Roundabout - Etherless

Norme di Progetto

Versione	1.0.2
Approvazione	Veronica Barbieri
Redazione	Veronica Barbieri
	Luca Benetazzo
	Nicoletta Fabro
	Egon Galvani
Verifica	Luca Benetazzo
	Alessandro Sgreva
	Nicoletta Fabro
Stato	Approvato
Uso	Interno
Destinato a	*Roundabout*
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin

Descrizione

Questo documento racchiude le regole, le convenzioni e gli strumenti utilizzati dal gruppo Roundabout per il progetto Etherless

`team.roundabout.13@gmail.com`

Registro delle modifiche

Versione	Data	Nominativo	Ruolo	Descrizione
1.0.2	2020-05-01	Alessandro Sgreva	*Amministratore*	Approfondimento sezione §2.1, introduzione appendice A e revisione generale.
1.0.1	2020-04-30	Luca Benetazzo	*Responsabile*	Stesura §3.3 e aggiornamento indice.
1.0.0	2020-04-11	Veronica Barbieri	*Responsabile*	Approvazione del documento.
0.5.0	2020-04-26	Luca Benetazzo	*Verificatore*	Revisione modifiche apportate al documento.
0.4.3	2020-04-25	Alessandro Sgreva	*Amministratore*	Approfondimento sezioni §2.1, §3.4, §3.6 e Processi Organizzativi.
0.4.2	2020-04-25	Egon Galvani	*Amministratore*	Approfondimento sezioni §2.2, §3.1.7 e §3.5
0.4.1	2020-03-23	Veronica Barbieri	*Amministratore*	Stesura §3.4.
0.4.0	2020-03-23	Alessandro Sgreva	*Verificatore*	Revisione e modifica Processi Primari e Processi di Supporto.
0.3.2	2020-03-22	Veronica Barbieri	*Amministratore*	Correzioni a seguito della verifica e Stesura §3.2, §3.4 e §3.5.
0.3.1	2020-03-22	Nicoletta Fabro	*Amministratore*	Correzione imprecisioni Processi Primari.
0.3.0	2020-03-22	Alessandro Sgreva	*Verificatore*	Revisione e modifica §2.1 §2.2 e Processi Organizzativi.
0.2.3	2020-03-21	Nicoletta Fabro	*Amministratore*	Stesura §2.2 Sviluppo.
0.2.2	2020-03-20	Nicoletta Fabro	*Amministratore*	Stesura §2.1 Fornitura.
0.2.1	2020-03-19	Luca Benetazzo	*Amministratore*	Correzione imprecisioni Processi Organizzativi.
0.2.0	2020-03-19	Alessandro Sgreva	*Verificatore*	Revisione e modifica §3.1 Documentazione.
0.1.1	2020-03-18	Veronica Barbieri	*Amministratore*	Stesura §3.1 Documentazione.

Versione	Data	Nominativo	Ruolo	Descrizione
0.1.0	2020-03-17	Alessandro Sgreva	*Verificatore*	Revisione e modifica del documento.
0.0.3	2020-03-16	Luca Benetazzo	*Amministratore*	Stesura Processi Organizzativi.
0.0.2	2020-03-16	Nicoletta Fabro	*Amministratore*	Stesura Introduzione.
0.0.1	2020-03-12	Nicoletta Fabro	*Amministratore*	Creazione documento $\LaTeX_G$ e struttura generale.

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di fissare le regole, le convezioni e le tecnologie che i membri del gruppo *Roundabout* devono adottare durante tutto il corso del progetto, al fine di garantire uniformità nello svolgimento del lavoro e collaborazione tra tutti i membri del team. Verrà utilizzato un approccio incrementale, in modo da normare ogni decisione discussa e applicata dal team. Le attività presenti all'interno di questo documento sono state prese da processi appartenenti allo standard ISO/IEC 12207:1995.
Ciascun componente ha il dovere di prendere visione di tale documento e a rispettare le norme in esso descritte.

1.2 Scopo del prodotto

L'applicativo che si vuole sviluppare è *Etherless*, una piattaforma cloud$_G$ che sfrutta la tecnologia degli smart contract$_G$ caratteristica del network Ethereum$_G$. Lo scopo di *Etherless* è duplice: da una parte permettere agli *sviluppatori software* di rilasciare funzioni Javascript$_G$ nel cloud$_G$, dall'altra permettere agli *utenti* di beneficiare di queste funzioni in seguito ad un pagamento per il loro uso. *Etherless* è gestita e mantenuta dai suoi *amministratori*.

1.3 Glossario

Al fine di evitare possibili ambiguità, i termini tecnici utilizzati nei documenti formali vengono chiariti ed approfonditi nel *Glossario 2.0.0*. Per facilitare la lettura, i termini presenti in tale documento sono contrassegnati in tutto il resto della documentazione da una 'G' a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato$_G$ d'appalto C2 - Etherless**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C2.pdf;
- **Standard ISO 8601**:
  https://it.wikipedia.org/wiki/ISO_8601.

1.4.2 Riferimenti informativi

- **Standard ISO/IEC 12207:1995**:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf;
- **Guide to the Software Engineering Body of Knowledge (SWEBOK), V3.0**:
  https://www.computer.org/education/bodies-of-knowledge/software-engineering;
- **Ian Sommerville. Software Engineering. 9th Edition. 2010. 615 pages**;
- **Slide del corso Ingegneria del Software - Diagrammi delle classi**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01b.pdf;

- **Slide del corso Ingegneria del Software - Diagrammi dei package**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01c.pdf;
- **Slide del corso Ingegneria del Software - Diagrammi di sequenza**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E02a.pdf;
- **Slide del corso Ingegneria del Software - Diagrammi di attività**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E02b.pdf;
- **Slide del corso Ingegneria del Software - Analisi dei Requisiti**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L08.pdf;
- **Documentazione LaTeX**$_G$: https://www.latex-project.org/help/documentation/;
- **Airbnb**$_G$ **JavaScript**$_G$ **Style Guide**: https://github.com/airbnb/javascript/blob/master/README.md;
- **Solidity**$_G$ **Style Guide**: https://solidity.readthedocs.io/en/v0.5.7/style-guide.html
- **Sito ufficiale Git**$_G$: https://git-scm.com/.

2 Processi Primari

2.1 Fornitura

2.1.1 Descrizione

Il processo di fornitura ha lo scopo di determinare l'insieme delle attività e dei compiti necessari allo svolgimento del progetto. In questa sezione vengono esposte le regole che i membri del team *Roundabout* si impegnano a rispettare nel corso delle fasi di progettazione, sviluppo e consegna della piattaforma *Etherless* per diventare fornitori nei confronti del Proponente$_G$ *RedBabel* e dei Committenti$_G$ Prof. Tullio Vardanega e Prof. Riccardo Cardin.
Le attività di cui questo processo è composto, descritte poi nel dettaglio, sono:

- avvio;
- preparazione della proposta;
- contrattazione;
- pianificazione;
- esecuzione e controllo;
- revisione e valutazione;
- consegna e completamento.

L'obiettivo del gruppo è mantenere un costante dialogo con il Proponente$_G$ al fine di:

- instaurare un rapporto di collaborazione;
- comprenderne a fondo le richieste;
- determinare vincoli sui processi e sui requisiti;
- stimare i costi;
- promuovere una verifica continua;
- avere un riscontro efficace sul lavoro svolto.

2.1.2 Attività

2.1.2.1 Avvio

Durante quest'attività il gruppo conduce un'accurata analisi e valutazione dei requisiti presenti nelle varie proposte di capitolato$_G$, sotto coordinazione del Responsabile di Progetto. Tale valutazione culmina con l'accettazione o meno di ogni singola proposta.

Studio di Fattibilità

Lo *Studio di Fattibilità* viene redatto dagli Analisti ed indica il risultato dell'attività di valutazione di ogni capitolato$_G$ proposto. Tra le informazioni da esso fornite sono presenti:

- **informazioni generali**: presenta il nome del progetto, del Proponente$_G$ e del Committente$_G$;
- **descrizione**: descrive sinteticamente il capitolato$_G$ sotto analisi;
- **obiettivo finale**: rappresenta il prodotto$_G$ risultante dal completamento del progetto con soddisfacimento dei requisiti;
- **tecnologie coinvolte**: descrive le tecnologie necessarie al raggiungimento dell'obiettivo finale;
- **aspetti positivi**: espone gli elementi del capitolato$_G$ che hanno suscitato entusiasmo nel team;
- **criticità**: analizza i punti critici ed i fattori di rischio relativi alla realizzazione del progetto;
- **esito**: illustra la posizione finale del team nei confronti del capitolato$_G$.

2.1.2.2 Preparazione della proposta

Il gruppo definisce ed appronta una proposta da consegnare in risposta alle richieste del Proponente$_G$.

2.1.2.3 Contrattazione

Il gruppo entra in contatto con il Committente$_G$ per negoziare ed infine stipulare il contratto di fornitura del prodotto$_G$ software.

2.1.2.4 Pianificazione

Il gruppo conduce una revisione dei requisiti di acquisizione, andando poi a definire un piano di gestione del progetto in grado di garantire la qualità del prodotto$_G$ software che si va a concretizzare. È durante lo svolgimento di questa attività che ha inizio il rapporto di collaborazione tra il gruppo ed il Proponente$_G$, la cui corretta gestione garantisce una più approfondita comprensione delle sue richieste e, conseguentemente, una più precisa determinazione dei vincoli da porre sui processi e dei requisiti del prodotto$_G$.

Piano di Progetto

Il Responsabile di Progetto, con l'aiuto degli Amministratori, redige un *Piano di Progetto* volto a pianificare le attività del team. Tale documento contiene:

- **analisi dei rischi**: viene effettuata un'approfondita attività di analisi dei fattori di rischio e vengono approntate eventuali procedure attraverso cui limitare la loro occorrenza ed i loro effetti. Le valutazioni effettuate vertono principalmente verso la probabilità che un rischio si verifichi e l'entità del danno che questo è in grado di di provocare;
- **modello di sviluppo**: a seguito delle opportune valutazioni viene scelto un modello di sviluppo da utilizzare e seguire per la pianificazione e lo svolgimento del progetto;

- **pianificazione**: in funzione del modello di sviluppo scelto si effettua una pianificazione quanto più dettagliata possibile delle attività da eseguire nelle diverse fasi del progetto, stabilendo preventivamente opportune scadenze temporali e tenendo giusta considerazione degli obiettivi di qualità;
- **preventivo**: sulla base delle attività pianificate e della loro redistribuzione ai vari ruoli di progetto viene data una stima della quantità di lavoro necessaria per ogni fase. In funzione di tale stima viene stilata una proposta di preventivo per il costo totale del progetto, da presentare a Proponente$_G$ e Committente$_G$;
- **consuntivo di periodo**: vengono rendicontate le spese, totali e per ruolo, sostenute all'interno di ciascuna fase e data una valutazione dell'andamento rispetto a quanto era stato preventivato.

Piano di Qualifica

I Progettisti ed i Verificatori redigono il *Piano di Qualifica*, il cui scopo è raccogliere le metriche e le strategie impiegate per garantire la qualità sia dei processi attuati che del prodotto$_G$ nel tempo. Tale documento contiene:

- **qualità di processo**: vengono individuati processi utili allo svolgimento del progetto a partire dagli standard, vengono identificati degli obiettivi funzionali e di qualità, ed infine stabilite procedure di controllo a garanzia del raggiungimento di tali obiettivi in funzione delle risorse disponibili;
- **qualità di prodotto$_G$**: vengono identificate le caratteristiche principali necessarie affinché il prodotto$_G$ soddisfi i requisiti previsti. Vengono poi stabilite procedure e metriche di controllo in grado di garantire l'ottenimento di tali caratteristiche e verificarne la qualità;
- **specifiche dei test**: devono essere definiti nel dettaglio dei test utili al collaudo del prodotto$_G$, e metodologie da implementare per garantire il loro corretto svolgimento;
- **standard di qualità**: una volta scelti adeguati standard di qualità, questi vengono presentati e descritti;
- **valutazioni per il miglioramento**: vengono riportate eventuali problematiche riscontrate nello svolgimento di una determinata attività o nel ricoprire uno specifico ruolo, corredate dalla relativa soluzione;
- **resoconto delle attività di verifica**: i risultati delle attività di verifica, eseguite con riferimento alle metriche previste, devono essere inseriti in un report. Tale documentazione viene poi fornita ai soggetti interessati ed eventualmente al Proponente$_G$, qualora sia coinvolto nell'attività.

2.1.2.5 Esecuzione e controllo

Il gruppo implementa il piano di gestione del progetto stilato durante la fase di pianificazione e procede con lo sviluppo del prodotto$_G$ software. Durante il ciclo di vita di tale prodotto$_G$ è fondamentale monitorare e controllare iterativamente il progresso del progetto, nonché il suo livello di qualità. Importante è anche il coinvolgimento del Proponente$_G$ per la dimostrazione e valutazione dei prototipi prodotti, così da aggiornare quest'ultimo sul progresso del lavoro e sugli obiettivi raggiunti.

2.1.2.6 Revisione e valutazione

È compito del fornitore condurre iterativamente procedure di verifica e validazione del software prodotto, qualora possibile anche in collaborazione con il Proponente$_G$, allo scopo di dimostrare il soddisfacimento dei requisiti concordati.

2.1.2.7 Consegna e completamento

Il gruppo consegna il prodotto$_G$ software completato e fornisce eventuale servizio di supporto, come specificato dal contratto.

2.1.3 Metriche

Non sono state attualmente individuate metriche specifiche al processo di fornitura.

2.1.4 Strumenti

Non sono stati attualmente individuati strumenti specifici al processo di fornitura.

2.2 Sviluppo

2.2.1 Descrizione

Il processo di sviluppo contiene le attività ed i compiti che devono essere svolti per produrre il software richiesto. Esso viene svolto in conformità allo standard ISO/IEC 12207:1995, comprendendo perciò le seguenti attività:

- Analisi;
- Progettazione;
- Codifica.

In particolare, l'obiettivo del team è sviluppare un prodotto$_G$ software che superi i test, soddisfi le richieste ed i requisiti, fissati dal Proponente$_G$.

2.2.2 Attività

2.2.2.1 Analisi

L'attività di analisi consiste nello studio dei bisogni e delle fonti del dominio applicativo. Ha come obiettivo una modellazione concettuale del sistema mediante l'assegnazione di requisiti a parti distinte del sistema.

Analisi dei Requisiti

Nell'*Analisi dei Requisiti* gli Analisti individuano ed elencano tutti i requisiti richiesti dal Proponente$_G$ per il progetto in questione. Tali requisiti possono derivare da:

- capitolato$_G$ d'appalto;
- verbali di riunioni interne o esterne;
- casi d'uso$_G$.

E sono fondamentali per:

- descrivere lo scopo del lavoro;
- fornire ai Progettisti riferimenti specifici ed affidabili;
- fissare funzionalità concordate con il cliente;
- fornire una base per raffinamenti successivi al fine di garantire un miglioramento continuo;
- facilitare le revisioni del codice;
- stimare i costi in base alla quantità di lavoro prevista.

Casi d'uso

Dopo aver identificato i casi d'uso$_G$, è compito degli Analisti elencare questi ultimi con un grado di precisione che va dal generale al particolare, usando la struttura:

- **codice identificativo**;
- **titolo**;
- **diagramma UML$_G$**;
- **attori primari**;
- **attori secondari**;
- **descrizione**;
- **scenario principale**;
- **specializzazioni (se presenti)**;
- **inclusioni (se presenti)**;
- **estensioni (se presenti)**;
- **precondizione**;
- **postcondizione**.

Ogni caso d'uso$_G$ dev'essere corredato da un codice identificativo, che segue la dicitura:

UC[codice_padre].[codice_figlio]

Dove:

- **Codice padre**: numero che identifica univocamente i casi d'uso$_G$ generici;
- **Codice figlio**: numero progressivo che identifica i sottocasi. Può a sua volta includere altri livelli.

Requisiti

Ogni requisito emerso durante l'attività di analisi dev'essere descritto dalla seguente struttura:

- codice identificativo;
- fonti;
- relazioni di dipendenza con altri requisiti;
- descrizione;
- importanza.

Ogni requisito dev'essere corredato da un codice identificativo, che segue la dicitura:

R[Importanza][Tipologia][Codice]

Dove:

- **Importanza**: indica il grado di importanza del requisito ai fini del progetto. Può assumere i valori:
  - **1**: requisito obbligatorio ai fini del progetto, irrinunciabile per gli stakeholders;
  - **2**: requisito desiderabile: non strettamente necessario ai fini del progetto ma che porta valore aggiunto;
  - **3**: requisito opzionale, contrattabile più avanti nel progetto.
- **Tipologia**: classe a cui appartiene il requisito in questione. Può assumere i valori:
  - **F**: funzionale;
  - **P**: prestazionale;
  - **Q**: qualitativo;
  - **V**: vincolo.
- **Codice**: identificatore univoco del requisito.

Il codice stabilito secondo la convenzione precedente, una volta associato ad un requisito, non può più essere modificato.

Inoltre per ogni requisito bisogna indicare:

- **descrizione**: breve descrizione del requisito, strutturata in maniera da evitare ambiguità;
- **classificazione**: indica il grado di importanza del requisito considerato. Sebbene tale informazione sia già presente nell'identificativo, la sua ripetizione rende la lettura più semplice e scorrevole;
- **fonti**:
  - *capitolato*$_G$: requisito indicato nel capitolato$_G$;
  - *interno*: requisito individuato dagli analisti;
  - *caso d'uso*$_G$: il requisito è stato estrapolato da uno o più casi d'uso$_G$. In questo caso vengono riportati gli identificativi dei casi d'uso$_G$ considerati;
  - *verbale*$_G$: si tratta di un requisito individuato a seguito di un incontro tra i membri del gruppo o di una richiesta di chiarimento con il Proponente$_G$. In questo caso è riportato il codice identificativo presente nella tabella delle decisioni dei verbali considerati.

Diagrammi UML

I diagrammi UML$_G$ devono essere realizzati utilizzando la versione del linguaggio *v2.0*. Per garantire leggibilità è richiesto che gli Analisti si occupino di creare i diagrammi UML$_G$ rispettando le seguenti indicazioni:

- distribuire in modo omogeneo gli elementi all'interno dello spazio disponibile, mantenendo un margine minimo;
- dove possibile, allineare i vari elementi sia in senso verticale sia in orizzontale;
- impostare i collegamenti in uscita da ogni elemento ad angolo retto.

2.2.2.2 Progettazione

L'attività di progettazione definisce una soluzione del problema presentato soddisfacente per tutti gli stakeholders, in relazione ai requisiti specificati nel documento *Analisi dei Requisiti 2.0.0*. Questo garantisce che il prodotto$_G$ sviluppato soddisfi le proprietà e i bisogni specificati dal Proponente$_G$. La progettazione ha quindi come obiettivo l'elaborazione di una struttura adeguata per il sistema, e si divide in:

- **Progettazione Architetturale**: vengono svolte operazioni di incremento, verifica dei requisiti e progettazione ad alto livello di un'architettura adeguata alla realizzazione del progetto. La Technology Baseline del progetto si ottiene alla fine di questo periodo;
- **Progettazione di Dettaglio**: vengono svolte operazioni di incremento, verifica dei documenti redatti; vengono definite le specifiche di dettaglio dell'architettura del prodotto$_G$, partendo dalla Tecnology Baseline. Particolare attenzione viene data alla codifica di tale prodotto$_G$ ed alla creazione di un manuale utente. La Product Baseline del progetto si ottiene alla fine di questo periodo;

I rispettivi output sono:

- ***Technology Baseline***$_G$: contiene le specifiche della progettazione ad alto livello, i diagrammi UML$_G$ utilizzati per la realizzazione dell'architettura ed i test di verifica;
- ***Product Baseline***$_G$: approfondisce l'attività di progettazione precedentemente trattata nella Technology Baseline$_G$, specifica le definizioni delle classi e definisce i test necessari alla verifica.

Technology Baseline

Deve contenere:

- **Tracciamento delle componenti**: viene rappresentata la relazione tra ogni requisito ed il componente che lo soddisfa, dimostrando concretamente il loro soddisfacimento;
- **Test di integrazione**: vengono definite delle classi di verifica per accertarsi che ogni componente del sistema funzioni come previsto;
- **Tecnologie utilizzate**: devono essere descritte le tecnologie utilizzate, specificandone l'utilizzo nel progetto e le motivazioni per cui sono state scelte.
- **Poc (Proof of Concept)**: eseguibile funzionale alla dimostrazione dell'adeguatezza delle scelte architetturali;

Product Baseline

Deve contenere:

- **Diagrammi UML**$_G$: usati per rendere più chiare le scelte progettuali adottate e ridurre le ambiguità. Possono essere:
  - diagrammi di attività: descrivono la logica procedurale di un flusso di operazioni, aiutando a descrivere gli aspetti dinamici dei casi d'uso$_G$;
  - diagrammi delle classi: descrivono le classi presenti all'interno del sistema, soffermandosi sui loro metodi, attributi e relazioni;
  - diagrammi dei package: descrivono le relazioni di dipendenza presenti tra classi raggruppate in package diversi, ossia in raggruppamenti di un numero arbitrario di elementi in unità di livelllo più alto;
  - diagrammi di sequenza: descrivono la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento;
- **Design pattern**$_G$: vengono esplicitati chiaramente i design pattern$_G$ utilizzati per l'architettura, accompagnandoli con una descrizione ed un diagramma così da esporne il significato e la struttura;
- **Definizione delle classi**: ogni classe viene descritta illustrandone lo scopo e le funzionalità;
- **Tracciamento delle classi**: ogni requisito viene tracciato, in modo da garantire che ogni classe ne soddisfi almeno uno;
- **Test di unità**: vengono definiti dei test di unità per verificare che le componenti del sistema funzionino come previsto.

Diagrammi delle classi

I diagrammi delle classi devono descrivere le tipologie di oggetti presenti all'interno del sistema e le relazioni di dipendenza tra essi presenti. Ogni classe viene rappresentata tramite un rettangolo tripartito in senso orizzontale, che conterrà rispettivamente:

1. **nome della classe**: che deve essere univoco, scritto con la lettera maiuscola e in inglese. Nel caso in cui la classe sia astratta, il nome deve essere scritto in italico; nel caso in cui si tratti invece di un'interfaccia, il nome dovrà essere preceduto dal termine `«interface»`;
2. **attributi (opzionali)**: rappresentano lo stato interno della classe. Ogni attributo deve essere indicato nel seguente modo:

   visibilità nome : tipo [molteplicità] = default {proprietà aggiuntive}

   dove:
   - **visibilità**: visibilità dell'attributo rispetto all'esterno della classe, può essere pubblica, protetta o privata;
   - **nome**: nome dell'attributo;
   - **tipo**: tipo dell'attributo;

- **molteplicità (opzionale)**: numero di occorrenze dell'attributo all'interno della classe;
- **default(opzionale)**: eventuale valore predefinito dell'attributo;
- **proprietà aggiuntive**: eventuali informazioni aggiuntive relative all'attributo considerato;

3. **operazioni (opzionali)**: rappresentano le azioni che la classe è in grado di compiere. Ogni operazione deve essere indicata nel seguente modo:

   visibilità nome (lista-parametri) : tipo-ritorno {proprietà aggiuntive}

   dove:

   - **visibilità**: visibilità dell'operazione rispetto all'esterno della classe, può essere pubblica, protetta o privata;
   - **nome**: nome dell'operazione;
   - **lista-parametri**: lista di parametri dell'operazione; per ogni parametro dovranno essere indicate le seguenti proprietà:
     - **direzione (opzionale)**: modalità di accesso al parametro, può essere in lettura, in scrittura o entrambe; di default è in lettura;
     - **nome**: nome del parametro;
     - **tipo**: tipo del parametro;
     - **default (opzionale)**: eventuale valore di default del parametro;
   - **tipo-ritorno**: tipo di ritorno della funzione considerata;
   - **proprietà aggiuntive(opzionali)**: eventuali informazioni aggiuntive relative alla funzione;

Le varie classi possono essere collegate tra di loro tramite apposite frecce, che esplicitano le relazioni di dipendenza presenti tra esse. In particolare, i gradi di dipendenza considerati sono:

- **dipendenza**: indicata con la freccia tratteggiata, si ha quando la classe utilizza un oggetto della classe con cui si relaziona;
- **aggregazione**: indicata con la freccia "a diamante" vuota, si ha quando la classe considerata presenta come attributo almeno un riferimento alla classe con cui si relaziona;
- **composizione**: indicata con la freccia "a diamante" piena, si ha quando la classe contiene come attributo almeno un oggetto della classe con cui si relaziona;
- **associazione**: indicata con una linea semplice, si ha quando la classe crea e utilizza un oggetto della classe con cui si relaziona;
- **generalizzazione**: indicata con la freccia vuota, viene usata per indicare relazioni di tipo "Is-A".

Diagrammi dei package

Ogni package deve essere rappresentato tramite un rettangolo con un'etichetta per il nome, in inglese e scritta in minuscolo. Ogni package può contenere al suo interno altri package o classi. Le dipendenze tra package devono essere indicate con frecce tratteggiate che dovrebbero seguire tutte le medesima direzione, in modo da evitare la creazione di dipendenze cicliche.

Diagrammi di attività

I diagrammi di attività permettono di descrivere i processi che compongono l'applicazione software attraverso dei grafi in cui i nodi rappresentano le azioni e gli archi l'ordine in cui esse sono eseguite. Gli elementi usati in tali diagrammi sono i seguenti:

- **nodo iniziale**: rappresentato da un pallino pieno, porta alla generazione di un token ed è il punto d'inizio dell'esecuzione dell'attività;
- **activity**: rappresenta un'azione all'interno dell'attività, viene indicata da un rettangolo che ne contiene la descrizione;
- **subactivity**: è rappresentata da un rettangolo che ne contiene il nome, viene usata per riferirsi ad una sottoattività il cui diagramma deve essere fornito separatamente;
- **branch**: tali nodi modellano delle decisioni, vengono indicati tramite dei rombi vuoti e generalmente distinguono due branch. Il test che deve essere soddisfatto normalmente viene indicato da un'etichetta posta di fianco al nodo di branch;
- **merge**: punto in cui i rami generati da un branch si uniscono, vengono rappresentati nello stesso modo dei nodi di branch;
- **fork**: punto in cui l'attività si parallelizza senza vincoli di esecuzione temporale, viene indicato da una lunga linea orizzontale o verticale. Consuma un token e ne genera uno per ogni percorso in uscita;
- **join**: rappresentato allo stesso modo del fork, è un punto in cui avviene la sincronizzazione di processi paralleli. Consuma un token per ogni percorso in entrata e genera un solo token;
- **pin**: rappresentato da un quadratino da cui entrano o escono frecce, indica la produzione o consumo di un parametro, il cui tipo va indicato di fianco;
- **segnali**: vengono rappresentati tramite due figure "a incastro", la prima (non bloccante) per l'emissione del segnale, la seconda (bloccante) per la ricezione dello stesso. Rappresenta un evento esterno;
- **timeout**: sono rappresentati da una clessidra, permettono di modellare timeout ed eventi ripetuti;
- **nodo di fine flusso**: rappresentato da un cerchio vuoto con una X al centro. Porta al consumo di un token e rappresenta un punto di terminazione di un percorso di esecuzione; non causa la terminazione dell'esecuzione dell'attività;
- **nodo finale**: viene rappresentato da due cerchi concentrici: il più esterno vuoto e il più interno pieno. Porta al consumo di un token e rappresenta il punto di terminazione dell'esecuzione dell'attività.

Diagrammi di sequenza

I diagrammi di sequenza permettono di descrivere la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento.
Tali diagrammi devono essere letti verticalmente, dall'alto al basso, in modo da simulare lo scorrere del tempo. Ognuno degli oggetti coinvolti viene indicato tramite un rettangolo, che contiene il nome a lui associato.

All'interno del diagramma è consigliato l'utilizzo delle barre di attivazione, in modo da rendere immediatamente visibile a chi osserva lo stato di attività del diagramma.
La collaborazione tra gli oggetti rappresentati è resa possibile tramite lo scambio di messaggi, che viene indicato da apposite frecce. Un messaggio può essere di una delle seguenti tipologie:

- **messaggio sincrono**: viene indicato tramite una freccia piena, corrisponde alla chiamata sincrona di un metodo. Il chiamante deve attendere la risposta del chiamato prima di continuare l'esecuzione;
- **messaggio asincrono**: viene indicato tramite una freccia, corrisponde ad una chiamata asincrona. Il chiamante non deve aspettare la risposta del chiamato per continuare l'esecuzione;
- **messaggio di ritorno**: indicato con una freccia tratteggiata, rappresenta il ritorno di un metodo chiamato;
- **messaggio di creazione**: indicato con una freccia tratteggiata sormonata da `«create»`, rappresenta la creazione di un nuovo oggetto;
- **messaggio di distruzione**: freccia piena sormontata da `«destroy»`, indica la distruzione di un oggetto.

Per modellare in maniera più dettagliata le interazioni tra gli oggetti si consiglia l'utilizzo dei frame d'interazione. Ognuno di tali frame è caratterizzato dalle seguenti proprietà:

- **guardia**: indica la condizione di attivazione del frame;
- **etichetta**: viene usata per indicare la tipologia del frame d'interazione considerato. Le tipologie possibili sono:
  - **alt**: rappresenta un'alternativa tra più frame; viene eseguito solo quello per cui si verifica la condizione;
  - **opt**: rappresenta l'esecuzione opzionale di un frame, che avviene solo se la condizione specificata si verifica;
  - **par**: indica frammenti da eseguire in parallelo;
  - **loop**: indica che il frammento può essere eseguito più volte, la base dell'iterazione è indicata dalla guardia;
  - **region**: rappresenta una regione critica, il frammento può essere eseguito da un solo thread alla volta;
  - **neg**: indica un'iterazione non valida;
  - **ref**: si riferisce ad un'interazione definita in un altro diagramma;
  - **sd**: utilizzato per racchiudere un intero diagramma di sequenza.

Qualità dell'architettura

In seguito alla definizione dei requisiti, che consolida le funzionalità richieste, ci si occupa della realizzazione dell'architettura del sistema. Tale architettura dovrà godere delle seguenti proprietà:

- **sufficienza**: deve soddisfare i requisiti definiti nel documento *Analisi dei Requisiti 2.0.0*;

- **comprensibilità**: deve essere capibile da tutti gli stakeholder;
- **modularità**: deve essere suddivisa in parti chiare e ben distinte;
- **robustezza**: deve essere in grado di sopportare ingressi diversi, sia da parte dell'utente che dell'ambiente;
- **flessibilità**: deve permettere di essere modificata senza sostanziali modifiche e a costi contenuti al variare dei requisiti;
- **efficienza**: deve gestire le risorse in maniera da ridurre eventuali sprechi;
- **affidabilità**: l'architettura deve garantire una buona usabilità del prodotto$_G$ una volta implementata;
- **disponibilità**: la manutenzione delle sue parti deve richiedere un tempo limitato e non andare ad affliggere il corretto funzionamento di tutto il sistema;
- **sicurezza**: non deve presentare gravi malfunzionamenti o essere vulnerabile ad intrusioni;
- **semplicità**: ogni sua parte deve contenere solo il necessario e nulla di superfluo;
- **incapsulazione**: costituita da componenti le cui informazioni interne non sono visibii da fuori;
- **coesione**: composta in maniera che le parti con obiettivo comune siano raggruppate insieme;
- **basso accoppiamento**: composta da parti distinte che devono essere indipendenti tra di loro, in modo che l'architettura possa subire modifiche a costi contenuti;

2.2.2.3 Codifica

Questa attività ha come scopo normare l'effettiva realizzazione del prodotto$_G$ software richiesto. Tramite questa attività si concretizza la soluzione architetturale elaborata durante la progettazione, mediante la programmazione vera e propria. Gli sviluppatori, durante l' implementazione, dovranno attenersi alle norme di programmazione stabilite in maniera tale da:

- ottenere codice leggibile ed uniforme per i programmatori;
- agevolare le fasi di manutenzione, verifica e validazione;
- fornire un prodotto$_G$ conforme ai requisiti indicati dal Proponente$_G$;
- migliorare la qualità del prodotto$_G$.

La scrittura del codice dovrà inoltre perseguire gli obiettivi di qualità stabiliti nel documento *Piano di Qualifica 2.0.0*, in modo da garantire una buona qualità del codice.

Stile di codifica

- **indentazione**: i blocchi innestati devono essere correttamente indentati, usando per ciascun livello di indentazione quattro (4) spazi, fatta eccezione per i commenti. Ogni programmatore dovrà configurare adeguatamente il proprio IDE$_G$ al fine di rispettare tale norma;
- **parentesizzazione**: le parentesi di delimitazione dei costrutti vanno inserite in linea e non al di sotto di essi;
- **lunghezza dei metodi**: ove possibile, preferire metodi brevi (poche righe di codice) e che assolvano il minor numero di compiti possibile;
- **univocità dei nomi**: classi, variabili e metodi devono avere un nome univoco ed esplicativo (nome parlante), al fine di evitare ambiguità e consentire ad eventuali lettori di comprendere lo scopo di quel preciso elemento, anche senza conoscenze informatiche pregresse;
- **classi**: le parole componenti i nomi delle classi devono iniziare con la lettera maiuscola (e.g. NomeClasse);
- **costanti**: le costanti devono essere scritte usando solo lettere maiuscole (e.g. COSTANTE). Nel caso siano composte da più parole, queste devono essere separate dal carattere speciale underscore (e.g. NOME_COSTANTE);
- **metodi**: i nomi dei metodi devono iniziare con una lettera minuscola e, nel caso siano composti da più parole, quelle successive devono iniziare con una lettera maiuscola (e.g. nomeMetodo);
- **lingua**: le parti testuali di codice ed i commenti ad esso riferiti devono essere scritti in lingua inglese;
- **ricorsione**: l'uso della ricorsione va evitato quanto più possibile.

I componenti *Etherless-cli* ed *Etherless-server* dovranno essere codificati in TypeScript$_G$, seguendo la Airbnb$_G$ JavaScript$_G$ Style Guide. Per quanto riguarda *Etherless-smart*, questo dovrà essere sviluppato in Solidity$_G$, seguendo la relativa style guide presente nel sito ufficiale.

2.2.3 Metriche

2.2.3.1 Metriche di analisi dei requisiti

Percentuale dei Requisiti Obbligatori Soddisfatti

- **Descrizione**: la metrica indica la percentuale di requisiti obbligatori soddisfatti sui requisiti totali;
- **unità di misura**: la metrica viene espressa in percentuale;
- **formula**: $PROS = \frac{\#requisiti\_obbligatori\_soddisfatti}{\#requisti\_obbligatori\_totali} \times 100$;
- **risultato**: il risultato è compreso tra 0% e 100%, in particolare:
  - una valore inferiore al 100%, indica la presenza di alcuni requisiti obbligatori non ancora soddisfatti;
  - una valore pari al 100%, indica che tutti i requisiti obbligatori presentati sono stati soddisfatti.

2.2.3.2 Metriche di progettazione

Coupling Between Objects (CBO)

- **Descrizione**: la metrica CBO indica il numero di classi accoppiate ad una data classe;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **risultato**: in linea generale si può dire che:
  - un valore pari a 0 indica che la classe non ha alcuna relazione con nessun'altra classe presente all'interno del sistema;
  - un valore compreso tra 1 e 4 indica che la classe presenta un basso livello di accoppiamento;
  - un numero maggiore di 4 indica che la classe presenta un livello medio/alto di accoppiamento, che potrebbe complicarne il test e la manutenzione.

Structural Fan-In (SFIN)

- **Descrizione**: la metrica SFIN indica il numero di componenti che utilizzano un dato modulo$_G$;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **risultato**: un valore elevato di SFIN è indice di un riutilizzo consistente della componente considerata;

Structural Fan-Out (SFOUT)

- **Descrizione**: la metrica SFOUT indica il numero di componenti utilizzate dal modulo$_G$ considerato;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **risultato**: un valore elevato di SFOUT è indice di un eccessivo livello di accoppiamento della componente;

2.2.3.3 Metriche di codifica

Complessità ciclomatica

- **Descrizione**: la metrica è indice della complessità di una porzione di codice; in particolare rappresenta il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **formula**: $v(G) = e - n + 2p$, dove:
  - v(G): complessità ciclomatica del grafo G;
  - e: numero di archi del grafo;
  - n: numero di nodi del grafo;
  - p: numero di componenti connesse.

- **risultato**:
  - un valore troppo elevato del risultato indica un'eccessiva complessità del codice, possibile causa di difficile manutenzione;
  - un valore eccessivamente basso può indicare una scarsa efficienza dei metodi considerati.

Rapporto linee di codice per linee di commento (RCC)

- **Descrizione**: la metrica indica il rapporto tra il numero di linee di codice totali, escluse quelle vuote, e il numero di linee di commento presenti;
- **unità di misura**: la metrica viene espressa tramite un numero decimale;
- **formula**: $RCC = \frac{\#linee\_totali}{\#linee\_di\_commento}$.
- **risultato**: un valore troppo basso del risultato è indice di scarse informazioni necessarie alla comprensione del codice scritto.

Livello di annidamento

- **Descrizione**: la metrica indica il livello di annidamento dei metodi;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **risultato**: un alto valore del risultato indica un eccessivo livello di annidamento dei metodi, che risulteranno essere complessi e di difficile manutenzione;

Numero di parametri per metodo

- **Descrizione**: indica il numero di parametri del metodo considerato;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **risultato**: un numero eccessivo di parametri è indice di un'elevata complessità del metodo.

Numero di attributi per classe

- **Descrizione**: indica il numero totale di attributi presenti all'interno della classe;
- **unità di misura**: la metrica viene espressa tramite un numero intero;
- **risultato**: un numero elevato di attributi denota un carico di responsabilità eccessivo per la classe. In questo caso si consiglia di scomporre la classe seguendo principi quali il *Single Responsability Principle*.

2.2.4 Strumenti

Di seguito vengono elencati gli strumenti usati dal gruppo durante il processo di sviluppo:

2.2.4.1 Draw.io

Per la creazione dei diagrammi dei casi d'uso$_G$ è stato scelto Draw.io; esso permette una facile integrazione con Google Drive ed è già conosciuto dalla maggior parte dei membri del gruppo.

https://app.diagrams.net/

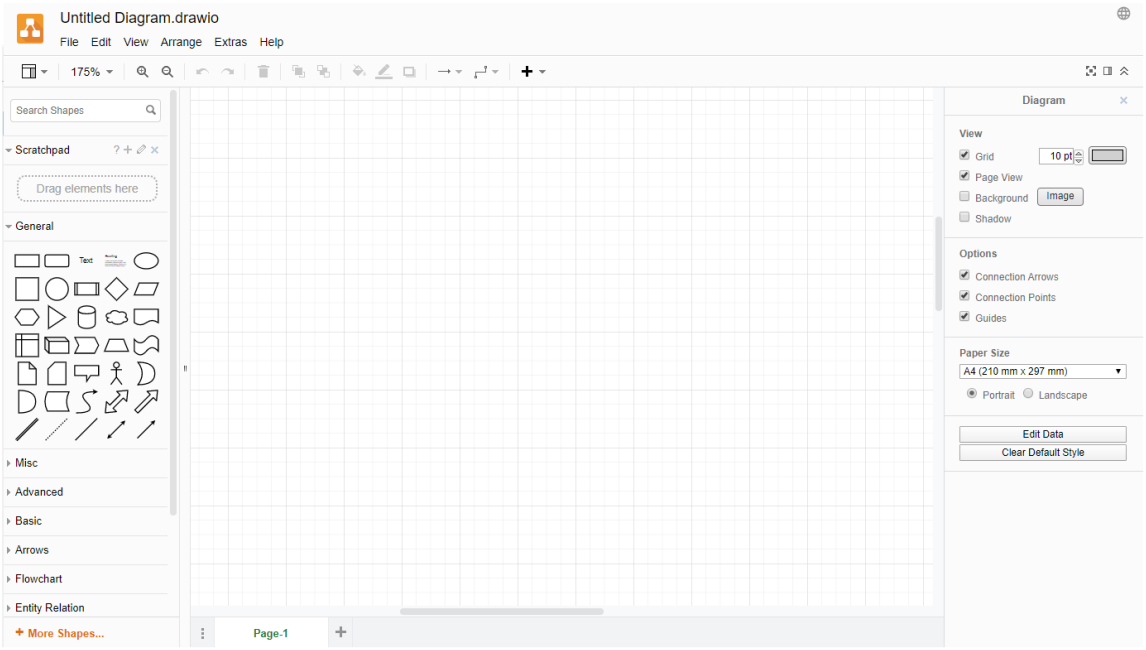

Figura 2.2.1: Schermata principale draw.io

2.2.4.2 ESLint

ESLint$_G$ è uno strumento di analisi statica per identificare pattern problematici all'interno di codice JavaScript$_G$. Tale strumento permette inoltre di descrivere e utilizzare regole personalizzate. ESLint$_G$ controlla sia la code quality che eventuali problemi riguardanti lo stile di codifica.

https://eslint.org/

2.2.4.3 Ethers.js

Ethers.js è una libreria che consente l'interazione con una blockchain Ethereumed il suo ecosistema.

https://docs.ethers.io/ethers.js

2.2.4.4 Ganache

Si tratta di un Ethereum client, con la particolarità di consentire la creazione di una blockchain$_G$ privata finalizzata all'esecuzione di test in locale.

https://www.trufflesuite.com/ganache

2.2.4.5 Husky

Strumento utilizzato per la prevenzione di bad $_G$commit nella Repository$_G$ Git$_G$.

https://github.com/typicode/husky

2.2.4.6 Node.js

Runtime di JavaScript open source, particolarmente utilizzata all'interno del progetto per l'ascolto di eventi.

https://nodejs.org/it/

2.2.4.7 NPM

Package manager$_G$ utilizzato per l'installazione di strumenti utilizzati durante il progetto.

https://docs.npmjs.com/

2.2.4.8 Open Zeppelin

Open Zeppelin è una libreria composta da smart contract$_G$ ben documentati, utilizzata all'interno del progetto.

https://openzeppelin.com/

2.2.4.9 Ropsten

Testnet pubblica usata per test e sperimentazioni su rete Ethereum$_G$. Permette di essere utilizzata senza preoccuparsi dei costi e senza possedere monete di reale valore.

https://ropsten.etherscan.io/

2.2.4.10 Serverless

Framework$_G$ utilizzato per lo sviluppo, il deploy$_G$ ed il monitoraggio della comunicazione con AWS Lambda$_G$.

https://www.serverless.com/

2.2.4.11 Solidity

Linguaggio utilizzato per lo sviluppo di smart contract$_G$.

https://solidity.readthedocs.io/en/v0.6.4/

2.2.4.12 Truffle

Framework$_G$ per lo sviluppo, test e gestione di smart contract$_G$. Permette la scrittura di test automatici sia in JavaScript$_G$ che in Solidity$_G$.

https://www.trufflesuite.com/truffle

2.2.4.13 Visual Studio Code

Visual Studio Code è l'IDE$_G$ scelto per la parte di codifica, in particolare tale scelta è dovuta a:

- completa compatibilità con Windows, Linux e macOs: in questo modo tutti i membri del gruppo possono riferirsi ad un unico strumento senza essere vincolati dal sistema operativo da loro utilizzato;
- supporto per sistemi di debugging;
- integrazione del sistema di versionamento Git$_G$;
- numerose estensioni facilmente installabili in grado di coprire la maggior parte delle necessita del gruppo.

https://code.visualstudio.com/

3 Processi di Supporto

3.1 Documentazione

3.1.1 Descrizione

Questa sezione fornisce le norme per la stesura, la verifica e l'approvazione dei documenti. Tali regole vanno seguite in tutti i documenti ufficiali prodotti durante il ciclo di vita del software, garantendo così la coerenza e la validità degli stessi.

3.1.2 Ciclo di vita dei documenti

Ogni documento attraversa diversi stadi:

- **Creazione e strutturazione del documento:** il documento viene creato nella sua directory di appartenenza (secondo le indicazioni presenti nella sezione "Documenti interni ed esterni"), e viene stesa la sua struttura generale. Viene utilizzato un template$_G$ LaTeX$_G$ (descritto nella sezione §3.1.5.1);
- **Stesura:** scrittura effettiva dei contenuti del documento da parte di un redattore$_G$;
- **Verifica:** attività eseguita dai Verificatori, i quali si occupano di controllare che il documento sia conforme alle *Norme di Progetto 2.0.0*, sia sintatticamente che semanticamente. Alla fine di ogni controllo, il resoconto della verifica viene consegnato al Responsabile di Progetto che provvede a notificare il redattore$_G$ in caso di errori, riportando il documento al passo di "Stesura". Quando la fase di verifica finale non rileva ulteriori errori, il Responsabile passa il documento al processo di "Approvazione";
- **Approvazione:** in questa fase i Verificatori hanno terminato i controlli finali con esito positivo, comunicandoli al Responsabile, il quale si occupa di approvare il documento e preparare il rilascio.

3.1.3 Documenti interni ed esterni

Ogni documento deve essere classificato come Interno o Esterno:

- **Interno:** il documento viene utilizzato all'interno del team;
- **Esterno:** il documento viene condiviso con il Committente$_G$ ed il Proponente$_G$.

3.1.4 Documenti presenti

Di seguito sono elencati i documenti ufficiali che verranno prodotti e la loro classificazione in uso Interno o Esterno.

3.1.4.1 Norme di Progetto

Documento ad uso Interno.
Lo scopo delle *Norme di Progetto* è descritto nella sezione §1.1 "Scopo del Documento", di questo stesso documento.

3.1.4.2 Studio di Fattibilità

Documento ad uso Interno.
Lo *Studio di Fattibilità* ha l'obiettivo di esporre (brevemente) ogni capitolato$_G$ e di elencare per ognuno gli aspetti positivi e le criticità che il team ha individuato.

3.1.4.3 Glossario

Documento ad uso Esterno.
Il *Glossario* ha lo scopo di disambiguare alcuni termini che compaiono all'interno dei documenti e vengono utilizzati nelle comunicazioni interne.

3.1.4.4 Analisi dei Requisiti

Documento ad uso Esterno.
Lo scopo dell'*Analisi dei Requisiti* è di esporre dettagliatamente i requisiti individuati per lo sviluppo del capitolato$_G$ scelto.

3.1.4.5 Piano di Progetto

Documento ad uso Esterno.
Lo scopo del *Piano di Progetto* è di organizzare le attività in modo da gestire le risorse disponibili in termini di tempo e "forza lavoro".

3.1.4.6 Piano di Qualifica

Documento ad uso Esterno.
Lo scopo del *Piano di Qualifica* è di presentare i metodi di verifica e validazione implementati dal gruppo, per garantire la qualità del prodotto$_G$ e dei processi adottati.

3.1.5 Struttura dei documenti

3.1.5.1 Template LaTeX

Per uniformare la struttura dei documenti il gruppo ha deciso di creare un template$_G$ LaTeX$_G$, da utilizzare per la stesura di tutti i documenti ufficiali. Il template$_G$ è contenuto nella cartella LaTeX$_G$, la cui struttura è la seguente:

- **common_commands.tex:** file che contiene la definizione di nuovi comandi LaTeX$_G$ per l'inserimento nel flusso di testo di termini ricorrenti:
  - **informazioni sul gruppo:** nome del gruppo e indirizzo email;
  - **informazioni sul progetto:** nome del progetto e nomi del Proponente$_G$ e del Committente$_G$;
  - **membri del gruppo:** nomi dei membri del gruppo *Roundabout*;
  - **nomi dei documenti:** nomi dei documenti ufficiali.
- **configs.tex:** contiene i comandi per l'inclusione dei necessari pacchetti LaTeX$_G$ e la definizione dell'aspetto grafico generale dei documenti;
- **copertina.tex:** contiene il codice LaTeX$_G$ per la copertina, ovvero la prima pagina di ogni documento (la cui struttura è descritta nella sezione §3.1.5.2);

- **Template:** directory che contiene la struttura "classica" di un documento (quest'ultima viene descritta di seguito).

La directory Template ha la seguente struttura:

- **config:** directory che contiene un solo file: "commands.tex", all'interno del quale sono inseriti dei comandi specifici per il documento considerato (es. nome del documento, stato di approvazione, ecc...). I comandi devono essere adeguatamente modificati per ogni rispettivo documento;
- **res:** directory che contiene due cartelle:
  - **img:** contiene le immagini utilizzate all'interno del documento;
  - **sections:** contiene le varie sezioni del documento e un file "changelog.tex", ovvero il codice per il registro delle modifiche.
- **document.tex:** il file contiene la struttura generica di un documento, includendo le sezioni necessarie contenute nella cartella res.

3.1.5.2 Copertina

La copertina è la prima pagina di ogni documento e contiene alcune informazioni generali:

- Logo del gruppo *Roundabout*;
- Nome del gruppo e nome del capitolato$_G$ scelto: *Roundabout* - *Etherless*;
- Nome del documento (nel caso dei Verbali$_G$ accompagnato dalla data della riunione).

Queste informazioni sono seguite da una struttura tabellare, che contiene dettagli pertinenti al singolo documento:

- **Versione:** versione attuale del documento (vedi sezione §3.2);
- **Approvazione:** nome e cognome del Responsabile di progetto;
- **Redazione:** nome e cognome dei redattori$_G$ del documento;
- **Verifica:** nome e cognome dei Verificatori;
- **Stato:** stato del ciclo di vita in cui si trova il documento (vedi sezione §3.1.4.1 paragrafo "Ciclo di vita dei documenti");
- **Uso:** indica se il documento è ad uso Interno o Esterno;
- **Destinato a:** elenco dei destinatari del documento (vedi sezione §3.1.4.1 paragrafo del singolo documento);

Tale struttura è infine seguita da alcune informazioni aggiuntive, allineate al centro del documento:

- **Descrizione:** breve descrizione del documento;
- **Email:** indirizzo email del gruppo *Roundabout*.

3.1.5.3 Registro delle modifiche

Inizia nella seconda pagina del documento e contiene un resoconto delle modifiche apportate al documento, strutturate in forma tabellare.
Questa sezione è presente anche nei Verbali$_G$ di riunione, nei quali si limita alla stesura, la verifica e l'approvazione del documento in questo ordine, per evitare modifiche retroattive ai Verbali.
Ogni riga rappresenta una modifica ed è suddivisa in cinque colonne:

- **Versione:** aggiornamento progressivo della versione;
- **Data:** data della modifica;
- **Nominativo:** nome e cognome del membro del gruppo che ha effettuato la modifica;
- **Ruolo:** ruolo ricoperto in quel momento dal membro del team che ha effettuato la modifica;
- **Descrizione:** breve descrizione della modifica effettuata.

3.1.5.4 Indice

L'indice si trova nella pagina successiva e ha lo scopo di aiutare nella navigazione del documento e riassumerne la struttura in maniera visuale.
Nell'indice sono elencati i numeri delle sezioni, seguiti dal titolo e dal numero di pagina. Ogni riga dell'indice è un link che porta alla sezione specificata del documento.
L'indice dei contenuti puó essere seguito da un indice delle immagini e un indice delle tabelle.

3.1.5.5 Contenuto

Tutte le pagine successive sono occupate dal contenuto e sono strutturate come segue:

- in alto a destra il nome del documento;
- in alto a sinistra il logo del gruppo *Roundabout*;
- il contenuto della pagina diviso da intestazione e piè di pagina con una riga orizzontale;
- in basso a destra il numero di pagina nel formato:

  Pagina [numero di pagina] di [numero totale di pagine].

3.1.5.6 Verbali

Seguono la stessa struttura generale degli altri documenti, ma hanno un'organizzazione specifica del contenuto, in particolare sono suddivisi in:

- **Informazioni generali:** che contengono le informazioni dell'incontro e l'ordine del giorno.

  Le informazioni dell'incontro sono:

  - **Luogo:** il luogo dove si è svolta la riunione (in alternativa il mezzo utilizzato es. Skype$_G$);
  - **Data:** il giorno in cui si è svolta la riunione;
  - **Ora di inizio:** l'orario di inizio della riunione;
  - **Ora di fine:** l'orario di fine della riunione;

  - **Partecipanti:** elenco dei partecipanti alla riunione;
  - **Segretario:** redattore$_G$ del *Verbale*$_G$.

  L'ordine del giorno contiene un elenco degli argomenti di discussione previsti per l'incontro.

- ***Verbale*$_G$:** che contiene la descrizione e un riassunto degli argomenti elencati nell'ordine del giorno, suddivisi in sezioni;
- **Riepilogo delle decisioni:** contiene il resoconto delle decisioni prese durante la riunione, in forma tabellare. Ogni decisione è identificata da un codice scritto in forma:

  V[T]_[numero dell'incontro].[numero progressivo]

  dove [T] indica il tipo di *Verbale*$_G$ che puó essere Interno (I) o Esterno (E).

3.1.6 Norme tipografiche

3.1.6.1 Nomi dei file

I nomi dei file seguono le seguenti regole:

- sono composti di diverse parole, con la prima lettera minuscola, fatta eccezione per i Verbali che, come descritto in seguito, iniziano per VI oppure VE;
- ogni parola è separata da un underscore;
- il nome del file corrisponde al nome del documento senza escludere le preposizioni, ad eccezione dei Verbali$_G$, come già menzionato.

I Verbali seguono quindi delle regole differenti, a causa della loro unicità rispetto ad altri documenti. Sono nominati seguendo la struttura:

V[T]_[data]

- **[T]:** indica il tipo di *Verbale*$_G$ che puó essere Interno (I) o Esterno (E);
- **[data]:** indica la data della riunione in formato "YYYY_MM_DD", ovvero anno (YYYY), mese (MM) e giorno (DD), separati da underscore.

Esempi di nomi corretti:

- per un documento qualunque: nome_di_file.tex;
- per un *Verbale*$_G$: VI_2020_03_16.tex.

Inoltre, si è deciso di adottare delle convenzioni anche per la parola "verbale", in particolare:

- nel caso in cui essa compaia al singolare, deve essere riportata in corsivo e con la prima lettera maiuscola;
- nel casi in cui essa compaia al plurale, deve essere riportata solamente con la prima lettera maiuscola.

3.1.6.2 Stile del testo

Le seguenti norme vanno seguite per l'utilizzo di particolari stili di testo:

- **Grassetto:** usato se necessario all'inizio delle voci di un elenco puntato, a titoli o a termini significativi;
- **Corsivo:** usato per evidenziare proposizioni particolari all'interno del testo, in particolare il nome dei documenti ufficiali, il nome del Proponente$_G$, il nome del progetto *Etherless* e il nome del gruppo *Roundabout*;
- **Maiuscoletto:** usato per la "G" a pedice che indica le parole presenti nel *Glossario*.

Quando all'interno del testo vengono riferiti dei particolari documenti (es.: *Analisi dei Requisiti*), vanno seguite le seguenti regole:

- indicare con lettera maiuscola le iniziali (es. *Analisi dei Requisiti*), senza la versione del documento, il tutto in corsivo;
- se si fa riferimento al documento vero e proprio o a qualcosa in esso contenuto, va aggiunta la versione del documento nel formato:

  v[X].[Y].[Z]

  (es. Riferendosi ad una sezione specifica "[...] come indicato nella §5.1 dell'*Analisi dei Requisiti* v1.1.0[...]").

3.1.6.3 Termini del Glossario

Le norme relative ai termini da inserire nel *Glossario* sono:

- ogni termine del *Glossario* deve essere contrassegnato, in ogni sua istanza, da una G maiuscola a pedice;
- le istanze dei termini del *Glossario* presenti nei titoli non necessitano della G maiuscola a pedice.

3.1.6.4 Elenchi puntati

Ogni voce di un elenco puntato deve aderire alle norme seguenti:

- deve iniziare con la lettera minuscola;
- deve essere seguita da un ";", fatta eccezione per l'ultimo elemento che deve essere seguito da un punto;
- puó iniziare con dei termini in grassetto e/o con prima lettera maiuscola nel caso in cui il resto della voce sia una descrizione di quei termini.

3.1.6.5 Date e orari

In conformità allo standard ISO 8601, le date devono essere scritte usando il formato:

YYYY-MM-DD

ovvero anno in quattro cifre (YYYY), mese in due cifre (MM) e giorno in due cifre (DD).
Gli orari devono essere scritti usando il formato:

HH.MM

ovvero l'ora in due cifre (HH) e i minuti in due cifre (MM).

3.1.6.6 Tabelle e immagini

Le tabelle sono sempre corredate da un titolo ed un'indicizzazione separata dal normale contenuto.
Le immagini sono accompagnate da una didascalia ed un'indicizzazione, anch'essa separata dal normale contenuto.

3.1.6.7 Sigle e abbreviazioni

Nella stesura dei documenti possono venire menzionate diverse sigle:

- sigle per i ruoli di progetto:
  - **Rp:** Responsabile di Progetto;
  - **As:** Amministratore;
  - **An:** Analista;
  - **Pt:** Progettista;
  - **Pr:** Programmatore;
  - **Vf:** Verificatore.
- sigle per i nomi dei documenti:
  - **SdF:** Studio di Fattibilità;
  - **PdQ:** Piano di Qualifica;
  - **PdP:** Piano di Progetto;
  - **NdP:** Norme di Progetto;
  - **AdR:** Analisi dei Requisiti;
  - **G:** Glossario.
- sigle per revisioni del progetto:
  - **RR:** Revisione dei Requisiti;
  - **RP:** Revisione di Progettazione;
  - **RQ:** Revisione di Qualifica;
  - **RA:** Revisione di Accettazione.

3.1.7 Metriche

3.1.7.1 Indice di Gulpease

- **Descrizione**: è un indice di leggibilità di un testo tarato sulla lingua italiana. Considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero di lettere;
- **unità di misura**: la metrica è espressa tramite un numero intero;
- **formula**: $IG = 89 + \frac{300 \times \#frasi - 10 \times \#lettere}{\#parole}$;
- **risultato**: il risultato è compreso tra 0 e 100, dove il valore 100 indica il grado più alto di leggibilità e 0 il più basso. In particolare:
  - se il risultato è inferiore a 80, il testo è considerato di difficile lettura per chi ha la licenza elementare;
  - se il risultato è inferiore a 60, il documento è considerato di difficile lettura per chi ha la licenza media;
  - se il risultato è minore di 40, il documento è considerato di difficile lettura per chi ha un diploma superiore.

3.1.7.2 Formula di Flesch

- **Descrizione**: formula utilizzata per misurare la leggibilità di un testo inglese;
- **unità di misura**: la metrica è espressa tramite un numero decimale;
- **formula**: $F = 206,835 - (84,6 \times S) - (1,015 \times P)$, dove:
  - S indica il numero medio di sillabe per parola;
  - P indica il numero medio di parole per frase.
- **risultato**: il risultato è compreso tra 0.0 e 100.0, in particolare:
  - se il risultato è compreso tra 0.0 e 30.0 il documento è considerato molto difficile da leggere, adatto a lettori laureati;
  - se il risultato è compreso tra 30.0 e 50.0 il documento è leggibile per studenti universitari;
  - se il risultato è compreso tra 50.0 e 70.0 il documento è leggibile per chi possiede un diploma superiore;
  - se il risultato è compreso tra 70.0 e 90.0 il documento è leggibile per chi possiede una licenza media;
  - se il risultato è compreso tra 90.0 e 100.0 il documento viene considerato di facile lettura per un qualsiasi lettore.

3.1.7.3 Correttezza ortografica

- **Descrizione**: permette di misurare la correttezza a livello lessicografico del documento;
- **unità di misura**: la metrica è espressa tramite un numero intero;
- **formula**: il valore coincide con il numero di errori ortografici presenti, cioè: $CO = \#numero\_errori\_ortografici$;
- **risultato**:
  - se il risultato è pari a 0, il documento è considerato corretto e non presenta errori ortografici;
  - se il risultato è maggiore di 0, allora il documento presenta almeno un errore ortografico.

3.1.8 Strumenti

3.1.8.1 LaTeX

Per la stesura della documentazione si è scelto il linguaggio di markup LaTeX$_G$. Nonostante non sia di immediata comprensione, quest'ultimo permette di scrivere documenti in modo collaborativo, modulare e scalabile.

3.1.8.2 File condivisi di Microsoft Teams

Si utilizza la condivisione di file interna a Microsoft Teams$_G$ per file utili non ufficiali. Questa piattaforma offre infatti funzionalità per il lavoro simultaneo sugli stessi file, che risulta utile per l'organizzazione interna del lavoro di gruppo.

3.1.8.3 Editor di testo

Si è demandata la scelta dell'editor di testo ai singoli membri del gruppo per permettere un lavoro più efficiente, data la familiarità di ognuno con diversi editor e ambienti di lavoro.

3.1.8.4 Microsoft Excel

Software per la creazione e la gestione di fogli elettronici, distribuito da Microsoft come parte della suite Microsoft Office. Utilizzato per funzioni di calcolo, produzione di tabelle, grafica e diagrammi.

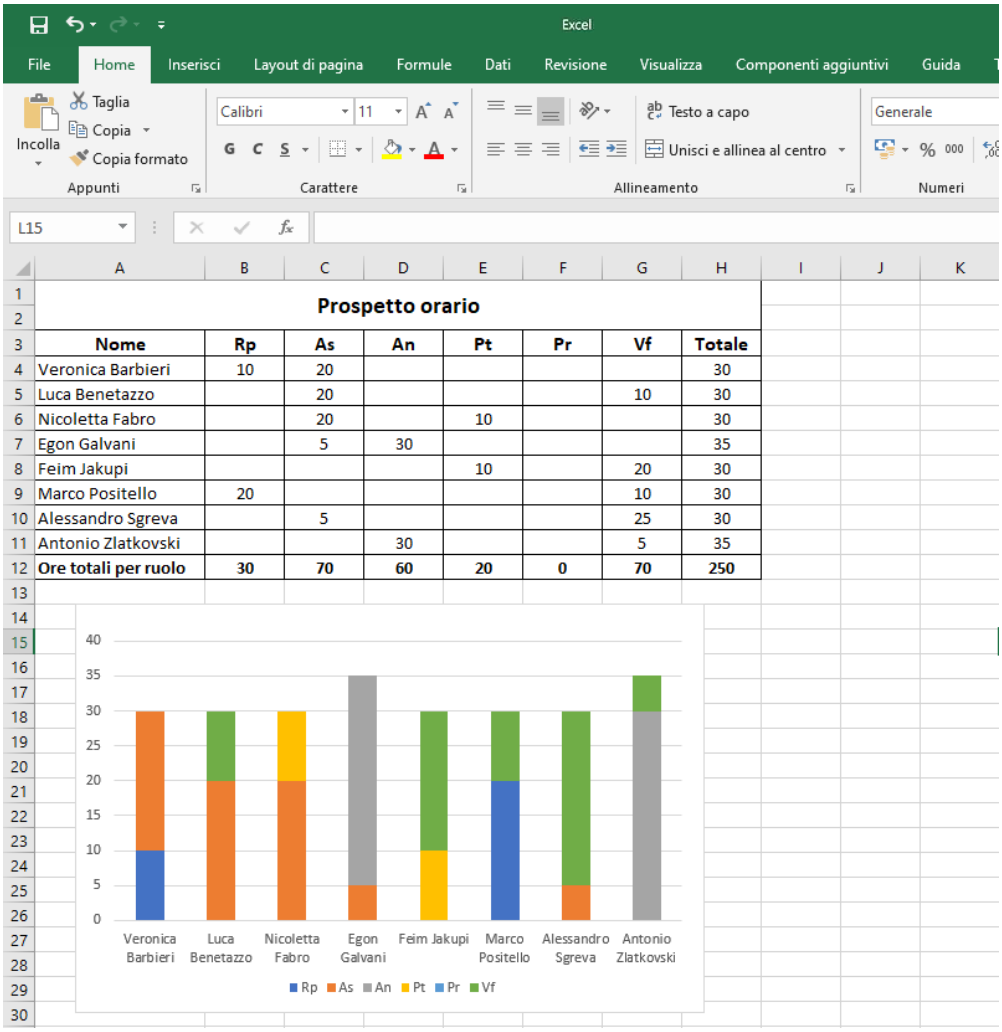

Figura 3.1.1: Microsoft Excel

3.1.8.5 GanttProject

Software per la gestione dei progetti basato su Java$_G$. Utilizzato principalmente per la creazione di diagrammi di Gantt, ovvero strumenti in grado di rappresentare le tempistiche e le attività di un progetto, per tracciare quindi il suo avanzamento.

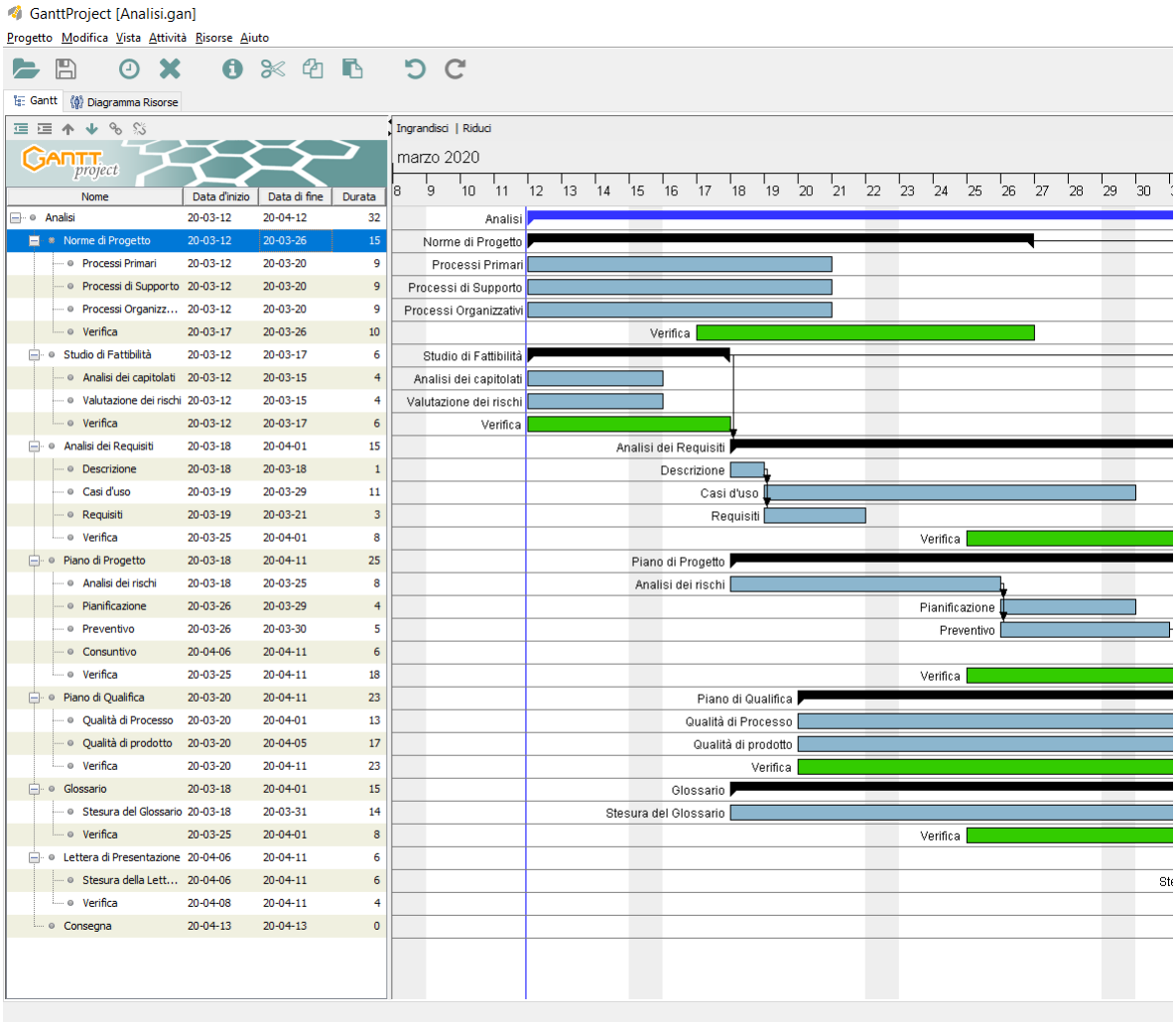

Figura 3.1.2: GanttProject

3.2 Gestione della configurazione

3.2.1 Descrizione

Questo processo, definisce le norme utili alla predisposizione di un workspace ordinato e accessibile, andando a controllare e automatizzare lo stato dei documenti e delle componenti software. Si compone di una serie di attività, di seguito descritte.

3.2.2 Versionamento

3.2.2.1 Codice di versione

Tutti i documenti prodotti vengono conservati in una repository$_G$ e devono essere versionati tramite un sistema identificativo. I numeri di versione rispettano la struttura seguente:

X.Y.Z

Dove:

- **X:** rappresenta una versione completa del documento pronta al rilascio esterno:
  - parte da 0 (puó soltanto aumentare);
  - viene incrementato soltanto dopo l'approvazione del documento da parte del Responsabile di progetto.
- **Y:** rappresenta una versione del documento che è stata oggetto di verifica, da parte di un Verificatore:
  - parte da 0;
  - viene incrementato ad ogni verifica;
  - si resetta ad ogni incremento di X.
- **Z:** rappresenta una versione del documento in fase di stesura:
  - parte da 0;
  - incrementato dal redattore$_G$ ad ogni modifica;
  - si resetta ad ogni incremento di Y.

3.2.2.2 Strumenti

Vengono utilizzati Git$_G$ come VCS$_G$ distribuito e GitHub$_G$ per ospitare la repository$_G$ di progetto.

3.2.2.3 Repository

La versione ufficiale della documentazione viene mantenuta in una repository$_G$ remota nel sito GitHub$_G$, appartenete alla *GitHub$_G$ organization* del gruppo *Roundabout*, al link:

https://github.com/Rounadbout/Documentazione

Ogni membro del gruppo lavora su una copia locale della repository$_G$ sul proprio computer, interagendo con il VCS$_G$ e la repository$_G$ remota sia attraverso linea di comando, sia tramite software come GitHub$_G$ Desktop e GitKraken.

3.2.2.4 Utilizzo di Git

Per sfruttare in modo efficace le funzionalità offerte da Git$_G$ e promuovere il lavoro collaborativo, la repository$_G$ è strutturata in vari branch$_G$, ognuno relativo ad una particolare feature o componente del progetto (es. Un branch$_G$ per la stesura delle *Norme di Progetto* chiamato feature/norme_di_progetto). Dunque ogni membro del gruppo che necessita di operare su una certa componente del progetto deve:

- scegliere il branch$_G$ adatto su cui lavorare;
- spostarsi su tale branch$_G$;
- effettuare un pull$_G$ dal repository$_G$ remoto per trasferire una copia aggiornata nella propria repository$_G$ locale;

- svolgere il lavoro;
- eseguire un commit$_G$ del lavoro svolto, allegando una descrizione;
- eseguire il push$_G$ delle modifiche sulla repository$_G$ remota.

3.2.2.5 Gestione delle modifiche

Tutte le modifiche effettuate nei documenti vengono memorizzate all'interno del "Registro delle modifiche" situato in ogni documento nella pagina successiva alla copertina, come descritto nella sezione §3.1.5.3.
Ad ogni modifica corrisponde un commit$_G$ Git$_G$, al quale è opportuno allegare un commento in modo da facilitare il tracciamento delle modifiche e del lavoro svolto, tramite Issue$_G$. Ogni modifica apportata ai documenti deve essere discussa con i collaboratori che si stanno occupando del medesimo documento; nel caso in cui la modifica sia di rilevante importanza, è opportuno discuterne con tutto il gruppo durante gli appositi incontri settimanali.

3.3 Gestione dei cambiamenti

3.3.1 Descrizione

La gestione dei cambiamenti risulta fondamentale a seguito della rilevazione di uno o più errori. Tramite questo processo è possibile definire ed attuare attività volte a correggere le problematiche riscontrate.

3.3.2 Attività

3.3.2.1 Gestione delle Issue

Al fine di dover risolvere uno o più errori nel corso del progetto viene attivato un piano di mitigazione degli stessi: l'uso di un sistema di Issue$_G$ Tracking System (ITS) permette di trasformare in Issue quanto rilevato erroneo. Grazie a tale strumento è possibile risolvere tali problematiche assegnandole ai rispettivi responsabili per essere svolte entro scadenze prestabilite.

3.3.3 Metriche

3.3.3.1 Numero di Issue aperte

Si è convenuto che il numero di Issue$_G$ aperte debba essere pari a 0 alla fine di ogni periodo pianificato. Nel caso in cui siano presenti pendenze, sarà necessario attuare quanto possibile per risolverle, sempre rimanendo entro la scadenza prefissata dalle stesse.

3.3.4 Strumenti

3.3.4.1 GitHub

Per attuare questo processo il team ha scelto di utilizzare la componente Issue$_G$ Tracking System (ITS) di GitHub$_G$. Principalmente è stato scelto di utilizzare questa piattaforma in quanto già utilizzata per il sistema di versionamento$_G$.

3.4 Gestione della qualità

3.4.1 Descrizione

Un'adeguata implementazione del processo di gestione della qualità è fondamentale per il corretto svolgimento del progetto. Questo processo intende garantire software e documenti di buona qualità, sviluppati attraverso processi chiari e ordinati. Il processo di gestione di qualità viene descritto nel dettaglio all'interno del *Piano di Qualifica 2.0.0*. In particolare per ogni processo ed ogni prodotto$_G$ vengono descritti gli obiettivi e le metriche per la valutazione del raggiungimento degli stessi.

3.4.2 Attività

3.4.2.1 Pianificazione

Il presupposto per l'implementazione del processo di gestione della qualità è la presenza di un *Piano di Qualifica*, a cui sia possibile fare riferimento per coordinare tutte le attività periodiche ed operazioni di controllo qualità. Tale piano resta valido per tutta la durata del progetto e viene costantemente aggiornato con le nuove metriche e procedure integrate.

3.4.2.2 Garanzia di qualità del prodotto

La qualità del prodotto$_G$ che si va a sviluppare viene garantita coordinatamente dai processi di Verifica e Validazione (descritti rispettivamente nelle sezioni §3.4 e §3.5), i quali puntano ad un miglioramento continuo e a verificare il rispetto delle metriche di qualità stabilite. Importante è anche il confronto costante con il Proponente$_G$, per garantire che il software prodotto sia in grado di soddisfare i requisiti concordati.

3.4.2.3 Garanzia di qualità dei processi

La qualità dei processi che compongono il ciclo di vita del software deriva dallo svolgimento corretto e normato delle attività che compongono tali processi. Ottenendo quindi risultati in grado di soddisfare i requisiti previsti da contratto, rispettando norme e standard scelti come riferimento. È compito degli Amministratori di progetto monitorare lo svolgimento delle attività, ed intervenire per garantire il rispetto dei piani e delle procedure di gestione della qualità. L'obiettivo è il miglioramento continuo dei processi, perseguendo i principi di efficacia ed efficienza del prodotto$_G$ durante tutto il suo ciclo di vita.

3.4.3 Metriche

3.4.3.1 Metriche di processo e di prodotto

Percentuale di metriche soddisfatte (PMS)

- **Descrizione**: è un valore percentuale che punta a rappresentare la qualità del processo/prodotto$_G$ sotto analisi. È basato sul numero di metriche che hanno raggiunto un valore considerato accettabile, rapportato con il numero totale di metriche applicate nella valutazione del processo/prodotto$_G$;
- **unità di misura**: la metrica è espressa tramite un numero percentuale;
- **formula**: PMS $= \frac{\#metriche\_soddisfatte}{\#totale\_di\_metriche} \times 100$;

- **risultato**:
  - se il risultato è minore di 60, la qualità del processo/prodotto$_G$ è considerata non accettabile. Vanno considerate procedure di risoluzione quali: ricalcolo del PMS, revisione del processo/prodotto$_G$ sotto analisi, revisione delle metriche applicate;
  - se il risultato è minore di 90, la qualità del processo/prodotto$_G$ è considerata accettabile, ma ancora migliorabile;
  - se il risultato è maggiore di 90, la qualità del processo/prodotto$_G$ è considerata ideale.

3.4.4 Strumenti

Gli strumenti di riferimento per la qualità sono:

- parte dei processi forniti dallo standard ISO 12207;
- standard ISO/IEC 15504;
- ciclo di Deming;
- standard ISO/IEC 9126;
- metriche di qualità stabilite.

Un'accurata descrizione di tali stumenti e modelli è presente nell'appendice A di questo stesso documento.
Gli Amministratori del progetto sfruttano inoltre l'apparato Issue$_G$ Tracking System di **GitHub**$_G$, per monitorare costantemente lo svolgimento delle attività del progetto.

3.5 Verifica

3.5.1 Descrizione

Il processo di Verifica determina se il prodotto$_G$ di un'attività sia conforme o meno alle specifiche e alle aspettative già confermate, individuando quindi possibili difetti nel software e nella documentazione, e garantendo prodotti$_G$ corretti e completi.
Il gruppo si impegna inoltre a garantire una verifica costante dei requisiti mantenendosi in stretto contatto con il Proponente$_G$.

3.5.2 Attività

3.5.2.1 Analisi

Analisi statica

L'analisi statica riguarda sia la documentazione che il codice e valuta la conformità alle norme stabilite e la sua correttezza formale, senza l'esecuzione del prodotto$_G$ software. Questo tipo di analisi si serve di metodi formali (implementati tramite macchine) e metodi manuali (implementati solo per prodotti$_G$ semplici).
I metodi manuali sono:

- **walkthrough:** analisi dei documenti (o diversi file) nella loro interezza, per ricercare eventuali difetti identificati da un Verificatore, che andranno eventualmente corretti dallo Sviluppatore (o redattore$_G$ del documento). Nel caso di semplici modifiche o correzioni potrà intervenire direttamente il Verificatore;

- **inspection:** analisi mirata dell'oggetto di verifica da parte di un Verificatore, che utilizza delle liste di controllo per cercare difetti specifici in sezioni specifiche. Alcuni errori comuni sono elencati di seguito:
  - formato di date (YYYY-MM-DD);
  - formato di elenchi puntati (punteggiatura alla fine di ogni elemento ";" o ".");
  - stile del testo per termini particolari (es. termini in corsivo);
  - tempo verbale (il presente è sempre preferibile).

Analisi dinamica

L'analisi dinamica richiede l'esecuzione del prodotto$_G$ software e viene effettuata tramite una suite di test, che garantisce nel complesso la verifica del prodotto$_G$ stesso.

3.5.2.2 Test

Sono parte costituente dell'analisi dinamica e hanno lo scopo di verificare il corretto funzionamento del codice. I test, per poter essere considerati ben scritti, devono essere:

- **Ripetibili**, quindi devono specificare:
  - l'ambiente di esecuzione$_G$;
  - l'input e l'output atteso;
  - metodi di interpretazione dei risultati.
- **Automatizzati:** devono basarsi su strumenti che ne permettano l'esecuzione automatica, ne registrino i risultati e provvedano a notificare i soggetti ad essi interessati.

Esistono diversi tipi di test:

- **Test di unità**: verificano l'unità, cioè il più piccolo sottosistema possibile che può essere testato separatamente. L'approccio più comune al test di unità prevede la scrittura di driver$_G$ e stub$_G$, dove il primo simula un'unità chiamante mentre il secondo un'unità chiamata. Tali test devono essere indicati nel seguente modo:

  TU[id]

  dove *id* rappresenta il codice identificativo crescente dell'unità da verificare. È responsabilità del singolo Programmatore scrivere i test delle unità più semplici da lui sviluppate. Nel caso in cui l'unità sia più complessa la scrittura del test di unità deve essere compiuta da un Verificatore.
- **Test di integrazione**: test eseguiti per verificare che siano rispettati i contratti di interfaccia tra più moduli o sub-system. La forma più semplice di questo tipo di test prevede che le componenti vengano verificate in maniera incrementale, con l'aggiunta di altre unità o gruppi di unità corretti, fino ad arrivare al sistema completo. Tali test devono essere indicati nel seguente modo:

  TI[id]

  dove *id* indica il codice identificativo crescente del componente da verificare.

- **Test di sistema:** il sistema viene testato nella sua interezza e i risultati confrontati con quanto richiesto dall'*Analisi dei Requisiti 2.0.0*. In questo modo è possibile accertarsi delle caratteristiche del sistema e della copertura dei requisiti funzionali. Tali test devono essere indicati nel seguente modo:

  TS[id]

  dove *id* rappresenta il codice identificativo crescente del componente da verificare.

- **Test di regressione**: si effettua in seguito ad una modifica del sistema, in modo da controllare che essa non abbia compromesso le funzionalità già testate in precedenza. Un test di regressione consiste nella reiterazione dei test già esistenti sia per l'unità modifica sia per quelle che si relazionano con essa.

3.5.3 Metriche

3.5.3.1 Metriche sui Test

Code coverage (CC)

- **Descrizione**: la metrica CC indica la percentuale di righe di codice percorse durante il test rispetto alle linee di codice totali. A granularità più fine, la code coverage misura quattro aspetti:
  - **statement coverage**: percentuale di statement percorsi dai test;
  - **branch coverage**: percentuale di diramazioni percorse dai test;
  - **function converage**: percentuale di funzioni percorse dai test;
  - **line coverage**: percentuale di righe di codice percorse dai test;
- **unità di misura**: la metrica viene espressa tramite una percentuale;
- **formula:** $CC = \frac{\#linee\_di\_codice\_eseguite\_dal\_test}{\#linee\_di\_codice\_totali} \times 100$;
- **risultato**: il risultato è compreso tra 0.0% e 100.0%, in particolare:
  - se il risultato è pari a 0%, allora non sono presenti test all'interno dell'applicativo;
  - se il risultato è pari a 100%, allora i test coprono in maniera completa il codice che costituisce l'applicativo.

3.5.4 Strumenti

3.5.4.1 Verifica ortografica

Per la verifica ortografica durante la redazione dei documenti viene utilizzato lo strumento integrato negli editor di testo scelti dai singoli componenti del gruppo. In questo modo, durante la fase di scrittura, è possibile individuare e prontamente correggere gli errori ortografici.

3.6 Validazione

3.6.1 Descrizione

Il processo di validazione stabilisce se il prodotto$_G$ soddisfa i requisiti richiesti, eseguendo un test completo sul sistema. Di conseguenza va eseguito in seguito al processo di verifica, il quale predispone il software per l'esecuzione di tale test. La definizione dei test da eseguire è di competenza dei Progettisti, mentre la loro esecuzione è compito dei Verificatori, che sono tenuti ad interpretarne e documentarne i risultati.

3.6.2 Attività

3.6.2.1 Pianificazione

Prerequisito per l'adeguata gestione del processo di validazione è un'adeguata pianificazione. Tale pianificazione dovrebbe focalizzarsi principalmente su:

- oggetti sottoposti a validazione;
- operazioni di validazione da eseguire;
- risorse, responsibilità e gestione delle scadenze legate alla pianificazione;
- procedure standardizzate per l'inoltro della documentazione risultante a Proponente$_G$ e Committente$_G$.

3.6.2.2 Test di sistema [TS]

Già precedentemente descritti nella sezione §3.5.2.2.

3.6.2.3 Test di accettazione [TA]

Nel processo di validazione viene eseguito il test di accettazione (detto anche collaudo), ovvero un test molto simile a quello di sistema, ma eseguito in collaborazione con il Committente$_G$. Questo test, nello specifico, è mirato a accertare e confermare il soddisfacimento dei requisiti specificati nel capitolato$_G$ e analizzati nel documento *Analisi dei Requisiti 2.0.0*. Tali test vengono rappresentati con la seguente struttura:

TA[Importanza][Tipologia][Codice].

Dove:

- **Importanza**: indica il grado di importanza del requisito sotto test ai fini del progetto. Può assumere i valori:
  - **1**: per un requisito obbligatorio;
  - **2**: per un requisito desiderabile;
  - **3**: per un requisito opzionale.
- **Tipologia**: rappresenta la classe a cui appartiene il requisito sotto test. Può assumere i valori:
  - **F**: funzionale;

  - **P**: prestazionale;
  - **Q**: qualitativo;
  - **V**: vincolo.

- **Codice**: identificatore univoco del requisito sotto test.

3.6.3 Metriche

Un'accurata descrizione delle metriche di qualità implementate dal gruppo è presente all'interno del documento *Piano di Qualifica 2.0.0*.

3.6.4 Strumenti

Non sono attualmente stati individuati strumenti per l'attuazione dei test di sistema ed accettazione.

4 Processi Organizzativi

Questa sezione descrive tutti i processi propedeutici o ausiliari, utili affinché i processi primari funzionino in modo adeguato. Di seguito vengono illustrate tutte le norme da rispettare per un'efficace organizzazione tra le parti operanti.

4.1 Gestione Organizzativa

4.1.1 Descrizione

La gestione organizzativa è il processo che descrive le scelte sottostanti la suddivisione e il coordinamento del lavoro all'interno del progetto. Lo scopo principale di questo processo è fornire ai membri del gruppo un *Piano di Progetto* per l'organizzazione del lavoro, con focus sull'efficacia e l'efficienza. Nello specifico, le attività che costituiscono questo processo sono:

- inizializzazione e definizione degli obiettivi;
- pianificazione e gestione di scadenze, rischi e risorse;
- esecuzione e controllo dei processi;
- revisione e valutazione;
- Definition of Done$_G$.

Le operazioni svolte durante queste attività vengono in seguito descritte indirettamente, attraverso l'esplicitazione delle mansioni dei vari ruoli di progetto.

4.1.2 Ruoli di progetto

Ciascun componente del gruppo ricopre un ruolo di progetto a rotazione, facendo sì che ogni membro possa assumere almeno una volta ciascuno di essi nel corso del progetto. Nel documento *Piano di Progetto*2.0.0 vengono organizzate e pianificate le attività assegnate ai specifici ruoli previsti nell'attività di progetto. Ciascun ruolo viene descritto di seguito.

4.1.2.1 Responsabile di Progetto

Il Responsabile di Progetto è una figura essenziale, che partecipa al progetto dall'inizio fino alla fine. È responsabile delle decisioni e scelte che vengono intraprese, coordinando l'intero progetto. Rappresenta il gruppo di fronte a soggetti esterni, come Committente$_G$ e Proponente$_G$. Riassunto delle mansioni:

- coordina, pianifica e controlla le attività;
- gestisce le risorse umane;
- approva la documentazione;
- approva l'offerta economica;
- preventiva l'analisi dei rischi e la loro eventuale gestione;
- determina e verifica le condizioni di completezza.

4.1.2.2 Amministratore

L'Amministratore è la figura che ha come compito principale il controllo e l'amministrazione dell'ecosistema lavorativo. Inoltre ha diretta responsabilità sull'efficienza e sulla capacità operativa dell'ambiente di lavoro. Riassunto delle mansioni:

- studia e ricerca strumenti che riducano il più possibile l'impiego di risorse umane e che automatizzino tutto ciò che è possibile fare attraverso l'utilizzo di software;
- ricerca soluzioni ai problemi legati alla difficoltà di gestione dei processi e risorse, attraverso la realizzazione o la ricerca di strumenti adatti a tale scopo;
- controlla le versioni e le configurazioni del prodotto$_G$;
- gestisce il versionamento della documentazione del progetto e della sua archiviazione;
- fornisce procedure e strumenti di monitoraggio/segnalazione, in modo da garantire un corretto controllo di qualità.

4.1.2.3 Analista

L'Analista è il responsabile delle attività di analisi. Deve effettuare studi e ricerche in maniera molto approfondita per conoscere bene il dominio$_G$ del problema. Non è necessario che partecipi al progetto fino al termine, ma il suo operato è fondamentale fin dalla fase iniziale, in quanto le sue scelte decisionali hanno un grande impatto sul successo dell'intero progetto. Riassunto delle mansioni:

- studia e definisce il problema da risolvere, rilevandone la complessità;
- compie l'analisi del dominio$_G$ delle richieste tramite lo studio dei bisogni, espliciti ed impliciti;
- compie l'analisi del dominio$_G$ applicativo determinando gli utilizzatori e l'ambiente di utilizzo;
- redige i documenti *Analisi dei Requisiti* e *Studio di Fattibilità*.

4.1.2.4 Progettista

Il Progettista è la figura che si occupa delle scelte architetturali del progetto e ne influenza gli aspetti tecnici e tecnologici. Utilizzando le attività svolte dall'Analista, il Progettista ha il compito di trovare una soluzione attuabile, comprensibile e motivata. Riassunto delle mansioni:

- produce una soluzione attuabile, comprensibile e motivata;
- effettua scelte su aspetti progettuali, applicando al prodotto$_G$ soluzioni note ed ottimizzate;
- effettua scelte che portino ad avere un prodotto$_G$ facilmente manutenibile.

4.1.2.5 Programmatore

Il Programmatore è la figura responsabile della codifica del codice e della creazione delle componenti di supporto, indispensabili per poter effettuare le prove di verifica e di validazione. Riassunto delle mansioni:

- implementa in maniera precisa e scrupolosa le soluzioni generate dal Progettista;
- scrive codice sorgente altamente e facilmente manutenibile;
- si occupa del versionamento e della documentazione del codice prodotto$_G$;
- realizza strumenti per la verifica e la validazione del software.

4.1.2.6 Verificatore

Il Verificatore è la figura che ha il compito di effettuare una attenta verifica del prodotto$_G$, dando particolare attenzione al rispetto delle normative di progetto. Oltre ad una grande conoscenza di tali norme, il verificatore deve avere buone capacità di giudizio. Riassunto delle mansioni:

- controlla che le attività svolte siano conformi alle normative stabilite;
- vigila sull'integrità del prodotto$_G$ ad ogni stadio del suo ciclo di vita.
- comunica eventuali errori identificati al responsabile dell'oggetto preso in esame.

4.1.3 Procedure

4.1.3.1 Gestione delle comunicazioni

Comunicazioni interne

Le comunicazioni interne sono gestite principalmente attraverso tre canali:

- **gruppo Telegram**$_G$: dove è possibile scambiare rapidamente comunicazioni brevi ed informali, coinvolgendo l'intero team.
- **Microsoft Teams**$_G$: piattaforma dotata di funzioni di messaggistica istantanea, con possibilità di creare canali specifici divisi per argomento, e videochiamata. Viene utilizzata per lo scambio di comunicazioni più consistenti e tecniche, come aggiornamenti sulle attività di Verifica o discussioni legate a specifiche attività.
- **GitHub**$_G$: anche se meno focalizzato sulla comunicazione, il sistema di gestione Issue$_G$ di GitHub$_G$ consente di aggiornare rapidamente l'intero gruppo sull'avanzamento dei lavori.

Comunicazioni esterne

Le comunicazioni esterne, gestite principalmente dal Responsabile di progetto, avvengono attraverso due canali:

- **Gmail**$_G$: per comunicare con il Committente$_G$ e, almeno inizialmente, con il Proponente$_G$ viene utilizzata la casella di posta elettronica creata appositamente per il team *Roundabout*.
- **canale Slack**$_G$: è stato concordato con il Proponente$_G$ *RedBabel* l'utilizzo della piattaforma Slack$_G$, con un canale dedicato al gruppo *Roundabout*. Viene utilizzata per rapidi scambi di informazioni o la pianificazione di eventuali incontri.

4.1.3.2 Gestione delle riunioni

Le riunioni sono indette dal Responsabile di Progetto, il quale ha il compito di:

- definire la data e l'orario delle riunioni, sia interne che esterne, considerando la disponibilità dei partecipanti;
- stabilire l'oggetto della riunione;
- valutare le richieste relative alle riunioni da parte dei componenti del Team *Roundabout* e dai soggetti esterni;
- verificare ed approvare il verbale$_G$ redatto dal Segretario della riunione, il quale verrà nominato ad inizio incontro;

I partecipanti devono presentarsi puntuali alle riunioni e, in caso di imprevisti, comunicarli con congruo preavviso. Le decisioni da intraprendere durante le riunioni, sono ritenute approvate nel caso di maggioranza da parte dei partecipanti.
Al termine di ciascuna riunione viene redatto un *Verbale*$_G$ da parte del Segretario, che deve contenere tutte le informazioni sulla riunione.

Riunioni interne

Le riunioni interne sono aperte esclusivamente agli 8 membri del gruppo *Roundabout*. Le occasioni di incontro sono ritenute valide esclusivamente in due modalità:

- **incontri in video-conferenza:** Effettuati tramite l'applicativo Microsoft Teams$_G$;
- **incontri di persona:** Effettuati trovandosi fisicamente in uno stesso luogo.

Considerata la situazione extra-progettuale relativa all'emergenza COVID-19$_G$, le riunioni iniziali saranno effettuate esclusivamente a distanza tramite video-conferenza. Affinché le riunioni siano ritenute valide, all'incontro dovranno essere presenti almeno 6 componenti del team.

Riunioni esterne

Le riunioni esterne comprendono tutti gli incontri che coinvolgono, oltre ai membri del team *Roundabout*, anche altri soggetti esterni.
Questi incontri sono tenuti telematicamente, fino al termine dell'emergenza COVID-19$_G$ e, successivamente, potranno tenersi attraverso riunioni fisiche. In caso di video-conferenza è da privilegiare lo strumento di comunicazione proposto dai soggetti esterni, similmente per le riunioni fisiche, in luoghi proposti dai soggetti esterni.
Nel caso si scelga di usufruire dei locali di Torre Archimede è necessario chiedere il permesso al Prof. Tullio Vardanega all'indirizzo mail tullio.vardanega@math.unipd.it.

4.1.3.3 Gestione degli strumenti di coordinamento e versionamento

Issue Tracking System

Come già menzionato, il gruppo ha deciso di utilizzare come strumento per l'organizzazione del carico di lavoro, la componente Issue$_G$ Tracking System (ITS) della piattaforma GitHub$_G$. Grazie a tale strumento, è possibile rappresentare ogni singolo compito tramite Issue$_G$, che verranno assegnate ai rispettivi responsabili per essere svolte entro scadenze prestabilite. È possibile inoltre avere una visione d'insieme delle Issue$_G$ (aperte o chiuse), così che il Responsabile di Progetto o

eventuali Amministratori possano facilmente gestire e monitorare l'andamento del progetto. La procedura seguita per l'assegnazione di una Issue$_G$ è la seguente:

- vengono decisi uno o più responsabili per la nuova Issue$_G$;
- viene inserito un titolo alla Issue$_G$, che renda chiaro il suo obiettivo;
- viene impostata una scadenza per la Issue$_G$, se precedentemente concordata;
- la Issue$_G$ viene concretamente assegnata ai responsabili scelti;
- viene associata un'etichetta alla Issue$_G$, utile ad identificare il contesto in cui questa opera o a classificarne la tipologia;
- qualora il titolo non sia sufficientemente esplicativo, o sia necessario aggiungere ulteriori informazioni, viene aggiunto un breve riassunto dell'obiettivo della Issue$_G$;

Il processo viene completato con una conferma, che provoca un'immediata notifica a tutti i componenti del gruppo.

Repository

Per la gestione del versionamento e l'archiviazione dei file di progetto, viene sfruttata la componente VCS$_G$ della piattaforma GitHub$_G$. Gli Amministratori si sono occupati della creazione del repository$_G$ **Documentazione**, dove vengono mantenuti i file di documentazione del progetto, ed è loro compito garantire l'ordine e la pulizia di quest'ultimo. È prevista l'integrazione di tale repository$_G$ per l'archiviazione del prodotto$_G$ software in sviluppo.
La struttura del repository$_G$ Documentazione è la seguente:

- **Cartella Esterni**: contiene i file e sotto-cartelle relativi ai documenti esterni del progetto;
- **Cartella Interni**: contiene i file e sotto-cartelle relativi ai documenti interni del progetto;
- **Cartella Latex$_G$**: contiene i file template$_G$ e configurazioni per il layout dei documenti LaTeX$_G$;
- **File .gitignore**: specifica i tipi di file che devono essere ignorati e non presenti all'interno del repository$_G$. In particolare si desidera conservare solo file di tipo .tex, .pdf, .jpg e .png.

Al fine di tracciare in maniera più efficacie le modifiche effettuate nel repository$_G$, è richiesto di specificare alla fine di ogni commit$_G$ le Issue$_G$ che quest'ultimo va a chiudere tramite il commento: `"close #"` seguito dall'ID della Issue$_G$.

4.1.3.4 Gestione dei rischi

Tutte le problematiche che potrebbero ostacolare il corretto proseguimento del progetto devono essere opportunamente analizzate e gestite. È compito del Responsabile di Progetto svolgere tale mansione e documentare i risultati all'interno del *Piano di Progetto*2.0.0. La procedura da seguire per la gestione dei rischi comprende:

- individuazione dei potenziali fattori di rischio;
- analisi dei fattori di rischio, con documentazione nel *Piano di Progetto*2.0.0;
- pianificazione di controllo rischi e mitigazione effetti;

- monitoraggio costante, per individuare nuovi rischi e gestire i conosciuti.

Per la classificazione dei fattori di rischio, si è deciso di dividerli in:

- **RT**: Rischi Tecnologici;
- **RO**: Rischi Organizzativi;
- **RI**: Rischi Interpersonali;
- **RR**: Rischi legati ai Requisiti.

4.1.4 Metriche

4.1.4.1 Metriche per la gestione dei rischi

Budget at Completion (BAC)

- **Descrizione**: valore che indica il budget inizialmente allocato per la realizzazione del progetto;
- **unità di misura**: la metrica è espressa tramite un numero intero;
- **risultato**: l'obiettivo è ottenere un valore pari al preventivato, accettando un errore massimo del $\pm 5\%$. Un errore maggiore di tale valore indica un'inadeguatezza del preventivo stilato o della gestione delle risorse, si rende quindi necessaria una revisione di quest'ultimi.

Estimated at Completion (EAC)

- **Descrizione**: rappresenta il budget stimato per la realizzazione del progetto, aggiornato allo stato attuale, con quindi conoscenza dei costi già sostenuti.
- **unità di misura**: la metrica è espressa tramite un numero intero;
- **formula**: $EAC = AC + ETC$;
- **risultato**: l'obiettivo è ottenere un valore pari al preventivato, accettando un errore massimo di $\pm 5\%$. Un errore maggiore di tale valore indica un'inadeguatezza del preventivo stilato o della gestione delle risorse, si rende quindi necessaria una revisione di quest'ultimi.

Estimated to Complete (ETC)

- **Descrizione**: rappresenta il budget stimato per la realizzazione delle rimanenti attività necessarie al completamento del progetto;
- **unità di misura**: la metrica è espressa tramite un numero intero;
- **risultato**: l'obiettivo è ottenere un valore pari o inferiore al preventivato. Nel caso sia stato speso un budget maggiore, è necessario fornire una o più valide motivazioni a giustificare il fatto.

Planned Value (PV)

- **Descrizione**: costo pianificato per la realizzazione delle attività di progetto fino a quel momento;
- **unità di misura**: la metrica è espressa tramite un numero di giorni o di €;
- **formula**: $PV = \%lavoro\_pianificato \times BAC$;
- **risultato**: si attende un valore che sia maggiore o uguale a 0. In caso contrario si presume sia presente un errore nel calcolo ed è necessaria una sua reiterazione.

Actual Cost (AC)

- **Descrizione**: costo effettivamente sostenuto fino al momento del calcolo;
- **unità di misura**: la metrica è espressa tramite un numero di giorni o di €;
- **risultato**: si attende un valore maggiore o uguale di zero e minore di BAC. In caso contrario, qualora il valore sia negativo si presume sia presente un errore di calcolo, nel caso sia maggiore di BAC è necessario fornire valide motivazioni per il superamento del budget previsto.

Earned Value (EV)

- **Descrizione**: valore totale delle attività portate a termine al momento del calcolo;
- **unità di misura**: la metrica è espressa tramite un numero di giorni o di €;
- **formula**: $EV = \%lavoro\_completato \times BAC$;
- **risultato**: ci si aspetta di ottenere un valore maggiore o uguale di 0. In caso contrario si presume sia presente un errore nel calcolo ed è necessaria una sua reiterazione.

Cost Variance (CV)

- **Descrizione**: è un indicatore di produttività o efficienza nella gestione del progetto. Stabilisce se il budget effettivamente speso è maggiore, uguale o minore, rispetto a quanto si era previsto di spendere;
- **unità di misura**: la metrica è espressa tramite un numero di giorni o di €;
- **formula**: $CV = EV - AC$;
- **risultato**: si attende un valore uguale o maggiore di zero, ad indicare che il progetto produce con efficienza pari o maggiore rispetto a quanto pianificato. In caso questo valore sia negativo, significa che ci si trova in una situazione *over budget*, ed è necessario rivedere la gestione delle risorse nel progetto.

Schedule Variance (SV)

- **Descrizione**: è un indicatore di efficacia nei confronti del Proponente$_G$ e della capacità del gruppo di mantenersi in linea con la pianificazione temporale delle attività del progetto.
- **unità di misura**: la metrica è espressa tramite un numero di giorni o di €;
- **formula**: $SV = EV - PV$;
- **risultato**: l'obiettivo è ottenere un valore uguale o maggiore di zero, ad indicare una produzione in pari o in anticipo rispetto alla pianificazione temporale. Un valore negativo rappresenta un ritardo nella tabella di marcia e la necessità di ottimizzare eventuali attività successive, per poter tornare a lavorare coerentemente al piano stabilito.

Correlazione tra CV e SV

Lo stato di un progetto è esprimibile dalla correlazione tra *Cost Variance* e *Schedule Variance*, in particolare:

1. **SV e CV positive**: il progetto è in anticipo rispetto alla pianificazione e rientra nel budget previsto;
2. **SV positiva, CV negativa**: il progetto è in anticipo rispetto alla pianificazione ma ha superato il budget allocato;
3. **SV negativa, CV positiva**: il progetto è in ritardo rispetto alla pianificazione ma rientra nel budget previsto;
4. **SV e CV negative**: il progetto è in ritardo rispetto alla pianificazione e ha superato il budget previsto.

4.1.5 Strumenti

I membri del gruppo *Roundabout* possono lavorare indifferentemente su Windows, Mac OS X o Linux in quanto i principali strumenti necessari ai fini del progetto sono disponibili per tutti i sistemi operativi citati. Gli applicativi organizzativi utilizzati sono di seguito descritti.

4.1.5.1 Microsoft Teams

Si è individuato Microsoft Teams$_G$ come strumento di condivisione e comunicazione interno. La decisione è motivata dalla notevole multifunzionalità dell'applicativo, in quanto consente video-chiamate, condivisione di file e chat divise per argomento.

4.1.5.2 Git e GitHub

Come software di controllo versione$_G$ si è deciso di impiegare Git$_G$, che rappresenta uno dei migliori strumenti attualmente esistenti per quanto riguarda performance e facilità di utilizzo. Per lo sviluppo collaborativo abbiamo deciso di appoggiarci al servizio GitHub$_G$ il quale fornisce non solo un repository$_G$ Git$_G$, ma anche funzionalità utili per la cooperazione fra più persone.

4.1.5.3 Gmail

Per la gestione della corrispondenza si è scelto di creare una casella mail su dominio Gmail$_G$.

4.1.5.4 Slack

In accordo con il Proponente$_G$, si è concordato l'utilizzo della piattaforma Slack$_G$ per avere un canale di comunicazione più diretto e veloce rispetto alle mail.

4.1.5.5 Zoom

Per la gestione delle video-chiamate con il Proponente$_G$ ed il Committente$_G$, si è scelto di utilizzare l'applicazione Zoom$_G$.

4.1.5.6 Telegram

Utilizzato come mezzo di comunicazione interno per scambiare informazioni velocemente ed in maniera informale.

4.2 Formazione

4.2.1 Descrizione

Per avere i membri del gruppo *Roundabout* allineati relativamente alle conoscenze degli strumenti applicativi utilizzati, si rende necessario procedere ad un adeguata formazione dei singoli attraverso lo studio autonomo. Lo scopo di questo processo è fornire ai diversi membri del gruppo una formazione sufficiente.
In particolare, oltre al materiale indicato nella sotto sezione *Riferimenti Informativi*, si segnala anche la seguente documentazione:

- LaTeX$_G$: latex-project.org
- Solidity$_G$: solidity.readthedocs.io
- TypeScript$_G$: www.typescriptlang.org

Oltre allo studio autonomo, un altro metodo di formazione è l'apprendimento dall'operato altrui, di modo che le proprie conoscenze possano essere arricchite da quelle di altre persone maggiormente preparate in specifici argomenti.

4.2.2 Condivisione documentazione

Oltre ai riferimenti già citati è prevista una condivisione fra i membri del gruppo di informazioni utili alla formazione personale, il tutto gestito attraverso la piattaforma Microsoft Teams$_G$. È stato infatti creato un documento all'interno dello spazio di archiviazione condiviso dal team, che può venire costantemente aggiornato da un qualsiasi membro con eventuali nuove informazioni interessanti o link ipertestuali a siti di formazione. Anche lo stesso spazio di archiviazione può divenire luogo di condivisione di documentazione tecnica o utili video-tutorial.

A Standard di qualità

A.1 ISO/IEC 15504

ISO/IEC 15504, conosciuto anche come SPICE (*Software Process Improvement and Capability Determination*), è lo standard internazionale usato per valutare la qualità dei processi software e perseguirne il miglioramento continuo. La qualità di ogni processo viene valutata oggettivamente mediante la misurazione della capacità dello stesso tramite specifici attributi.
Lo standard ISO/IEC 15504 definisce un modello di riferimento che si suddivide in:

- dimensione del processo;
- dimensione della capacità.

A.1.1 Dimensione del processo

La *process dimension* divide i processi in cinque categorie:

- Customer/Supplier;
- Engineering;
- Supporting;
- Management;
- Organization.

A.1.2 Dimensione della capacità

Per ogni processo, ISO/IEC 15504 definisce un livello di maturità basato sulla scala rappresentata di seguito:

Tabella A.1.1: Scala di maturità dello standard ISO/IEC 15504

Livello	Nome
5	Optimizing process
4	Predictable process
3	Established process
2	Managed process
1	Performed process
0	Incomplete process

La capacità (o maturità) di ciascun processo viene misurata tramite gli attributi descritti di seguito:

Tabella A.1.2: Attributi per la misurazione della capacità dello standard ISO/IEC 15504

Appartenenza al livello ed identificativo	Nome
1.1	Process Performance
2.1	Performance Management
2.2	Work Product Management
3.1	Process Definition
3.2	Process Deployment
4.1	Process Measurement
4.2	Process Control
5.1	Process Innovation
5.2	Process Optimization

Ciascun attributo consiste di una o più pratiche generiche che aiutano nella fase di valutazione. Inoltre, ciascun attributo è valutato secondo una scala a quattro valori (N-P-L-F):

Tabella A.1.3: Scala di valutazione degli attributi dello standard ISO/IEC 15504

Stato	Range di valori corrispondenti
Not achieved	0 - 15%
Partially achieved	>15 - 50%
Largely achieved	>50 - 85%
Fully achieved	>85 - 100%

La valutazione viene fatta sulla base di evidenze oggettive acquisite durante la fase di assessment. La figura di seguito mostra la relazione tra livello di maturità, attributi di misurazione e relativa scala di valutazione nell'attività di valutazione della qualità di processo.

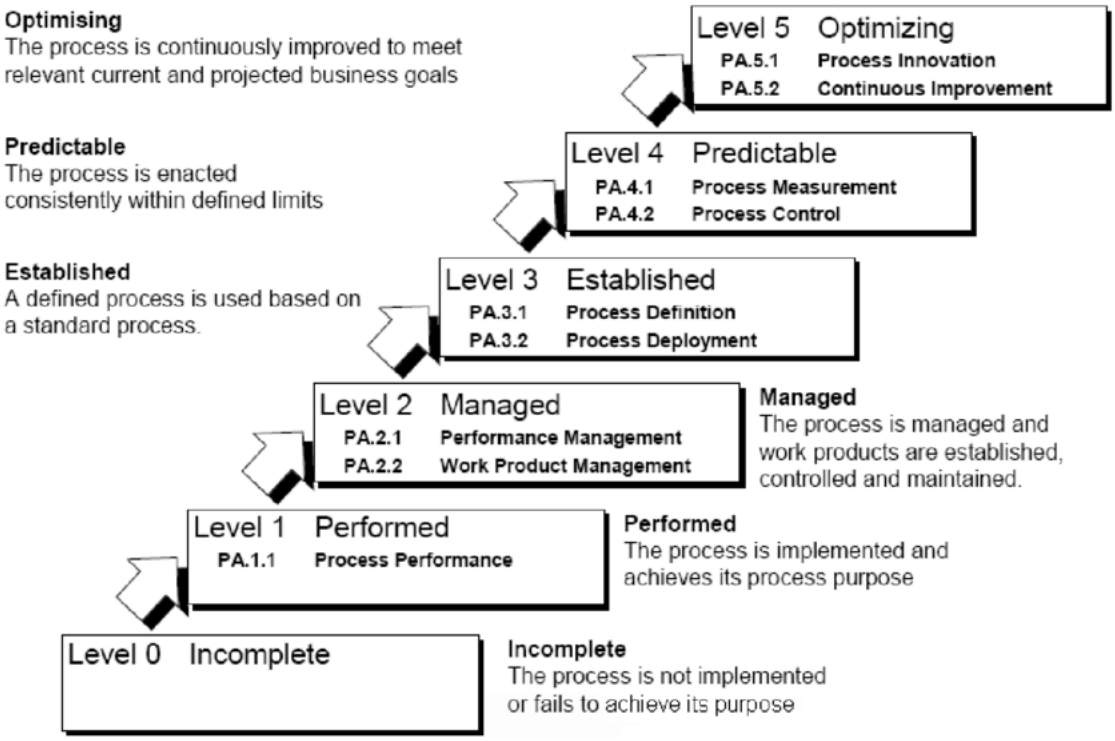

Figura A.1.1: SPICE Capability (fonte: Janos Ivanyos, `researchgate.net`)

A.2 Ciclo di Deming

Il ciclo di Deming (o ciclo di PDCA, acronimo dall'inglese di *Plan-Do-Check-Act*) è un metodo di gestione iterativo utilizzato per il controllo ed il miglioramento continuo dei processi e, di riflesso, anche dei prodotti da questi risultanti. Il ciclo di Deming è strutturato in quattro fasi, come illustrate di seguito:

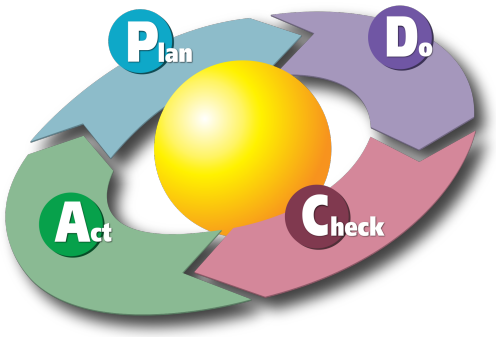

Figura A.2.1: Ciclo PDCA per il miglioramento continuo (fonte: Wikipedia)

1. **Plan**: è la fase di pianificazione in cui vengono stabiliti gli obiettivi ed i processi necessari a raggiungerli. In questa fase vengono definite tutte le attività da svolgere, le risorse da assegnarvi e le scadenze da rispettare;
2. **Do**: è la fase di esecuzione del programma precedentemente stilato, dapprima -se possibile- in contesti circoscritti. E' possibile ed auspicabile in questa fase raccogliere dati utili alle fasi di *Check* e *Act*;
3. **Check**: è la fase di controllo per accertarsi che la fase *Do* sia stata eseguita in accordo con ciò che era stato deciso nella fase *Plan*. In questa fase vengono anche studiati i risultati confrontandoli con quelli attesi;
4. **Act**: è la fase di attuazione, che permette di rendere definitivi i processi i cui esiti sono stati positivi o apportare modifiche migliorative in caso contrario.

I quattro punti sopra indicati costituiscono una sequenza logica che viene ripetuta finché l'obiettivo finale non è raggiunto.

A.3 ISO/IEC 9126

ISO/IEC 9126 è lo standard internazionale usato per valutare la qualità del software. Esso è articolato in quattro parti:

1. modello per la qualità del software, che a sua volta è suddiviso in:
   - modello per la qualità esterna ed interna;
   - modello per la qualità in uso;
2. metriche per la qualità interna;
3. metriche per la qualità esterna;
4. metriche per la qualità in uso.

A.3.1 Modello per la qualità del software

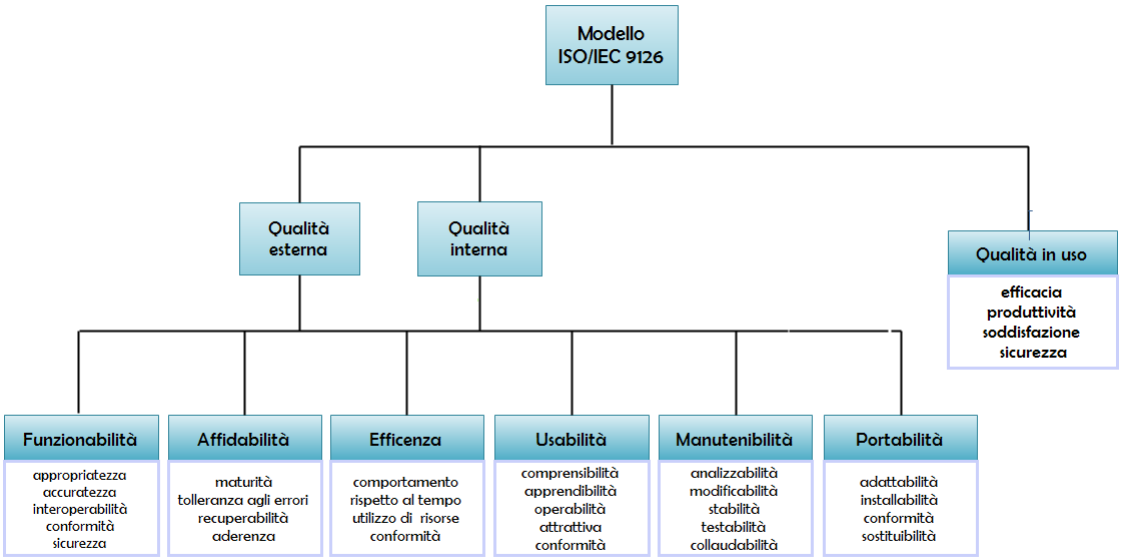

Figura A.3.1: Modello ISO/IEC 9126 (fonte: Wikipedia)

A.3.1.1 Modello per la qualità esterna ed interna

Il modello per la definizione della qualità interna ed esterna è composto da sei caratteristiche generali e varie sotto caratteristiche misurabili attraverso delle metriche. Tali caratteristiche sono:

Funzionalità

E' la capacità del software di fornire funzioni che soddisfano esigenze stabilite nell'*Analisi dei Requisiti* e che permettono di operare in condizioni specifiche. Questa capacità si traduce nelle seguenti sotto caratteristiche:

- **appropriatezza**: capacità di fornire funzioni appropriate per attività specifiche che permettano di raggiungere gli obiettivi prefissati;
- **accuratezza**: capacità di fornire risultati corretti e con la precisione richiesta;
- **interoperabilità**: capacità di interagire con uno o più sistemi specificati;
- **conformità**: capacità di aderire a standard rilevanti al settore in esame;
- **sicurezza**: capacità di proteggere informazioni e dati.

Affidabilità

E' la capacità del software di mantenere uno dato livello di prestazioni quando usato in specifiche condizioni. Questa capacità si traduce nelle seguenti sotto caratteristiche:

- **maturità**: capacità di evitare il verificarsi di errori o malfunzionamenti in fase di esecuzione;
- **tolleranza agli errori**: capacità di mantenere livelli predeterminati di prestazioni anche in presenza di malfunzionamenti o errori;
- **recuperabilità**: capacità di ripristinare il livello di prestazioni e di recupero delle informazioni rilevanti in seguito ad un malfunzionamento;
- **aderenza**: capacità di aderire a standard e regole inerenti all'affidabilità.

Efficenza

E' la capacità del software di eseguire le funzioni prefissate minimizzando il tempo necessario e sfruttando al meglio le risorse disponibili. Questa capacità si traduce nelle seguenti sotto caratteristiche:

- **nel tempo**: capacità di fornire appropriati tempi di risposta;
- **nello spazio**: capacità di utilizzare un appropriato numero di risorse.

Usabilità

E' la capacità del software di essere capito, appreso, usato ed accettato positivamente dall'utente. Questa capacità si traduce nelle seguenti sotto caratteristiche:

- **comprensibilità**: capacità di essere chiaro riguardo le proprie funzionalità ed il proprio utilizzo;
- **apprendibilità**: capacità di essere facilmente apprendibile;
- **operabilità**: capacità di permettere all'utente di raggiungere i suoi scopi e controllarne l'uso;
- **attrattività**: capacità di essere piacevole per l'utente che ne fa uso;
- **conformità**: capacità di aderire a standard o convenzioni relativi all'usabilità.

Manutenibilità

E' la capacità del software di essere modificato includendo correzioni, miglioramenti od adattamenti. Questa capacità si traduce nelle seguenti sotto caratteristiche:

- **analizzabilità**: capacità di essere facilmente analizzato al fine di individuare un errore;
- **modificabilità**: capacità di essere agevolmente modificato nel codice, nella progettazione o nella documentazione;
- **stabilità**: capacità di evitare effetti indesiderati a seguito di una modifica;
- **testabilità**: capacità di essere facilmente testato al fine di validare le modifiche apportate.

Portabilità

E' la capacità del software di essere trasportato da un ambiente hardware/software ad un altro seguendo le evoluzioni tecnologiche. Questa capacità si traduce nelle seguenti sotto caratteristiche:

- **adattabilità**: capacità di essere facilmente adattato a differenti ambienti operativi senza applicare modifiche;
- **installabilità**: capacità di essere installato in uno specifico ambiente;
- **conformità**: capacità di aderire a standard e convenzioni relative alla portabilità;
- **sostituibilità**: capacità di essere utilizzato al posto di un altro software per svolgere gli stessi compiti nello stesso ambiente.

A.3.1.2 Modello per la qualità in uso

Il modello per la definizione della qualità in uso elenca quattro caratteristiche generali che permettono agli utenti di ottenere specifici obiettivi. Tali caratteristiche sono:

- **efficacia**: capacità del software di permettere agli utenti di raggiungere gli obiettivi specificati con accuratezza e completezza;
- **produttività**: capacità del software di essere efficiente rispetto alle risorse necessarie;
- **soddisfazione**: capacità del software di soddisfare gli utenti;
- **sicurezza**: capacità del software di avere dei livelli di rischio accettabili rispetto a danni nei confronti di persone, apparecchiature e ambiente operativo.

A.3.2 Metriche per la qualità interna

La qualità interna viene rilevata tramite analisi statica, ovvero le metriche scelte vengono applicate a software non eseguibile e permettono di individuare eventuali problemi che potrebbero influire sulla qualità finale del prodotto. Le misure effettuate permettono di prevedere il livello di qualità esterna ed in uso del prodotto finale.

A.3.3 Metriche per la qualità esterna

La qualità esterna viene rilevata tramite analisi dinamica. Le metriche sono applicate al software in esecuzione e ne misurano il comportamento attraverso attività di test in funzione degli obbiettivi prefissati. Idealmente la qualità esterna determina la qualità in uso.

A.3.4 Metrica per la qualità in uso

Si tratta di metriche applicabili solo al prodotto finito ed in uso in condizioni reali. La qualità in uso viene raggiunta solo se è stato raggiunto sia il livello di qualità interna che di qualità esterna.