2 MPC14 - Labor Efficiency (LE)

Codice	MPC14
Formula	$$LE_k = \frac{H_{pianificate}}{H_{effettive}} \cdot 100$$ Dove: • $H_{pianificate}$ = Somma delle ore stimate per le attività completate nello *sprint* k; • $H_{effettive}$ = Somma delle ore rendicontate dai membri del gruppo per lo *sprint* k; • k = Numero dello sprint di riferimento.
Descrizione	Rappresenta il rapporto tra le ore di lavoro pianificate per lo svolgimento di determinate attività e le ore effettivamente impiegate per il loro completamento nello *sprint* di riferimento. Interpretazione dei valori: • LE $>$ 100%: il gruppo è stato più veloce del previsto; • LE $<$ 100%: il team ha impiegato più tempo del previsto; • LE = 100%: le stime sono perfettamente in linea con la capacità produttiva.
Utilità	Mentre lo SPI (si veda §**??**) valuta l'avanzamento in termini di valore economico, questa metrica permette di monitorare direttamente la precisione delle stime temporali effettuate dal team durante la pianificazione.

Tabella 16: MPC14 - Labor Efficiency (LE)

6 Metriche di qualità di prodotto

In questa sezione vengono definite le metriche di qualità adottate per garantire che i prodotti di progetto (documentazione e *software*) soddisfino i requisiti qualitativi stabiliti.
Ogni metrica è identificata da un codice univoco nella forma *MPDX*, dove *MPD* è l'acronimo di *Metrica di ProDotto* e *X* rappresenta un numero intero progressivo.

6.1 Funzionalità

6.1.0.1 MPD1 - Percentuale Requisiti Obbligatori Soddisfatti (PROS)

Codice	MPD1
Formula	$$PROS = \frac{RO_{soddisfatti}}{RO_{totali}} \cdot 100$$ Dove: • $RO_{soddisfatti}$ = Requisiti obbligatori soddisfatti; • RO_{totali} = Requisiti obbligatori totali.
Descrizione	Percentuale di requisiti obbligatori soddisfatti rispetto al numero totale di requisiti obbligatori definiti. Interpretazione dei valori: • PROS = 100%: tutti i requisiti obbligatori sono stati implementati, il prodotto è completo; • 90% ≤ PROS < 100%: il prodotto è quasi completo, mancano poche funzionalità; • 70% ≤ PROS < 90%: il prodotto è parzialmente funzionante; • PROS < 70%: il prodotto è incompleto, non rilasciabile.
Utilità	Determina se il prodotto ha raggiunto il livello minimo di funzionalità necessario per il rilascio. Permette inoltre di identificare tempestivamente ritardi che potrebbero compromettere la consegna del prodotto nei tempi concordati.

Tabella 17: MPD1 - Percentuale Requisiti Obbligatori Soddisfatti (PROS)

6.1.0.2 MPD2 - Percentuale Requisiti Opzionali Soddisfatti (PRPS)

Codice	MPD2
Formula	$$PRPS = \frac{RP_{soddisfatti}}{RP_{totali}} \cdot 100$$ Dove: • $RP_{soddisfatti}$ = Requisiti opzionali soddisfatti; • RP_{totali} = Requisiti opzionali totali.
Descrizione	Percentuale di requisiti opzionali soddisfatti rispetto al numero totale di requisiti opzionali definiti. L'implementazione di requisiti opzionali aumenta il valore del prodotto ma non è essenziale per il suo funzionamento base.
Utilità	Misura il valore aggiunto fornito dal prodotto oltre le funzionalità minime richieste e permette di comunicare al proponente il livello di arricchimento del prodotto rispetto alle aspettative base.

Tabella 18: MPD2 - Percentuale Requisiti Opzionali Soddisfatti (PRPS)

6.1.0.3 MPD3 - Percentuale Requisiti Desiderabili Soddisfatti (PRDS)

Codice	MPD3
Formula	$$PRDS = \frac{RD_{soddisfatti}}{RD_{totali}} \cdot 100$$ Dove: • $RD_{soddisfatti}$ = Requisiti desiderabili soddisfatti; • RD_{totali} = Requisiti desiderabili totali.
Descrizione	Percentuale di requisiti desiderabili soddisfatti rispetto al numero totale di requisiti desiderabili definiti. I requisiti desiderabili migliorano significativamente l'esperienza utente e la competitività del prodotto.
Utilità	Valuta il valore aggiunto fornito dal prodotto oltre le funzionalità minime e opzionali.

Tabella 19: MPD3 - Percentuale Requisiti Desiderabili Soddisfatti (PRDS)

6.2 Affidabilità

6.2.0.1 MPD4 - Line Coverage (LC)

Codice	MPD4
Formula	$$LC = \frac{L_{eseguite}}{L_{totali}} \cdot 100$$ Dove: • $L_{eseguite}$ = Numero di linee di codice eseguite dai test; • L_{totali} = Numero di linee di codice totali.
Descrizione	Indica la percentuale di linee di codice eseguite dai test automatici rispetto alle linee di codice totali. Interpretazione dei valori: • LC $\geq$ 80%: copertura eccellente, codice ben testato; • 60% $\leq$ LC $<$ 80%: copertura buona, ma migliorabile; • 40% $\leq$ LC $<$ 60%: copertura insufficiente, è necessario aumentare i test; • LC $<$ 40%: copertura critica, il codice è poco testato.
Utilità	Quantifica l'efficacia dei test automatici nell'individuare difetti e garantire la correttezza del codice. Permette inoltre di identificare porzioni di codice non testate che potrebbero contenere errori.

Tabella 20: MPD4 - Line Coverage (LC)

6.2.0.2 MPD5 - Branch Coverage (BC)

Codice	MPD5
Formula	$$BC = \frac{B_{eseguite}}{B_{totali}} \cdot 100$$ Dove: • $B_{eseguite}$ = Numero di rami (*branch*) decisionali eseguiti dai test; • B_{totali} = Numero di rami decisionali totali.
Descrizione	Indica la percentuale di rami decisionali del codice eseguiti dai test. Interpretazione dei valori: • BC $\geq 75\%$: copertura eccellente dei percorsi decisionali; • $60\% \leq$ BC $< 75\%$: copertura buona; • $40\% \leq$ BC $< 60\%$: copertura insufficiente; • BC $< 40\%$: molti percorsi decisionali non testati, rischio elevato di errori.
Utilità	Fornisce una misura più rigorosa della qualità dei test rispetto alla semplice copertura delle linee, garantendo che la logica condizionale sia correttamente testata.

Tabella 21: MPD5 - Branch Coverage (BC)

6.2.0.3 MPD6 - Statement Coverage (SC)

Codice	MPD6
Formula	$$SC = \frac{S_{eseguite}}{S_{totali}} \cdot 100$$ Dove: • $S_{eseguite}$ = Numero di istruzioni (*statement*) eseguite dai test; • S_{totali} = Numero di istruzioni totali.
Descrizione	Indica la percentuale di istruzioni nel codice eseguite dai test. Interpretazione dei valori: • SC $\geq 80\%$: copertura eccellente delle istruzioni; • $60\% \leq$ SC $< 80\%$: copertura buona; • $40\% \leq$ SC $< 60\%$: copertura insufficiente; • SC $< 40\%$: molte istruzioni non testate.
Utilità	Garantisce che ogni singola istruzione del codice sia stata eseguita almeno una volta durante i test, identificando codice inutilizzato o non raggiungibile.

Tabella 22: MPD6 - Statement Coverage (SC)

6.3 Usabilità

6.3.0.1 MPD7 - Profondità Massima di Navigazione (PMN)

Codice	MPD7
Formula	$$PMN = \max_{i=1,\dots,k}\{C_i\}$$ Dove: • C_i = Numero di click necessari per completare l'operazione i-esima; • k = Numero totale di operazioni analizzate.
Descrizione	Misura il massimo numero di click necessari per completare qualsiasi operazione, date k operazioni analizzate. Valuta il caso peggiore di navigazione, garantendo che anche le operazioni più profonde rimangano accessibili. Interpretazione dei valori: • PMN ≤ 3: tutte le operazioni sono immediate; • $3 <$ PMN ≤ 5: la navigazione è buona anche nel caso peggiore; • $5 <$ PMN ≤ 7: accettabile, nonostante alcune operazioni richiedano troppi passaggi; • PMN > 7: inefficiente, sono presenti percorsi eccessivamente profondi.
Utilità	Garantisce che nessuna funzionalità del sistema sia eccessivamente nascosta o difficile da raggiungere, rischiando di compromettere l'usabilità del prodotto.

Tabella 23: MPD7 - Profondità Massima di Navigazione (PMN)

6.4 Efficienza

6.4.0.1 MPD8 - Tempo Medio di Risposta (TMR)

Codice	MPD8
Formula	$$TMR = \frac{\sum_{i=1}^{k} T_i}{k}$$ Dove: • T_i = Tempo di risposta per la richiesta i-esima; • k = Numero di richieste testate;
Descrizione	Tempo medio che intercorre tra l'invio di una richiesta al sistema e la ricezione della risposta completa. Interpretazione dei valori: • TMR ≤ 1 secondo: eccellente, risposta immediata; • $1 <$ TMR ≤ 3 secondi: buono, risposta rapida; • $3 <$ TMR ≤ 5 secondi: accettabile, ma migliorabile; • TMR > 5 secondi: inaccettabile, l'utente percepisce lentezza.
Utilità	Garantisce che il sistema soddisfi le aspettative degli utenti in termini di reattività e fluidità d'uso.

Tabella 24: MPD8 - Tempo Medio di Risposta (TMR)

6.5 Manutenibilità

6.5.0.1 MPD9 - Complessità Ciclomatica (CC)

Codice	MPD9
Formula	$$CC = E - N + 2P$$ Dove: • E = Numero di archi del grafo di controllo di flusso; • N = Numero di nodi del grafo di controllo di flusso; • P = Numero di componenti connesse.
Descrizione	Misura la complessità strutturale del codice calcolando il numero di percorsi linearmente indipendenti attraverso il codice sorgente. Viene calcolata per ciascuna funzione o metodo del sistema. Una bassa complessità indica codice più leggibile, testabile e manutenibile. Interpretazione dei valori: • CC $\leq$ 10: complessità bassa, codice semplice e manutenibile; • 10 $<$ CC $\leq$ 20: complessità moderata, ancora accettabile; • 20 $<$ CC $\leq$ 30: complessità elevata; • CC $>$ 30: complessità critica, il codice è difficile da testare e mantenere.
Utilità	Permette di misurare la complessità delle funzioni per evitare difficoltà nel *testing* del codice e nella sua comprensione.

Tabella 25: MPD9 - Complessità Ciclomatica (CC)

6.6 Portabilità

6.6.0.1 MPD10 - Compatibilità Cross-Browser (CCB)

Codice	MPD10
Formula	$$CCB = \frac{B_{supportati}}{B_{target}} \cdot 100$$ Dove: • $B_{supportati}$ = Numero di browser su cui il prodotto funziona correttamente; • B_{target} = Numero totale di browser target definiti per il progetto.
Descrizione	Percentuale di browser testati su cui il prodotto funziona correttamente rispetto al totale dei browser target definiti nel progetto. Interpretazione dei valori: • CCB = 100%: piena compatibilità, il prodotto funziona su tutti i browser target; • 80% $\leq$ CCB $<$ 100%: compatibilità buona, problemi solo su browser marginali; • 60% $\leq$ CCB $<$ 80%: compatibilità parziale, alcuni browser principali presentano problemi; • CCB $<$ 60%: compatibilità insufficiente, il prodotto non è utilizzabile su molti browser.
Utilità	Garantisce che il prodotto sia utilizzabile dalla maggior parte dei browser.

Tabella 26: MPD10 - Compatibilità Cross-Browser (CCB)