Norme di progetto

Gruppo	Alt+F4
Data	27 Febbraio 2025
Versione	v0.14

Registro modifiche

Versione	Data	Autore/i	Verificatore/i	Descrizione
v0.14	27 Febbraio 2025	Marko Peric	Francesco Savio	Scrittura sezioni: StarUML, Requisiti Software
v0.13	9 Gennaio 2025	Enrico Bianchi, Matteo Eghosa	Marko Peric	Scrittura sezione: Processo di accertamento qualità
v0.12	8 Gennaio 2025	Pedro Leoni	Marko Peric	Modifica sezione: Casi d'uso, Comandi personalizzati, Ambienti personalizzati
v0.11	7 Gennaio 2025	Marko Peric	Enrico Bianchi	Modifica sezione: Creazione delle tabelle, Aggiunta sezione: Creazione dei grafici
v0.10	31 Dicembre 2024	Marko Peric	Francesco Savio	Modifica sezioni: Github action controllo delle ore di lavoro, Github action del controllo ortografico dei documenti LaTeX
v0.9	26 Dicembre 2024	Marko Peric	Enrico Bianchi	Modifica sezione: Documentazione, introdotta descrizione struttura documento Piano di Progetto
v0.8	23 dicembre 2024	Francesco Savio	Marko Peric	Modifica sezione: Processi di supporto, Processo di infrastruttura
v0.7	22 dicembre 2024	Enrico Bianchi, Guirong Lan	Marko Peric	Scrittura sezione: Processo di Verifica

v0.6	21 dicembre 2024	Guirong Lan	Marko Peric	Scrittura sezione: Descrizione Processi Primari, Processo di acquisizione, Processo di fornitura
v0.5	11 dicembre 2024	Pedro Leoni	Francesco Savio, Enrico Bianchi	Scrittura sezione: Documentazione
v0.4	11 dicembre 2024	Pedro Leoni	Francesco Savio, Enrico Bianchi	Scrittura sezione: Gestione della configurazione
v0.3	11 dicembre 2024	Pedro Leoni, Francesco Savio	Francesco Savio, Enrico Bianchi	Scrittura sezione: Processo di infrastruttura
v0.2	11 dicembre 2024	Pedro Leoni, Enrico Bianchi	Francesco Savio, Enrico Bianchi	Scrittura sezione: Gestione di progetto
v0.1	11 dicembre 2024	Pedro Leoni, Francesco Savio, Matteo Eghosa	Francesco Savio, Enrico Bianchi	Scrittura sezione: Processo di sviluppo

Indice

1 Processi primari

I processi primari definiti dalla norma $ISO/IEC\ 12207_G$ sono essenziali per la realizzazione di un progetto e sono strettamente legati all'acquisizione, alla realizzazione, alla gestione, all'operazione e al ritiro del software durante l'intero ciclo di vita. Un processo primario consiste in un insieme di attività fondamentali e interconnesse, e variano a seconda della tipologia di processo. Questi processi sono cruciali nelle fasi operative del ciclo di vita del software e sono determinanti per garantire una produzione, manutenzione e dismissione in modo efficace. La norma distingue cinque principali processi primari, ognuno dei quali comprende attività chiave per lo sviluppo e la gestione del software, dalla sua concezione fino al suo ritiro. I cinque processi sono:

1. **Processo di acquisizione**;
2. **Processo di fornitura**;
3. **Processo di sviluppo**;
4. **Processo di operazione**;
5. **Processo di manutenzione**;

Nota bene: Per la questione didattica il processo di manutenzione non verrà implementato.

1.1 Processo di acquisizione

Il processo di acquisizione, secondo la norma ISO/IEC 12207, comprende l'insieme delle attività che un'organizzazione, cioè il $Fornitore_G$, deve svolgere per ottenere un sistema, un prodotto software o un servizio software. Esso prevede l'identificazione dei requisiti, la selezione di un Fornitore e un $Proponente_G$, la stipula di un contratto, la supervisione della fornitura e, infine, l'accettazione e il completamento del prodotto o servizio acquisito. Dunque, per il nostro gruppo, il processo di acquisizione consiste nelle seguenti attività principali:

- **Iniziazione**;
- **Studio di fattibilità**;
- **Selezione del capitolato**;
- **Preparazione della candidatura**;
- **Accettazione della candidatura**;

1.1.1 Iniziazione

Il Fornitore effettua un'analisi preliminare dei capitolati d'appalto, la comprensione degli obiettivi e dei requisiti iniziali per lo sviluppo, includendo il confronto e la discussione con i proponenti.

1.1.2 Studio di fattibilità

Il Fornitore analizza i capitolati per identificare eventuali punti critici e valuta le idee ricevute dai proponenti.

1.1.3 Selezione del capitolato

Il Fornitore sceglie e prepara il capitolato d'appalto che meglio soddisfa le sue aspettative.

1.1.4 Preparazione della candidatura

Il Fornitore prepara e definisce la redazione dei documenti necessari:

- **Lettera di candidatura**;
- **Stima dei costi**;
- **Assunzione d'impegni**;
- **Valutazione dei capitolati**.

Revisionando il contenuto per correggere eventuali errori e, se necessario, aggiorna il documento prima della presentazione ai $Committenti_G$.

1.1.5 Accettazione della candidatura

L'approvazione della soluzione selezionata e la conseguente formalizzazione del contratto spettano ai committenti, i quali, con la loro approvazione, rendono ufficiale il contratto.

1.2 Processo di fornitura

Il processo di fornitura, definito dalla norma ISO/IEC 12207, comprende le attività svolte dal Fornitore e inizia con la presentazione di una proposta al Proponente o la stipula di un contratto. Nel nostro caso, il processo riguarda la Proponente Zucchetti S.p.A. e si avvia al termine del processo di acquisizione, ossia al completamento della fase di accettazione della candidatura. Il processo prevede la pianificazione e

l'organizzazione delle risorse, delle procedure e dei piani necessari per la gestione del progetto, fino alla consegna del sistema, prodotto o servizio software. Questo processo è fondamentale per garantire che il software soddisfi i requisiti del cliente, sia di alta qualità e venga realizzato rispettando tempi e costi concordati. L'obiettivo principale è allineare costantemente le aspettative dell'acquirente con i risultati ottenuti durante l'esecuzione del progetto. Dunque, per il nostro gruppo, il processo si articola nelle seguenti attività principali:

- **Pianificazione**;
- **Esecuzione e controllo**;
- **Revisione e valutazione**;
- **Consegna e completamento**;

1.2.1 Pianificazione

Il Fornitore definisce obiettivi, risorse e procedure necessari per l'esecuzione del progetto, identificando i requisiti per la gestione, la misurazione della qualità e lo svolgimento delle attività. Questa fase sta principalmente nella redazione del Piano di Progetto, che pianifica l'utilizzo delle risorse e documenta i risultati attesi.

1.2.2 Esecuzione e controllo

Il Fornitore attua le attività pianificate nel Piano di Progetto, rispettando le norme stabilite. Inoltre, viene effettuato un controllo continuo sullo stato di avanzamento e sulla gestione delle risorse, garantendo la rendicontazione rispetto agli obiettivi prefissati.

1.2.3 Revisione e valutazione

Il Fornitore definisce criteri e procedure per la revisione ed esegue le operazioni in conformità a tali criteri. L'obiettivo è verificare che il progetto soddisfi i requisiti e rispetti gli standard stabiliti.

1.2.4 Consegna e completamento

Consegna del progetto, verifica finale e accettazione da parte dell'acquirente.

1.2.5 Contatti con la Proponente dell'azienda

La Proponente, Zucchetti S.p.A., fornisce l'indirizzo email del proprio rappresentante per la comunicazione asincrona e per la pianificazione di videochiamate su Google Meet con il Fornitore. Le comunicazioni tra Proponente e Fornitore riguardano vari aspetti del progetto, tra cui la raccolta dei requisiti, la raccolta di feedback sui risultati ottenuti e le indicazioni sull'avanzamento del progetto. Durante il corso del progetto, che ha carattere didattico, la Proponente assume il ruolo di cliente, mantenendo un ampio grado di libertà per il Fornitore, il quale è responsabile della realizzazione del prodotto. Per ogni colloquio con l'azienda Proponente, ovvero per ogni videochiamata tramite Google Meet, sarà redatto un Verbale Esterno che riassume i punti chiave discussi durante l'incontro.

1.2.6 Documentazione fornita

Di seguito viene descritta la documentazione che il gruppo rende disponibile alla Proponente Zucchetti S.p.A. e ai committenti, Prof. Tullio Vardanega e Prof. Riccardo Cardin.

1.2.6.1 Piano di progetto

Il Piano di Progetto è un documento redatto dal responsabile del progetto, che fornisce una visione dettagliata della gestione e dell'organizzazione del gruppo di lavoro. La sua funzione principale è quella di pianificazione, monitoraggio e controllo delle attività, al fine di garantire il raggiungimento degli obiettivi nei tempi e nei costi stabiliti. Inoltre, il piano include l'analisi e la gestione dei rischi, nonché la pianificazione, il preventivo e il consuntivo di ciascun sprint, monitorando costantemente l'avanzamento del progetto e le risorse utilizzate. Il documento è suddiviso nelle seguenti sezioni:

- **Introduzione**: Una breve descrizione dello scopo del documento e delle varie sezioni che lo compongono.
- **Analisi dei rischi**: Riguarda l'identificazione dei potenziali rischi che potrebbero sorgere durante il corso del progetto, i quali potrebbero causare ritardi o ostacoli nella sua progressione. Vengono inoltre sviluppate strategie di prevenzione per evitare che tali rischi si manifestino, nonché strategie di mitigazione per ridurne l'impatto nel caso in cui si verificassero, al fine di garantire la continuità del progetto.
- **Stima dei costi**: Calcolo delle risorse necessarie per il completamento del progetto, che viene aggiornato a ogni sprint.
- ***Milestone*$_G$ principali**: Sezione dedicata all'indicazione delle milestone fondamentali e delle baseline di riferimento del progetto.

- **Primo periodo**: Rappresenta la fase iniziale del gruppo.
- ***Sprint*$_G$**: Gli sprint rappresentano fasi del progetto in cui vengono definiti obiettivi specifici e attività da completare. Durante ogni sprint, il gruppo si impegna a raggiungere tali obiettivi. Inoltre, si stabiliscono una revisione retrospettiva e la data per lo sprint successivo. La sottosezione si articola nelle seguenti parti:
  1. **Obiettivi**: Definisce gli obiettivi specifici dello sprint.
  2. **Pianificazione**: Presenta il preventivo dei costi e delle risorse previste per lo sprint.
  3. **Consuntivo**: Riporta il resoconto dei costi e delle risorse effettivamente utilizzati.
  4. **Retrospettiva**: Riporta la conclusione dello sprint e propone miglioramenti sui punti deboli emersi.
  5. **Aggiornamento risorse rimaste**: Aggiorna la situazione delle risorse disponibili per il progetto.

1.2.6.2 Analisi dei requisiti

L'analisi dei requisiti è un documento redatto dall'analista che fornisce una visione chiara delle richieste e delle aspettative dell'azienda Proponente. Va a includere un elenco dettagliato delle funzionalità da sviluppare e implementare, nonché i casi d'uso che definiscono le interazioni tra il sistema e l'utente. Il documento è suddiviso nelle seguenti sezioni:

1. **Introduzione**: Una breve descrizione dello scopo del documento e delle varie sezioni che lo compongono.
2. **Descrizione del prodotto**: Descrive gli obiettivi del prodotto, le sue funzioni principali e le caratteristiche.
3. **Casi d'uso**: Indica tutti i casi d'uso individuati dal gruppo durante analisi.
4. **Requisiti**: Elenco completo dei requisiti del prodotto, organizzato per categorie e con riferimento alle fonti da cui proviene il tracciamento.

1.2.6.3 Piano di Qualifica

Il Piano di Qualifica è un documento che definisce le attività del verificatore nel progetto, stabilendo le strategie e gli approcci per garantire la qualità del prodotto software in fase di sviluppo. Redatto dal verificatore, questo documento descrive le modalità di verifica e validazione. Tutti i membri del team si baseranno su questo documento

per garantire il raggiungimento della qualità desiderata. Il documento è suddiviso nelle seguenti sezioni:

1. **Introduzione**: Una breve descrizione dello scopo del documento e delle varie sezioni che lo compongono.
2. **Obiettivi di qualità**: Questa sezione presenta i valori accettabili e gli ambiti per le metriche definite dal team, le metriche sono divise:Qualità di processo, Qualità di prodotto, Qualità per obiettivo.
3. **Metodologie di testing**: Include tutti i test necessari per verificare che il prodotto rispetti i requisiti specificati.
4. **Cruscotto di valutazione della qualità**: Vengono scritte tutte le attività di verifica effettuate e le problematiche emerse durante lo sviluppo del software.

1.2.6.4 Glossario

Il Glossario è un elenco dettagliato e organizzato di tutti termini, acronimi e definizioni utilizzati nella documentazione. L'obiettivo principale di questo documento è fornire una comprensione chiara dei concetti e dei termini specifici impiegati nel progetto, garantendo una comunicazione coerente e precisa tra tutti i membri del gruppo e con gli $Stakeholder_G$. In questo modo, si facilita il lavoro collaborativo e si assicura un allineamento efficace su linguaggio e significati durante l'intero ciclo di vita del progetto.

1.2.6.5 Lettera di presentazione

La Lettera di Presentazione è il documento con cui il gruppo comunica la propria candidatura alle revisioni di avanzamento $Requirements\ and\ Technology\ Baseline_G$ e $Product\ Baseline_G$. Nel primo caso essa include informazioni sui repository di documentazione e codice sorgente, il riferimento al $Proof\ of\ Concept_G$ e un aggiornamento sugli impegni con la stima del preventivo "a finire".

1.3 Processo di sviluppo

Il processo di sviluppo riguarda l'insieme di attività necessarie per produrre e implementare il software, assicurandosi che risponda ai requisiti definiti e sia di qualità adeguata.

1.3.1 Analisi dei requisiti

L'analisi dei requisiti è una attività cruciale nel ciclo di vita del software, in cui vengono raccolte, analizzate e documentate le esigenze degli stakeholder per definire chiaramente cosa il sistema dovrà fare. Questa attività produce il documento di analisi dei requisiti descritto nella sezione Documentazione.

1.3.1.1 Requisiti

I requisiti vengono categorizzati in:

1. **Requisiti funzionali**.

   Rappresentano ciò che un sistema deve fare per soddisfare le esigenze degli utenti o degli stakeholders.

2. **Requisiti qualitativi**.

   I requisiti di qualità (requisiti non funzionali) descrivono caratteristiche qualitative del sistema, come prestazioni, sicurezza, usabilità e manutenibilità.

3. **Di vincolo**.

   I requisiti di vincolo rappresentano limitazioni o condizioni obbligatorie che influenzano lo sviluppo, l'implementazione o l'operatività del sistema. A differenza dei requisiti funzionali o di qualità, non descrivono cosa deve fare il sistema, ma come deve essere realizzato rispettando regole o restrizioni specifiche.

4. **Prestazionali**.

   I requisiti prestazionali sono un tipo specifico di requisiti non funzionali che definiscono le aspettative relative alle prestazioni del sistema. Alcuni esempi sono: la velocità, il tempo di risposta, la capacità di carico, e altre caratteristiche misurabili legate all'efficienza.

1.3.1.2 Casi d'uso

I casi d'uso forniscono una descrizione dettagliata delle funzionalità del sistema dal punto di vista degli utenti, delineando come il sistema risponda a specifiche azioni o scenari. In particolare un caso d'uso rappresenta un uso completo del sistema dalla prospettiva dell'utente. I casi d'uso vengono rappresentati tramite una combinazione di descrizione e diagramma UML. Durante la definizione di un caso d'uso possono emergere nuove informazioni che richiedono una modifica dei casi d'uso definiti in precedenza. La definizione dei casi d'uso non è quindi lineare.

1.3.1.2.1 Descrizione

Nella Tabella 2 viene mostrato il template per la descrizione dei casi d'uso. Ogni descrizione di caso d'uso deve includere:

1. **Identificativo**.

   `UC-<ID caso principale>.<ID sotto caso>`.

   Dove:

   (a) `<ID caso principale>`: identificativo numerico del caso d'uso principale (numero crescente).

   (b) `<ID sotto caso>`: identificativo numerico del sotto caso d'uso (presente solo se si tratta di un sotto caso d'uso).

2. **Requisito utente**.

   Requisito utente a cui fa riferimento il diagramma.

3. **Pre-condizioni**.

   Stato iniziale del sistema o vincoli necessari per l'attivazione del caso d'uso.

4. **Post-condizioni**.

   Stato finale del sistema dopo un esecuzione normale(priva di errori) del caso d'uso.

5. **Attori**. Un attore è un entità esterna(umana o meno) al sistema che interagisce con il sistema per iniziare o per contribuire al compimento dell'esecuzione di un caso d'uso. Gli attori si dividono in:

   (a) **Attori primari**: attori principali che partecipano al caso d'uso, solitamente contengono gli attori che iniziano l'interazione e/o sono i destinatari delle informazioni in output del caso d'uso.

   (b) **Attore secondario**: altri attori coinvolti che non sono protagonisti nell'esecuzione del caso d'uso(es. Sistemi esterni).

6. **Trigger**.

   Evento eseguito da un attore che attiva il caso d'uso.

7. **Scenario principale**.

   La sequenza di step importanti che vengono portati a termine in un esecuzione normale del caso d'uso.

8. **Scenari alternativi**.

   Eventuali deviazioni o eccezioni rispetto allo scenario principale.

<identificativo caso d'uso>: <descrizione caso d'uso>	
Requisito utente	<link requisito utente>
Pre-condizioni	• <pre condizione> • <pre condizione>
Post-condizioni	• <post condizione> • <post condizione>
Attore principale	<attore principale>
Attori secondari	<lista attori secondari>
Trigger	<trigger>
Casi d'uso inclusi	<lista casi d'uso inclusi>
Caso d'uso base	<caso d'uso di base>
Scenario principale	1. <1^a azione> 2. <2^a azione> 3. <<include:id>>
Scenari secondari	1.1 <condizione branch dalla 1^a azione> a. 1^a azione del scenario secondario b. 2^a azione del scenario secondario 2.1 <condizione branch dalla 2^a azione> a. 1^a azione del scenario secondario b. 2^a azione del scenario secondario

Tabella 2: Template casi d'uso.

1.3.1.2.2 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono strumenti grafici utilizzati nell'ingegneria del software per rappresentare in modo intuitivo le interazioni tra gli utenti (attori) e un sistema. Basati sul linguaggio UML (Unified Modeling Language), questi diagrammi illustrano i principali scenari operativi del sistema e aiutano a identificare e chiarire i requisiti funzionali. Ogni diagramma è composto da:

- **Sistema**: delimita i confini del sistema software, indicando quali funzionalità sono incluse e quali sono esterne a esso. Il sistema viene rappresentato nei diagrammi come mostrato in Figura 1.

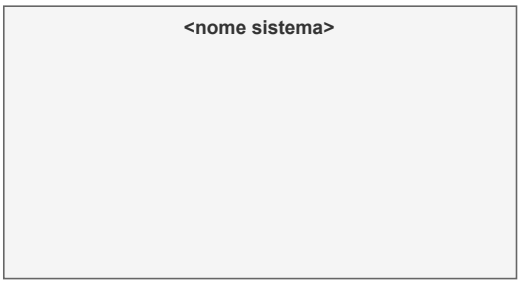

Figura 1: Rappresentazione del sistema in UML.

- **Attori**: rappresentano soggetti che interagiscono con il sistema software. Gli attori possono essere persone, altri applicativi o dispositivi che utilizzano le funzionalità del sistema. Un attore viene rappresentato nei diagrammi usando una delle due notazioni mostrate in Figura 2 posta al di fuori del sistema.

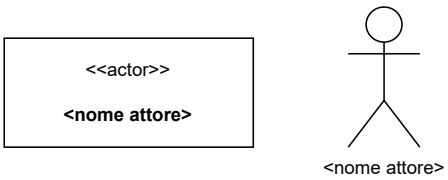

Figura 2: Rappresentazione degli attori in UML.

- **Casi d'uso**: rappresentano le funzionalità o i servizi offerti dal sistema che producono un risultato di valore per un attore. Ogni caso d'uso descrive una sequenza specifica di interazioni tra gli attori e il sistema. Ogni caso d'uso viene rappresentato nei diagrammi usando la notazione mostrata in Figura 3 posta all'interno del sistema.

Figura 3: Rappresentazione caso d'uso in UML.

- **Relazioni**.

  Sono delle notazioni che permettono di rendere più modulari i diagrammi di casi d'uso e di rendere chiara l'interazione degli attori con il sistema.

  – **Associazione**: è la relazione fondamentale che collega un attore a un caso d'uso a cui esso partecipa. Uno stesso attore può partecipare a `N` casi d'uso. Le associazioni vengono rappresentate in UML come mostrato in Figura 4.

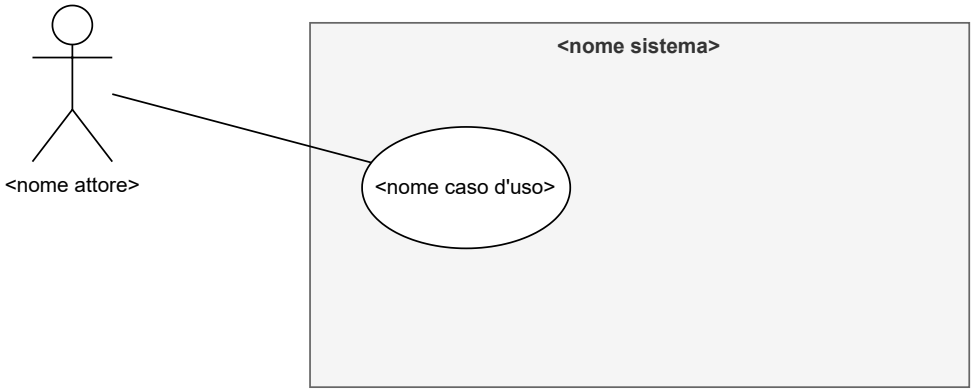

Figura 4: Rappresentazione associazione in UML.

  – **Inclusione**: rappresenta una dipendenza in cui il comportamento del caso d'uso incluso è incorporato ogni volta che viene eseguito il caso d'uso base. Il caso d'uso incluso contiene funzionalità che sono riutilizzate in più casi d'uso principali, permettendo di evitare la duplicazione di comportamenti

comuni. L'inclusione viene rappresentata in UML come mostrato in Figura 5.

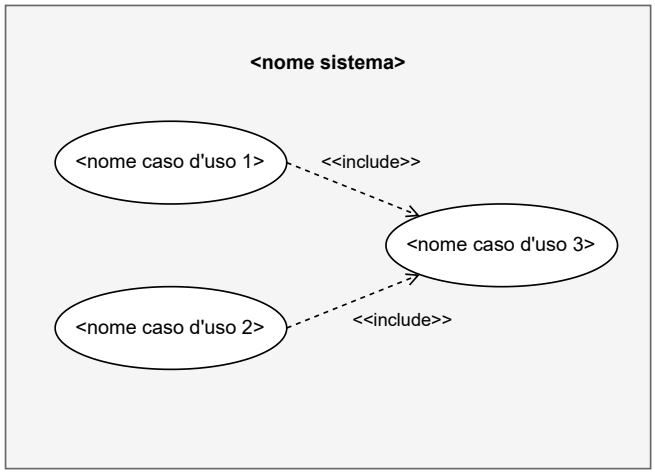

Figura 5: Rappresentazione inclusione in UML.

Come si può vedere nella Tabella 2 è necessario:

1. Indicare il caso d'uso incluso come valore della colonna "Casi d'uso inclusi".
2. Indicare lo step dello scenario principale o delle estensioni che utilizza il caso d'uso incluso. Questo viene fatto tramite la sintassi `include::<nome caso d'uso>`.

– **Estensione**: mostra una relazione in cui un caso d'uso esteso aumenta le funzionalità del caso d'uso principale. Il caso d'uso esteso viene eseguito solo sotto determinate condizioni, interrompendo l'esecuzione del caso d'uso principale. L'estensione viene rappresentata in UML come mostrato in Figura 6.

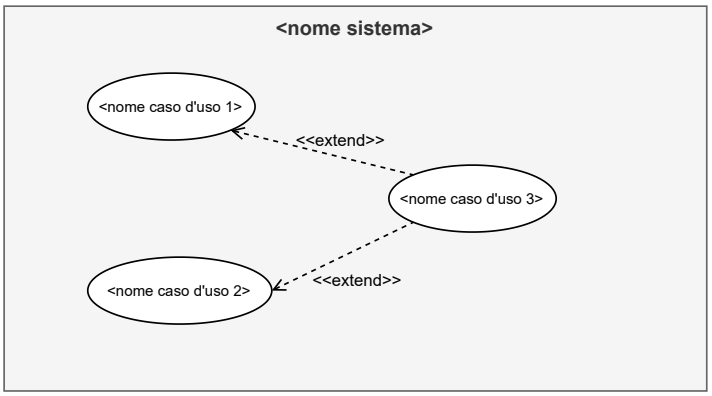

Figura 6: Rappresentazione estensione in UML.

– **Generalizzazione**: rappresenta una relazione in cui un caso d'uso figlio può aggiungere funzionalità o modificare il comportamento di un caso d'uso genitore. Tutte le funzionalità definite nel caso d'uso genitore si mantengono nel caso d'uso figlio se queste non vengono ridefinite. La generalizzazione viene rappresentata in UML come mostrato in Figura 7.

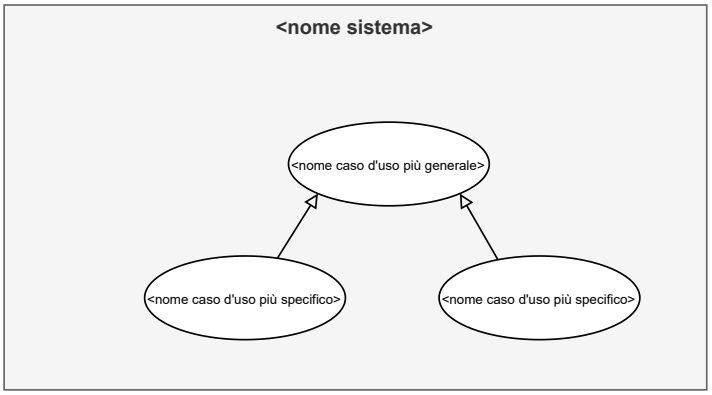

Figura 7: Rappresentazione generalizzazione in UML.

Come si può vedere nella Tabella 2 è necessario indicare il caso d'uso genitore come valore della colonna "Caso d'uso base".

- **Sottocasi d'uso**: i sottocasi d'uso rappresentano scenari specifici e dettagliati che si sviluppano all'interno di un caso d'uso principale. La loro funzione è di descrivere in modo approfondito le diverse situazioni o varianti operative che

possono verificarsi nel contesto del caso d'uso generale. In particolare, ogni caso d'uso può essere suddiviso in più sottocasi, ciascuno associato a una specifica pagina o componente funzionale correlata al caso d'uso principale.

1.3.1.3 Fonti per l'Individuazione dei Casi d'Uso

L'individuazione dei casi d'uso si basa principalmente sul capitolato fornito, che rappresenta una fonte essenziale per comprendere le caratteristiche generali del software da realizzare. Il capitolato, descrive le funzionalità principali e gli obiettivi del sistema, specificando alcuni requisiti chiave.

Tuttavia, non tutti gli aspetti operativi sono dettagliati nel documento, il che rende necessaria un'integrazione attraverso deduzioni basate su necessità logiche e conoscenze pregresse. Il capitolato guida l'identificazione delle macro interazioni, consentendo di delineare i casi d'uso principali e il contesto generale in cui il sistema opererà. Tuttavia, nei punti in cui le specifiche risultano incomplete o generiche, è indispensabile adottare un approccio pro attivo, immaginando scenari d'uso che derivano dall'analisi delle esigenze tipiche di sistemi simili e delle best practices nel settore.

Questa attività di interpretazione si traduce nella definizione di ulteriori dettagli operativi, come eccezioni o varianti necessarie per garantire una piena copertura dei processi e un corretto funzionamento del sistema.

1.3.1.3.1 Individuazione dei casi d'uso

L'identificazione dei casi d'uso avviene attraverso i seguenti passaggi:

1. **Identificazione degli attori**: Gli attori sono individuati in base ai loro ruoli e alle loro necessità operative. Ad esempio, in un sistema gestionale possono essere individuati attori come l'amministratore, l'utente finale e un sistema di terze parti che si integra per la gestione dei pagamenti. L'identificazione degli attori è fondamentale per comprendere chi interagirà con il sistema e quali sono le operazioni che gli utenti devono poter eseguire. In Figura 8 viene mostrata una tecnica utile per decidere se un concetto che appare nei requisiti è un attore o meno.

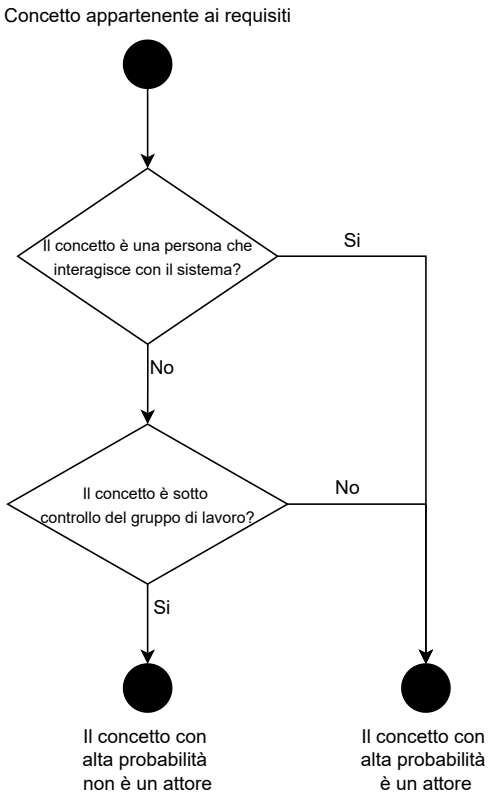

Figura 8: Tecnica per identificare gli attori.

2. **Raffinamento attori**: Una coppia di attori può essere in una particolare relazione chiamata generalizzazione. In questa relazione l'attore "più generale" può eseguire un insieme di interazioni con il sistema che sono un sotto insieme delle interazioni che può eseguire l'attore "più specializzato". In altre parole l'attore "più specializzato" può eseguire tutte le interazioni che può eseguire l'attore "più generico" più eventuali ulteriori interazioni. In questo passaggio è importante scoprire queste relazioni tra gli attori.

3. **Analisi degli obiettivi degli attori**: Una volta identificati gli attori e le relazioni tra essi, vengono analizzati i loro obiettivi in relazione al sistema. Questo processo si basa sulla comprensione dei risultati che ogni attore desidera ottenere, come ad esempio la gestione di un ordine, l'accesso a report personalizzati o

la possibilità di modificare parametri di configurazione. Gli obiettivi degli attori aiutano a delineare il perimetro dei casi d'uso e a definire le funzionalità che il sistema deve supportare per soddisfare tali obiettivi.

4. **Identificazione dei casi d'uso principali**: Partendo dagli obiettivi individuati, si procede con la definizione dei casi d'uso principali. Questi rappresentano le macro funzionalità del sistema, cioè le interazioni essenziali e più frequenti tra gli attori e il sistema stesso. Ogni caso d'uso principale viene descritto in termini di azioni, e risultati attesi, garantendo che le funzionalità fondamentali siano chiaramente delineate.

5. **Scomposizione in sotto-casi**: I casi d'uso principali possono venire successivamente scomposti in sotto-casi, se necessario, per dettagliare specifiche eccezioni, varianti o processi secondari. Questa scomposizione permette di descrivere scenari più specifici o complessi che possono verificarsi durante l'interazione. Ad esempio, il caso d'uso principale "Gestione di un ordine" può essere suddiviso nei sotto-casi "Modifica di un ordine esistente" e "Cancellazione di un ordine in attesa". Questo livello di dettaglio consente di rappresentare in modo accurato e completo tutti i comportamenti attesi del sistema.

1.3.1.4 Requisiti Software

I requisiti software sono le specifiche che il sistema software deve soddisfare per essere accettato dagli utenti finali, dagli stakeholder e dagli altri sistemi con cui interagisce. In particolare i requisiti software rappresentano le funzionalità, le prestazioni, la sicurezza e le altre caratteristiche che il sistema deve avere per soddisfare i requisiti utente analizzati. Presentano una struttura del tipo:

```
\begin{swreq}
    [] {<...>}

    {RXY-Z}{<Descrizione della funzione del requisito>}

    (<X e Y indicano il tipo di requisito, Z la versione>)

    \subreq{RZY-Z.01}{<Descrizione della sottofunzione>}

    \subreq{RXY-Z.02}{<Descrizione della sottofunzione>}

\end{swreq}
```

2 Processi di supporto

2.1 Documentazione

Il processo di documentazione ha lo scopo di registrare le informazioni prodotte da un processo primario garantendo la produzione di documenti coerenti e di qualità.

2.1.1 Standard di formato

I seguenti standard di formato sono validi per tutte le tipologie di documenti. Questi standard devono sottostare agli standard specifici per ogni tipologia di documento indicata nel piano di documentazione. Gli standard più specifici possono sovrascrivere i seguenti o specializzarli(es. data).

2.1.1.1 Standard di scrittura

- Le tabelle e le immagini devono preferibilmente comparire nella posizione in cui sono specificate all'interno del codice sorgente.
- Le tabelle e le immagini devono essere identificate da una label che deve essere usata ogni qual volta sia necessario farvi riferimento.
- Le tabelle e le immagini devono essere accompagnate da una caption che ne riassume il contenuto.
- Si devono utilizzare frasi non più lunghe di tre righe.
- I termini che appartengono al glossario devono essere indicati con una G a pedice e in corsivo.
- Si devono usare dei link per fare riferimento a elementi del documento.

Per informazioni pratiche sui punti sopra indicati leggere la sezione Comandi di base.

2.1.1.2 Standard di forma

2.1.1.2.1 Intestazione

Ogni pagina deve contenere nell'intestazione le seguenti informazioni:

```
Nome documento - <versione>
```

Dove la versione rispetta le regole indicate alla sezione Versione dei documenti.

2.1.1.2.2 Prima pagina

La prima pagina deve contenere le seguenti informazioni:

- Nome documento.
- Nome gruppo.
- Data.
- Versione.
- Logo del gruppo.

2.1.1.2.3 Seconda pagina

La seconda pagina deve contenere il registro delle modifiche per ogni file sottoposto a controllo di configurazione. Il contenuto e la gestione del registro è indicato alla sezione Registro delle modifiche.

2.1.1.2.4 Terza pagina

La terza pagina deve contenere l'indice.

2.1.2 Macro categorie di documenti

Di seguito vengono elencate le due macro categorie di documenti.

2.1.2.1 Documenti interni

Servono al team di lavoro. Sono:

- Verbali interni.
- Norme di progetto.

2.1.2.2 Documenti esterni

Servono al team di lavoro e alla proponente. Sono:

- Piano di progetto.
- Verbali esterni.
- Analisi dei requisiti.
- Piano di qualifica.

2.1.3 Piano di documentazione

Di seguito viene riportato il piano di documentazione in cui si definiscono le tipologie di documenti che verranno prodotti durante il ciclo di vita del prodotto e le loro caratteristiche.

2.1.3.1 Verbali interni

2.1.3.1.1 Scopo

I verbali interni sono documenti che hanno lo scopo di registrare il prodotto di una riunione interna al team di lavoro. Permettono quindi di avere una conoscenza condivisa delle decisioni prese e delle cose da fare.

2.1.3.1.2 Autore

L'autore dei verbali interni deve essere il Responsabile.

2.1.3.1.3 Input

Gli input per la stesura di questi documenti derivano direttamente dalla riunione interna eseguita solitamente a fine sprint ovvero ogni giovedì.

2.1.3.1.4 Struttura

I verbali interni sono composti dalle seguenti sezioni e sottosezioni:

1. **Registro presenze**.

   Contiene le seguenti informazioni:

   (a) Data.
   (b) Ora inizio.
   (c) Ora fine.
   (d) Piattaforma usata.
   (e) Tabella che attesta la presenza dei membri del team. Intestazioni: Componente e Presenza.

2. **Verbale**.

   (a) **Argomenti trattati**
   Contiene un riassunto dei temi trattati durante la riunione. Indica anche eventuali perplessità o difficoltà da introdurre nel diario di bordo.

(b) **Decisioni prese**. Contiene un riassunto delle decisioni prese durante la riunione indicando anche una giustificazione.

3. **To Do**.

   Indica una lista delle cose da fare nel prossimo sprint collegandole alle informazioni di gestione della configurazione.

2.1.3.1.5 Standard di scrittura specifici

- La data del documento riguarda il giorno in cui è avvenuta la riunione interna.

2.1.3.2 Verbali esterni

2.1.3.2.1 Scopo

I verbali esterni sono documenti che hanno lo scopo di registrare la conoscenza del team sulle necessità della proponente a seguito di una riunione esterna. Permettono quindi la conferma di una conoscenza comune(team e proponente) sulle necessità che il prodotto colma. Funzionano quindi da garanzia al team sul fatto che la sua direzione sia corretta.

2.1.3.2.2 Autore

L'autore dei verbali esterni deve essere il Responsabile.

2.1.3.2.3 Input

Gli input per la stesura di questi documenti derivano direttamente dalla riunione esterna che deve essere concordata in anticipo con la proponente.

2.1.3.2.4 Struttura

I verbali esterni sono composti dalle seguenti sezioni e sottosezioni:

1. **Registro presenze**.

   Contiene le seguenti informazioni:

   (a) Data.
   (b) Ora inizio.
   (c) Ora fine.
   (d) Piattaforma.
   (e) Tabella che attesta la presenza dei membri del team. Intestazioni: Componente e Presenza.

(f) Tabella che attesta i rappresentati della proponente che hanno partecipato alla riunione. Intestazioni: Componente e Presenza.

A fondo pagina viene lasciato spazio per permettere al proponente di firmare il verbale esterno.

2. **Domande**.

   Lista delle domande esposte dal team.

3. **Conclusioni**.

   Lista delle conclusioni che il team ha tratto dalle risposte date dai rappresentati della proponente.

2.1.3.2.5 Standard di scrittura specifici

- La data del documento riguarda il giorno in cui è avvenuta la riunione esterna.

2.1.3.3 Analisi dei requisiti

2.1.3.3.1 Scopo

Questo documento ha lo scopo di racchiudere i requisiti utente e i requisiti software sul prodotto oggetto del capitolato.

2.1.3.3.2 Autore

Gli autori di questo documento sono gli Analisti.

2.1.3.3.3 Input

Gli input per la stesura del documento di analisi dei requisiti derivano dall'attività di analisi dei requisiti.

2.1.3.3.4 Struttura

La struttura del documento di analisi dei requisiti è composta dalle seguenti sezioni:

1. **Descrizione prodotto**.

   Ha lo scopo di descrivere il sistema a un alto livello di astrazione. Questa sezione è composta dalle seguenti sottosezioni:

   (a) **Obiettivi prodotto**.
   Descrive il bisogno che ha portato alla stesura del capitolato e come il prodotto le soddisfa.

(b) **Funzioni prodotto**.
Descrive le funzioni principali del prodotto.

(c) **Caratteristiche utente**.
Descrive gli utenti che useranno il sistema e le loro caratteristiche.

2. **Use case**.
Descrive i requisiti utente e l'analisi che porta ai requisiti software.

3. **Requisiti funzionali di qualità e di vincolo**.
Elenca i requisiti funzionali e non.

2.1.3.4 Norme di progetto

Il presente documento ha lo scopo indicato nell'introduzione.

2.1.3.4.1 Autore

L'autore di questo documento è l'Amministratore.

2.1.3.4.2 Input

Gli input per la stesura del documento delle norme di progetto derivano dal processo di gestione.

2.1.3.4.3 Struttura

Deve esistere una sezione per ogni categoria di processo e una sottosezione per ogni processo indicato nello standard IEEE 12207:1996 usato dal team.

2.1.3.5 Piano di progetto

Il documento piano di progetto contiene le informazioni che permettono la gestione di progetto da parte del Responsabile.

2.1.3.5.1 Autore

L'autore di questo documento è il Responsabile.

2.1.3.5.2 Input

Gli input per la stesura del documento delle norme di progetto derivano dal processo di gestione.

2.1.3.5.3 Struttura

Il piano di progetto è composto dalle seguenti sezioni:

1. **Analisi dei rischi**.

   Contiene l'output dell'attività di analisi dei rischi.

2. **Stima dei costi**.

   Contiene la stima delle ore che il gruppo ritiene di consumare e i costi che derivano dalle stesse.

3. **Milestone principali**.

   Definisce le milestone principali che devono essere sottoposte alla validazione del proponente e/o del committente. Per ogni milestone vengono indicate:

   - Data di consegna.
   - Baseline.
   - Risorse preventivate.

4. **Primo periodo**.

   Indica l'insieme di risorse utilizzate nel primo periodo di progetto durante il quale la pianificazione era ancora confusionaria.

   Viene quindi mostrato lo stato delle risorse col termine del primo periodo nelle sottosezioni:

   (a) **Consuntivo**.
   Tabella composta dalle colonne: "Ruolo", "Ore consumate".

   (b) **Aggiornamento rimaste**.
   Tabella composta dalle colonne: "Ruolo", "Ore rimaste".

5. **Sprint**

   Per ogni sprint contiene una sottosezione chiamata `sprint n` che a sua volta contiene le seguenti sottosezioni:

   (a) **Obbiettivi**.
   Descrizione degli obbiettivi dello sprint tramite una lista puntata contenente la descrizione di essi e le issue assegnate a ogni obbiettivo.

   (b) **Pianificazione**.
   Tabella composta dalle colonne: "Identificativo richiesta di modifica", "Ore preventivate" e "Ruolo".

(c) **Consuntivo**.

Tabella composta dalle colonne: "Identificativo richiesta di modifica", "Ruolo", "Ore sviluppo", "Ore verifica" e "Stato". Dove "Stato" viene indicato seguendo la convenzione Stati delle richieste di modifica.

(d) **Retrospettiva**.

Descrizione dei problemi e dei rischi verificati durante lo sprint.

(e) **Aggiornamento risorse rimaste**.

Aggiornamento delle risorse restanti in seguito allo sprint tramite la tabella: "Ruolo", "Ore rimanenti".

2.2 Gestione della configurazione

Il processo di gestione della configurazione si occupa di applicare procedure amministrative e tecniche durante tutto il ciclo di vita del software. In particolare identifica e definisce gli elementi software di un sistema per tenere traccia: delle modifiche, dello stato e dei rilasci dei file. Inoltre garantisce la completezza, la coerenza e la correttezza degli elementi che compongono il prodotto software.

2.2.1 Repository

I prodotti del progetto vengono memorizzati in più $repository_G$. Di seguito vengono elencati i repository utilizzati dal gruppo, le loro caratteristiche e la loro struttura.

2.2.1.1 SorgentiDocumentazione

Questo repository contiene i sorgenti della documentazione scritti usando il linguaggio LaTeX. Il repository segue la struttura indicata in Figura 9.

```
SorgentiDocumentazione
├── .github
│   ├── DizionariLatex
│   │   └── <dizionari>
│   └── workflows
│       └── <file workflows>
├── Candidatura
├── RTB
├── Packages
├── Immagini
├── README.md
└── .gitignore
```

Figura 9: Struttura repository SorgentiDocumentazione.

Dove:

1. `Candidatura`, `RTB` e `Packages` sono delle cartelle il cui contenuto viene spiegato nelle successive sezioni.
2. `<file workflows>` sono dei file che definiscono le automazioni associate al repository e vengono denominati usando la sintassi $Pascal\ case_G$.
3. `Immagini` è una cartella che contiene il logo del gruppo.
4. `README.md` è un file che contiene la descrizione del repository e del gruppo.
5. `.gitignore` è un file speciale che contiene una lista di file che il sistema di versionamento deve ignorare.

2.2.1.1.1 Candidatura

La cartella candidatura ha la struttura indicata in Figura 10.

```
Candidatura
├── VerbaliEsterni
│   └── <verbali esterni esplicativi>
├── VerbaliInterni
│   └── <verbali interni>
├── <lettera di candidatura>
├── <preventivo dei costi assunzione impegni>
└── <valutazione capitolati>
```

Figura 10: Struttura cartella Candidatura.

2.2.1.1.2 RTB

La cartella RTB ha la struttura indicata in Figura 11.

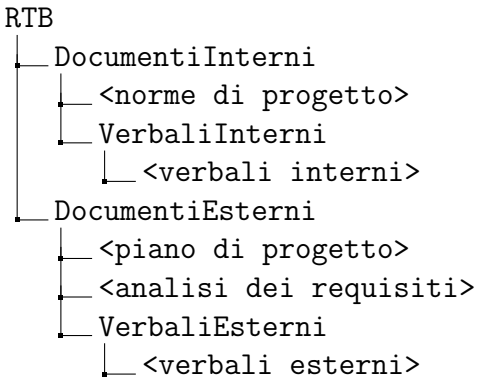

Figura 11: Struttura cartella RTB.

2.2.2 Identificazione configuration item

Di seguito vengono elencate le regole usate per nominare i $configuration\ item_G$ elencati nella sezione repository. I file che non riguardano direttamente il progetto non vengono spiegati ulteriormente.

2.2.2.1 Sorgenti documenti

Nome	Identificativo	Registro modifiche
verbali interni	`<data>-<versione>`	sì
verbali esterni	`<data>-<versione>`	sì
verbali esterni esplicativi	`<data>-<proponente>-<versione>`	sì
lettera di candidatura	`LetteraDiCandidatura`	no
valutazione dei capitolati	`ValutazioneDeiCapitolati-<versione>`	sì
preventivo dei costi e assunzione impegni	`CostiImpegni-<versione>`	sì
piano di progetto	`PianoDiProgetto-<versione>`	sì
norme di progetto	`NormeDiProgetto-<versione>`	sì
Piano di qualifica	`PianoDiQualifica-<versione>`	sì
analisi dei requisiti	`AnalisiDeiRequisiti-<versione>`	sì

Tabella 3: Configuration item del repository SorgentiDocumentazione.

Dove:

- Le date seguono lo schema `aaaa_mm_gg`.
- Il nome della proponente segue la sintassi $Pascal\ case_G$.
- Le versioni seguono lo schema indicato alla sezione Versione dei documenti.

Il motivo dell'utilizzo di data e proponente all'interno dei nomi di alcune tipologie di documenti è che permettono un ordinamento sensato degli stessi.

2.2.3 Controllo di versione

Di seguito vengono riportati i metodi usati dal gruppo per tenere traccia delle versioni dei file contenuti nei repository.

2.2.3.1 Software di controllo di versione

Il gruppo ha deciso di utilizzare Git come software per il controllo di versione. I motivi di questa scelta sono:

1. Git è abbastanza conosciuto dai membri del gruppo.

2. Git è molto usato e quindi una conoscenza approfondita dello stesso è molto utile.

Di seguito vengono fornite delle informazioni teoriche sulle scelte fatte dal gruppo sulla gestione del repository. La separazione netta tra concetti teorici e pratici pur essendo "fastidiosa" per il lettore permette la modularità degli strumenti usati.

2.2.3.1.1 Strategia di branching

Come *strategia di branching*$_G$ il gruppo ha scelto l'utilizzo di Gitflow dato che permette di massimizzare il lavoro parallelo riducendo la complessità di risoluzione dei conflitti. In Figura 12 viene mostrato un esempio di utilizzo della strategia di branching Gitflow.

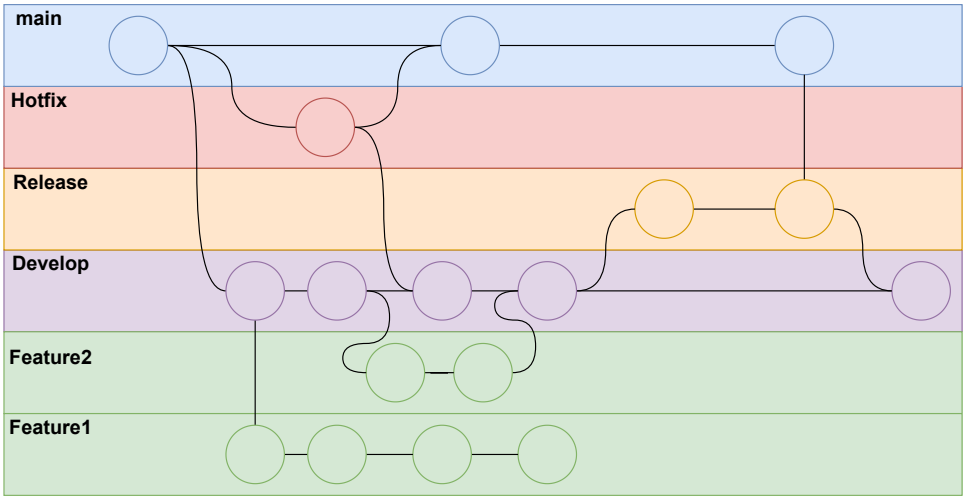

Figura 12: Gitflow in azione(i cerchi sono commit e le linee i collegamenti tra essi)

Di seguito vengono descritti i rami utilizzati da Gitflow e le convenzioni usate dal gruppo.

1. **main**

   Il ramo `main` contiene la codeline principale ovvero le versioni dei file che compongono l'ultima baseline raggiunta. Questo ramo è un ramo "stabile" nel senso che persiste nel repository lungo tutta la sua vita.

2. **develop**

   Il ramo `develop` contiene la codeline secondaria ovvero i cambiamenti verificati che si sommeranno formando la prossima baseline. Anche questo ramo è un ramo "stabile" e viene creato a partire dal ramo `main`.

3. **feature** I rami `feature` vengono usati dai membri del gruppo per implementare le modifiche. Sono dei rami "provvisori" nel senso che esistono finché il loro contenuto non viene verificato. I rami `feature` vengono creati a partire dal ramo `develop` e confluiscono in esso quando il loro contenuto passa con successo la verifica. Il gruppo ha deciso di utilizzare la seguente convenzione per la denominazione di questi rami:

   `feature/#<idRichiestaDiModifica>`

   Così facendo si ottiene un collegamento immediato tra ramo di `feature` e richiesta di modifica.

4. **hotfix**

   I rami `hotfix` vengono usati per eseguire modifiche rapide e di dimensioni ridotte. Queste modifiche non influenzano quindi le versioni dei documenti. Questi rami sono "provvisori" nel senso che vengono eliminati dopo che le loro modifiche sono state allineate ai rami `main` e `develop`.

5. **release**

   I rami `release` vengono usati dal Responsabile del gruppo per approvare il contenuto del ramo `develop` consolidandolo in una baseline nella codeline principale. Sono dei rami "provvisori" nel senso che esistono finché il loro contenuto non viene approvato. I rami `release` vengono creati a partire dal ramo `develop` e confluiscono nei rami `main` e `develop`. Il gruppo ha deciso di utilizzare la seguente convenzione per la denominazione di questi rami:

   `feature/nomeMilestone`

   Così facendo si ottiene un collegamento immediato tra ramo di `release` e la milestone che raggiunge.

2.2.3.1.2 Risoluzione dei conflitti

Utilizzando Gitflow i conflitti possono presentarsi nelle seguenti fusioni(merge) di rami(indicate come "partenza → destinazione"):

1. **release → main**

   I conflitti si presentano quando nel ramo di `release` esistono modifiche su uno o più file presenti nel ramo `main`. In questo caso il contenuto del ramo `release` ha la precedenza dato che è più aggiornato.

2. **hotfix → develop**

I conflitti si presentano quando la correzione rapida eseguita nel ramo `hotfix` riguarda uno o più file che nel mentre sono stati modificati nel ramo `develop`. In questo caso è importante che le correzioni del ramo `hotfix` persistano nel ramo `develop` dopo la fusione. Il ramo `hotfix` ha quindi la precedenza.

3. **feature → develop**

   I conflitti possono presentarsi se due membri del gruppo lavorano sullo stesso documento allo stesso tempo. In questo caso entrambe le modifiche verificate devono raggiungere il ramo `develop`. Bisogna però porre attenzione a mantenere il corretto ordine del Registro delle modifiche.

2.2.3.2 Registro delle modifiche

Oltre all'utilizzo di un software di controllo di versione il gruppo ha deciso di dotare alcuni documenti di una tabella chiamata registro delle modifiche. Tale tabella se il repository fosse gestito in modo impeccabile sarebbe una copia dei dati già registrati nella stessa. Tuttavia il metodo di lavoro del gruppo è in continua evoluzione quindi possono accadere modifiche distruttive che eliminano le versioni dei configuration item appartenenti alla baseline precedente. Il registro delle modifiche è quindi uno strumento a supporto del controllo di configurazione ed è più "attendibile" rispetto alla storia del repository Git. I documenti per cui viene usato il registro delle modifiche sono indicati nella sezione Identificazione configuration item.

Di seguito viene indicata la struttura del registro delle modifiche:

1. **Versione**: versione del documento dopo la verifica della modifica.
2. **Data**: data in cui è avvenuta la modifica.
3. **Autore/i**: nome e cognome dei componenti del gruppo che hanno eseguito le modifiche.
4. **Verificatore/i**: nome e cognome dei verificatori.
5. **Descrizione**: breve descrizione delle modifiche indicando un link alle sezioni modificate.

   La descrizione permette ai membri del team di capire velocemente la necessità di allinearsi a nuove informazioni.

Nota bene: Le righe del registro delle modifiche sono ordinate per versione decrescente.

2.2.3.3 Versione

Le versioni dei documenti vengono indicate seguendo lo schema di versionamento:

`vX.Y`

Dove:

1. `X`: intero positivo che indica la versione major del documento. La versione major indica a quante volte il documento ha raggiunto uno stato ritenuto come "stabile" dal gruppo.

2. `Y`: intero positivo che indica la versione minor del documento. La versione minor indica il numero di modifiche effettuate al documento a seguito dell'ultima versione major.

Uno schema di versionamento a due valori risulta più semplice dato che si concentra solo sulle modifiche significative in termini di contenuto dei documenti.

Nota bene: per evitare ambiguità con le estensioni dei documenti il gruppo ha deciso di rappresentare la versione nei nomi dei documenti usando la notazione `vX_Y`.

2.2.4 Gestione delle modifiche

Una modifica è intesa come un azione che ha lo scopo di modificare lo stato di uno o più file appartenenti all'ultima baseline raggiunta in un repository.

2.2.4.1 Richieste di modifica

Ogni modifica deve essere preceduta da una richiesta di modifica che deve essere discussa e accettata tramite votazione di maggioranza. Le richieste di modifica vengono registrate e gestite utilizzando il servizio di issue tracking offerto da GitHub. Le informazioni tecniche riguardanti la gestione delle issue vengono indicate alla sezione Issue Tracking System. Una richiesta di modifica deve contenere le seguenti informazioni:

- **Nome del file da modificare**: indicandone il percorso all'interno del repository.
- **Data della richiesta**.
- **Tempo stimato per implementare il cambiamento**: indicata nel piano di progetto nella pianificazione dello sprint attuale.
- **Tipo di modifica**.
- **Descrizione della modifica da fare**.
- **Stato**: indica lo stato della richiesta al momento e permette al gruppo di capire cosa fare.

2.2.4.1.1 Descrizione modifica

La descrizione di una modifica deve essere rappresentata tramite un check list contenente i compiti da svolgere per implementare la modifica. Questo permette:

a. A chi implementa la modifica di avere una linea guida su cosa fare anche dopo tanto tempo rispetto alla discussione della modifica nel gruppo.

b. A chi verifica la modifica di sapere che cosa doveva essere fatto.

2.2.4.2 Ciclo di vita delle richieste di modifica

Gli stati che una richiesta di modifica attraversa durante il suo ciclo di vita sono riassunti nel diagramma in Figura 13.

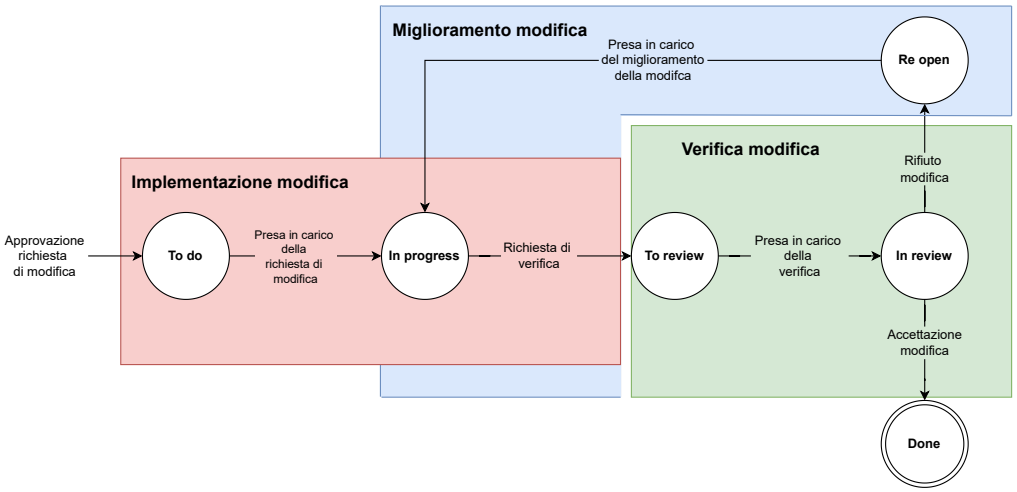

Figura 13: Ciclo di vita di una richiesta di modifica

2.2.4.3 Stati

In particolare gli stati di una richiesta di modifica sono:

1. **To do**: la richiesta di modifica è stata approvata dal gruppo, registrata in GitHub e appartiene ai compiti che portano al raggiungimento di una prossima baseline.

2. **In progress**: la richiesta di modifica è stata presa in carico da un membro del gruppo.

3. **To review**: la modifica è stata implementata dal membro del gruppo che la presa in carico. Prima di contribuire alla prossima baseline deve essere verificata.

4. **In review**: un Verificatore sta procedendo a eseguire la verifica della modifica.
5. **Re open**: la modifica non è stata accettata dal Verificatore e deve quindi essere migliorata.
6. **Done**: la modifica è stata accettata dal Verificatore e quindi è stata integrata nella prossima baseline.

2.2.4.4 Cambiamenti di stato

I cambiamenti di stato di una richiesta di modifica durante al suo ciclo di vita sono:

1. **Presa in carico della richiesta di modifica(To do → In progress)**

   Avviene manualmente seguendo la procedura a carico del modificatore indicata alla sezione Presa in carico di una modifica.

2. **Richiesta di verifica(In progress → To review)**

   Avviene manualmente seguendo la procedura a carico del modificatore indicata alla sezione Richiesta di verifica.

3. **Presa in carico della verifica(To review → In review)**

   Avviene manualmente seguendo la procedura indicata alla sezione Presa in carico di una verifica.

4. **Rifiuto modifica(In review → Re open)**

   Avviene in automatico nel caso in cui la verifica della modifica abbia esito negativo.

5. **Accettazione modifica(In review → Done)**

   Avviene in automatico nel caso in cui la verifica della modifica abbia esito positivo.

6. **Presa in carcio del migliroamento della modifica(Re open → In progress)**

   Avviene manualmente seguendo la procedura a carico del modificatore indicata alla sezione Presa in carico di una modifica.

2.2.4.5 Fasi

Le fasi del ciclo di vita di una modifica sono:

1. **Implementazione modifica**: comprende gli stati "To do", "In progress", "To review" e le transizioni tra gli stessi.

2. **Verifica modifica**: comprende gli stati "To review", "In review", "Done", "Re open" e le transizioni tra gli stessi.

3. **Miglioramento modifica**: comprende gli stati "Re open", "In progress", "To review" e le transizioni tra gli stessi.

2.2.4.5.1 Implementazione modifica

Per implementare una modifica è necessario eseguire i seguenti passi:

1. Presa in carico della modifica.

2. Pull ramo `develop`.

3. Creazione ramo `feature`(vedere rami feature).

4. Colmare la necessità della richiesta di cambiamento assicurandosi di soddisfare la check list. Questa operazione deve avvenire seguendo il processo che regola l'implementazione della tipologia di file da modificare.

   Nota bene: Per alcuni documenti è necessario aggiungere al registro delle modifiche una nuova riga contenente le informazioni per le colonne "Autore/i" e "Descrizione".

5. Commit delle modifiche nel ramo `feature`.

6. Push ramo `feature`.

7. Richiesta di verifica aggiungendo nel primo commento della pull request i seguenti campi:

   (a) closes #<ID della issue che richiede la verifica>

   (b) Tempo impiegato: <numero delle ore effettive usate per svolgere la issue>

2.2.4.5.2 Verifica modifica

Per verificare un cambiamento è necessario eseguire i seguenti passi:

1. Presa in carico della verifica.

2. Pull ramo `feature`(vedere rami feature).

3. Verificare i cambiamenti seguendo le regole indicate nella sezione Verifica.

   Nota bene: Nel caso di rifiuto per alcuni documenti è necessario aggiornare l'ultima riga del Registro delle modifiche specificando la colonna "Verificatore/i".

Nel caso di accettazione oltre a fare ciò è necessario specificare la colonna "Versione" e aggiornare la versione mostrata nel documento(leggere sezione Struttura di base dei documenti).

4. Commit delle modifiche sul ramo `feature`.
5. Push del ramo `feature`.
6. Accettazione o rifiuto delle modifiche.

2.2.4.5.3 Miglioramento modifica

Per eseguire il miglioramento di una modifica è necessario eseguire i seguenti passi:

1. Presa in carico del miglioramento.
2. Pull branch `feature`(vedere rami feature).
3. Modificare il documento assicurandosi di soddisfare la check list e le informazioni aggiuntive date dal verificatore che ha rifiutato la modifica. Questa operazione deve avvenire seguendo il processo che regola l'implementazione della tipologia di configuration item.

   Nota bene: Per alcuni documenti è necessario aggiungere all'ultima riga del registro delle modifiche le informazioni per la colonna "Autore/i".
4. Commit delle modifiche nel ramo `feature`.
5. Push ramo `feature`.
6. Richiesta di verifica.

2.2.5 Approvazione

Di seguito viene documentata l'attività di approvazione delle versioni dei file che devono essere consolidate in una baseline. L'approvazione deve essere eseguita dal Responsabile del gruppo di lavoro e prevede le seguenti operazioni:

1. Creazione ramo `release` seguendo le regole indicate nella sezione Strategia di branching.
2. Eseguire eventuali modifiche correttive minori.

   Nota bene: Aggiungere una nuova riga nel registro delle modifiche indicando i valori per le colonne "Autore/i", "Versione" e "Data". Inoltre usare il valore "Approvazione modifiche" per la colonna "Descrizione".

3. Commit delle modifiche.
4. Merge del ramo `release` nel ramo `main` e nel ramo `develop` seguendo le regole indicate alla sezione Risoluzione dei conflitti.
5. push dei rami `main` e `develop`.

2.2.6 Distribuzione dei documenti

La distribuzione dei documenti avviene mediante l'utilizzo di un sito web disponibile al link https://alt-f4-eng.github.io/Documentazione/. I documenti distribuiti vengono aggiornati ogni volta che viene modificata la baseline ovvero ogni volta che viene modificato il ramo `main`. L'aggiornamento dei documenti segue la procedura indicata alla sezione Pubblicazione dei documenti

2.3 Accertamento della Qualità

Il processo di accertamento della qualità ha l'obiettivo di garantire che il progetto rispetti gli standard e i requisiti definiti, sia riguardo ai prodotti sviluppati sia ai processi adottati. Questo processo rappresenta un elemento centrale nella gestione del progetto, in quanto assicura il monitoraggio continuo e la verifica della conformità alle specifiche tecniche, agli obiettivi di progetto e alle normative applicabili. Il processo si basa su principi fondamentali quali tracciabilità, trasparenza e miglioramento continuo. Utilizza inoltre metriche e strumenti oggettivi per valutare sia l'efficacia dei processi che la qualità dei prodotti. Attraverso una pianificazione accurata e un monitoraggio costante, il processo di accertamento della qualità contribuisce a ridurre i rischi, migliorare le performance e a soddisfare le aspettative degli stakeholder.

2.3.1 Attività

Per l'accertamento della qualità, il gruppo adotta il ciclo di Deming, o PDCA (Plan-Do-Check-Act), un approccio iterativo e continuo volto a migliorare i processi e garantire il raggiungimento degli obiettivi di qualità. Il Ciclo di Deming è composto da 4 fasi fondamentali:

- **Plan**: In questa fase, si definiscono gli obiettivi di qualità e le strategie per garantirne il raggiungimento. È essenziale pianificare le attività che saranno svolte durante il ciclo di vita del progetto, stabilendo le metriche da monitorare e i criteri di accettazione.
- **Do**: In questa fase, le attività pianificate vengono effettivamente eseguite. Si mettono in atto le procedure di controllo della qualità stabilite per monitorare la qualità del progetto in fase di sviluppo.

- **Check**: Questa fase consiste nel confrontare i risultati ottenuti con gli obiettivi di qualità definiti. Si verificano i dati raccolti durante la fase "Do" per determinare se i processi e i risultati sono allineati con gli standard di qualità e i requisiti.
- **Act**: In questa fase finale, vengono intraprese azioni correttive o preventive in base ai risultati della fase "Check".

2.3.2 Standard di riferimento per la qualità di prodotto del software

Per la valutazione della qualità del software con conseguente stesura delle metriche viene seguita la normativa ISO/IEC 9126, secondo lo standard le qualità sono suddivise nelle seguenti categorie:

- **qualità esterne**: misurano i comportamenti del software durante la sua esecuzione;
- **qualità interne**: si applicano al software non eseguibile, permettono di individuare eventuali problemi che potrebbero influire sulla qualità finale del prodotto prima che sia realizzato il software eseguibile;
- **qualità in uso**: rappresentano il punto di vista dell'utente sul software, consentono di stabilire i seguenti obiettivi:
  - *Efficacia*: capacità del software di permettere agli utenti di raggiungere gli obiettivi specificati con accuratezza e completezza;
  - *Produttività*: capacità del software di permettere agli utenti di spendere una quantità opportuna di risorse in relazione all'efficacia ottenuta;
  - *Soddisfazione*: capacità del software di soddisfare gli utenti;
  - *Sicurezza*: capacità del software di rimanere entro livelli accettabili di rischi di danni a persone e apparecchiature.

Lo standard normativo stabilisce un modello definendo un set di caratteristiche che consentono di misurare e valutare diversi aspetti della qualità del prodotto software:

- **Funzionalità**: capacità del software di fornire le funzioni adatte a soddisfare le esigenze stabilite;
- **Affidabilità**: capacità del software di mantenere un livello di prestazioni specifico quando usato in date condizioni per un dato periodo di tempo;
- **Efficienza**: capacità del software di fornire adeguate prestazioni relativamente alla quantità di risorse utilizzate;

- **Usabilità**: capacità del software di essere propriamente compreso e utilizzato dall'utente;
- **Manutenibilità**: capacità del software di essere modificato per introdurre migliorie o adattamenti;
- **Portabilità**: capacità del software di essere di essere trasportato da un ambiente di lavoro a un altro.

2.3.3 Notazione Metriche di Qualità

Ogni metrica è identificata in un modo univoco seguendo la notazione: $M.[Tipo].[Abbreviazione\ Nome]$ Dove:

- **M**: indica "metrica"
- **Tipo**: sarà PC per indicare che la metrica misura la qualità di un processo o PR per indicare che la metrica misura la qualità di un prodotto
- **Abbreviazione Nome**: viene inserito l'acronimo basato sul nome completo della metrica

2.3.4 Descrizione Metriche di Qualità

Le metriche sono descritte tramite i seguenti campi:

- **Notazione**: seguendo le indicazioni sopra elencate;
- **Nome**: nome completo della metrica;
- **Descrizione**: descrizione della metrica;
- **Caratteristiche**: presente unicamente nelle metriche di prodotto, indica a quale caratteristica, tra quelle indicate nella sezione relativa allo standard di riferimento, fa riferimento la metrica;
- **Formula di misurazione**: formula per la misurazione del valore della metrica.

Verranno poi indicati all'interno del documento "Piano di Qualifica" i valori tollerabili e i valori ottimali per ogni metrica e il processo a cui fanno riferimento.

2.3.5 Elenco delle Metriche di Qualità

L'elenco delle metriche di qualità da adottare per il progetto è dettagliato secondo la struttura definita e copre diverse aree rilevanti per il processo di accertamento della qualità. Le metriche includono:

2.3.5.1 Metriche di processo

Planned Value

- **Notazione**: M.PC.PV
- **Descrizione**: rappresenta il costo stimato del lavoro programmato entro un determinato momento del progetto, secondo il piano di progetto originale
- **Formula**: $BAC \times (\%LavoroPianificato)$, dove BAC indica Budget at Completion cioè il budget previsto per la realizzazione del progetto

Earned Value

- **Notazione**: M.PC.EV
- **Descrizione**: rappresenta il valore del lavoro effettivamente completato alla data corrente
- **Formula**: $BAC \times (\%LavoroCompletato)$, dove BAC indica Budget at Completion cioè il budget previsto per la realizzazione del progetto

Actual Cost

- **Notazione**: M.PC.AC
- **Descrizione**: rappresenta il costo sostenuto fino alla data corrente
- **Formula**: costo speso per la realizzazione delle attività svolte fino data corrente

Schedule Variance

- **Notazione**: M.PC.SV
- **Descrizione**: misura la variazione (in percentuale) tra il valore del lavoro effettivamente completato e il valore del lavoro pianificato, indica se il progetto è in ritardo o in anticipo rispetto a quanto preventivato
- **Formula**: $\frac{EV-PV}{EV} \times 100$

Cost Variance

- **Notazione**: M.PC.CV
- **Descrizione**: misura la variazione (in percentuale) tra il valore del lavoro completato e il costo effettivo sostenuto per tale lavoro, indica se il progetto sta rispettando il budget pianificato

- **Formula**: $\frac{EV-AC}{EV} \times 100$

Variazione del piano

- **Notazione**: M.PC.VP
- **Descrizione**: misura la variazione (in percentuale) tra il numero di task pianificati per un determinato periodo di tempo e il numero di task realmente realizzati, misura la qualità della pianificazione del lavoro per un periodo di tempo
- **Formula**: $\frac{T_p-T_c}{T_p} \times 100$, dove T_p indica il numero di task pianificati e T_c indica il numero di task completati

Estimated at Completion

- **Notazione**: M.PC.EAC
- **Descrizione**: rappresenta una stima aggiornata del costo totale previsto per la realizzazione del progetto
- **Formula**: $AC + (BAC - EV)$, (costo sostenuto + stima costi da sostenere), dove BAC indica il Budget at Completion cioè il budget previsto per la realizzazione del progetto

Rischi inattesi

- **Notazione**: M.PC.RI
- **Descrizione**: indica il numero dei rischi che si sono verificati in un determinato periodo e che non erano stati preventivati tramite l'Analisi dei Rischi
- **Formula**: Numero di rischi occorsi e non preventivati

Risk Mitigation Rate

- **Notazione**: M.PC.RMR
- **Descrizione**: indica la percentuale dei rischi occorsi durante lo sviluppo in un determinato periodo che sono stati mitigati correttamente tramite le strategie di mitigazione indicate nella Analisi dei Rischi
- **Formula**: $\frac{R_g}{R_t} \times 100$, dove R_g indica il numero di rischi gestiti correttamente e R_t il numero totale di rischi verificati nel periodo in analisi

Metriche Soddisfatte

- **Notazione**: M.PC.MS

- **Descrizione**: indica la percentuale di metriche soddisfatte, cioè con un valore calcolato che rispetta il valore minimo ammissibile indicato all'interno del Piano di Qualifica
- **Formula**: $\frac{M_s}{M_t} \times 100$, dove M_s indica il numero di metriche soddisfatte mentre M_t il numero totale delle metriche analizzate

2.3.5.2 Metriche di prodotto

Percentuale Requisiti Obbligatori Soddisfatti

- **Notazione**: M.PR.PRM
- **Descrizione**: rappresenta la percentuale di requisiti obbligatori soddisfatti rispetto i requisiti obbligatori totali inseriti all'interno del documento Analisi dei Requisiti
- **Caratteristiche**: Funzionalità
- **Formula**: $\frac{RMS}{RMT} \times 100$, dove RMS indica il numero di requisiti obbligatori soddisfatti e RMT il numero di requisiti obbligatori totale

Percentuale Requisiti Opzionali Soddisfatti

- **Notazione**: M.PR.PRO
- **Descrizione**: rappresenta la percentuale di requisiti opzionali soddisfatti rispetto al numero totale di requisiti opzionali inseriti all'interno del documento di Analisi dei Requisiti
- **Caratteristiche**: Funzionalità
- **Formula**: $\frac{ROs}{ROT} \times 100$, dove ROS indica il numero di requisiti opzionali soddisfatti e ROT il numero di requisiti opzionali totale

Correttezza Ortografica

- **Notazione**: M.PR.CO
- **Descrizione**: rappresenta il numero di errori ortografici rilevato all'interno di un documento
- **Caratteristiche**: Usabilità
- **Formula**: N° errori ortografici rilevati

2.4 Processo di Verifica

Il processo di verifica ha come obiettivo fondamentale l'accertamento che:

- i **prodotti di lavoro** generati durante il ciclo di vita del software rispettino i requisiti specificati
- i **processi seguiti** siano conformi allo standard, alle linee guida e ai piani definiti

2.4.1 Descrizione

Questo processo consiste nel fornire prove oggettive che i risultati di una specifica fase dello sviluppo software rispettino tutti i requisiti previsti. Si basa sull'analisi e revisione del contenuto per valutare la coerenza, la completezza e la correttezza dei risultati. Nel caso di codice, include anche il testing per assicurarsi che i risultati siano conformi alle aspettative definite. Le task fondamentali previste dal processo sono:

- verifica dei processi;
- verifica dei requisiti;
- verifica della progettazione;
- verifica del codice;
- verifica dell'integrazione;
- verifica della documentazione.

Per garantire l'accertamento della conformità, ogni volta che si apporta una modifica, è necessario sottoporre l'intero contenuto aggiornato a una verifica. L'incremento della versione del prodotto aggiornato avviene esclusivamente se la modifica viene verificata e la verifica ha esito positivo. Il processo di verifica viene svolto dai membri incaricati come verificatori (che si segneranno all'interno del registro delle modifiche del documento). i quali non possono essere la stessa persona a cui è stata assegnata la realizzazione del prodotto da verificare.

2.4.2 Analisi statica

L'analisi statica è un approccio alla verifica che non richiede l'esecuzione del codice dell'oggetto di verifica per individuare i difetti del prodotto software e accertarne la sua completezza e coerenza. Si applica non solo al codice, ma anche alla documentazione, verificando la conformità alle regole del prodotto, l'assenza di difetti e la presenza delle proprietà desiderate. Dal team viene utilizzata una tecnica standard per l'analisi statica dei prodotti: **Walkthrough**.

2.4.2.1 Walkthrough

Il walkthrough è uno dei metodi di lettura nell'analisi statica utilizzato per esaminare e verificare una parte del prodotto, che sia documento o codice, per accertarne la conformità ai requisiti o vincoli stabiliti precedentemente. Si tratta di un approccio collaborativo tra autore e verificatore durante il quale viene esaminato un prodotto o una sua parte, seguendo un percorso prestabilito e cercando di identificare difetti attraverso una lettura critica ad ampio spettro, priva di assunzioni. Nel caso del controllo del codice, il verificatore deve simulare diverse possibili esecuzioni, mentre per i documenti deve analizzarne il contenuto. Le fasi che vengono svolte durante questa tecnica di analisi statica sono:

- **lettura**: il verificatore effettua una lettura critica dell'oggetto in esame cercando eventuali errori;
- **discussione**: al termine della lettura, nel caso vengano rilevati problemi, il verificatore comunica con gli autori e propone eventuali suggerimenti, con l'obiettivo di correggere i difetti;
- **correzione e repeat**: una volta terminata la discussione e rilevati i difetti, gli autori sono pregati di correggere tali difetti seguendo le indicazioni discusse. Successivamente, si passa di nuovo al passo 2.

2.4.2.2 verifica della documentazione

La verifica della documentazione, composta solamente da analisi statica dei documenti realizzati e/o modificati, ha lo scopo di verificare la qualità, completezza, coerenza e conformità dei documenti tecnici relativi al prodotto software da realizzare. Durante l'esecuzione della tecnica di walkthrough descritta precedentemente, il verificatore ha lo scopo di assicurarsi che vengano seguiti correttamente gli standard di scrittura e di forma, sia generici che specifici, indicati all'interno del processo di Documentazione. Oltre a ciò, è stata realizzata una GitHub Action che, all'apertura di una pull request, realizza un'analisi grammaticale dei file modificati e coinvolti nella pull request, fornendo successivamente un feedback al gruppo di lavoro. La GitHub Action assicura la correttezza grammaticale dei documenti verificati migliorando il processo di verifica, aumentandone l'efficienza e riducendo il tempo richiesto per le revisioni manuali, permettendo al verificatore di concentrarsi maggiormente su aspetti di forma e di contenuto.

2.4.3 Analisi dinamica

L'analisi dinamica è un approccio all'analisi di sitemi informatici e prodotti software che si concentra sull'osservazione e verifica del loro comportamento durante l'esecuzione.

Quest'ultima è ampiamente impiegata per individuare errori a runtime, ottimizzare l'efficienza e testare la robustezza contro scenari imprevisti. Grazie alla sua capacità di fornire dati concreti e rilevanti, rappresenta un elemento fondamentale nel processo di sviluppo e manutenzione di applicazioni e sistemi complessi. Essa prevede la definizione di una suite di test, generalmente automatizzati e riproducibili, che vengono eseguiti a runtime per valutare il comportamento del sistema in risposta a specifici input. Questi test verificano la correttezza delle funzionalità, l'efficienza delle prestazioni e l'assenza di errori o anomalie operative. I principali tipi di test che vengono utilizzati per l'analisi dinamica sono:

- **test di unità**: Mirano a verificare il corretto funzionamento di singole unità o componenti del software (ad esempio, funzioni o metodi). Sono solitamente automatizzati e si concentrano su un ambito ristretto per identificare errori locali;
- **test di integrazione**: Valutano come le diverse unità o moduli del software interagiscono tra loro. L'obiettivo è assicurarsi che le componenti integrate funzionino correttamente come un sistema coerente;
- **test di sistema**: Analizzano il comportamento dell'intero sistema per verificare che soddisfi i requisiti specificati. Considerano il software come un unico blocco, includendo interazioni con l'ambiente e altre applicazioni.

All'interno del processo di Sviluppo, nello specifico nella sezione di testing, vi è una descrizione più approfondita delle varie tipologie di test adottate durante lo sviluppo del prodotto software, ogni test utilizzato verrà poi codificato e indicato all'interno del documento "Piano di Qualifica".

2.5 Processo di infrastruttura

Il processo di infrastruttura ha il compito di definire e mantenere l'infrastruttura e gli strumenti necessari allo svolgimento di tutti gli altri processi di ciclo di vita.

2.5.1 Documentazione

L'infrastruttura del processo di documentazione contiene tutti gli strumenti software, i linguaggi e i pacchetti usati per la stesura, la compilazione e la distribuzione dei documenti.

2.5.1.1 Linguaggio

Per stesura dei documenti è stato deciso di utilizzare LaTeX. LaTeX è un linguaggio di markup per la creazione di testi basato sul software di composizione tipografica chiamato TeX. LaTeX è il linguaggio più utilizzato per la produzione di documenti in formato PDF professionali.

2.5.1.2 Distribuzione TeX

Il gruppo sviluppando la documentazione su sistema operativo Windows ha scelto l'utilizzo della distribuzione TeX chiamata **TeX Live**. Questa distribuzione oltre che contenere TeX fornisce pacchetti, font e software a supporto tra cui un gestore di pacchetti grafico chiamato **TLShell**. L'installazione di TeX Live può essere fatta seguendo le istruzioni indicate al link Installazione Tex Live.

2.5.1.2.1 TLShell

L'installazione di pacchetti tramite TLShell è abbastanza intuitiva e segue i passaggi:

1. Avvio TLShell.
2. Selezionare il radio button `Not installed` Figura 14 (1).
3. Indicare il nome del pacchetto da installare nella barra di ricerca Figura 14 (2).
4. Selezionare il radio button del pacchetto da installare Figura 14 (3).
5. Cliccare il pulsante `Install marked` Figura 14 (4).

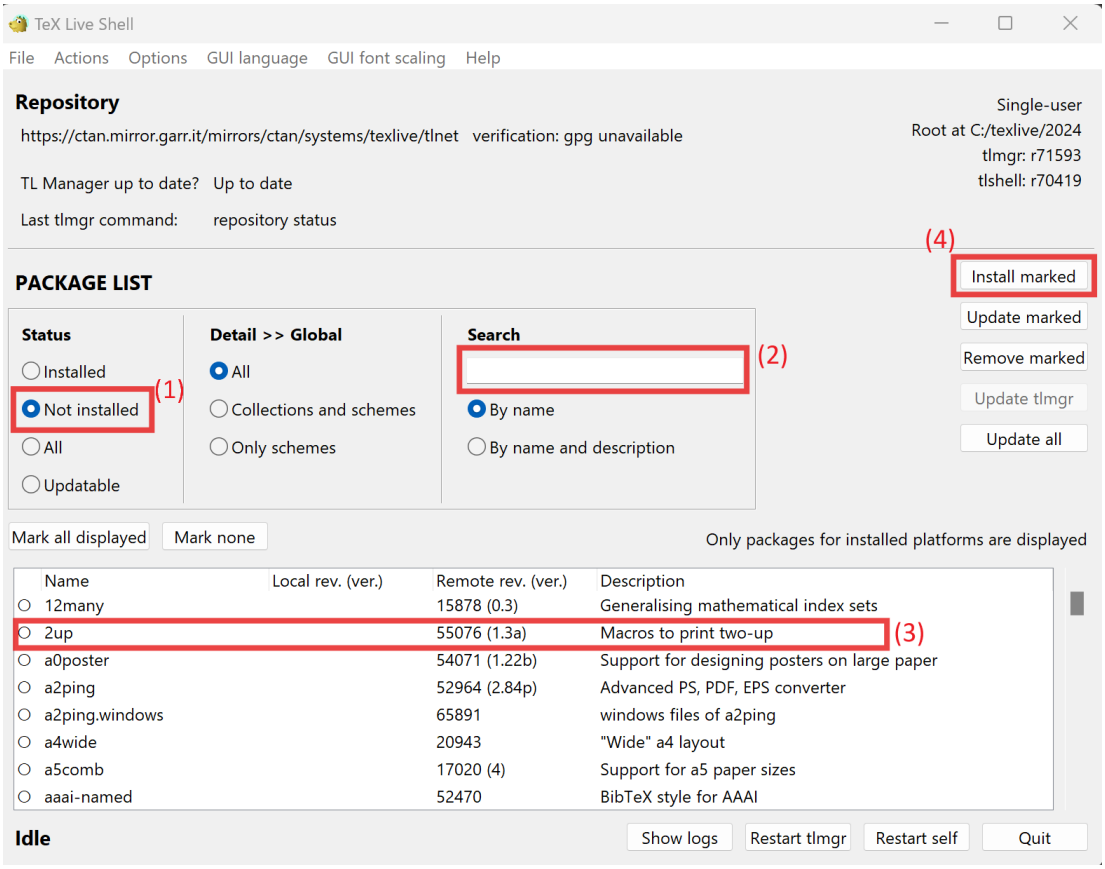

Figura 14: Installazione pacchetti LaTeX..

2.5.1.3 Comandi di base

Di seguito vengono elencati i comandi di base che devono essere noti per la stesura di documenti in LaTeX.

2.5.1.3.1 Tabelle

```
\begin{table}[H]
    \begin{tabularx}{| X | c | l | r |}
        \hline
        \makecell{Intestazione1 \\ Intestazione1} &
        Intestazione2 &
        Intestazione3 &
        Intestazione4 \\
```

```
            \hline

            prima colonna prima riga   &
            seconda colonna prima riga &
            terza colonna prima riga  &
            quarta colonna prima riga  &

            \hline

            ...

            \hline
        \end{tabularx}
        \caption{Riassunto contenuto.}
    \end{table}
```

Dove:

- `[H]` indica che il posizionamento della tabella nel pdf deve essere dove si trova nel codice.
- `[| X | c | l | r |]` indica le colonne della tabella.

  `|` indica una riga di separazione tra le colonne.

  `X` indica una colonna che si allarga dinamicamente.

  `c`, `l` e `r` indicano rispettivamente colonne in cui il posizionamento del testo è centrale, a sinistra e a destra.
- `\makecell{... \\ ...}` permette di creare celle con testo su più righe, usando `\\` per andare a capo.
- `\hline` crea una linea orizzontale.

2.5.1.3.2 Link

Link a elementi interni alla pagina:

```
\label[sec:nome]

...

\hypperref[sec:nome]{nomeLink}
```

Dove:

- Convezione standard per nominare le label: `tipoElemento:nome`. Dove i possibili tipi di elemento sono: paragraph(par), subparagraph(subpar), section(sec), subsection(subsec) e figure(fig).

Link a elementi esterni alla pagina:

```
\href{URL}{nomeLink}
```

2.5.1.3.3 Listati di codice

Listati di codice su più righe:

```
\begin{lstlisting}
    ...
\end{lstlisting}
```

Listati di codice inline:

```
\texttt{code}
\lstinline+code+
```

2.5.1.3.4 Figure

```
\begin{figure}[H]
    \includegraphics[scale=1.2]{pathImmagine}
    \caption{Riassunto immagine.}
    \label{fig:IdFigura}
\end{figure}
```

Dove:

- `[H]` permette di posizionare la figura nel pdf dove si trova nel codice.
- `scale=x.y` indica la scala da applicare all'immagine.

2.5.1.3.5 Inclusione file

L'inclusione di file .tex permette di dividere un documento in più file rendendone la modifica più semplice.

```
\input{PercorsoAlFile}
```

2.5.1.3.6 Grafici

```
\begin{axis}[
    title={Titolo del grafico},
    xlabel={Nome dell'asse x},
    ylabel={Nome dell'asse y},
    xmin=, xmax=,
    ymin=, ymax=,
    xtick={assex1, assex2, ...},
    ytick={assey1, assey2, ...},
    grid=,
    grid style={},
    width=, height=,
    legend pos=
]

\addplot[color=] coordinates {
    (x1, y1) (x2, y2) ...
};
\addlegendentry{}
```

Dove:

- `title` indica il titolo del grafico.
- `xlabel` e `ylabel` indicano i nomi degli assi x e y.
- `xmin` e `xmax` indicano i valori minimi e massimi dell'asse x.
- `ymin` e `ymax` indicano i valori minimi e massimi dell'asse y.
- `xtick` e `ytick` indicano i valori sull'asse x e y.
- `grid` indica se mostrare la griglia che può essere `major`, `minor` o `both`.
- `grid style` indica lo stile della griglia che può essere `dotted`, `dashed`, `solid`, `none` e se ne può specificare il colore, es. `gray`.
- `width` e `height` indicano la larghezza e l'altezza del grafico.
- `legend pos` indica la posizione della legenda che può essere `north east`, `north west`, `south east`, `south west`.
- `addplot coordinates` indica i valori da aggiungere al grafico.

- `[]` subito dopo addplot contiene le caratteristiche del grafico tracciato, come il colore.
- `addlegendentry` indica il nome del grafico tracciato.

2.5.1.4 Pacchetto LaTeX

Per rendere i file sorgenti meno complessi è stato deciso di definire un pacchetto LaTeX che contiene:

1. La definizione di comandi e ambienti di uso comune.
2. Le dichiarazioni dei pacchetti usati.

Oltre a diminuire la verbosità dei documenti questo metodo permette di centralizzare la gestione dei pacchetti, dei comandi e degli ambienti semplificando la manutenzione. I pacchetti usati devono essere installati manualmente usando il processo spiegato nella sezione TLShell. In Figura 15 viene mostrata la struttura del pacchetto LaTeX. **Nota bene**: Il pacchetto per poter essere importato deve essere indicato con il comando `\usepackage` e il suo percorso deve essere indicato nella variabile di ambiente TEXINPUTS. La spiegazione della configurazione dell'ambiente per l'utilizzo del pacchetto è spiegata nella sezione Integrated Development Environment IDE.

```
Packages
├── Immagini
│   └── logo.jpeg
├── custom.sty
└── TitlePage.tex
```

Figura 15: Struttura pacchetto LaTeX.

2.5.1.4.1 Comandi personalizzati

Nel pacchetto sono definiti i seguenti comandi necessari per la scrittura di un documento:

1. `\primapagina` : inserisce nella posizione di invocazione la pagina iniziale dei documenti. Questo comando sfrutta le variabili indicate alla sezione Struttura di base documenti.
2. `\glossario{<termine>}` : modifica lo stile del termine per indicare la sua presenza nel glossario

Per la stesura delle descrizioni dei casi d'uso il gruppo ha deciso di definire un insieme di comandi da utilizzare all'interno del ambiente personalizzato `\begin{usecase} ...\end{usecase}`:

1. `\req{<link requisito utente>}`: genera in automatico la riga della tabella relativa al requisito di riferimento.
2. `\pre{ \item <pre condizione 1> ... }`: genera in automatico la riga delle pre-condizioni mostrandole in una lista.
3. `\post{ \item <post condizione 1> ... }`: genera in automatico la riga delle post-condizioni mostrandole in una lista.
4. `\actor{<attore principale>}`: genera in automatico la riga per l'attore principale.
5. `\subactors{<attori secondari>}`: genera in automatico la riga per gli attori secondari.
6. `\trigger{<trigger>}`: genera in automatico la riga per il trigger.
7. `\inc{<casi d'uso inclusi>}`: genera in automatico la riga per i casi d'uso inclusi.
8. `\base{<caso d'uso base>}`: genera in automatico la riga per il caso d'uso base.
9. `\scenario{\item <azione 1> ...}`: genera in automatico la riga per lo scenario principale e la popola indicando le azioni in una lista.
10. `\subscenario{\item <condizione 1> ...}`: genera in automatico la riga per gli scenari secondari e la popola indicandoli in una lista.

2.5.1.4.2 Ambienti personalizzati

Il pacchetto definisce anche i seguenti ambienti necessari per la scrittura di un documento:

1. `\begin{registromodifiche} ... \end{registromodifiche}` : permette di definire il registro delle modifiche tralasciando le impostazioni della tabella. All'interno di questo ambiente devono essere indicate le righe del registro delle modifiche.
2. `\begin{usecase}{<id>}{<descrizione>} ... \end{usecase}`: genera in automatico la tabella per la descrizione dei casi d'uso.

2.5.1.5 Struttura di base documenti

Di seguito viene mostrato il codice necessario per la definizione dello scheletro di un documento soggetto al processo di Gestione della configurazione:

```
\documentclass[a4paper, 12pt]{article}
\usepackage{custom}

%--------------------VARIABILI---------------------
\def\lastversion{ Versione documento }
\def\title{ Nome documento }
\def\date{ Data }
%--------------------------------------------------

\begin{document}

\primapagina

\begin{registromodifiche}

\end{registromodifiche}

\tableofcontents

\newpage
\end{document}
```

Nota bene: I valori delle variabili devono seguire le regole indicate nelle sezioni Documentazione e Versione dei documenti.

2.5.1.5.1 Gestione documenti "lunghi"

Per rendere più semplice la gestione di documenti molto lunghi il gruppo ha deciso di dividere il contenuto in più file .tex. I documenti soggetti a questa divisione sono:

1. Norme di progetto.
2. Analisi dei requisiti.
3. Piano di progetto.

Ognuno di questi file verrà diviso in sezioni contenute in una cartella avente nome uguale al documento in Pascal case. Queste cartelle sono contenute nella cartella padre comune `Sezioni`. Le sezioni saranno divise a loro volta in sottosezioni contenute nella cartella `Sottosezioni`. Inoltre ogni cartella di sottosezione conterrà una cartella

chiamata `Immagini`. La divisione segue la struttura indicata in Figura 16 dove `F` indica il nome di un file da dividere:

```
cartella di F
├── F.tex
└── Sezioni
    ├── SezioneA
    │   ├── SezioneA.tex
    │   ├── Sottosezioni
    │   │   ├── Sottosezione_a.tex
    │   │   └── Sottosezione_b.tex
    │   └── Immagini
    └── SezioneB
        ├── SezioneB.tex
        ├── Sottosezioni
        │   ├── Sottosezione_a.tex
        │   └── Sottosezione_b.tex
        └── Immagini
```

Figura 16: Struttura file "lunghi".

2.5.1.6 Integrated Development Environment IDE

Come IDE per la stesura della documentazione viene utilizzato Visual Studio Code che può essere installato seguendo le istruzioni al seguente link VsCode.

Visual Studio Code dispone di un grande numero di estensioni che lo rendono molto versatile. In particolare sono stati scelti dal gruppo le seguenti estensioni:

1. **LaTeX Workshop**: https://github.com/James-Yu/latex-workshop/wiki.
2. **LTeX+**: https://ltex-plus.github.io/ltex-plus.

2.5.1.6.1 LaTeX Workshop

Questa estensione permette di integrare la compilazione dei file LaTeX in Visual Studio Code. LaTeX Workshop richiede che sia installato TeX Live. Dopo aver installato l'estensione è necessario modificare la procedura di compilazione di modo che vengano cercati i pacchetti usati e i file .tex importati nella cartella Packages. Per fare ciò è necessario seguire le azioni:

1. Aprire Visual Studio Code.
2. Premere la combinazioni di tasti `Ctrl+,`.

3. Scrivere sulla barra di ricerca il testo `latex-workshop.tools` Figura 17 (1).
4. Cliccare il link `Edit in settings.json` del primo risultato Figura 17 (2).
5. Aggiungere al oggetto tool chiamato `latexmk` a seguito dell'attributo `args` l'attributo `env` indicando il seguente attributo Figura 17 (3):

```
"TEXINPUTS":"%WORKSPACE_FOLDER%/Packages;"
```

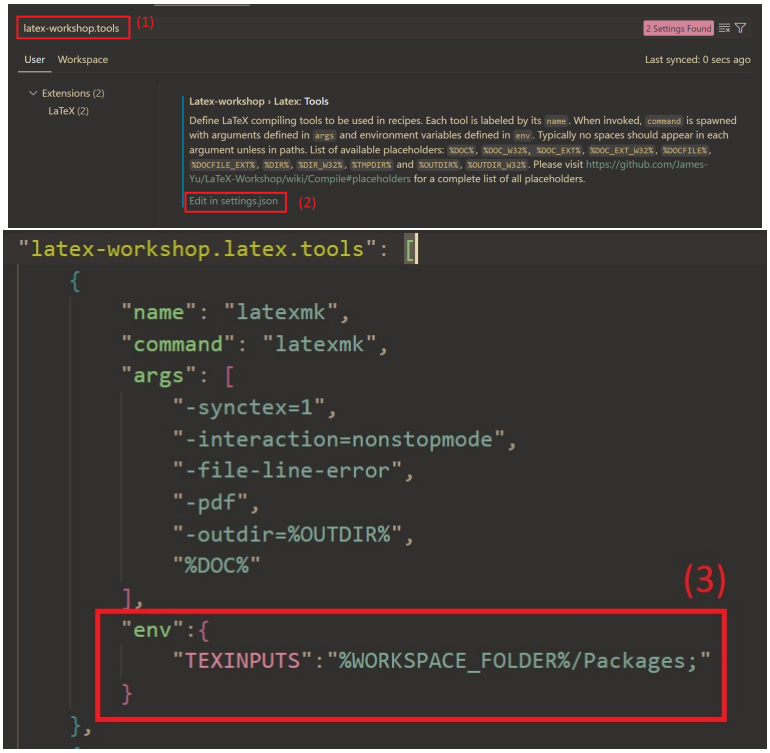

Figura 17: Modifica impostazioni LaTex Workshop.

Nota bene: Per riuscire ad accedere al pacchetto locale è necessario lavorare in Visual Studio Code usando come "workspace folder" la cartella relativa alla repository.

2.5.1.6.2 LTeX+

LTeX+ è un'estensione che esegue lo "spell checking" dei documenti .tex per installarla basta cercarla nel marketplace di Visual Studio Code. Per eventuali problemi di set up le procedure sono spiegate in modo chiaro e dettagliato al link installazione LTeX+.

2.5.2 Gestione della configurazione

A supporto del processo di gestione della configurazione il gruppo ha scelto l'utilizzo di due strumenti:

1. Git: https://git-scm.com/doc.
2. GitHub: https://docs.github.com/en.

2.5.2.1 Git

Git è un Distributed Version Control System DVCS utilizzabile tramite interfaccia a linea di comando. I DVCS hanno la caratteristica di permettere ai membri del gruppo di lavoro di mantenere una propria copia locale del repository e della sua storia. Una volta modificata la copia locale del repository è necessario registrare le modifche nella copia centralizzata del repository. Le operazioni e gli elementi di base comuni a tutti i DVCS: **repository**, **ramo**, **push**, **pull**, **conflitto** e **commit**. Nelle sezioni successive viene mostrato come Git implementa questi concetti e altri concetti caratteristici.

2.5.2.1.1 repository

Un repository in Git è una cartella contenente:

1. Una cartella `.git` che contiene l'insieme di strutture dati e file necessari per la gestione della storia delle versioni dei file utente.
2. Un insieme di file utente e cartelle di cui Git deve tenere la storia delle versioni.

In git esistono due diverse declinazioni di repository:

1. **remota**: esiste in un server remoto a cui i membri del team possono attingere tramite le operazioni di push e pull.
2. **locale**: esiste solo sul file system locale e può essere associata a un repository remoto usato come default per le operazioni di psuh e pull.

Per creare un repository è possibile eseguire uno dei seguenti comandi:

```
git clone <URL>
git init <NomeRepository>
```

Dove:

1. Il primo comando permette di copiare la repository remota nella current working directory proprio file system.
2. Il secondo comando permette di creare una repository nella current working directory nel proprio file system.

2.5.2.1.2 commit

Un commit rappresenta una istantanea del contenuto di uno o più file utente di cui è stata indicata l'esistenza a Git. In Git ogni commit è associato alle seguenti informazioni:

1. **hash SHA-1**: sequenza di 40 caratteri alfanumerici che identifica univocamente il commit all'interno del repository.
2. **autore**.
3. **data di creazione**.
4. **messaggio**: usato per segnalare cosa fa il contenuto del commit.
5. **commit padre**: hash SHA-1 dei commit padre.

I commit sono quindi messi in relazione "padre-figlio" da Git andando a creare un albero. L'unico commit che non ha padre è il primo commit chiamato **commit radice**. I commit con più di un padre sono dei commit generati da operazioni di merge. Il comando Git per generare un commit con commento è:

```
git commit -m "<commento>"
```

Per comprendere meglio i commit leggere sezione 3 aree di Git. Questo comando utilizza come commit padre il commit foglia del ramo corrente.

2.5.2.1.3 ramo

Un ramo in Git è un commit a cui è stata associata un etichetta mobile che funziona da alias per lo stesso. Tramite questa etichetta mobile è possibile fare riferimento al commit a essa associato. Un ramo è un etichetta mobile nel senso che Git la sposta in automatico quando viene generato un commit figlio. Questo crea una biforcazione nella storia dei commit(ramo) dove l'etichetta punta sempre all'ultimo commit(foglia) del nuovo flusso.

Git mantiene internamente un puntatore all'etichetta del ramo corrente chiamato commit **HEAD**. Così facendo riesce a operare nel modo atteso quando si eseguono comandi che agiscono rispetto al ramo corrente. Git permette di ottenere riferimenti a commit padre del commit HEAD tramite le seguenti sintassi:

```
# riferimento al commit padre di HEAD
# si possono concatenare arbitrari ^
HEAD^

# riferimento ad n-esimo parente di HEAD
HEAD~n
```

Per creare un ramo esiste il comando:

```
git branch <nomeRamo>
```

Per eliminare un ramo esiste il comando:

```
# il ramo non deve avere commit
git branch -d <nomeRamo>

# il ramo puo avere commit
git branch -D <nomeRamo>
```

Per spostarsi in un ramo esiste il comando:

```
git checkout <nomeRamo>
```

Per ottenere una lista dei rami del repository locale esiste il comando:

```
git branch
```

Per ottenere una lista dei rami dei repository locale e remoto è necessario applicare l'opzione -a al comando precedente.

2.5.2.1.4 3 aree di Git

Git lavora su tre diversi livelli:

1. **Commit HEAD**: rappresenta il ramo corrente. Per gestire il ramo corrente esistono i seguenti comandi:

```
# modifica etichetta ramo correte usando il
# commit indicato le modifiche vengono
# eliminate dal file system
git reset --hard <hashCommit>

# modifica etichetta ramo correte usando il
# commit indicato le modifiche tornano
# nell'area di staging
git reset --soft <hashCommit>

# ottenere la storia dei commit
# del ramo corrente
git log --graph

# ottenere la storia dei commit
# di tutti i rami
git log --all --graph

# ottenere differenze con area di staging
```

```
git diff --cached

# ottenere differenze con albero di lavoro
git diff HEAD
```

2. **Area di staging**: anche chiamata indice o cache, è una struttura dati dinamica che contiene le modifiche ai file utente che Git deve aggiungere nel prossimo commit.

   Di seguito vengono mostrati i comandi per la gestione dell'area di staging:

   ```
   # aggiunge file
   git add <percorsoFile>

   # rimuove i file separati da ,
   git checkout -- <files>

   # ottenere differenze con albero di lavoro
   git diff
   ```

3. **Albero di lavoro**: albero di cartelle e file che Git "conosce" e quindi ritiene come facenti parte del repository.

   Questi file e cartelle vengono chiamati **file tracciati** e devono per forza essere memorizzati all'interno della cartella relativa al repository.

   Un file diventa tracciato quando è passato dall'area di staging.

2.5.2.1.5 push

L'operazione di push permette di pubblicare le modifiche apportate a un repository locale al repository remoto a cui è associato. Le declinazioni del comando di Git che permette di eseguire un push sono:

```
# pubblica il contenuto del ramo locale
# corrente nel ramo remoto indicato
git push origin <ramoDestinazione>

# pubblica il contenuto del ramo locale
# indicato nel ramo remoto indicato
git push <ramoSorgente> origin <ramoDestinazione>
```

Dove `origin` è il nome di default del riferimento al repository remoto e può essere gestiti usando i comandi:

```
# aggiunge nuovo riferimento remoto
git remote add <nome> <URL>

# rimuove il riferimento remoto
git remote remove <nome>

# modifica il riferimento remoto
git remote set-url <nome> <URL>
```

2.5.2.1.6 pull

L'operazione di pull permette aggiornare un repository locale con le modifiche registrate al repository remoto a cui è associato. Git per eseguire un operazione di pull mette a dispozione il seguente comando:

```
# esegue il pull del ramo remoto indicato
# nel ramo locale corrente
git pull origin <nomeRamo>
```

In realà il comando `git pull` è un comando composto che accorpa i comandi:

1. `git fetch origin <nomeRamo>`: recupera localmente il ramo remoto indicato.
2. `git merge <nomeRamo>`.

2.5.2.1.7 merge

L'operazione di merge esegue una fusione di due o più rami sorgente verso un ramo di destinazione. A seguito dell'operazione di merge la cronologia del ramo di destinazione conterrà le modifiche dei rami sorgente. In una fusione possono accadere le seguenti situazioni:

1. I commit HEAD dei rami contengono file con nomi uguali ma contenuto diverso. In questo caso sorge un **conflitto**.
2. Non esistono conflitti.
3. Il commit HEAD del ramo di destinazione è in dietro rispetto ai commit HEAD dei rami sorgente. In questo caso avviene un **fast-forward** e Git risolve il merge in automatico spostando l'etichetta del ramo di destinazione.

Nei primi due casi Git ci guiderà nell'assemblaggio di un nuovo commit HEAD per il ramo di destinazione chiamato **merge commit**.

Per eseguire un merge è necessario usare il comando:

```
# merge ramo indicato nel ramo corrente
git merge <nomeAlbero>
```

Nel caso in cui esista un conflitto Git richiede la risoluzione manuale dello stesso tramite la modifica dei file. Git indica questa situazione nella console in cui appare la keyword `MERGING` che indica il fatto che siamo nel mezzo di un operazione di fusione. In questa situazione il comando `git diff` ci mostra i conflitti. Per uscire dallo stato di `MERGING` eliminando l'operazione di merge è possibile eseguire il comando `git merge --abort`.

Una volta sistemato il conflitto si rientra nel caso in cui non esistano conflitti. In questo caso è necessario eseguire il **merge commit** che avrà come commit padre i commit HEAD del ramo di destinazione e del ramo sorgente.

2.5.2.2 GitHub

GitHub permette la memorizzazione e la gestione di repository Git. Attorno alle funzionalità offerte da Git implementa nuove funzionalità e servizi a supporto dello sviluppo di progetti.

2.5.2.2.1 GitHub organization

GitHub offre degli account particolari chiamati **organization account** che permettono definire più repository pubblici a cui uno stesso gruppo di lavoro ha accesso. Il gruppo ha deciso di utilizzare un account di questo tipo dato che è pensato per la realizzazione di progetti. L'account del gruppo si trova al link https://github.com/ALT-F4-eng/.

Nell'account sono state definite le seguenti repository pubbliche:

1. SorgentiDocumentazione: contiene la documentazione scritta in LaTeX.
2. Documentazione: contiene i documenti pdf e il sito web per la loro pubblicazione.

2.5.2.2.2 Gestione dei rami

GitHub permette di gestire tramite la pagina web del repository i rami in esso definiti. In particolare permette l'esecuzione delle seguenti operazioni:

1. **Creazione ramo**

   La creazione di un ramo segue le operazioni:

   (a) Selezionare la scheda "Code".
   (b) Premere il bottone "Branches".
   (c) Premere il bottone "New branch".
   (d) Indicare il nome del nuovo ramo.

(e) Indicare il ramo di partenza tramite il menù a tendina con label "Source".

(f) Premere il bottone "Create new branch".

2. **Eliminazione ramo**

   L'eliminazione di un ramo segue le operazioni:

   (a) Selezionare la scheda "Code".

   (b) Premere il bottone "Branches".

   (c) Trovare il ramo da eliminare.

   (d) Premere il bottone raffigurante un cestino.

2.5.2.2.3 Issue

GitHub fornisce un ITS ovvero un sistema che permette di registrare e gestire le richieste di modifica. Le richieste di modifica sono rappresentate tramite il concetto di **issue**. Un issue è composta dai seguenti elementi:

1. **Titolo**: nome della richiesta di modifica.
2. **Id**: numero univoco che identifica la richiesta di modifica all'interno del sistema.
3. **Stato**: indica lo stato attuale della richiesta di modifica tra gli stati che essa può attraversare durante il suo ciclo di vita.
4. **Etichetta**: rappresenta in modo schematico la tipologia di richiesta di modifica.
5. **Assegnatario**: indica il membro del gruppo di lavoro che sta implementando la richiesta di modifica.
6. **Milestone**: indica la $milestone_G$ a cui la richiesta di modifica appartiene.
7. **Commenti**: lista di commenti che specificano con più precisione la richiesta di modifica.

Di seguito vengono indicate l'insieme di operazioni che possono essere eseguite su un issue:

1. **Creazione**

   Per creare una issue è necessario eseguire i seguenti passi a partire dalla pagina web del repository:

   (a) Selezione finestra "Issues".

   (b) Cliccare tasto "New issue".

(c) Inserire titolo della issue.

(d) Inserire descrizione della issue. La descrizione supporta un linguaggio di markup chiamato Markdown, questo aiuta a rendere la descrizione più strutturata.

La descrizione di una issue deve seguire la struttura:

```
# File da modificare
<percorso primo file>
<percorso secondo file>
...

# Cosa fare
- [ ] <primo compito>
- [ ] <secondo compito>
...

# Ruolo
<ruolo>

# Ore preventivate
<ore preventivate>
```

Il significato del codice viene indicato nella sezione Markdown.

(e) Assegnare una o più etichette.

(f) Assegnare il progetto chiamato **ArtificalQI**.

(g) Cliccare il pulsante "Submit new issue".

2. **Eliminazione**

Per eliminare una issue è necessario eseguire i seguenti passi a partire dalla pagina web del repository:

(a) Selezionare finestra "Issues".

(b) Selezionare la issue che si vuole eliminare.

(c) Cliccare il bottone "Delete issue".

3. **Chiusura**

Per chiudere una issue è necessario eseguire i seguenti passi a partire dalla pagina web del repository:

(a) Selezionare finestra "Issues".

(b) Selezionare la issue che si vuole chiudere.

(c) Cliccare il bottone "Close issue".

Chiudere una issue equivale a indicare che la richiesta di modifica che rappresenta è stata implementata e verificata correttamente.

4. **Modifica**

   Per modificare una issue è necessario eseguire i seguenti passi a partire dalla pagina web del repository:

   (a) Selezionare finestra "Issues".

   (b) Selezionare la issue che si vuole modificare.

   (c) Ora è possibile modificare: commenti, titolo, etichetta, assegnatario, progetto ecc.

5. **Commentare un issue**

   Per aggiungere un commento a una issue è necessario eseguire i seguenti passi a partire dalla pagina web del repository:

   (a) Selezionare finestra "Issues".

   (b) Selezionare la issue che si vuole commentare.

   (c) Scrivere il commento nella barra con label "Add a comment"(I commenti supportano il linguaggio Markdown).

   (d) Premere il bottone "Comment"

2.5.2.2.4 Markdow

Markdown è un linguaggio di markup che permette di strutturare il testo. Gli elementi di base utili per l'utilizzo di questo linguaggio sono:

- **Check list**

```
- [ ] elemento 1
- [ ] elemento 2
- [ ] elemento 3
...
```

- **Lista non numerata**

```
- elemento 1
- elemento 2
- elemento 3
...
```

- **Lista numerata**

```
1. elemento 1
2. elemento 2
3. elemento 3
...
```

- **Headings**

```
# Titolo livello 1
## Titolo livello 2
### Titolo livello 3
...
```

2.5.2.2.5 Etichette

Di seguito viene fornita una lista delle etichette relative al repository SorgentiDocumentazione:

1. **bug**: riguarda la sistemazione di un documento.
2. **enhancement**: riguarda l'implementazioni di automazioni della gestione e organizzazione di progetto.
3. **requirement**: riguardano il documento di analisi dei requisiti.
4. **documentation**: riguardano gli altri documenti.

2.5.2.2.6 pull request

Una pull request in GitHub è una richiesta di merge di un ramo sorgente verso un ramo destinazione. Di seguito vengono elencate e descritte le operazioni per la gestione delle pull requet:

1. **Creazione**

   Per creare una pull request bisogna seguire questi passaggi a partire dal sito web del repository:

   (a) Selezionare la scheda "Pull requests".
   (b) Premere il pulsante "New pull request".
   (c) Selezionare il ramo di destinazione tramite il menù a tendina "base:".
   (d) Selezionare il ramo di sorgente tramite il menù a tendina "compare:".
   (e) Premere il pulsante "Create pull request".

2. **Chiusura**

   Per chiudere una pull request bisogna seguire questi passaggi a partire dal sito web del repository:

   (a) Selezionare la scheda "Pull requests".

   (b) Selezionare la pull request che si vuole chiudere.

   (c) Cliccare il bottone "Close pull request".

3. **Approvazione**

   L'approvazione di una pull request implica l'esecuzione dell'operazione di merge dal ramo sorgente al ramo destinazione. Come nel caso dell'esecuzione di un merge nel repository locale possono presentarsi tre casi:

   (a) **Esistono dei conflitti**.

   (b) **Fast-forward**.

   (c) **Non esistono dei conflitti**.

   Nel primo caso sarà necessario risolvere i conflitti eseguendo le seguenti operazioni a partire dalla pagina relativa alla pull request:

   (a) Cliccare il bottone "Resolve conflicts".

   (b) GitHub mostrerà una lista composta dai documenti che sono in conflitto. Per ognuno dei conflitti viene indicata la differenza tra il documento presente nel ramo sorgente e quello presente nel ramo destinazione.

   (c) Una volta sistemato il conflitto premere il bottone "Mark as resolved".

   (d) Ripetere il passo 3. finchè tutti i conflitti sono risolti.

   (e) Cliccare il bottone "Commit merge".

   (f) Ora la procedura si ricollega ai casi **Fast-foward** e **Non esistono confilitti**.

   Nel caso di **Fast-forward** o nel caso in cui **Non esistono dei conflitti** seguire i passi:

   (a) Cliccare il bottone "Merge pull request".

   (b) Indicare un commento per il merge commit.

   (c) Cliccare il pulsante "Confirm merge".

   (d) Cliccare il pulsante "delete branch".

2.5.2.2.7 Project

GitHub permette di raggruppare le issue in progetti. Un progetto può offrire più "rappresentazioni" dell'insieme di issue che contiene. Le rappresentazioni si basano sul valore di proprietà associate alle issue(assegnatario, etichetta, milestone ecc). I progetti introducono anche la possibilità di creare proprietà personalizzate. In particolare esistono 3 tipologie di rappresentazioni:

1. **Task-board**

   Una task-board è una tabella composta da un insieme di colonne. Ogni colonna corrisponde a un valore per una proprietà e contiene tutte le issue che hanno tale valore per la proprietà stessa. Ogni membro del team può modificare il valore della proprietà per una issue spostandola nella rispettiva colonna.

   Per una guida più completa su questa tipologia di rappresentazione viene lasciato il seguente link.

2. **Table layout**

   Le issue vengono organizzate in una tabella. Ogni riga contiene una issue. Le colonne contengono i valori delle proprietà selezionate delle issue. Questa rappresentazione permette di eseguire diverse operazioni sulle issue quali: ordinamento per proprietà, raggruppamento per proprietà, separazione per proprietà ecc.

   Per una guida più precisa alle operazioni esistenti e come applicarle viene lasciato il seguente link.

3. **Road map**

   Le issue vengono organizzate in un $diagramma\ di\ Gantt_G$.

Il gruppo ha deciso di utilizzare tre rappresentazioni del progetto:

1. **Progetto**

   task-board definita sulla proprietà Stato che viene utilizzata come project backlog.

2. **Sprint backlog**

   task-board generata applicando un filtro sul valore della proprietà **Iteration** delle issue. Questo filtro seleziona solo le issue che hanno per valore lo sprint attuale per la proprietà **Iteration**. Questa task-board contiene quindi le richieste di modifica da portare a termine nello sprint corrente. Per informazioni sulle proprietà personalizzate leggere la sezione Proprietà personalizzate.

3. **Tabella progetto**

   Un table layout che rappresenta il project backlog in modo alternativo per semplificare l'assegnazione di valori alle proprietà delle issue.

2.5.2.2.8 Proprietà personalizzate

Le proprietà personalizzate anche chiamate issue possono essere gestiste a partire dalla pagina di progetto. Per accedere alle proprietà personalizzate è necessario eseguire i seguenti passi:

1. Spostarsi sulla pagina di progetto.
2. Premere il pulsante "...".
3. Selezionare l'opzione "Settings".
4. Ora è possibile:
   - **Aggiungere un campo**.
     (a) Selezionare il pulsante "+ New field".
     (b) Indicare il nome del campo.
     (c) Indicare il tipo del campo.
     (d) Cliccare il pulsante "Save".
   - **Modificare un campo**.
     (a) Selezionare il campo da modificare.
     (b) Aggiungere un possibile valore o eliminarne uno esistente.

Per assegnare un valore a una proprietà personalizzata di una issue è necessario eseguire i seguenti passaggi:

1. Spostarsi sulla pagina di progetto.
2. Selezionare la vista del progetto chiamata "TabellaProgetto".
3. Individuare la issue per cui si vuole assegnare un valore.
4. Spostasi sulla colonna nominata come il campo personalizzato.
5. Cliccare il pulsante "▽".
6. Selezionare il valore per la proprietà tra le scelte del menù a tendina.

Le proprietà utilizzate dal gruppo sono:

1. Sprint: usata per associare le issue agli sprint.
2. Priority: usata per indicare la priorità delle issue.

2.5.2.2.9 Stato

Lo stato di una issue è una proprietà personalizzata che corrisponde allo stato della richiesta di modifica che la issue rappresenta. I valori che questa proprietà può assumere sono:

1. **To do**.
2. **In progress**.
3. **To review**.
4. **In review**.
5. **Re open**.
6. **Done**.

Per informazioni dettagliate sullo stato delle richieste di modifica leggere la sezione Ciclo di vita delle richieste di modifica.

2.5.2.2.10 Presa in carico di una modifica

Per poter prendere in carico una modifica essa deve appartenere alla rappresentazione Sprint backlog.

Per prendere in carico l'esecuzione di una modifica è necessario trascinarla dalla colonna di appartenenza alla colonna che indica l'esecuzione della stessa.

In accordo con la sezione Ciclo di vita delle richieste di modifica le possibili colonne di partenza sono: "To do" e "Re open" mentre l'unica colonna di destinazione possibile è "In progress".

2.5.2.2.11 Presa in carico di una verifica

Per poter prendere in carico una verifica essa deve appartenere alla rappresentazione Sprint backlog.

Per prendere in carico una verifica il Verificatore deve trascinarla dalla colonna "To review" alla colonna "In review".

2.5.2.2.12 Richiesta di verifica

Per richiedere la verifica di una modifica è necessario aprire una pull request avente come:

- **Ramo sorgente**: il ramo contenete la modifica da verificare che deve seguire le indicazioni fornite nella sezione Strategia di branching.
- **Ramo destinazione**: il ramo `develop`.

2.5.2.2.13 Accettazione modifiche

Per accettare le modifiche di cui è stata richiesta la verifica è necessario eseguire il merge della pull request. Ed eliminare il ramo sorgente della pull request.

Per informazioni sulla risoluzione dei conflitti nella strategia di branching scelta dal gruppo leggere la sezione Risoluzione dei conflitti.

2.5.2.2.14 Rifiuto modifiche

Per rifiutare le modifiche di cui è stata richiesta la verifica è necessario chiudere la pull request.

Il verificatore in questo caso ha l'onere di indicare in un commento della issue i miglioramenti da apportare la modifica affinchè venga accettata nella verifica successiva.

2.5.2.2.15 Pubblicazione dei documenti

La pubblicazione dei documenti avviene al raggiungimento di ogni baseline. Questa operazione è automatizzata tramite il meccanismo delle GitHub action. In particolare i documenti vengono pubblicati seguendo la stessa struttura del repository contenente i sorgenti.

2.5.2.2.16 Distribuzione dei documenti

La distribuzione dei documenti avviene tramite un sito web reso accessibile tramite il servizio di hosting offerto da GitHub. Il sito segue la struttura indicata in Figura 18.

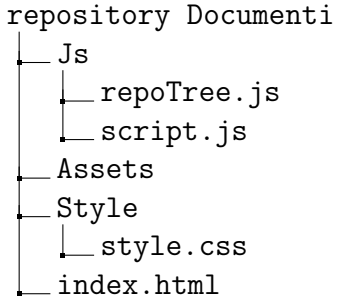

Figura 18: Struttura sito web.

2.5.3 Github action

Le GitHub action sono un meccanismo messo a disposizione da GitHub che permette di automatizzare dei compiti dei diversi processi di sviluppo. L'automazione avviene indicando un flusso di lavoro che viene eseguito da GitHub alla necessità su un container.

2.5.3.1 Teoria

2.5.3.1.1 Workflow

Ogni workflow ha un proprio nome che viene usato da GitHub per identificarlo nei report delle esecuzioni. Una workflow è composto dai seguenti elementi fondanti:

- **events**: ogni workflow risponde a determinati eventi sul repository a cui esso appartiene. Di seguito viene indicato un link alla lista completa dei possibili eventi: events.
- **jobs**: insieme di compiti che devono essere eseguiti in risposta all'evento. Di default i job vengono eseguiti in parallelo su container diversi.

2.5.3.1.2 Jobs

Ogni job ha un nome che viene usato da GitHub per identificarlo nei report delle esecuzioni. Un job è composto dai seguenti elementi fondanti:

- **env**: una o più variabili d'ambiente accessibili all'interno del job stesso. runner: l'identificativo del runner(container) hostato da GitHub che deve eseguire il job. GitHub offre diversi container che variano nella versione/tipo di OS utilizzato. Di seguito viene indicato un link che contiene la lista dei runner hostati da GitHub runners.
- **steps**: insieme di compiti atomici che formano il job e vengono eseguiti sequenzialmente sul runner.

2.5.3.1.3 Step

Ogni step è identificato da GitHub tramite un nome. Uno step è composto dai seguenti componenti presenti in modo mutuamente esclusivo:

- **action**: nome di una GitHub action appartenente al marketplace. Il marketplace delle GitHub action si trova al link actions.
- **comandi**: uno o più comandi che vengono eseguiti nel runner.

2.5.3.2 Pratica

2.5.3.2.1 Definizione workflow

I workflow devono essere definiti nella cartella .github/workflows del repository. Ogni workflow viene specificato in un apposito file che contiene la definizione del workflow tramite il linguaggio YAML. I file dei workflow hanno quindi estensione .yaml. La sintassi per la definizione di un workflow è:

```
name: <nomeWorkflow>

on:
    <evento>
jobs:
    <jobs>
```

2.5.3.2.2 Definizione job

Un job viene definito seguendo la sintassi:

```
<nome>:
    if: <condizione>
    runs-on: <runner>
    env:
        <env1>
        <env2>
        ...
    steps:
        <step1>
        <step2>
        ...
```

Dove:

- La condizione determina se il job viene eseguito o meno e serve per analizzare aspetti più specifici dell'evento(non è obbligatoria).
- Le variabili d'ambiente(env) vengono definite usando la sintassi:

  ```
  <nome>: <valore>
  ```

2.5.3.2.3 Definizione step

Uno step che utilizza una action viene definito seguendo la sintassi:

```
    name: <nomeStep>
    uses: <nomeAction>
        with:
            <arg1>
            <arg2>
            ...
```

Dove:

- Il nome dell'action segue solitamente la seguente forma standard:

`<proprietarioRepository>/<nomeRepository>@<vers>`

Dove la repository è quella in cui è definita la action da utilizzare e la versione è appunto la versione della action che si vuole usare.

- Ogni argomento viene indicato seguendo la sintassi: `<nome>: <valore>`.

Le action sono solitamente documentate in modo estensivo nel file README.md del repository che le contiene. In caso in cui ciò non sia vero all'interno del repository deve esistere un file chiamato action.yaml(ogni action deve averlo così GitHub la riconosce come tale) che definisce: argomenti e output della action.

2.5.3.3 Variabili GitHub

Le variabili definite a livello di repository sono accessibili da parte di tutte le GitHub action contenute nel repository stesso. Le variabili permettono di modificare i valori usati dalle action senza dover modificare il file in cui esse sono definite.

2.5.3.3.1 Definizione

Per definire una variabile a livello di repository è necessario seguire i passaggi:

1. Spostarsi sulla pagina della repository in cui si vuole definire la variabile.
2. Selezionare la scheda "Settings".
3. Selezionare la voce "Secrets and variables" nel menù laterale.
4. Selezionare la sottovoce "Actions" del menù a tendina.
5. Selezionare la scheda "Variables".
6. Cliccare il bottone "New repository variable"
7. Indicare Nome e Valore della variabile.
8. Cliccare il pulsante "Add variable".

2.5.3.3.2 Gestione

Per modificare una variabile definita a livello di repository è necessario seguire i passaggi:

1. Spostarsi sulla pagina della repository in cui si vuole modificare una variabile.
2. Selezionare la scheda "Settings".

3. Selezionare la voce "Secrets and variables" nel menù laterale.
4. Selezionare la sottovoce "Actions" del menù a tendina.
5. Selezionare la scheda "Variables".
6. Cliccare il bottone con l'icona della matita che compare a fianco alla variabile.
7. Indicare Nome e/o Valore della variabile.
8. Cliccare il pulsante "Update variable".

2.5.3.4 Secrets GitHub

I secrets sono delle variabili "speciali" il cui valore viene crittografato e mantenuto da GitHub. Il valore di un secret definito a livello di repository è accessibile alle GitHub action che vi appartengono. Una volta assegnato un valore ad un secret esso non è più visibile dalle impostazioni del repository, tuttavia il suo valore può essere modificato.

2.5.3.4.1 Definizione

Per definire un secret a livello di repository è necessario seguire i passaggi:

1. Spostarsi sulla pagina della repository in cui si vuole definire il secret.
2. Selezionare la scheda "Settings".
3. Selezionare la voce "Secrets and variables" nel menù laterale.
4. Selezionare la sottovoce "Actions" del menù a tendina.
5. Selezionare la scheda "Secrets".
6. Cliccare il bottone "New repository secret"
7. Indicare Nome e Valore del secret.
8. Cliccare il pulsante "Add secret".

2.5.3.4.2 Gestione

Per modificare un secret definito a livello di repository è necessario seguire i passaggi:

1. Spostarsi sulla pagina della repository in cui si vuole modificare un secret.
2. Selezionare la scheda "Settings".

3. Selezionare la voce "Secrets and variables" nel menù laterale.
4. Selezionare la sottovoce "Actions" del menù a tendina.
5. Selezionare la scheda "Secrets".
6. Cliccare il bottone "Manage organization secrets".
7. Cliccare il bottone con l'icona della matita che compare a fianco del secret.
8. Indicare il nuovo Nome e/o Valore del secret.
9. Cliccare il pulsante "Save changes".

2.5.3.5 Pubblicazione dei documenti

La pubblicazione utilizza un workflow definito nel file `PublicazioneDocumenti.yaml`. Questo workflow viene eseguito ogni volta che avviene la pubblicazione di un commit nel ramo `main` che contiene la modifica di un file `.tex`. Il workflow utilizza due variabili definite a livello di repository:

1. `DIRS_TO_DEL`: contiene i nomi(non percorsi) delle cartelle da eliminare e non compilare separati da spazio.
2. `DIRS_TO_IGNORE`: contiene i nomi(non percorsi) delle cartelle da non compilare e da rimuovere con le omonime cartelle della repository del sito separati da spazio.

Notare che quindi una modifica alla struttura delle cartelle del progetto può richiedere:

1. La modifica delle variabili `DIRS_TO_IGNORE` e `DIRS_TO_DEL`.
2. La modifica manuale della struttura della repository che contiene il sito.

Oltre alle variabili questa GitHub action utilizza un **Personal Access Token** memorizzato nel secret chiamato `PAT_COMPILATI`.

2.5.3.6 Correzione grammaticale dei documenti

La correzione grammaticale dei documenti viene facilitata tramite un workflow definito nel file `ControlloOrtografico.yaml`. Questo workflow viene eseguito ogni volta che viene effettuato il push o viene creata una pull request di un file .tex nei rami *feature* e *develop*. Il workflow utilizza aspell per effettuare il controllo e in caso trovasse errori questi vengono segnalati come output della action e anche come commento alla pull request. Il workflow non ha lo scopo di correggere automaticamente gli errori ma di segnalarli per una correzione manuale andando a rendere il lavoro efficente ed efficace.

2.5.3.7 Calcolo delle ore di lavoro

Il workflow è definito nel file `CalcoloOre.yaml` e viene eseguito ogni volta che viene effettuato il merge di una pull request. Per facilitare la determinazione delle ore di lavoro impiegate da ogni ruolo si è deciso di utilizzare un file excel condiviso. Questo file excel contiene una dashboard che permette di tenere traccia delle ore preventivate ed effettive per ogni ruolo. Ciò facilita la stesura degli sprint all'interno del documento Piano di progetto. La compilazione dei campi della dashboard nel file excel è automatizzata tramite una GitHub action che preleva lo sprint di riferimento della issue, l'id della issue, il ruolo che ha risolto la issue, le ore preventivate dal primo commento della issue, le ore impiegate dal primo commento della pull request relativa.

2.5.4 Gestione

2.5.4.1 Canali di comunicazione

I canali di coumunicazione si dividono in canali interni usati per le riunioni del gruppo e canali esterni usati per le riunioni tra il gruppo e il proponente.

2.5.4.1.1 Canali di comunicazione interni

La comunicazione tra membri del gruppo avviene utilizzando le seguenti tecnologie:

1. **Telegram (comunicazione asincrona)**: applicazione di messaggistica che permette comunicazione rapide di interesse generale e di contenuto breve. Usato per segnalare al responsabile, tramite chat individuali, rischi che si sono verificati o eventuali problemi accorsi.

2. **Discord (comunicazione sincrona)**: utilizzata per effettuare chiamate di gruppo sia formali che informali.

2.5.4.1.2 Canali di comunicazione esterni

La comunicazione tra i membri del gruppo e l'azienda proponente avviene tramite i seguenti canali di comunicazione:

1. **Gmail (comunicazione asincrona)**: verrà utilizzato principalmente per la condivisione di file/documenti e per organizzare chiamate sincrone tra gruppo ed azienda.

2. **Google Meet (comunicazione sincrona)**: utilizzata per effettuare videochiamate per il contatto diretto con l'azienda proponente, durante le quali verranno discussi eventuali dubbi dei membri del gruppo o verrà presentato il lavoro svolto dal gruppo per ricevere un feedback.

2.5.5 Sviluppo

A supporto del processo di sviluppo il gruppo ha scelto l'utilizzo dei seguenti strumenti:

1. StarUML: https://staruml.io/blog/staruml-6-3-1-release/.

2.5.5.1 StarUML

StarUML è un software che permette la creazione di diagrammi di diverso tipo. Questi diagrammi possono poi essere esportati usando diversi formati tra cui: PDF, PNG e JPEG. Il gruppo ha deciso di utilizzare questo software invece di Draw.io come deciso precedentemente per la sua maggiore semplicità d'uso. In particolare, StarUML offre strumenti dedicati per la creazione e gestione di relazioni tra elementi, ereditarietà e associazioni, rendendo il processo di modellazione più chiaro e strutturato.

2.5.6 Dashboard excel

2.5.6.1 Dashboard excel per il calcolo delle ore

Il gruppo si è munito di un file excel condiviso dove è stata realizzata una dashboard per tenere traccia, in un unico file, degli sprint. Ogni sprint traccia, a sua volta, le seguenti informazioni per ogni issue appartenente allo sprint:

1. ID issue.
2. Ruolo di chi ha svolto la issue.
3. Ore preventivate.
4. Ore effettive.

Questo file excel aiuta il gruppo ad accumulare le informazioni utili per compilare e tenere aggiornato il documento Piano di progetto. La dashboard viene aggiornata in modo automatico attraverso una GitHub Action che preleva, dalla issue, il relativo sprint associato, l'ID, il ruolo e le ore preventivate. Le ore effettive, invece, vengono prese all'apertura della pull request della relativa issue e, per convenzione, devono essere riportate nel primo commento.

2.6 Gestione di progetto

2.6.1 SCRUM

Il gruppo ha deciso di utilizzare alcuni concetti del $framework_G$ di gestione di progetto $agile_G$ chiamato SCRUM. SCRUM richiede l'utilizzo di un $modello\ di\ sviluppo\ a\ periodi_G$ chiamati **sprint**. Il gruppo ha deciso di utilizzare sprint della durata di una settimana(da giovedì a giovedì) per i seguenti motivi:

- Aumenta il coinvolgimento dei membri del gruppo.
- Ottiene feedback continui riducendo la gravità di eventuali problematiche.

Il framework definisce un insieme di:

1. **Artefatti**: informazioni e strumenti a supporto delle cerimonie.
2. **Ruoli**: descrivono le responsabilità chiave dei membri del gruppo di lavoro.
3. **Cerimonie**: incontri interni al gruppo eseguite in momenti precisi del ciclo di vita del software.

Il gruppo come detto non segue il framework alla lettera ma prende spunto da alcuni dei suoi elementi. SCRUM è stato scelto perché è molto flessibile e molto utilizzato.

2.6.2 Ruoli

Il gruppo seguendo le regole di progetto abbraccia una definizione e distinzione più rigida dei ruoli dei membri del gruppo rispetto a quella data da SCRUM. Di seguito vengono elencate le definizioni dei ruoli e la regola di rotazione.

2.6.2.1 Responsabile

Il responsabile del progetto è incaricato di coordinare le attività del gruppo di lavoro, pianificare e monitorare i progressi, e gestire efficacemente le risorse disponibili. In sintesi, egli si assicura che il progetto venga portato a termine nei tempi stabiliti e in conformità con le risorse assegnate.

2.6.2.2 Amministratore

L'amministratore di progetto ha il compito di gestire il way of working del gruppo e gli strumenti a supporto dello stesso.

2.6.2.3 Analista

L'analista è responsabile dell'analisi delle funzionalità del software, definendo i requisiti e i casi d'uso pertinenti.

2.6.2.4 Progettista

Il progettista è responsabile della definizione dell'architettura del software, identificando le componenti e le relazioni tra di esse, sulla base dei requisiti stabiliti dall'analista.

2.6.2.5 Programmatore

Il programmatore si occupa di scrivere il codice sorgente del software, seguendo le specifiche elaborate dal progettista.

2.6.2.6 Verificatore

Il verificatore di un progetto software ha il compito di garantire che il software prodotto e la documentazione associata siano conformi alle normative e alle specifiche definite.

2.6.2.7 Rotazione dei ruoli

Per aumentare la produttività il gruppo ha deciso di preassegnare solo il ruolo di responsabile. Ogni richiesta di modifica pianificata durante lo sprint specificherà il ruolo per il quale è competenza svolgerla. Di conseguenza, il membro che si auto-assegnerà la relativa issue assumerà il ruolo a essa associata per il tempo necessario al suo compimento. Così facendo è possibile massimizzare l'efficienza del gruppo, infatti ogni membro ha la possibilità di ottimizzare il tempo a disposizione realizzando le richieste di modifica libere.

2.6.3 Artefatti

Le cerimonie usate dal gruppo sono: product backlog e sprint backlog. Di seguito vengono spiegati questi artefatti indicando:

- Descrizione.
- Scopo.

2.6.3.1 Product backlog

Il product backlog è una $Kanban\ board_G$ che contiene tutte le richieste di modifica che riguardano il progetto. Le colonne del product backlog come implementato dal gruppo sono: "To Do", "In Progress", "To Review", "In Review", "Re Open" e "Done". Il significato delle colonne viene spiegato alla sezione Ciclo di vita richieste di modifica. L'implementazione concreta è indicata alla sezione GitHub projects.

2.6.3.2 Sprint backlog

Kanban board uguale alla product backlog con la differenza che contiene le richieste di modifica dello sprint corrente. L'implementazione concreta è indicata alla sezione GitHub projects.

2.6.4 Cerimonie

Le cerimonie usate dal gruppo sono: retrospettiva, revisione e pianificazione. Di seguito vengono spiegate le cerimonie indicando:

- Momento in cui viene eseguita.
- Membri del gruppo coinvolti.
- Scopo.
- Artefatti o documenti influenzati.

2.6.4.1 Retrospettiva

Riunione eseguita a fine sprint, coinvolge tutto il gruppo e ha lo scopo di discutere le problematiche riscontrate. I responsabili designati per lo sprint appena terminato devono riportare le problematiche riscontrate nella sezione retrospettiva dello sprint nel documento piano di progetto.

2.6.4.2 Revisione

Eseguita a fine sprint, coinvolge i Responsabili designati per lo sprint. Questa cerimonia produce la modifica del **piano di progetto** eseguendo:

1. Per ogni richiesta di modifica facente parte della sprint board indicare nella sezione consuntivo dello sprint corrente:
    - Identificativo.
    - Figura professionale che lo ha preso in carico.
    - Risorse utilizzate.
    - Confronto tra risorse utilizzate e quelle preventivate.
    - Stato in cui si trova.
2. Aggiornamento delle risorse rimaste nella relativa sezione dello sprint corrente.
3. Modifica dei rischi nel piano di progetto.
4. Documentare gli eventuali rischi riscontrati e l'efficacia delle azioni di mitigazione applicate.
5. Analisi ed eventuale modifica del preventivo dei costi.

2.6.4.3 Pianificazione

La pianificazione è la cerimonia principale, viene realizzata all'inizio di ogni sprint e coinvolge l'intero gruppo di lavoro. Lo scopo è quello di aggiornare il product backlog e definire quali richieste di modifica in esso contenute devono essere poste nello sprint backlog. La pianificazione delle attività segue la seguente mappatura:

1. **Identificazione delle attività**.

   Durante questa fase il gruppo guidato dal responsabile ha il compito di valutare l'avanzamento del progetto identificando le nuove richieste di modifica.

2. **Analisi delle attività**.

   Il responsabile deve associare a ogni richiesta di modifica il ruolo che deve implementarla tra i ruoli definiti nella sezione Ruoli. Il responsabile deve anche associare a ogni richiesta di modifica un valore di priorità in una scala da 1(minimo) a 3(massimo) e una stima delle ore previste per il loro completamento. Se una richiesta di modifica richiede una quantità di ore che supera le 8 è necessario suddividerla in più richieste di modifica. Il risultato della identificazione e della analisi delle attività è un documento condiviso realizzato su Google Docs dove per ogni attività ne vengono indicate le seguenti caratteristiche:

   - File coinvolti.
   - Descrizione a punti del lavoro da svolgere.
   - Ruolo di competenza.
   - Priorità.
   - Ore stimate per l'implementazione.

3. **Modifica product backlog**.

   Le richieste di modifica risultanti dalla precedente fase vengono registrate nel product backlog. Questa operazione viene implementata usando le informazioni indicate alla sezione Issue.

4. **Pianificazione sprint**.

   Questa fase consiste nella realizzazione di un piano da seguire durante lo svolgimento dello sprint successivo. Il responsabile deve:

   (a) **Calcolo ore disponibili per lo sprint**.

   Il responsabile deve chiedere a ogni membro le ore che può dedicare al successivo sprint. Così facendo riesce a calcolare le ore totali produttive che il gruppo si impegna a dedicare nella settimana a venire. Se le ore a

disposizione si discostano troppo dal valore $\frac{OreTotaliRTB}{SettimaneRTB}$ il responsabile deve identificare se possibile alcuni membri che danno disponibilità maggiore. Le ore usate da ogni membro devono essere memorizzate di modo che nei successivi sprint l'impegno di ognuno venga ribilanciato.

(b) **Calcolo ore totali svolgimento attività**.

Il responsabile deve selezionare il maggior numero di attività con priorità maggiore che possono essere portate a termine nel prossimo sprint.

5. **Popolazione sprint backlog**.

Popolare lo sprint backlog usando le richieste di modifica risultanti dalla precedente fase. Il popolamento viene fatto usando le informazioni indicate alla sezione Issue.

2.6.4.4 Controllo

Il responsabile ha il compito di mantenersi in contatto con i membri del gruppo durante ogni sprint. In particolare deve monitorare giornalmente lo stato di avanzamento del lavoro prefissato contattando tutti i membri tramite i canali di comunicazione interni indicati nella sezione Canali di comunicazione interni. Ogni giorno il responsabile deve verificare di quali attività si è già iniziato lo svolgimento e monitorarne lo stato di avanzamento. Il responsabile deve:

1. **Aggiornare la pianificazione sprint**.

Se di uno o più membri del gruppo notificano la necessità di più ore di lavoro rispetto alla previsione per terminare un'attività, il responsabile se lo ritiene necessario, deve aggiornare la pianificazione dello sprint.

Per fare ciò il responsabile deve richiedere ai membri del gruppo le ore in "eccedenza" necessarie per poi:

(a) Identificare le attività di priorità minima che devono ancora essere assegnate a un membro.

(b) Valutare il numero di attività che devono essere posticipate allo sprint successivo, basandosi sulle ore di lavoro totali disponibili rimaste e il preventivo di tempo necessario al completamento delle attività ancora da svolgere.

(c) Aggiornare lo sprint backlog togliendo le attività che dovranno essere posticipate.

In questo modo si porteranno a termine il maggior numero di attività possibili.

2. **Controllo dei rischi**.

   Il responsabile deve monitorare che i rischi individuati tramite l'attività di Gestione dei rischi vengano gestiti correttamente.

   Il responsabile deve rendersi conto del verificarsi dei rischi. Per fare ciò deve:

   (a) Tenersi in contatto con i membri del gruppo per assicurarsi dell'assenza di rischi.

   (b) Conoscere le strategie di rilevazione di ogni rischio che devono essere applicate giornalmente.

   Nel caso in cui un rischio si presenti il responsabile deve attuare nel minor tempo possibile tutte le attività di mitigazione specifiche indicate per il rischio stesso. Le attività di mitigazione possono richiedere il coinvolgimento con uno o più membri del gruppo. In tal caso il responsabile deve accertarsi che le attività vengano applicate correttamente.

2.7 Gestione dei rischi

Lo standard ISO 31000 è uno standard internazionale per la gestione dei rischi, progettato per essere applicabile in qualsiasi tipo di contesto, compresa la progettazione software. Nello specifico fornisce un quadro strutturato per la gestione dei rischi, composto dalle seguenti fasi principali:

- **definizione del contesto**: comprendere l'ambiente esterno e interno in cui si sta operando e gli obiettivi del sistema.
- **identificazione dei rischi**: individuare i rischi e i fattori che potrebbero compromettere il raggiungimento degli obiettivi e dei requisiti necessari allo sviluppo.
- **analisi dei rischi**: comprendere la natura dei rischi, compresa la causa per cui potrebbero verificarsi e la probabilità, le conseguenze sul sistema e sull'attività di lavoro.
- **ponderazione dei rischi**: valutare la gravità che comporterebbe il verificarsi di un rischio, identificare se è accettabile il verificarsi di uno o più rischi e quali invece dovrebbero richiedere un trattamento immediato.
- **trattamento dei rischi**: identificare i ruoli che hanno responsabilità nel trattamento dei rischi, i processi di mitigazione e le azioni da svolgere, nel caso dovesse verificarsi un rischio, per ridurre l'impatto sul sistema e sullo svolgimento del prodotto software.

- **monitoraggio**: identificazione delle metodologie che permettono un continuo monitoraggio e rilevamento dei rischi, identificazione anche di eventuali nuovi rischi con conseguente aggiornamento della documentazione relativa alla analisi dei rischi.
- **comunicazione**: indicare strategie di comunicazione e ruoli coinvolti a seguito del rilevamento di un rischio indicato all'interno dell'analisi dei rischi.