Università degli Studi di Padova

Laurea: Informatica

Corso: Ingegneria del Software
Anno Accademico: 2025/26

Gruppo 17

Nome: BitByBit
Email: swe.bitbybit@gmail.com

Norme di progetto

Stato:	in lavorazione
Versione:	0.11.5
Responsabile:	Riccardo Manisi
Redattori:	Ferdinando Fracasso
	Riccardo Manisi
	Gabriele Scaggiante
	Dennis Parolin
Verificatori:	Dennis Parolin
	Marco Sanguin
	Giovanni Visentin
	Ferdinando Fracasso
Destinatari:	Gruppo BitByBit

Registro delle modifiche

Versione	Data	Autore	Descrizione	Verificatore
0.11.5	2026-02-09	Dennis Parolin	Aggiunta una descrizione mancante.	Marco Sanguin
0.11.4	2026-02-06	Dennis Parolin	Modificata la sezione "Stesura dei reqauisiti" modificando la nomenclatura degli identificatori dei requisiti. Risolti problemi di compilazione.	Giovanni Visentin
0.11.3	2026-02-03	Dennis Parolin	Fatti dei controlli generici del documento. Sistemati riferimenti a sezioni non esistenti o errate. Aggiunti i placeholder per raggiungere sezioni esistenti ma non raggiungibili (perché senza riferimento). Sistemati vari errori di battitura, errori grammaticali, refusi e errori logici. Aggiunti ad esempio degli strumenti citati ma che non erano presenti nella tabella degli strumenti utilizzati	Giovanni Visentin

0.11.2	2026-01-11	Dennis Parolin	Aggiunti gli strumenti di supporto delle sezioni "Processo di **Verifica**$^{\mathbf{G}}$", "Processo di **Validazione**$^{\mathbf{G}}$" e "Codifica". Aggiunta la lista delle tabelle.	Riccardo Manisi
0.11.1	2026-01-10	Dennis Parolin	Sistemate le sezioni "Processo di **Verifica**$^{\mathbf{G}}$" e "Processo di **Validazione**$^{\mathbf{G}}$" tra i Processi di Supporto. Sistemata la sezione "Codifica" tra i Processi Primari.	Riccardo Manisi
0.11.0	2026-01-06	Dennis Parolin	Aggiunta sezioni "Processo di **Verifica**$^{\mathbf{G}}$" e "Processo di **Validazione**$^{\mathbf{G}}$" tra i Processi di Supporto. Aggiunta la sezione "Codifica" tra i Processi Primari.	Riccardo Manisi
0.10.1	2026-01-06	Ferdinando Fracasso	Aggiunta "**Piano Di Qualifica**$^{\mathbf{G}}$" nella sezione Documentazione fornita	Riccardo Manisi
0.10.0	2025-12-31	Riccardo Manisi	Aggiunta sezione con metriche per QA	Ferdinando Fracasso
0.9.0	2025-12-30	Riccardo Manisi	Aggiunto processo di accertamento qualità	Ferdinando Fracasso

0.8.0	2025-12-23	Riccardo Manisi	Aggiunte le sezioni di attivita previste e analisi dei requisiti per processo di sviluppo	Ferdinando Fracasso
0.7.0	2025-12-22	Riccardo Manisi	Aggiunta attività di aggiornamento glossario	Giovanni Visentin
0.6.1	2025-12-16	Ferdinando Fracasso	Rimossa sezione ripetuta	Giovanni Visentin
0.6.0	2025-12-13	Gabriele Scaggiante	Aggiunta Processo di miglioramento e Processo di formazione	Marco Sanguin
0.5.0	2025-12-11	Ferdinando Fracasso	Aggiunta sezione "Processo di fornitura"	Giovanni Visentin
0.4.1	2025-12-11	Riccardo Manisi	Aggiunta prefazione con info generali del doc	Marco Sanguin
0.4.0	2025-12-10	Riccardo Manisi	Aggiunta attività di istituzione per Git e GitHub	Marco Sanguin
0.3.1	2025-12-10	Riccardo Manisi	Corretti errori grammaticali	Marco Sanguin
0.3.0	2025-12-07	Riccardo Manisi	Aggiunta attività di implementazione nel processo di infrastruttura	Giovanni Visentin
0.2.0	2025-12-02	Riccardo Manisi	Aggiunta sezioni per processo di configurazione e di gestione.	Marco Sanguin
0.1.1	2025-12-02	Riccardo Manisi	Sistemati errori lessicali	Marco Sanguin

0.1.0	2025-12-01	Riccardo Manisi	Creazione del documento. Aggiunto processo di documentazione e bozza per le sezioni successive	Marco Sanguin

Indice

Elenco delle tabelle

1. Introduzione

1.1. Scopo del documento

Il presente documento è volto a definire i processi del **Way Of Working**^{G} per il gruppo BitByBit che si impegna a mantenerlo per tutta la durata del progetto. Per la sua redazione è stato preso come riferimento lo Standard ISO/IEC 12007:1995 che identifica 3 tipologie di processo:

1. **processi primari:** atti alla produzione di un prodotto software;
2. **processi di supporto:** supportano altri processi come parte integrante e contribuiscono al successo e alla qualità del prodotto software;
3. **processi organizzativi:** definiscono i metodi organizzativi del gruppo, per stabilire e implementare la struttura costituita dai processi di ciclo di vita.

Ogni membro del gruppo si impegna a visionare costantemente il presente documento e a rispettare rigorosamente i processi che vengono esposti di seguito, per ottenere un prodotto che sia professionale, coerente e uniforme.

1.2. Scopo del prodotto

L'azienda **Miriade S.r.l.** ha proposto lo sviluppo di un'applicazione mobile, denominata "L'app che Protegge e Trasforma" finalizzata alla prevenzione e al supporto delle vittime di violenze di genere.
Il prodotto punta a stabilirsi come uno strumento intelligente e sicuro che aiuta l'utilizzatore a riconoscere segnali di pericolo e a fornire risorse per tutelarsi da atti, che siano fisici o psicologici, che potrebbero nuocere alla sua salute. L'applicazione implementa metodologie volte alla tutela delle persone che sono già vittima di violenze, ma anche per la valutazione e l'individuazione di situazioni che potrebbero sfociare in atti di violenza.

1.3. Glossario

I documenti prodotti nei processi di ciclo di vita sono corredati da un glossario che costituisce un riferimento condiviso. Il suo scopo è ridurre le ambiguità lessicali che possono emergere a causa dei diversi livelli di competenze tecniche tra i destinatari dei documenti. Poiché le Norme di Progetto, l'Analisi dei Requisiti, il **Piano Di Progetto**^{G} e il **Piano Di Qualifica**^{G} sono rivolti a figure eterogenee, tra cui acquirenti, fornitori, sviluppatori, gestori e utenti del prodotto software, è necessario che ogni termine potenzialmente ambiguo, tecnico o soggetto a interpretazioni multiple sia riportato in modo chiaro e univoco. Per favorire la reperibilità dei termini e rendere immediata la loro identificazione all'interno dei documenti, ogni voce del glossario è richiamata nel testo mediante la seguente convenzione stabilita dal gruppo BitByBit:

termine_nel_glossario[G]

Il glossario è considerato parte integrante del processo di documentazione: viene aggiornato ogni volta che emergono nuovi termini rilevanti, che vengono introdotti nuovi concetti tecnici o che si rende necessario chiarire ambiguità riscontrate durante la redazione o la **Verifica**[G] dei documenti.

1.4. Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato**[G] **d'appalto C4: *L'app che Protegge e Trasforma***
  Parti consultate: documento completo
  Versione: 1.0
  Ultimo accesso: *2026-01-12*
  https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C4.pdf

- **ISO/IEC 12207:1997 - Software Life Cycle**[G] **Processes**
  Parti consultate: documento completo
  Versione: edizione 1997
  Ultimo accesso: *2025-12-30*
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf

1.4.2 Riferimenti informativi

- **Ian Sommerville, *Software Engineering***
  Parti consultate: Capitoli 2, 24
  Versione: *X Edizione*
  Ultimo accesso: *2026-01-12*

2. Processi primari

La presente sezione espone i seguenti processi primari:

(a) **Fornitura;**

(b) **Sviluppo;**

2.1. Fornitura

2.1.1 Scopo

Il processo primario di fornitura definisce le attività da intraprendere da parte del fornitore e l'analisi e pianificazione delle risorse necessarie per garantire l'efficienza e la conformità del progetto ai requisiti discussi con il proponente.

2.1.2 Implementazione di processo

Il processo di fornitura è suddiviso nelle seguenti attività:

- **Inizializzazione:** Il fornitore effettua una analisi dei requisiti nelle richieste del proponente, tenendo conto di eventuali vincoli e valutando la propria capacità di realizzare la proposta, valutando possibili requisiti da ritrattare con il proponente.
- **Pianificazione della risposta:** Il fornitore definisce una proposta in risposta alla richiesta del proponente, tenendo conto dei risultati della fase precedente.
- **Contrattazione:** Il fornitore presenta la propria risposta, definita nell'attività precedente, al proponente al fine di stipulare un accordo per la fornitura del progetto finale.
- **Pianificazione:** Il fornitore stabilisce un framework per la gestione del progetto e per garantire la qualità del prodotto finale, e, nel caso non fosse definito negli accordi con il proponente, il fornitore sceglie anche un modello di ciclo di vita del software appropriato per il progetto. Il fornitore inoltre si occupa anche di individuare le risorse e tecnologie necessarie per il progetto, considerandone anche i relativi rischi.
- **Esecuzione e controllo:** Il fornitore esegue e implementa il piano definito nella fase precedente, monitorandone progresso e qualità del prodotto durante il ciclo di vita precedentemente definito, interfacciandosi con il proponente quando necessario.
- **Revisione e Validazione[G]:** Il fornitore coordina attività di revisione con il proponente, anche durante lo sviluppo del prodotto poiché necessario per avere feedback per piani futuri, ed effettua le necessarie attività di **Verifica[G]** e **Validazione[G]** per garantire la qualità del prodotto.
- **Consegna e completamento:** Il fornitore consegna il prodotto finale al proponente, fornendo inoltre il necessario supporto.

2.1.3 Contatti con l'azienda proponente

La comunicazione avverrà principalmente tramite uno spazio su Google Chat messo a disposizione dal proponente Miriade S.r.l., al fine di permettere una comunicazione più

rapida in caso di eventuali dubbi o domande. Inoltre il proponente si rende disponibile a incontri da remoto, tramite Google Meet, o in presenza, quando necessario per la **Verifica**^{G} dello stato di avanzamento del progetto.

2.1.4 Documentazione fornita

Questa sezione elenca e descrive brevemente tutta la documentazione che verrà prodotta dal gruppo BitByBit che verrà consegnata al proponente Miriade S.r.l. e ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.

- **Lettera di presentazione:**
  - Documento in cui il gruppo BitByBit ufficializza la propria candidatura al **Capitolato**^{G} dell'azienda Miriade S.r.l.
    Il documento contiene una versione riassunta delle ragioni per la scelta del **Capitolato**^{G}, un resoconto generale degli incontri preliminari effettuati con delle aziende proponenti e informazioni riguardanti il budget e la data di consegna prevista del progetto.
- **Valutazione dei capitolati:**
  - Documento in cui viene fatta una analisi dettagliata di ciascun **Capitolato**^{G} reso disponibile dai proponenti, valutandone punti a favore e punti critici, con maggiore attenzione posta sul **Capitolato**^{G} scelto dal gruppo.
- **Valutazione di costi e rischi:**
  - Documento in cui il gruppo BitByBit ha svolto una analisi preliminare dei rischi durante lo svolgimento del progetto e descritto dei possibili metodi di mitigazione di tali rischi.
    Il documento inoltre contiene una analisi dei ruoli richiesti dal progetto e delle ore assegnate per ciascun ruolo ai membri del gruppo, il calcolo del preventivo costi per lo svolgimento del progetto, e la data di consegna prevista per il prodotto finale.
- **Analisi dei requisiti:**
  - Documento che ha l'obiettivo di analizzare in modo esaustivo i requisiti del prodotto del progetto, mediante l'identificazione dei casi d'uso, classificandoli in base alla loro importanza come obbligatori, desiderabili o opzionali.
    Questo in modo da avere un punto di riferimento primario per le attività di progettazione, sviluppo, **Verifica**^{G} e **Validazione**^{G}, e garantire la conformità del prodotto finale alle richieste del proponente.
- **Norme di progetto:**

  - Il presente documento, avente l'obiettivo di definire i processi del **Way Of Working**[G] e le norme che il gruppo BitByBit si impegna a seguire per l'intera durata del progetto.

- **Piano Di Progetto**[G]**:**
  - Documento in cui viene definita la pianificazione strategica e operativa delle attività intraprese dal gruppo BitByBit durante lo sviluppo del progetto.
    Il **Piano Di Progetto**[G] ha l'obiettivo di tenere sotto controllo l'avanzamento delle attività e di garantire l'efficienza e la qualità dello sviluppo, mediante la pianificazione delle attività e relative scadenze, monitorando i costi di ciascun periodo e identificando le potenziali criticità che ostacolerebbero lo sviluppo.

- **Piano Di Qualifica**[G]**:**
  - Documento in cui viene definita la strategia adottata dal gruppo al fine di garantire la qualità del prodotto finale, sotto forma di metriche e relativi obiettivi, metodi di testing e resoconti.

- **Glossario:**
  - Documento contenente le definizioni dei termini usati nei documenti prodotti durante lo svolgimento del progetto, per permetterne una facile consultazione.

2.2. Processo di sviluppo

Il **processo di sviluppo** comprende le attività svolte dal team di sviluppo. Esso include le attività per l'analisi dei requisiti, il design, lo sviluppo del prodotto software, l'integrazione dei test e l'accettazione del prodotto da rilasciare.

2.2.1 Strumenti a supporto

Per sostenere i processi di sviluppo sono stati utilizzati i seguenti strumenti:

- **Lucidchart**: piattaforma per la modellazione UML. Lo strumento consente la creazione di diagrammi UML (tra cui diagrammi dei casi d'uso, delle classi e di sequenza) direttamente in ambiente web, favorendo la collaborazione sincrona tra i membri del gruppo sullo stesso documento.

2.2.2 Attività previste

Le attività che compongono il **processo di sviluppo** adattate al progetto sviluppato dal gruppo BitByBit sono elencate di seguito.

- **Implementazione di processo**, l'organizzazione è tenuta a scegliere un modello di ciclo di vita se non ne ha ancora adottato uno in particolare, tenendo conto del campo di applicazione, del campo e della complessità del progetto.
- **Analisi dei requisiti di sistema**, l'analisi del sistema da implementare deve essere condotta in modo dettagliato, al fine di produrre una descrizione delle funzionalità e delle capacità del sistema, nonché dei requisiti **Utente**^{G} e di sistema.
- **Progettazione dell'Architettura**^{G}, deve essere identificata un'**Architettura**^{G} di alto livello per identificare gli elementi software. Deve essere garantito che tutti i requisiti siano allocati tra gli elementi.
- **Codifica e testing**, per ogni elemento software il programmatore deve implementare e documentare ogni elemento software, andando a garantire che ogni unità software soddisfi i suoi requisiti implementando i corrispettivi test.
- **Integrazione software**, il programmatore è tenuto a definire le unità software, i test le tempistiche e i dati che devono essere implementati per il particolare elemento software.
- **Test di qualità del software**, comprende l'implementazione di test in conformità con le metriche di qualità del software.
- **Integrazione nel sistema**, il programmatore procede ad integrare l'elemento software nel sistema, assieme alle sue dipendenze se è necessario.
- **Test di qualità del sistema**, vengono eseguiti i test sull'intero sistema, per accertarsi che sia pronto per essere rilasciato.
- **Installazione del software**, comprende la consegna del software al cliente e la sua installazione nell'ambiente concordato dal **Contratto**^{G}.
- **Supporto per l'approvazione**, il fornitore e il cliente collaudano il prodotto per accertare che tutti i requisiti stabiliti sono presenti nel software consegnato.

2.2.3 Analisi dei requisiti

L'attività di analisi dei requisiti ha l'obiettivo di trasformare i requisiti **Utente**^{G} espressi nel **Capitolato**^{G}, nel nostro caso il **Capitolato**^{G} **C4** proposto da **Miriade S.r.l.**, in requisiti di sistema chiari, completi e verificabili. Tale attività produce il documento dell'analisi dei requisiti che stabilisce le basi per un accordo tra la proponente e il **gruppo BitByBit** su quello che il prodotto software è tenuto a fare.
L'analisi dei requisiti obbliga tutte le parti coinvolte a valutare in modo completo e condiviso ciò che il sistema dovrà realizzare prima dell'avvio della progettazione. Questo riduce significativamente il rischio di dover intervenire successivamente su design, codice o test.

Il gruppo si impegna inoltre a monitorare con continuità l'allineamento tra requisiti implementati e requisiti concordati con la proponente, aggiornandoli in modo controllato quando necessario.

2.2.3.1 Stesura dei casi d'uso La definizione dei casi d'uso segue un processo iterativo che comprende:

- **comprensione del problema** e del dominio applicativo;
- **identificazione degli scenari** attraverso la definizione delle interazioni tra attori e sistema;
- **stesura dei casi d'uso** andando ad identificare gli scenari principali e quelli secondari;
- **implementazione degli UML**.

Il gruppo effettua una revisione completa dei casi d'uso individuati, correggendo eventuali incoerenze. Se necessario, vengono condotti approfondimenti con la proponente per garantire un'interpretazione condivisa degli scenari critici.
Per ogni caso d'uso viene realizzato un diagramma UML dedicato, al fine di ridurre possibili ambiguità interpretative.

La nomenclatura adottata viene esposta di seguito.

`UC-<funzionalità>-X.Y`

- `UC` sigla per use case
- `<funzionalità>` sigla per la funzionalità a cui si riferisce quel caso d'uso;
- `X` numero intero identificativo per il caso d'uso principale;
- `Y` numero intero identificativo per il sotto-caso eventuale.

Questa codifica garantisce un'identificazione univoca e promuove tracciabilità e verificabilità.

2.2.3.2 Stesura dei requisiti I requisiti vengono organizzati per livello di importanza, definito con la proponente:

- **Obbligatori** (**OB**): necessari e irrinunciabili per la soddisfazione minima degli **Stakeholder**^{G};
- **Desiderabili** (**DE**): non essenziali ma ad alto valore aggiunto;
- **Opzionali** (**OP**): utili, ma pianificabili o negoziabili in fasi successive.

Ogni **Requisito**^{G} è identificato tramite una codifica standard:

`R<tipo>-<importanza>-X`

Dove:

- `R`: è per abbreviare la parola **Requisito**^{G}
- `<tipo>`: è la sigla che indica la tipologia di **Requisito**^{G}
  - `F`: per indicare che è un **Requisito Funzionale**^{G}
  - `Q`: per indicare che è un **Requisito**^{G} di Qualità
  - `V`: per indicare che è un **Requisito**^{G} di Vincolo
- `<importanza>`: è la sigla a due lettere che indica la categoria del **Requisito**^{G}
- `X`: è un identificatore numerico univoco

Tale classificazione supporta la tracciabilità, la **Verifica**^{G} e la gestione dell'evoluzione dei requisiti durante l'intero ciclo di vita.

2.2.4 Codifica

L'attività di codifica ha lo scopo di realizzare tecnicamente quanto definito nelle fasi di analisi e progettazione. L'obiettivo principale delle norme qui esposte è garantire l'**uniformità**: il codice deve risultare omogeneo, leggibile e manutenibile, indipendentemente dal membro del gruppo che lo ha scritto.

2.2.4.1 Automazione e controllo qualità Per garantire il rispetto rigoroso dello stile senza rallentare lo sviluppo, il gruppo affida il controllo formale agli strumenti di automazione, riducendo l'errore umano. Il processo di **Verifica**^{G} dello stile avviene su due livelli:

- **Locale**: l'ambiente di sviluppo (IDE) di ogni membro è configurato per formattare automaticamente il codice al salvataggio, applicando le regole di indentazione e spaziatura condivise.
- **Remoto**: la pipeline di **GitHub Actions Verifica**^{G} la conformità del codice ad ogni *push*. Come da policy, il mancato rispetto delle norme di formattazione impedisce tecnicamente il merge delle modifiche nel ramo principale.

2.2.4.2 Norme stilistiche e sintattiche Per favorire la comprensione globale del progetto, vengono imposte le seguenti regole di scrittura:

- **Lingua e Nomenclatura**: l'intera base di codice (variabili, funzioni, commenti) deve essere redatta in **lingua inglese**. I nomi scelti devono essere significativi ed auto-esplicativi, evitando abbreviazioni criptiche che richiedano interpretazione.
- **Formattazione**: l'indentazione è fissata a 4 spazi (o 1 tabulazione) ed è mantenuta coerente dagli strumenti automatici.
- **Leggibilità**: per facilitare il debugging e la lettura, è vietato scrivere più istruzioni sulla stessa riga.
- **Commenti**: si predilige un codice che si "autodocumenta" tramite nomi chiari. Tuttavia, è obbligatorio commentare tutte le interfacce pubbliche (metodi e classi esportate) specificando parametri e valori di ritorno.

2.2.4.3 Vincoli implementativi Al fine di mantenere il codice robusto e testabile, il gruppo adotta le seguenti pratiche ingegneristiche:

- **Assenza di stato globale**: è vietato l'utilizzo di variabili globali, in quanto introducono dipendenze nascoste e rendono imprevedibile il comportamento del software.
- **Semplicità delle funzioni**: le funzioni devono essere brevi e focalizzate su una singola responsabilità. Una funzione eccessivamente lunga o complessa è indice di cattiva progettazione e deve essere rifattorizzata.

2.2.5 Strumenti a supporto della codifica

L'applicazione delle norme di codifica è facilitata da un set di strumenti che accompagna lo sviluppatore dalla scrittura fino all'integrazione.

2.2.5.1 Ambiente di Sviluppo Il gruppo ha uniformato l'ambiente di lavoro adottando **Visual Studio Code** come IDE di riferimento.

2.2.5.2 Automazione e CI Gli strumenti critici per il mantenimento dello standard, le cui specifiche tecniche verranno consolidate durante la Product **Baseline**$^{\mathbf{G}}$ (**Pb**$^{\mathbf{G}}$), sono classificabili in:

- **Linter e Formatter**: strumenti di analisi statica specifici per linguaggio che rilevano violazioni stilistiche in tempo reale all'interno dell'IDE.
- **GitHub Actions**: piattaforma utilizzata per i controlli remoti bloccanti (Quality Gate), che assicura che nessun codice non conforme venga integrato nel ramo `develop`.

3. Processi di supporto

La presente sezione espone i seguenti processi di supporto:

(a) **documentazione;**

(b) **gestione della configurazione;**

(c) **gestione della qualità;**

(d) **Verifica[G];**

(e) **Validazione[G].**

3.1. Documentazione

Il processo di documentazione ha l'obiettivo di definire le modalità, gli strumenti e le convenzioni che il gruppo adotta per la produzione, la gestione, la revisione e l'approvazione di tutti i documenti del progetto.

3.1.1 Obiettivi

1. Garantire **completezza**, **chiarezza** e **coerenza** delle informazioni contenute nei documenti.
2. Assicurare la **tracciabilità** delle modifiche e delle versioni;
3. Favorire la **comprensibilità e manutenibilità** dei documenti da parte di tutti i membri del gruppo;
4. Fornire un **mezzo ufficiale di comunicazione** tra il gruppo, il **Committente[G]** e i revisori;
5. Rispettare la **uniformità formale e stilistica** tra i vari documenti prodotti.

L'implementazione del **Repository[G]** dedicato alla documentazione integra tutti gli strumenti necessari alla produzione dei documenti di ciclo di vita, garantendo che ogni nuovo membro del progetto possa svolgere le attività previste senza dipendere dal proprio ambiente di lavoro locale o incorrere in limitazioni tecniche.
È necessaria l'automazione dei processi dove è possibile, in modo da avere un sistema con pipeline per lo sviluppo che aiuti il membro del gruppo nei suoi compiti.

3.1.2 Strumenti a supporto

Per sostenere il processo di documentazione sono utilizzati i seguenti strumenti:

- **LaTeX**, per la stesura di documenti il gruppo ha scelto LaTeX, linguaggio di markup che permette di avere il controllo di tutte le parti di un documento, può essere eseguito nel **Repository**[G] senza la necessità di un editor di testo;
- **Overleaf**, piattaforma online utilizzata in fase iniziale per l'apprendimento della sintassi e i comandi di LaTeX;
- **Google Sheets**: utilizzato per la redazione di pianificazioni, tracciamento di requisiti preliminari, matrici decisionali e supporto alla raccolta dati;
- **Google Docs**, impiegato durante riunioni o fasi preliminari per appunti condivisi e raccolta rapida di contenuti;
- **Google Presentazioni**: utilizzato per la redazione e condivisione dei diari di bordo richiesti dal corso;
- **GitHub**, piattaforma di versionamento distribuito e piattaforma di collaborazione utilizzata per la gestione centralizzata dei documenti, delle modifiche, dei branch e delle **Pull Request**[G];
- **GitHub Actions**, strumento di automazione per l'aggiunta del template comune a tutti i documenti e per la compilazione dei file `.tex` in `PDF`.

3.1.3 Tipologie di documenti

- **Documenti interni**, destinati alla gestione e al coordinamento interno (verbali, norme di progetto, glossario);
- **Documenti esterni**, destinati a revisori, docenti o aziende (Analisi dei Requisiti, **Piano Di Progetto**[G], **Piano Di Qualifica**[G]);
- **Verbali interni ed esterni**, registrano rispettivamente, gli incontri tra i membri del gruppo e gli incontri tra il gruppo e gli enti esterni come l'azienda proponente o i professori.

Ogni documento, indipendentemente dalla sua categoria, deve essere redatto in conformità alle convenzioni formali definite nelle Norme di Progetto.

3.1.4 Implementazione di processo

Ogni documento, prima di essere integrato nella versione stabile del **Repository**^{G}, deve attraversare un ciclo strutturato di produzione che garantisce la qualità, la verificabilità e la tracciabilità delle modifiche. Le attività descritte in questa sezione definiscono il flusso operativo seguito dal gruppo per l'elaborazione, la revisione e la pubblicazione dei prodotti documentali.

3.1.4.1 Creazione delle issue GitHub Ogni azione che porta a dei cambiamenti ad un documento del gruppo deve iniziare con l'apertura di una issue su GitHub per permettere la tracciabilità nella board del GitHub **Project**^{G}. Ogni commit deve fare riferimento ad una sola azione in modo da garantire che il processo iterativo di revisione ed eventuali correzioni di errori sia il più veloce possibile.

3.1.4.2 Assegnazione della issue Nessuna issue può essere avviata senza che sia assegnata ad un membro del gruppo. L'assegnazione viene effettuata dal responsabile, che determina chi, nel gruppo, è incaricato dell'esecuzione di quella precisa attività. Questo passaggio garantisce che ogni modifica sia riconducibile a un autore e che la responsabilità operativa sia chiaramente definita.

3.1.4.3 Impostazione del documento Il membro del team può ora aprire un branch secondario da `develop`, seguendo la nomenclatura presente, ed eseguire le modifiche definite nella relativa issue. Se le modifiche implicano la creazione del file di documentazione il membro del gruppo è tenuto a creare il file sorgente, denominato secondo le convenzioni stabilite. Una volta inserito il nuovo file nel **Repository**^{G} remoto, il sistema provvede automaticamente ad applicare il template comune, garantendo l'integrazione delle componenti strutturali condivise.
Spetta quindi al redattore adattare il template al contenuto specifico del documento, completandone le sezioni pertinenti e verificando che l'impostazione risultante sia conforme agli standard definiti per la documentazione del progetto. Questa attività assicura uniformità, leggibilità e coerenza stilistica dell'intera produzione documentale lungo tutto il ciclo di vita del progetto.
Ogni membro del gruppo è quindi tenuto a verificare che i documenti prodotti rispettino un formato uniforme, verificabile e riconducibile agli standard scelti.
Questa attività garantisce la leggibilità, la coerenza e la mantenibilità dell'intera produzione documentale durante tutto il ciclo di vita del progetto.

Intestazione

Ogni pagina del documento include un'intestazione che riporta il titolo dello specifico documento e il nome del gruppo.

Frontespizio

Il frontespizio possiede i seguenti elementi:

- **logo dell'ateneo;**
- **informazioni dell'ateneo;**
- **logo del gruppo**;
- **contatto ufficiale** del gruppo;
- **nome del documento**;
- **informazioni generali del documento** le cui informazioni sono riportate di seguito.
  - Lo **stato del documento**;
  - la **versione** attuale;
  - il membro del gruppo **responsabile** per quel documento;
  - **redattori**: i membri del gruppo che hanno contribuito alla redazione
  - i **verificatori**, che identificano i membri che hanno eseguito attività di **Verifica**$^{\mathbf{G}}$ per quel documento;
  - i **destinatari** del documento.

Storico del documento

Nella prima pagina dopo il frontespizio deve essere presente il registro delle modifiche apportate al documento. Le entry della tabella devono essere in ordine cronologico in modo che la prima entry faccia riferimento alla modifica più recente.
Ogni entry deve essere composta da:

1. **versione**;
2. **data della modifica**;
3. **autore della modifica**;
4. **verificatore**, il nome del verificatore che ha verificato quella modifica.

Versione	Data	Autore	Descrizione	Verificatore

Tabella 2: Esempio di tabella dello storico di un documento

Indice

Ogni documento deve presentare un indice delle sezioni e sottosezioni per facilitare la navigazione dei lettori all'interno dello stesso documento.

Struttura dei verbali

I verbali rappresentano i rendiconti di riunioni ufficiali interne, a cui partecipano solo i componenti del gruppo, ed esterne, a cui invece partecipano anche enti esterni al gruppo come impiegati dell'azienda proponente del **Capitolato**$^{\mathbf{G}}$ o i professori del corso. Ogni **Verbale**$^{\mathbf{G}}$ oltre alle sezioni precedentemente elencate deve presentare al suo interno i seguenti elementi nell'ordine in cui sono descritti:

1. **Informazioni generali** per quell'incontro: contiene i dati identificativi dell'incontro quali:

   - la data in cui si è svolto l'incontro;
   - l'orario d'inizio;
   - la durata;
   - il luogo in cui si è svolto;
   - l'elenco dei presenti e degli assenti.

   Queste informazioni sono volte a dare un contesto formale alla riunione.

2. **Ordine del giorno**: riassume i punti principali che si intendono discutere durante l'incontro, predisposti in anticipo dal responsabile/i di progetto durante quel periodo. Questa sezione funge da guida per la conduzione del **Verbale**$^{\mathbf{G}}$.

3. **Discussioni**: riporta in modo sintetico ma chiaro i contenuti emersi durante il confronto tra i partecipanti. Devono essere descritte le osservazioni, le problematiche sollevate e le proposte di soluzione valutate durante l'incontro.

4. **Decisioni**: elenca tutte le risposte che il gruppo si dà per uno specifico punto, incluse quelle riguardanti modifiche ai documenti, scadenze o variazioni organizzative. Ogni decisione deve essere formulata in modo oggettivo e deve avere un codice identificativo univoco che segue il seguente formato:

   ```
   <VI/VE> <fase a cui appartiene> Y.Z
   ```

   dove:

   - **VI/VE**, indica se la decisione fa riferimento ad un **Verbale**$^{\mathbf{G}}$ interno o esterno, seguendo le convenzioni di scrittura stabilite.
   - **fase**, indica la fase di progetto a cui si riferisce;

- **Y**, indica il numero del **Verbale**$^{\mathbf{G}}$ in riferimento a quella revisione;
- **X** indica il numero della decisione interna a quel documento.

5. **To-do**: presenta, sotto forma di tabella, gli incarichi operativi pianificati a seguito della riunione. Ogni **To-do** è identificata da:
   - una descrizione sintetica;
   - Il codice della decisione da cui è stata creata (per ogni decisione ci possono essere 0 o più attività);
   - il numero della issue GitHub di quella attività presente nel GitHub **Project**$^{\mathbf{G}}$.

Convenzioni di scrittura

Le convenzioni di scrittura stabiliscono regole comuni per la redazione di tutti i documenti del progetto.

3.1.4.4 Date

- Le **date** devono essere nel formato **AAAA-MM-GG**.

Le ore hanno 2 formati:

- **Durata**: Per esprimere una durata usare il formato **Hh Mm** (es: 2h 15m)
- **Ora di orologio**: Per indicare che un evento è avvenuto in una specifica ora, usare il formato **HH:MM** (es: 15:45)

Nomi dei file

- Tutti i file devono avere nomi in minuscolo, senza spazi, e le parole devono essere separate da un **underscore (_)**. ¡¡¡¡¡¡¡ HEAD
- Il nome di ogni **Verbale**$^{\mathbf{G}}$ deve iniziare con `Verbale_`, segue il tipo di **Verbale**$^{\mathbf{G}}$ `interno` o `esterno` e la data in cui si è svolto. Un esempio di nominazione corretta è `Verbale_interno_2025-11-05`.
- L'unica eccezione è nel caso in cui usare questo formato generi due verbali con lo stesso nome all'interno della stessa cartella. In tal caso è necessario aggiungere, dopo la data, una breve descrizione del contesto relativo al **Verbale**$^{\mathbf{G}}$, sostituendo agli spazi il simbolo **underscore (_)**. Ad esempio `Verbale_2025-10-28_incontro_c8`.
- I `file PDF` mantengono lo stesso nome del file sorgente `.tex` =======
- Il nome di ogni **Verbale**$^{\mathbf{G}}$ deve iniziare con `Verbale_`, segue il tipo di **Verbale**$^{\mathbf{G}}$ `interno` o `esterno` e la data in cui si è svolto. Un esempio di nominazione corretta è `Verbale_interno_2025-11-05`.

- L'unica eccezione è nel caso in cui usare questo formato generi due verbali con lo stesso nome all'interno della stessa cartella. In tal caso è necessario aggiungere, dopo la data, una breve descrizione del contesto relativo al **Verbale**$^{\mathbf{G}}$, sostituendo agli spazi il simbolo **underscore** (_). Ad esempio `Verbale_2025-10-28_incontro_c8`.
- I `file PDF` mantengono lo stesso nome del file sorgente `.tex`. ¿¿¿¿¿¿ fca8c84bdab39622cee2e4b72

Sigle

Di seguito sono presenti le sigle usate all interno dei documenti.

- **Fasi di progetto**:
  - **C**, Candidatura;
  - **Rtb**$^{\mathbf{G}}$, Requirements and **Technology Baseline**$^{\mathbf{G}}$;
  - **Pb**$^{\mathbf{G}}$, Product **Baseline**$^{\mathbf{G}}$.
- **Tipologie di documenti**:
  - **AdR**, analisi dei requisiti;
  - **NdP**, norme di progetto;
  - **PdQ**, **Piano Di Qualifica**$^{\mathbf{G}}$;
  - **PdP**, **Piano Di Progetto**$^{\mathbf{G}}$;
  - **G**, glossario;
  - **VI**, **Verbale**$^{\mathbf{G}}$ interno;
  - **VE**, **Verbale**$^{\mathbf{G}}$ esterno.

Nomi dei membri

- I nomi dei membri devono essere riportati nel formato: **Nome Cognome** (iniziali maiuscole, nessuna abbreviazione).

3.1.4.5 Modifiche a documenti Per eseguire delle modifiche alla documentazione di progetto, il membro del gruppo è tenuto ad istanziare un nuovo branch a partire da quello di `develop`.
Ogni modifica a un documento deve essere accompagnata da uno scatto del numero di versione. Il membro del gruppo che si occupa delle modifiche è anche tenuto ad aggiornare lo **storico dei documenti** con le informazioni richieste.
Ogni modifica ad un documento deve essere tracciabile all'interno dello stesso tramite lo **storico del documento**, riportando:

- il numero di versione assegnato;

- la data della conclusione della modifica;
- il nome e cognome del membro del team che l'ha eseguita;
- una descrizione concisa, sintetica e formale delle modifiche avvenute;
- il nome del verificatore che è tenuto ad eseguire la **Verifica**$^{\mathbf{G}}$.

3.1.4.6 Aggiornamento del glossario Ogni membro del gruppo, nel momento in cui aggiunge un termine ad un documento che può essere fonte di ambiguità e che non sia già presente nel glossario, è tenuto ad aggiornare il glossario del progetto. Con il termine deve essere anche fornita una sua descrizione esplicativa.

3.1.4.7 Conclusione modifiche Una volta completate le modifiche nel nuovo branch aperto, il membro incaricato procede ad aprire una **Pull Request**$^{\mathbf{G}}$ dal proprio branch di feature verso il branch `develop`. Nella fase di apertura della **Pull Request**$^{\mathbf{G}}$ deve essere compilato il form per la descrizione della **Pull Request**$^{\mathbf{G}}$ con le informazioni richieste. La **Pull Request**$^{\mathbf{G}}$ costituisce la richiesta formale per l'integrazione delle modifiche nel **Repository**$^{\mathbf{G}}$ del gruppo, dopo l'avvenuta approvazione delle modifiche apportate.

3.1.4.8 Verifica$^{\mathbf{G}}$ Il verificatore esamina sia la correttezza formale sia sostanziale del documento.

- Se le modifiche risultano corrette, il verificatore approva la **Pull Request**$^{\mathbf{G}}$.
- Se emergono incongruenze o problemi, il verificatore richiede delle modifiche aggiungendo dei commenti al codice.

Le modifiche richieste devono essere applicate dall'autore originario della **Pull Request**$^{\mathbf{G}}$, che procede aggiornando il branch `develop` fino alla completa risoluzione. I verificatori sono tenuti a controllare che i documenti di ciclo di vita del progetto comprendano gli elementi di struttura elencati di seguito nell' ordine in cui si presentano.

3.1.4.9 Approvazione e Pull Request$^{\mathbf{G}}$ Una volta ottenuta l'approvazione del verificatore, la **Pull Request**$^{\mathbf{G}}$ viene sottoposta alla visione del responsabile, che effettua un ultima revisione formale. Solo il responsabile è autorizzato a procedere con il merge della **Pull Request**$^{\mathbf{G}}$ nel branch `develop`.
Il merge rappresenta l'integrazione definitiva del documento nella versione stabile del progetto.

3.1.4.10 Pubblicazione Con la conclusione del merge, il documento aggiornato diventa parte integrante della **Baseline**$^{\mathbf{G}}$ di progetto. Il branch `main` costituisce, infatti, il riferimento ufficiale per la versione stabile e coerente dei documenti.

3.2. Processo di gestione della configurazione

Il processo di gestione della configurazione ha l'obiettivo di applicare procedure per garantire il controllo sistematico degli artefatti prodotti dal gruppo *BitByBit* durante l'intero ciclo di vita del software.
Esso permette di tracciare le modifiche apportate ai configuration item, garantendo che ogni item venga rilasciato in modo coerente e dopo le opportune verifiche. Inoltre, consente di registrare, classificare e monitorare le richieste di modifica, assicurando che i prodotti di lavoro risultino completi, coerenti e corretti. Infine, supporta una gestione ordinata degli item approvati, curandone l'archiviazione, la conservazione e la distribuzione.

3.2.1 Strumenti a supporto

Per le attività previste, il gruppo BitByBit utilizza gli strumenti elencati di seguito.

- **GitHub**: piattaforma per il versionamento, la gestione dei **Repository**$^{\mathbf{G}}$, delle issue, delle **Pull Request**$^{\mathbf{G}}$ e dei flussi di lavoro collaborativi mediante GitHub Projects e Teams. È inoltre il punto ufficiale di archiviazione degli artefatti approvati.
- **GitHub Actions**: utilizzate per automatizzare la compilazione dei documenti in LaTeX e garantire la **Validazione**$^{\mathbf{G}}$ continua del **Repository**$^{\mathbf{G}}$.
- **Branch protection rules**: utilizzate per impedire modifiche non autorizzate sul branch principale e garantire che ogni integrazione avvenga secondo le regole di **Verifica**$^{\mathbf{G}}$ previste dal gruppo.

3.2.2 Implementazione di processo

Le attività principali del processo di gestione della configurazione sono:

- **identificazione della configurazione**, viene definito uno schema uniforme di versionamento e naming, per identificare ogni **Componente**$^{\mathbf{G}}$ del sistema;
- **controllo della configurazione**, metodologie per stabilire l'identificazione, l'approvazione o rifiuto, la **Verifica**$^{\mathbf{G}}$ e l'eventuale rilascio delle richieste di modifica;
- **registro di stato**, per predisporre registri di gestione e report di stato che mostrino lo stato e la cronologia degli elementi software;
- **valutazione della configurazione**, come determinare la completezza funzionale e fisica delle componenti software con i loro requisiti.

3.2.3 Identificazione della configurazione

Per il versionamento viene adottato il Semantic Versioning SemVer. Ogni documento riporta una versione nel seguente formato.

`X.Y.Z`

Dove:

- **X (major)**: incrementata solo alla pubblicazione ufficiale, corrispondente al merge sul branch `main`;
- **Y (minor)**: incrementata quando vengono effettuare modifiche sostanziali alla struttura o al contenuto del documento;
- **Z (patch)**: incrementata con correzioni minori, refusi o chiarimenti senza impatto tecnico.

Ogni documento di ciclo di vita presenta uno storico delle modifiche con associata la precisa versione. Questa scelta permette una chiara tracciabilità chiara, prevedibilità delle modifiche ed evoluzione controllata degli artefatti documentali.

3.2.4 Controllo della configurazione

Il controllo della configurazione definisce le modalità con cui il gruppo gestisce, valuta e approva tutte le modifiche agli elementi configurabili del progetto.
L'intero ciclo di lavoro è organizzato secondo la pratica del feature branching, secondo la quale ogni sviluppatore deve aprire un nuovo branch per iniziare a lavorare. Quando si è terminato il lavoro allora le modifiche possono essere integrate nel branch principale, nel nostro caso identificato su develop.
Per la nomenclatura del branch di `feature` ogni membro del gruppo è tenuto a seguire la nomenclatura di seguito esposta.

`Task/<descrizione delle modifiche>_<documento soggetto alle modifiche>`

Dove:

- `<descrizione delle modifiche>`, comprende una breve descrizione delle modifiche applicate in quel branch, senza utilizzare spazi e le parole separate da **trattini** (-);
- `<documento soggetto alle modifiche>`, sigla per il documento a cui avvengono apportate le modifiche.

3.2.4.1 Gestione issue e pianificazione del lavoro

Ogni modifica nasce dall'apertura di una issue GitHub, che costituisce il punto di riferimento per la tracciabilità dell'attività.
Al momento della creazione, la issue deve essere dotata di un titolo descrittivo e di una spiegazione chiara dell'obiettivo. Il responsabile assegna inoltre priorità, **Milestone**^{G}, iterazione (che corrisponde allo **Sprint**^{G} in cui l'attività deve essere completata) e l'assegnatario, selezionato in base al ruolo che il membro ricopre in quel momento. Label aggiuntive consentono una categorizzazione accurata del lavoro.
La pianificazione complessiva è supportata dalle GitHub **Project**^{G} Board, che permettono di consultare product **Backlog**^{G} e **Sprint**^{G} **Backlog**^{G} tramite viste personalizzate, semplificando la gestione dei carichi di lavoro e l'avanzamento delle attività.

3.2.4.2 Creazione della Pull Request^{G}

Una volta completata l'attività descritta nella issue GitHub, il membro incaricato apre una **Pull Request**^{G} dal branch di feature al `branch develop`.
Ogni **Pull Request**^{G} deve utilizzare il template comune del gruppo, che contiene le sezioni necessarie per descrivere contesto, modifiche introdotte, issue collegate e controlli effettuati. La descrizione del template presente nel file `pull_request_template.md`, deve essere compilato e adattato ogni volta, così da velocizzare il lavoro di revisione e rendere uniformi tutte le richieste. Il membro deve indicare tramite il campo `Reviewers` il singolo membro incaricato del compito di verificare quella modifica.

3.2.4.3 Verifica^{G} e revisione della Pull Request^{G}

All'apertura della **Pull Request**^{G} viene richiesta automaticamente la **Verifica**^{G} ai membri del team verificatori:

- se la **Verifica**^{G} produce esito positivo, la **Pull Request**^{G} viene approvata dal verificatore incaricato;
- se sono presenti dei problemi o miglioramenti da applicare, il verificatore richiede delle modifiche, tramite l'opzione di GitHub `request changes`, aggiungendo un commento di quello che è il problema trovato e di una possibile soluzione se è presente. Successivamente la persona che ha aperto la **Pull Request**^{G} si impegna a eseguire le correzioni e a notificare il completamento dell'azione per avere una **Verifica**^{G} il prima possibile.

3.2.4.4 Integrazione nella Baseline^{G}

Quando la **Pull Request**^{G} viene approvata dal verificatore, il responsabile esegue la revisione. Si impegna ad eseguire il merge nel branch `develop`. A questo punto, la modifica entra ufficialmente in **Baseline**^{G} e diventa parte del progetto.

3.2.5 Registro dello stato della configurazione

Lo stato aggiornato degli elementi di configurazione è tracciato tramite:

- le issue aperte, in corso e chiuse;
- le **Milestone**$^{\mathbf{G}}$ attive e completate;
- la cronologia delle **Pull Request**$^{\mathbf{G}}$;
- le versioni ufficiali dei documenti pubblicate sul branch `main`.

GitHub Projects funge da registro organizzato delle attività, offrendo una rappresentazione visiva dello stato corrente.

3.2.6 Valutazione della configurazione

L'**attività di valutazione della configurazione** ha lo scopo di garantire che gli elementi software sviluppati siano completi, coerenti e conformi ai **requisiti approvati**.
Il gruppo BitByBit assicura tale controllo attraverso un tracciamento sistematico dei requisiti, che consente di collegare ogni modifica e ogni **Componente**$^{\mathbf{G}}$ alla motivazione tecnica o funzionale che la giustifica. Durante lo sviluppo, i membri si impegnano inoltre a mantenere una chiara corrispondenza tra codice e requisiti, anche tramite annotazioni e riferimenti interni.
La valutazione viene svolta in modo continuativo, non solo nelle fasi conclusive, così da individuare rapidamente deviazioni dal design e garantire che:

- tutti i requisiti software siano effettivamente soddisfatti;
- non vengano introdotti comportamenti non richiesti;
- ogni parte del sistema risulti coerente con l'**Architettura**$^{\mathbf{G}}$ definita.

3.3. Processo di accertamento della qualità

Il processo di **accertamento della qualità** ha lo scopo di garantire che la qualità del **prodotto software** e dei **processi di ciclo di vita** adottati sia conforme ai requisiti specificati e ai piani prestabiliti. L'accertamento della qualità consente di fornire evidenze oggettive sul fatto che i sistemi software sviluppati siano idonei allo scopo per cui sono stati richiesti e soddisfino le aspettative degli **Stakeholder**$^{\mathbf{G}}$ coinvolti.

3.3.1 Implementazione di processo

Il gruppo *BitByBit* ha identificato come principale attività del processo di accertamento della qualità la redazione del documento di **Piano Di Qualifica**$^{\mathbf{G}}$. Tale documento definisce gli obiettivi quantitativi di qualità relativi sia al prodotto software sia ai processi di

ciclo di vita adottati, e raccoglie le misurazioni oggettive effettuate durante lo svolgimento del progetto.
Le misurazioni consistono nella quantificazione, mediante l'utilizzo di metriche, di specifici attributi dei prodotti software e dei processi di ciclo di vita. Per ciascuna metrica vengono inoltre individuati, nel **Piano Di Qualifica**$^{\mathbf{G}}$, dei valori ottimali o soglie di riferimento, al fine di identificare tempestivamente eventuali comportamenti anomali o scostamenti rispetto agli obiettivi di qualità prefissati.

3.3.2 Accertamento della qualità di prodotto

L'attività di **accertamento della qualità di prodotto** ha lo scopo di assicurare che i prodotti software e la documentazione correlata siano forniti in conformità ai termini e ai requisiti specificati con l'acquirente.
Nel **Piano Di Qualifica**$^{\mathbf{G}}$ vengono definite specifiche metriche di qualità di prodotto. Le misurazioni eseguite vengono confrontate con i valori ottimali stabiliti per ciascuna metrica, al fine di valutare in modo oggettivo la qualità del software realizzato.
Le metriche di prodotto costituiscono un supporto decisionale per determinare se il software implementato soddisfa i criteri di qualità richiesti ed è pertanto idoneo al rilascio.

3.3.3 Accertamento della qualità di processo

L'attività di **accertamento della qualità di processo** ha lo scopo di assicurare che i processi di ciclo di vita siano eseguiti in conformità agli standard e alle procedure definiti per ciascun processo.
Nel **Piano Di Qualifica**$^{\mathbf{G}}$ vengono formalizzate le metriche di processo e i relativi valori ottimali stabiliti dal gruppo. Le misurazioni effettuate vengono successivamente confrontate con tali valori di riferimento, consentendo di valutare l'efficacia e l'adeguatezza dei processi adottati.
Le metriche di processo sono utilizzate dal responsabile di progetto come strumento di supporto decisionale per individuare eventuali criticità e valutare la necessità di apportare modifiche o miglioramenti ai processi in uso.

3.4. Processo di Verifica$^{\mathbf{G}}$

Il processo di **Verifica**$^{\mathbf{G}}$ è l'insieme delle attività volte ad accertare che ogni prodotto del ciclo di vita (codice, documenti, **Architettura**$^{\mathbf{G}}$) sia conforme ai requisiti e agli standard di qualità stabiliti. L'obiettivo è rispondere alla domanda ≪Did we build the product right?≫ (Abbiamo costruito il prodotto nel modo corretto?), intercettando difetti ed errori prima che questi si propaghino alle fasi successive.

3.4.1 Strategia e Workflow

Il gruppo *BitByBit* integra la **Verifica**$^{\mathbf{G}}$ direttamente nel flusso di sviluppo quotidiano (Git Flow), adottando la politica del **Continuous Verification**. Le regole operative sono le seguenti:

- **Policy "No Verify, No Merge"**: nessun artefatto può essere integrato nel ramo di sviluppo principale (`develop`) se non ha superato con successo i controlli di **Verifica**$^{\mathbf{G}}$.
- **Pull Request**$^{\mathbf{G}}$ **come Quality Gate**: ogni modifica deve essere proposta tramite **Pull Request**$^{\mathbf{G}}$. In questa sede avviene la **Verifica**$^{\mathbf{G}}$ (automatica o manuale) e solo l'esito positivo autorizza il merge.
- **Rami Stabili**: il ramo `main` è protetto e accoglie solo versioni del software che hanno superato l'intero ciclo di test (inclusi quelli di sistema).

3.4.2 Verifica$^{\mathbf{G}}$ della Documentazione

La documentazione è soggetta esclusivamente ad **analisi statica**, ovvero controlli che non richiedono l'esecuzione. Per garantire la qualità dei documenti, il gruppo valuta due metodologie:

- **Walkthrough**: revisione guidata dall'autore. Utile per l'allineamento del team ma dispendiosa.
- **Ispezione (Inspection)**: lettura autonoma da parte del verificatore basata su liste di controllo (*checklist*).

Tra le due, il gruppo *BitByBit* adotta come standard l'**Ispezione**. L'uso di checklist predefinite garantisce infatti un processo ripetibile, oggettivo e meno soggetto a errori umani rispetto alla revisione libera.

3.4.2.1 Strumenti per la Verifica$^{\mathbf{G}}$ della documentazione Poiché il gruppo ha adottato la tecnica dell'**Ispezione** basata su checklist, la **Verifica**$^{\mathbf{G}}$ della documentazione non richiede strumenti di automazione complessi, ma si avvale delle piattaforme di sviluppo e collaborazione:

- **GitHub**: è lo strumento principale. L'interfaccia delle **Pull Request**$^{\mathbf{G}}$ permette ai verificatori di commentare righe specifiche del documento, richiedere modifiche e approvare il lavoro solo quando la checklist di **Verifica**$^{\mathbf{G}}$ è soddisfatta.
- **VS Code**: viene utilizzato dai verificatori per il controllo locale dei file sorgente (LaTeX/Markdown), permettendo di individuare errori di sintassi o formattazione non visibili dall'interfaccia web prima di dare l'approvazione.

3.4.3 Verifica[G] del Codice Software

La **Verifica[G]** del software si articola in due momenti distinti: l'analisi del codice sorgente (statica) e l'esecuzione dei test (dinamica).

3.4.3.1 Analisi Statica Automatizzata Prima di essere eseguito, il codice viene analizzato da strumenti automatici (Linter e Static Analyzers) integrati nella pipeline di CI/CD. Questo passaggio serve a rilevare:

- violazioni delle norme di codifica (indentazione, naming);
- potenziali **Bug[G]** logici o di sicurezza (es. variabili non utilizzate, cicli infiniti);
- calcolo delle metriche di complessità.

3.4.3.2 Analisi Dinamica (Testing) L'analisi dinamica **Verifica[G]** il comportamento del software tramite la sua esecuzione. Per essere valido, ogni test deve garantire due proprietà: **ripetibilità** (stesso output a parità di input) e **automatizzabilità** (esecuzione senza intervento umano). Il gruppo struttura i test secondo il modello a V, coprendo i seguenti livelli:

1. **Test di Unità**: **Verifica[G]** delle singole componenti (funzioni/classi) in isolamento.
    - *White-box*: test strutturali che mirano alla copertura dei percorsi logici interni (Statement Coverage).
    - *Black-box*: test funzionali che verificano l'output dato un input, analizzando valori validi, non validi e casi limite (Boundary Analysis).
2. **Test Di Integrazione[G]**: **Verifica[G]** della collaborazione tra unità già testate.
    - *Top-Down*: integrazione dalle componenti di alto livello a quelle basse (richiede **Stub**).
    - *Bottom-Up*: integrazione dalle componenti base verso l'interfaccia (richiede **Driver**).
3. **Test di Sistema**: **Verifica[G]** del sistema completo rispetto ai requisiti funzionali e prestazionali mappati nell'Analisi dei Requisiti.
4. **Test di Regressione**: Riesecuzione automatica della suite di test esistente ad ogni modifica, per garantire che le nuove feature non abbiano corrotto funzionalità preesistenti.
5. **Test di Accettazione**: Verifiche formali finali, condotte da utenti o **Stakeholder[G]**, per confermare che un software soddisfi i requisiti aziendali e funzionali prima del rilascio in produzione.

3.4.3.3 Strumenti per la Verifica[G] del codice

La definizione puntuale degli strumenti tecnici per la **Verifica[G]** del codice (es. framework di unit testing, linter specifici per linguaggio, strumenti di analisi statica e coverage) sarà effettuata e dettagliata durante lo svolgimento della **Product Baseline[G]** (**Pb[G]**), in coerenza con le tecnologie che verranno consolidate in quella fase.

3.4.4 Classificazione dei Test

Per garantire l'ordine nel **Piano Di Qualifica[G]**, il gruppo adotta una nomenclatura univoca per il tracciamento:

`T-[ID]-[Tipo]`

Dove **ID** è un progressivo numerico e **Tipo** indica la categoria: **U** (Unità), **I** (Integrazione), **S** (Sistema), **A** (Accettazione). Lo stato di ogni test viene tracciato come: **I** (Implementato), **NI** (Non Implementato) o **S** (Superato).

3.5. Processo di Validazione[G]

Il processo di **Validazione[G]** costituisce l'ultimo livello di controllo qualitativo prima del rilascio formale del prodotto. A differenza della **Verifica[G]**, che si configura come un controllo tecnico continuo sul codice, la **Validazione[G]** ha lo scopo di accertare che il sistema software realizzato dal gruppo *BitByBit* sia pienamente conforme alle aspettative e ai bisogni reali dell'azienda proponente **Miriade**.

3.5.1 Obiettivi del processo

Il gruppo adotta questo processo per rispondere in modo affermativo alla domanda «Did we build the right product?», ovvero «Abbiamo costruito il prodotto giusto?». Gli obiettivi perseguiti sono:

- dimostrare la totale copertura dei requisiti concordati nel documento di **Analisi dei Requisiti**;
- accertare l'idoneità del software nel contesto operativo reale tramite la simulazione di scenari d'uso;
- ottenere l'approvazione formale per il rilascio del prodotto.

3.5.2 Tracciamento come strumento di Validazione[G]

La prima fase operativa del processo consiste nell'assicurare che ogni funzionalità richiesta sia effettivamente presente nel sistema. Per fare ciò il gruppo utilizza la **matrice di tracciamento** come strumento di controllo principale. L'attività prevede di mappare

univocamente ogni **Requisito**^{G} (funzionale, di qualità o vincolo) ai relativi test di **Validazione**^{G}. Questo approccio metodologico garantisce che non venga sviluppato codice non richiesto e, soprattutto, che ogni **Requisito**^{G} concordato con **Miriade** sia verificabile tramite un test specifico. Un **Requisito**^{G} privo di tracciamento verso un test è considerato non validabile e quindi non accettabile.

3.5.3 Esecuzione dei test di accettazione

La **Validazione**^{G} si concretizza con l'esecuzione dei **Test di Accettazione** (UAT - User Acceptance Testing). In questa fase il gruppo non si limita a verificare l'assenza di errori, ma simula scenari di utilizzo reali (User Scenarios) per valutare il comportamento del sistema dal punto di vista dell'**Utente**^{G} finale. Il superamento di questi test certifica che il prodotto supporta efficacemente i processi lavorativi richiesti. In caso di fallimento di un test di accettazione il prodotto viene respinto e rimandato alla fase di sviluppo per le necessarie correzioni, seguite da una regressione per confermare la risoluzione delle non conformità.

3.5.4 Strumenti a supporto della Validazione^{G}

Analogamente alla **Verifica**^{G}, anche il processo di **Validazione**^{G} si avvale di strumenti specifici per gestire la complessità del tracciamento e l'esecuzione dei test.

3.5.4.1 Strumenti per il tracciamento La gestione manuale della matrice di tracciamento è soggetta ad errori umani. Per questo motivo il gruppo utilizza:

- **Script di automazione**: verranno impiegati script interni (o tool esterni integrati) capaci di scansionare i file sorgente della documentazione e del codice per generare e aggiornare automaticamente le tabelle di tracciamento (Requisiti-Fonti e Requisiti-Test).

3.5.4.2 Strumenti per l'esecuzione e il reporting Per l'effettuazione pratica dei test di accettazione:

- **Ambiente di Deployment**^{G}: i test verranno eseguiti su un'istanza del software deployata in un ambiente il più possibile simile a quello di produzione, accessibile tramite Browser Web standard (Chrome/Firefox).
- **GitHub Issues**: qualsiasi anomalia o comportamento inatteso riscontrato durante la **Validazione**^{G} (test fallito) verrà tracciato aprendo una Issue dedicata su GitHub, etichettata specificamente come **Bug**^{G} di **Validazione**^{G} per distinguerla dai difetti tecnici di sviluppo.

4. Processi organizzativi del ciclo di vita

I processi organizzativi di ciclo di vita rappresentano l'insieme delle attività che il gruppo BitByBit ha scelto di adottare per strutturare e supportare in modo sistematico tutti i propri progetti.
Questi processi forniscono un quadro stabile entro cui i progetti vengono sviluppati e permettono di mantenere strumenti, procedure e competenze costantemente aggiornati. Allo stesso tempo, l'esperienza maturata nei progetti può portare all'introduzione di miglioramenti nei processi stessi, contribuendo così all'evoluzione dell'intera organizzazione.
Nelle presenti *Norme di Progetto*, i processi organizzativi sono suddivisi nelle seguenti categorie, che verranno approfondite nelle sezioni successive:

- **Gestione di processo**;
- **Infrastruttura**;
- **Processo di miglioramento**;
- **Processo di formazione**.

4.1. Gestione dei processi

Il processo di **gestione dei processi** definisce le attività organizzative necessarie per pianificare, coordinare e monitorare il lavoro del gruppo. Esso stabilisce i compiti da svolgere, i ruoli incaricati di eseguirli e le modalità di comunicazione interna ed esterna. L'obiettivo è garantire che le attività procedano in modo efficace e controllato.

4.1.1 Strumenti a supporto

Il gruppo *BitByBit* utilizza le funzionalità delle **GitHub Organizations** per organizzare il lavoro e gestire i ruol. Ogni team corrisponde a un ruolo specifico e include solo i membri incaricati in quel periodo, facilitando la rotazione programmata dei ruoli.
Sul branch principale (`main`) solo i membri del team dei Responsabili possono eseguire push o merge, mentre tutti gli altri membri propongono modifiche tramite **Pull Request**^{G}. La funzione `CODEOWNERS` assegna automaticamente ai Verificatori la responsabilità di revisione dei documenti, garantendo che tutte le modifiche rispettino il workflow e i controlli qualitativi previsti.
Questa configurazione assicura una chiara separazione dei ruoli, un flusso di modifica strutturato e un efficace supporto automatizzato alla gestione della configurazione.

4.1.2 Implementazione di processo

Le attività previste dal processo di gestione dei processi sono articolate come segue.

- **Inizializzazione e definizione degli ambiti**, il responsabile del progetto assieme ai membri del team identifica i requisiti per l'attività da svolgere. Si procede poi alla valutazione della fattibilità verificando disponibilità delle risorse, adeguatezza delle competenze e tempistiche.
- **Pianificazione**, si definisce il piano operativo del processo, includendo la descrizione delle attività, i prodotti software da generare e la stima dei tempi necessari. Il piano specifica inoltre risorse richieste, assegnazione delle responsabilità, analisi dei rischi, misure di qualità da adottare, costi stimati e infrastruttura di supporto.
- **Esecuzione e controllo**, le attività pianificate vengono attuate. Il responsabile di processo produce report periodici sullo stato di avanzamento, destinati sia all'uso interno sia all'acquirente. Il responsabile monitora l'aderenza al piano e interviene per risolvere eventuali scostamenti.
- **Revisione e valutazione**, perché un prodotto entri in **Baseline**$^{\mathbf{G}}$, le modifiche apportate devono essere prima verificate da un verificatore e successivamente revisionate dal responsabile. Quest'ultimo accerta che i prodotti software e i piani risultino conformi ai requisiti stabiliti, valutando l'esito dei compiti completati e garantendo qualità e coerenza con gli obiettivi del processo.
- **Chiusura**, al completamento delle attività e dei prodotti previsti, si **Verifica**$^{\mathbf{G}}$ che i criteri concordati con l'acquirente siano soddisfatti. Il responsabile controlla la completezza dei risultati e dei report.

4.1.3 Ruoli

Ruoli	Compiti e responsabilità
Responsabile	È la figura che nel momento governa il team. Identifica le attività che sono state eseguite nello **Sprint**$^{\mathbf{G}}$ terminato e quelle che non sono state portate a termine. Durante l'evento di **Sprint**$^{\mathbf{G}}$ Planning stabilisce ed inserisce nello **Sprint**$^{\mathbf{G}}$ **Backlog**$^{\mathbf{G}}$ le attività da eseguire nello **Sprint**$^{\mathbf{G}}$ successivo, individuando i costi, i rischi e i membri del gruppo in base ai loro ruoli. Rappresenta il gruppo all'esterno, interfacciandosi con gli enti esterni quali i professori e l'azienda proponente.

Amministratore	È l'amministratore di sistema (Sys Admin), si occupa della gestione dell'infrastruttura utilizzata per eseguire le attività dei processi di ciclo di vita. Ha il compito di inserire nell'infrastruttura gli strumenti e le tecnologie per l'attuazione del **Way Of Working**^{G}. Deve accertarsi che l'infrastruttura si trovi in uno stato stabile e quindi risolvere eventuali problematiche che possono derivarne. Deve redigere il presente documento.
Analista	Il compito dell'analista è quello di identificare i requisiti del progetto, interpretando le necessità degli utenti finali in modo da ottenere una corretta definizione delle funzionalità richieste. Ha il compito di redigere il documento di analisi dei requisiti
Progettista	Il ruolo di progettista si occupa di tradurre i requisiti identificati dagli analisti in qualcosa di implementabile, e ne supervisiona l'implementazione da parte dei programmatori.
Programmatore	È la persona cui spetta il compito di tradurre le unità di design in codice funzionante. In concomitanza con il codice deve anche implementare i test automatici per la **Verifica**^{G} del corretto funzionamento del codice.
Verificatore	Ha il compito di assicurarsi che le attività vengano svolte correttamente seguendo i processi di ciclo di vita. Si occupa di eseguire test approfonditi e revisioni sul software. Ha il compito di verificare le modifiche apportate ai vari documenti ed eventualmente esponendo le criticità che presentano; deve perciò avere un'approfondita conoscenza dei requisiti per ogni processo di vita.

Tabella 3: Descrizione dei ruoli del progetto e delle loro responsabilità

4.1.4 Struttura dei team GitHub

L'organizzazione interna del gruppo (si veda sezione 4.1) è stata implementata tramite la funzione nativa di GitHub Teams, che consente di suddividere i membri in sotto–gruppi corrispondenti ai ruoli previsti dal progetto (si veda Sezione 4.1.3). Ogni ruolo forma un team composto da uno a più membri, così da distribuire responsabilità e carico di **Verifica**^{G}.
Le figure che interessano le **Pull Request**^{G} sono principalmente due:

- verificatori, scelti tra i membri del team con responsabilità di **Verifica**^{G} in quello **Sprint**^{G}, devono controllare la correttezza formale delle modifiche;

- responsabile, una volta ottenuta l'approvazione dei verificatori, è incaricato di completare l'operazione di merge sul `branch main` nel rispetto delle regole di protezione del branch.

Questa struttura garantisce che ogni modifica venga validata da più persone e che il processo rispetti il processo di qualità.

4.1.5 Coordinamento

Il coordinamento del gruppo è garantito attraverso riunioni periodiche, interne ed esterne, finalizzate a mantenere un allineamento continuo sullo stato dei lavori, sui problemi emersi e sulle decisioni organizzative che emergono in fase di lavoro.

4.1.5.1 Riunioni Le riunioni previste dal processo di gestione dei processi sono suddivise in due categorie principali:

- **Interne**: si svolgono tra i soli membri del gruppo e hanno luogo principalmente su Discord. Ogni **Sprint**$^{\mathbf{G}}$ ha una durata di due settimane e prevede tre incontri:
  - uno a fine **Sprint**$^{\mathbf{G}}$, comprendente **Sprint**$^{\mathbf{G}}$ Review **Sprint**$^{\mathbf{G}}$ Retrospective e **Sprint**$^{\mathbf{G}}$ Planning. In questa sede il **Responsabile Verifica**$^{\mathbf{G}}$ lo stato delle attività concluse, analizza eventuali problemi riscontrati e pianifica il lavoro del successivo **Sprint**$^{\mathbf{G}}$;
  - due durante lo svolgimento dello **Sprint**$^{\mathbf{G}}$, in cui vengono trattati i temi dell'ordine del giorno e ciascun membro espone le attività completate, quelle ancora in corso e le difficoltà incontrate.

  Per ogni riunione vengono assegnati specifici ruoli:
  - **Scriba**: i membri incaricati di prendere appunti durante gli incontri;
  - **Portavoce**: identificato nel responsabile, che modera e coordina la discussione;
  - **Redattore**: responsabile della stesura del **Verbale**$^{\mathbf{G}}$ e del caricamento nel **Repository**$^{\mathbf{G}}$.
- **Esterne**: coinvolgono anche figure esterne quali i docenti o l'azienda proponente **Miriade S.r.l.**. Queste riunioni vengono svolte generalmente tramite Google Meet. La frequenza concordata è di un incontro al mese, con la possibilità di richiederne ulteriori qualora fossero necessari chiarimenti o approfondimenti specifici.

L'esito di ogni incontro, interno o esterno, deve essere registrato mediante la redazione di un **Verbale**$^{\mathbf{G}}$, secondo le modalità previste dal processo di documentazione.

4.2. Infrastruttura

Il processo di infrastruttura ha lo scopo di stabilire e mantenere l'infrastruttura necessaria per eseguire gli altri processi di ciclo di vita.
L'infrastruttura, che sia hardware o software, comprende strumenti, standard e le strutture per lo sviluppo, il funzionamento e la manutenzione.

4.2.1 Attività previste

Il processo di infrastruttura consiste delle seguenti attività:

- **implementazione**;
- **istituzione**;
- **manutenzione**.

4.2.2 Implementazione

L'infrastruttura operativa del progetto viene implementata in modo da soddisfare pienamente i requisiti dei processi che la utilizzano.
Ogni tecnologia impiegata viene scelta in base alla sua capacità di supportare efficacemente le attività del ciclo di vita, migliorare la comunicazione tra i membri del team e garantire qualità, tracciabilità e mantenibilità dei prodotti generati.
La seguente sezione descrive gli strumenti adottati.

Strumento	Descrizione
Whatsapp	Piattaforma di messaggistica istantanea utilizzata per comunicazioni informali che non richiedono tracciamento. Scelta per la rapidità, l'immediatezza e la diffusione tra i membri del team. Facilita il coordinamento veloce su aspetti operativi di basso impatto.
Discord	Strumento di comunicazione tramite canali vocali e testuali, utilizzato per le riunioni interne che necessitano di tracciamento e organizzazione strutturata. Scelto per la stabilità, la possibilità di creare canali dedicati.
Google Meet	Piattaforma sincrona per videoconferenze utilizzata negli incontri con l'azienda proponente, in quanto standard scelto dal **Committente**$^{\mathbf{G}}$.
Google Chat	Piattaforma di messaggistica organizzata per stanze e conversazioni strutturate. Proposta dall'azienda proponente come canale asincrono per le comunicazioni.

Git	Sistema di versionamento distribuito utilizzato per tracciare l'evoluzione del progetto, gestire i rami di sviluppo e mantenere l'integrità dello storico. Scelto per la diffusione nel settore e la capacità di supportare efficacemente lavori collaborativi.
GitHub	Piattaforma cloud-based di hosting per **Repository**$^{\mathbf{G}}$ Git, utilizzata per la gestione collaborativa del codice. Scelta per le funzionalità integrate (issue tracking, **Pull Request**$^{\mathbf{G}}$, branching rules, code review) che supportano il processo di sviluppo.
LaTeX	Linguaggio di markup per la produzione di documenti tecnici e accademici. Scelto per l'elevata qualità tipografica e per la possibilità di definire template uniformi, garantendo coerenza e mantenibilità della documentazione.
Overleaf	Piattaforma di scrittura collaborativa online per file LaTeX. Utilizzata nelle fasi iniziali per l'apprendimento della sintassi e per la stesura rapida, prima di passare al workflow locale/Git.
Google Mail	Servizio di posta elettronica utilizzato come canale asincrono formale di comunicazione verso l'esterno. Scelto per la sua affidabilità e per l'integrazione con gli altri strumenti Google.
Google Drive	Spazio di archiviazione cloud utilizzato dal gruppo per raccogliere documenti correlati, appunti delle riunioni, fogli di calcolo in cui il gruppo registra le informazioni informali riguardanti i periodi di **Sprint**$^{\mathbf{G}}$. Scelto per la centralizzazione dei contenuti, l'accessibilità da qualsiasi dispositivo e la possibilità di modifica concorrente.
Google Sheets	Foglio di calcolo online utilizzato per i dati riguardanti i singoli **Sprint**$^{\mathbf{G}}$ e rappresentarli in appositi grafici per una visualizzazione migliore. Scelto per la collaborazione sincrona e la facilità di aggiornamento in tempo reale.
Google Docs	Editor di testi online per la stesura collaborativa di contenuti non ufficiali come bozze interne per i verbali. Scelto per la semplicità d'uso e la modifica concorrente.
Google Presentazioni	Strumento per la realizzazione di presentazioni utilizzate nelle esposizioni verso professori e **Stakeholder**$^{\mathbf{G}}$. Scelto per la capacità di collaborazione e per i template professionali disponibili.
Script Python	Script sviluppati per automatizzare attività ricorrenti nel processo, come operazioni di generazione automatica o **Validazione**$^{\mathbf{G}}$. Scelti per la flessibilità del linguaggio e la facilità di integrazione con altre tecnologie.

Lucidchart	Strumento per la creazione di diagrammi, utilizzato per la redazione dei diagrammi UML dei casi d'uso. Scelto per la possibilità di collaborazione in tempo reale, che ha consentito a più membri del gruppo di lavorare simultaneamente.

Tabella 4: Strumenti utilizzati per l'infrastruttura

4.2.3 Istituzione

La presente attività di **istituzione dell'infrastruttura** ha lo scopo di documentare la configurazione dell'infrastruttura per facilitarne la manutenzione e la modifica da parte dei membri del gruppo BitByBit.

4.2.3.1 Discord Discord è utilizzato come strumento per gli incontri del team e come archivio centralizzato delle risorse condivise, tra cui file e link relativi allo sviluppo e al materiale del corso. Il gruppo BitByBit ha organizzato la piattaforma tramite più canali testuali, suddivisi per tipologia di contenuto (risorse di sviluppo, file e chat generale), e un canale vocale dedicato allo svolgimento delle riunioni.

4.2.3.2 Git Per garantire coerenza operativa tra tutti i membri del team, è previsto un insieme minimo di configurazioni locali che ciascun **Componente**^{G} deve applicare al proprio ambiente Git.

- Ogni membro deve definire il proprio nome e indirizzo e-mail in Git, così che ogni commit sia attribuito in modo univoco e riconoscibile all'interno dello storico del progetto.
- Il membro deve assicurarsi che Git sia correttamente installato nel proprio ambiente, e che sia possibile accedere alla **Repository**^{G} remota tramite le chiavi SSH configurate.

4.2.3.3 GitHub La configurazione del **Repository**^{G} è definita per supportare il workflow adottato e garantire qualità, tracciabilità e sicurezza delle modifiche.
Al suo interno è possibile trovare varie directories, di seguito verranno descritte le più rilevanti.

- **.github** contiene la configurazione delle automazioni del progetto. Include il file `CODEOWNERS`, che definisce i responsabili delle diverse directory e consente a GitHub di assegnare automaticamente i revisori per le **Pull Request**^{G}. Contiene inoltre il file `latex_template.txt`, che rappresenta il template comune a tutti i documenti

del gruppo BitByBit, e `template_pull_request.md`, il modello utilizzato per la creazione automatica delle **Pull Request**[G].

- **.github/workflows** contiene i workflow GitHub Actions dedicati all'integrazione continua e all'automazione delle attività del **Repository**[G].
- **.github/scripts** contiene gli script Python incaricati dell'automazione dei processi, come la compilazione dei file `.tex` e l'inserimento automatico del template LaTeXdurante la creazione di nuovi documenti.

- **website** raccoglie gli elementi necessari alla pubblicazione del sito del progetto, tra cui fogli di stile e risorse grafiche.
- **resources** funge da archivio centralizzato delle risorse condivise, includendo immagini, loghi e altri materiali comuni utilizzati nel **Repository**[G].
- **src** contiene tutti i file sorgente `.tex`, ovvero il codice LaTeXprodotto dal gruppo BitByBit, organizzato in sezioni corrispondenti alle diverse fasi del progetto.
- **docs** replica la struttura della directory **src**, ma raccoglie i file `PDF` generati a partire dai documenti LaTeX.

4.3. Processo di miglioramento

Il processo di miglioramento è il processo tramite il quale vengono analizzati, valutati, misurati e migliorati i processi del ciclo di vita.

4.3.1 Scopo

Durante tutta la fase di lavoro il team si pone l'obiettivo di seguire i principi del **miglioramento continuo**. Lo scopo principale è identificare problematiche ricorrenti in processi, ruoli e situazioni dell'intero ciclo di vita in modo da poterli evitare in futuro. Dopo aver identificato tali punti critici, il team si impegna a discuterne insieme per trovare spunti di miglioramento da trasformare in **processi standardizzati**.

4.3.2 Descrizione

Il miglioramento continuo rappresenta un ciclo iterativo che consente al team di adattarsi in maniera flessibile alle esigenze in evoluzione del progetto, assicurando progressi costanti e un incremento dell'efficienza operativa. Ogni fase del ciclo di vita del progetto, dalla definizione delle attività alla redazione dei documenti, riflette questo approccio.

4.3.3 Implementazione

Durante l'esecuzione delle attività e la stesura dei documenti, il team applica sistematicamente il principio del **miglioramento continuo**. In caso di riscontro di un problema, viene convocata una riunione interna in cui tutti i membri del team discutono l'argomento, valutando soluzioni possibili e alternative. L'applicazione pratica del miglioramento viene registrata nella sezione **"Decisioni"** del corrispondente **Verbale**^{G} interno, comprensiva della giustificazione della scelta e del consenso tra tutti i membri del team.
Qualora la decisione comporti modifiche tecniche, è prevista la creazione di una issue specifica, secondo quanto stabilito dalle Norme di Progetto. La issue dovrà riportare l'etichetta "enhancement" e contenere tutte le informazioni necessarie a descrivere il cambiamento, le motivazioni alla base della decisione e l'impatto previsto sulle attività o sul software.
Inoltre, tutte le problematiche rilevate e le soluzioni adottate continuano a essere documentate e analizzate, garantendo una gestione consapevole e mirata delle criticità lungo l'intero processo di sviluppo. Questo approccio strutturato evita il ripetersi di errori già affrontati e favorisce il consolidamento di pratiche efficaci e condivise.

4.4. Processo di formazione

Il processo di formazione è il processo che garantisce che il team abbia le conoscenze tecniche necessarie per svolgere il lavoro che deve portare a termine.

4.4.1 Scopo

Lo scopo del processo di formazione è garantire che ogni membro del team sia adeguatamente formato e competente per lo svolgimento delle attività di progetto. La formazione è considerata un fattore essenziale per assicurare la corretta acquisizione, sviluppo, utilizzo e manutenzione del prodotto software, nonché per supportare efficacemente le attività di gestione e di natura tecnica.

4.4.2 Descrizione

Il processo di formazione prevede l'analisi dei requisiti di progetto al fine di individuare le competenze necessarie, i ruoli coinvolti e i livelli di conoscenza richiesti. Sulla base di tale analisi, vengono definite le tipologie di formazione e le categorie di personale che necessitano di istruzione. La formazione viene pianificata e avviata sin dalle prime fasi del progetto, in modo da garantire la disponibilità tempestiva di risorse qualificate durante l'intero ciclo di vita del software.

4.4.3 Aspettative formative

Il processo di formazione è finalizzato a mettere ciascun membro del gruppo nelle condizioni di:

- Ottenere dimestichezza con Git, GitHub e altri strumenti di controllo versione in uso durante il progetto.
- Acquisire competenze di scrittura con LaTeX per la redazione di documenti con formattazione e impaginazione adeguati.
- Acquisire competenze con i linguaggi di programmazione, le piattaforme, i programmi, le librerie, gli strumenti e gli ambienti di sviluppo necessari per la realizzazione del prodotto software concordato.

4.4.4 Piano di formazione

Ciascun membro del team è tenuto a studiare in maniera autonoma gli argomenti d'interesse. L'intero team stabilisce cosa studiare durante gli incontri interni, creando issue dedicate assegnate a ciascun membro del team. Ogni membro è tenuto ad avere una visione d'insieme delle tecnologie e delle metodologie necessarie per il completamento del progetto, ma ha completa libertà per quanto riguarda i metodi di apprendimento che preferisce utilizzare. Ciascun membro è tenuto ad approfondire argomenti specifici sulla base dei compiti che gli sono stati assegnati. Al completamento di tali compiti e al termine di ogni **Sprint**[G], i membri devono confrontarsi condividendo in maniera sintetica ciò che hanno imparato relativamente ai compiti che hanno portato a termine o sui quali hanno fatto progressi significativi. Questo approccio mira ad alleggerire il carico globale di studio, a garantire una maggiore velocità esecutiva e a promuovere una formazione meno teorica e più incentrata sulla comprensione globale del sistema sul quale si lavora.

5. Metriche

Le seguenti metriche compongono il manuale delle metriche che il gruppo ha identificato per questo specifico progetto. Le metriche identificate dipendono dal preciso software che si deve sviluppare e dalle tecnologie che vengono utilizzate.

5.1. Nomenclatura

La nomenclatura utilizzata per le metriche è la seguente:

`M-<tipo>X`

dove:

- `M`: indica la sigla per **Metrica**;

- `<tipo>`: indica che aspetto misura la metrica;
  - **PD** per Prodotto;
  - **PC** per Processo;
- `X`: un numero incrementale per identificare unicamente la metrica. Il conteggio è separato per le 2 tipologie.

5.2. Metriche di Prodotto

La qualità del software dipende da un insieme di attributi che caratterizzano il comportamento, la struttura e l'evolvibilità del sistema nel tempo.
Gli attributi di qualità considerati nel progetto sono i seguenti:

- **complessità (complexity)**, rappresenta il grado di articolazione logica e strutturale del sistema;
- **comprensibilità (understandability)**, indica la facilità con cui il software può essere letto e compreso;
- **manutenibilità (maintainability)**, rappresenta la capacità del prodotto software di essere modificato, corretto o esteso;
- **efficienza (efficiency)**, indica la capacità del sistema di fornire le funzionalità richieste utilizzando in modo ottimale le risorse disponibili;
- **affidabilità (reliability)**, rappresenta la capacità del sistema di mantenere un livello di prestazioni accettabile quando utilizzato in condizioni specificate.

Le metriche di prodotto adottate nel progetto sono suddivise nelle seguenti due categorie:

- **Metriche dinamiche**, raccolte mediante misurazioni effettuate su un programma in esecuzione. Tali metriche supportano la valutazione dell'efficienza e dell'affidabilità del sistema;
- **Metriche statiche**, raccolte mediante misurazioni effettuate su rappresentazioni del sistema, quali codice sorgente, progetto o documentazione. Esse supportano la valutazione della complessità, della comprensibilità e della manutenibilità del sistema o dei suoi componenti.

5.2.1 Metriche statiche

5.2.1.1 Requisiti obbligatori soddisfatti

- **Codice:** `M-PD1`
- **Formula:** $\frac{\text{\# di requisiti obbligatori soddisfatti}}{\text{\# di requisiti obbligatori totali}} * 100$

- **Descrizione:** fornisce un'indicazione del grado di requisiti obbligatori che sono stati implementati nel prodotto software fino a quel momento.

5.2.1.2 Requisiti desiderabili soddisfatti

- **Codice:** `M-PD2`
- **Formula:** $\frac{\text{\# di requisiti desiderabili soddisfatti}}{\text{\# di requisiti desiderabili totali}} * 100$
- **Descrizione:** fornisce un'indicazione del grado di requisiti desiderabili che sono stati implementati nel prodotto software fino a quel momento.

5.2.1.3 Requisiti opzionali soddisfatti

- **Codice:** `M-PD3`
- **Formula:** $\frac{\text{\# di requisiti opzionali soddisfatti}}{\text{\# di requisiti opzionali totali}} * 100$
- **Descrizione:** fornisce un'indicazione del grado di requisiti opzionali che sono stati implementati nel prodotto software fino a quel momento.

5.2.1.4 Defect Density (DD)

- **Codice:** `M-PD4`
- **Formula:** $DD = \frac{\text{\# di difetti}}{KLOC}$
- **Descrizione:** misura quanti difetti vengono trovati in 1000 linee di codice (**KLOC**, **Kilo Lines Of Code**). Migliora la qualità del codice e la sua mantenibilità.

5.2.1.5 Cyclomatic Complexity

- **Codice:** `M-PD5`
- **Formula:** $E - N + 2P$
- **Descrizione:** misura la complessità di un programma contandone i percorsi linearmente indipendenti. Fornisce un indice sulla complessità logica del codice.
  - E = numero di archi nel grafo
  - N = numero di nodi nel grafo
  - P = numero di componenti connesse ad ogni arco

5.2.1.6 Length of code

- **Codice:** `M-PD6`
- **Formula:** # di linee di codice per una **Componente**$^{\mathbf{G}}$ software
- **Descrizione:** misura la dimensione di un **Componente**$^{\mathbf{G}}$ software. Più è grande una **Componente**$^{\mathbf{G}}$ software maggiore sarà la possibilità di sviluppare errori.

5.2.1.7 Depth of Conditional Nesting (DCN)

- **Codice:** `M-PD7`
- **Formula:** $DCN = \max(\#$ di strutture annidate$)$
- **Descrizione:** misura la profondità massima di annidamento delle strutture condizionali (`if`, `else if`, `switch`) presenti nel codice. Un maggiore annidamento rende difficile la comprensione del codice.

5.2.2 Metriche dinamiche

5.2.2.1 Defect Leakeage

- **Codice:** `M-PD8`
- **Formula:** $\frac{\#\text{ difetti dopo il rilascio}}{\#\text{ difetti trovati durante fase di test}} * 100$
- **Descrizione:** Tiene traccia di quanti difetti vengono trasferiti in produzione dopo i test.

5.2.2.2 Defect Removal Efficiency (DRE)

- **Codice:** `M-PD9`
- **Formula:** $DRE = \frac{\text{difetti trovati prima del rilascio}}{\text{difetti prima del rilascio + difetti dopo}} * 100$
- **Descrizione:** mostra la percentuale di difetti rilevati e rimossi prima del rilascio. Aiuta a misurare l'efficacia del processo di accertamento della qualità.

5.2.2.3 Bug$^{\mathbf{G}}$ Rate per Test Case

- **Codice:** `M-PD10`
- **Formula:** $\frac{\#\ \textbf{Bug}^{\mathbf{G}}\text{ individuati}}{\#\text{ test case eseguiti}} \cdot 100$
- **Descrizione:** misura il numero medio di difetti rilevati. La metrica fornisce un'indicazione dell'efficacia dei test e della qualità del software nelle fasi di test.

5.3. Metriche di processo

5.3.1 Fornitura

5.3.1.1 Budget at Completion (BAC)

- **Codice:** `M-PC1`
- **Formula:** BAC = budget totale pianificato per il progetto
- **Descrizione:** rappresenta il costo totale pianificato del progetto al momento della definizione del piano. Rappresenta una metrica di riferimento per altre misurazioni.

5.3.1.2 Planned Value (PV)

- **Codice:** `M-PC2`
- **Formula:** PV = (BAC) $\cdot$ (% di lavoro pianificata nello **Sprint**$^{\mathbf{G}}$)
- **Descrizione:** rappresenta il preventivo per i costi del lavoro che si intende eseguire durante uno **Sprint**$^{\mathbf{G}}$ di progetto.

5.3.1.3 Actual Cost (AC)

- **Codice:** `M-PC3`
- **Formula:** AC = costo effettivo sostenuto in uno **Sprint**$^{\mathbf{G}}$
- **Descrizione:** rappresenta i costi effettivamente sostenuti per completare il lavoro svolto durante uno **Sprint**$^{\mathbf{G}}$ di progetto.

5.3.1.4 Earned Value (EV)

- **Codice:** `M-PC4`
- **Formula:** $EV = \frac{\text{\# di issue completate}}{\text{\# di issue pianificate}} * PV$
- **Descrizione:** rappresenta il valore del lavoro realmente completato rispetto a quanto pianificato. Consente di valutare l'avanzamento effettivo del progetto indipendentemente dai costi sostenuti.

5.3.1.5 Schedule Performance Index (SPI)

- **Codice:** `M-PC5`
- **Formula:** $SPI = \frac{\text{EV}}{\text{PV}}$
- **Descrizione:** misura l'efficacia della gestione del tempo. Esso confronta la quantità di lavoro effettivamente completata con la quantità di lavoro pianificata. Un valore $= 1$ indica che il progetto è in linea con il piano stabilito, un valore < 1 evidenzia un ritardo.

5.3.1.6 Cost Performance Index (CPI)

- **Codice:** `M-PC6`
- **Formula:** $CPI = \frac{\text{EV}}{\text{AC}}$
- **Descrizione:** misura l'efficienza con cui il progetto utilizza il proprio budget. Un valore ≥ 1 indica che si sta rispettando il budget prefissato.

5.3.1.7 Estimate at Completion (EAC)

- **Codice:** `M-PC7`
- **Formula:** $EAC = \frac{\text{BAC}-\text{EV}}{\text{CPI}}$
- **Descrizione:** stima il costo totale del progetto al completamento, assumendo che l'andamento futuro segua l'efficienza attuale. Viene confrontato con il Budget at Completion per individuare potenziali scostamenti.

5.3.1.8 Estimate to Complete (ETC)

- **Codice:** `M-PC8`
- **Formula:** $ETC = EAC - AC$
- **Descrizione:** misura il costo previsto per completare il lavoro rimanente del progetto escludendo i costi già sostenuti.

5.3.2 Verifica[G]

5.3.2.1 Code Coverage

- **Codice:** `M-PC9`
- **Formula:** $CC = \frac{\text{\# linee di codice testate}}{\text{\# linee di codice totali}} * 100$
- **Descrizione:** misura la percentuale di codice soggetto a test.

5.3.2.2 Requirements Coverage (RC)

- **Codice:** `M-PC10`
- **Formula:** $\frac{\text{\# requisiti testati}}{\text{\# requisiti totali}} * 100$
- **Descrizione:** Indica la percentuale di requisiti che hanno test case corrispondenti. Garantisce che tutti i requisiti documentati siano verificati.

5.3.3 Documentazione

5.3.3.1 Indice di Gulpease

- **Codice:** `M-PC11`
- **Formula:** $89 + \frac{300 \cdot (\text{numero di frasi}) - 10 \cdot (\text{numero di lettere})}{\text{numero di parole}}$
- **Descrizione:** indice di leggibilità tarato su testi in lingua italiana. In generale risulta che testi con un indice:
  - **inferiore a 80** sono difficili da leggere per chi ha la **Licenza**$^{\mathbf{G}}$ elementare
  - **inferiore a 60** sono difficili da leggere per chi ha la **Licenza**$^{\mathbf{G}}$ media
  - **inferiore a 40** sono difficili da leggere per chi ha un diploma superiore

5.3.3.2 Document Coverage Ratio (DCR)

- **Codice:** `M-PC12`
- **Formula:** $\frac{\text{\# sezioni documentate}}{\text{\# sezioni previste dallo standard o dal piano}} * 100$
- **Descrizione:** misura il grado di completezza della documentazione prodotta rispetto alle sezioni previste dai piani e dagli standard adottati. Supporta il processo di documentazione e l'accertamento della conformità agli standard di progetto.

5.3.4 Accertamento della qualità

5.3.4.1 Satisfied Metrics

- **Codice:** `M-PC13`
- **Formula:** $MS = \frac{\text{\# metriche soddisfatte}}{\text{\# metriche totali}} * 100$
- **Descrizione:** misura la percentuale di metriche che sono state soddisfatte durante uno **Sprint**$^{\mathbf{G}}$. Una metrica viene considerata soddisfatta quando presenta un valore accettabile.

5.3.5 Gestione dei processi

5.3.5.1 Sprint$^{\mathbf{G}}$ Commitment Reliability (SCR)

- **Codice:** `M-PC14`
- **Formula:** $\frac{\text{\# issue completate nello } \mathbf{Sprint}^{\mathbf{G}}}{\text{\# issue pianificate}} * 100$
- **Descrizione:** misura la capacità del team di rispettare gli impegni pianificati per lo **Sprint**$^{\mathbf{G}}$. Supporta la valutazione della qualità della pianificazione e della stabilità del processo di sviluppo.