Roundabout - Etherless

Norme di Progetto

Versione	0.4.1
Approvazione	
Redazione	Luca Benetazzo
	Nicoletta Fabro
	Veronica Barbieri
Verifica	Alessandro Sgreva
Stato	Non 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
0.4.1	2020-03-26	Veronica Barbieri	*Amministratore*	Stesura §3.3.
0.4.0	2020-03-25	Alessandro Sgreva	*Verificatore*	Revisione e modifica Processi Primari e Processi di Supporto.
0.3.3	2020-03-23	Veronica Barbieri	*Amministratore*	Stesura §3.2, §3.3 e §3.4.
0.3.2	2020-03-23	Veronica Barbieri	*Amministratore*	Correzioni a seguito della verifica in §3 "Processi di Supporto".
0.3.1	2020-03-23	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.
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

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 1.0.0*. 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**;
- **Documentazione LaTeX$_G$**:
  https://www.latex-project.org/help/documentation/;

- **Airbnb JavaScript style guide**:
  https://github.com/airbnb/javascript/blob/master/README.md;
- **Solidity 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 necessarie 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 *RedBabel* e dei Committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.
L'obiettivo del gruppo è quello di 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 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**: presentano 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**: espongono 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 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:

- *Mancano riferimenti*

2.1.2.3 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:

- *Mancano riferimenti*

2.1.3 Strumenti

Di seguito vengono elencati gli strumenti usati durante il 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 dei Requisiti;
- Progettazione;
- Codifica.

In particolare, l'obiettivo del team è quello di 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 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.

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$_G$, compito degli Analisti è quello di elencarli con un grado di precisione che va dal generale al particolare, usando la struttura:

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

Ogni caso d'uso 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 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*: requisito indicato nel capitolato;
  - *interno*: requisito individuato dagli analisti;
  - *caso d'uso*: il requisito è stato estrapolato da uno o più casi d'uso. In questo caso vengono riportati gli identificativi dei casi d'uso considerati;
  - *verbale*: 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. 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 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*. 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, poi descritta nei seguenti documenti:

- ***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

Tale documento viene redatto dal Progettista e contiene:

- **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;
  - 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;
- **Tracciamento delle componenti**: viene rappresentata la relazione tra ogni requisito ed il componente che lo soddisfa. Dimostrando quindi concretamente la loro completa esaustione;
- **Test di integrazione**: vengono definite delle classi di verifica, per accertarsi che ogni componente del sistema funzioni come previsto.

Product Baseline

Tale documento viene redatto dal Progettista e contiene:

- **Diagrammi UML$_G$**:
  - diagrammi di attività;
  - diagrammi delle classi;
  - diagrammi di sequenza.

- **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, che dovrà essere in inglese e scritto 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*;
- **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 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 fase ha come scopo normare l'effettiva realizzazione del prodotto software richiesto. In questa fase si concretizza la soluzione architetturale elaborata durante la fase di progettazione, mediante la programmazione vera e propria. Gli sviluppatori, durante la fase di 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 conforme ai requisiti indicati dal proponente;
- migliorare la qualità del prodotto$_G$.

La scrittura del codice dovrà inoltre perseguire gli obiettivi di qualità stabiliti nel documento *Piano di Qualifica v1.0.0*, 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.

Etherless-cli ed *Etherless-server* saranno scritte in TypeScript sono normate dalla Airbnb JavaScrit Style Guide, il cui uso per TypeScript è implemetnato tramite ESLint. Per quanto riguarda *Etherless-smart*, questo verrà sviluppato in Solidity, seguendo la Style Guide presente nella documentazione ufficiale di Solidity.

2.2.3 Metriche

2.2.3.1 Metriche di analisi dei requisiti

2.2.3.2 Metriche di progettazione

2.2.3.3 Metriche di codifica

2.2.4 Strumenti

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

2.2.4.1 ESLint

ESLint è uno strumento di analisi statica per identificare pattern problematici all'interno di codice JavaScript. Le regole in ESLint sono configurabili; è inoltre possibile descrivere e utilizzare regole personalizzate. ESLint controlla sia la code quality che eventuali problemi riguardanti lo stile di codifica.

https://eslint.org/

2.2.4.2 Draw.io

Per la creazione dei diagrammi dei casi d'uso è stato scelto Draw.io, a causa della sua facile integrazione con Google Drive e velocità e semplicità di creazione dei diagrammi di interesse.

https://app.diagrams.net/

2.2.4.3 Visual Studio Code

Visual Studio Code è l'IDE 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;
- numerose estensioni facilmente installabili in grado di coprire la maggior parte delle necessita espresse dal 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*, 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 l'attività di verifica finale non rileva ulteriori errori, il Responsabile passa il documento alla fase 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.

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 C2 - *Etherless*.

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:** 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.ext;
- per un *Verbale*$_G$: VI_2020_03_16.tex.

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

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 le fasi del progetto:
  - **RR:** Revisione dei Requisiti;
  - **RP:** Revisione di Progettazione;
  - **RQ:** Revisione di Qualifica;
  - **RA:** Revisione di Accettazione.

3.1.7 Strumenti

3.1.7.1 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.

3.1.7.2 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.

3.1.7.3 Editor di testo

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

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$_G$.

3.3 Gestione della qualità

3.3.1 Descrizione

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* v1.0.0. In particolare per ogni processo ed ogni prodotto vengono descritti gli obiettivi e le metriche per la valutazione del raggiungimento degli stessi.

Qualità di prodotto

La qualità del prodotto$_G$ viene garantita dai processi di Verifica e Validazione (descritti rispettivamente nelle sezioni §3.4 e §3.5), che devono portare ad un miglioramento continuo del prodotto.

Qualità di processo

L'obiettivo è il miglioramento continuo dei processi, perseguendo i principi di efficacia ed efficenza del prodotto durante tutto il ciclo di vita del software.

3.4 Verifica

3.4.1 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.

3.4.2 Attività

3.4.2.1 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);
- **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.

3.4.2.2 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:** quindi 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:** riguardano il funzionamento di singole unità di software;
- **Test di integrazione:** test eseguiti per verificare la corretta integrazione di multiple unità in un unica componente. I componenti vengono verificati in maniera incrementale, con l'aggiunta di altre unità o gruppi di unità corretti, fino ad arrivare al sistema completo;
- **Test di sistema:** test del sistema nella sua interezza e confronto dei risultati con quanto richiesto dall'*Analisi dei Requisiti*, così da accertare la copertura dei requisiti funzionali. Introduce il processo di Validazione;
- **Test di regressione:** utili a controllare che eventuali modifiche al sistema non compromettano funzionalità già testate in precedenza. Consiste nella reiterazione di test già esistenti, per l'unità modificata ed altre unità connesse a quest'ultima.

3.5 Validazione

3.5.1 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.

3.5.2 Attività

3.5.2.1 Test di accettazione

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*".

4 Processi Organizzativi

Questa sezione comprende tutti i processi propedeutici o ausiliari, utili affinché i Processi Principali 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 coordinazione del lavoro, all'interno del progetto. Lo scopo principale di questo processo è quello di fornire ai membri del gruppo un piano organizzativo efficace ed efficiente.

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* 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:

- coordinamento, pianificazione e controllo delle attività;
- gestione delle risorse umane;
- approvazione della documentazione;
- approvazione dell'offerta economica;
- preventiva l'analisi dei rischi e loro eventuale gestione.

4.1.2.2 Amministratore

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

- studio e ricerca di 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;
- trovare soluzioni ai problemi legati alla difficoltà di gestione dei processi e risorse, attraverso la realizzazione o la ricerca di strumenti adatti a tale scopo;
- controllo delle versioni e configurazioni del prodotto$_G$;
- gestione del 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;
- analizza il dominio$_G$ delle richieste tramite lo studio dei bisogni, espliciti ed impliciti;
- analizza il dominio$_G$ applicativo: gli utilizzatori e l'ambiente di utilizzo;
- ha il compito di stesura dei 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:

- produrre una soluzione attuabile, comprensibile e motivata;
- effettuare scelte su aspetti progettuali, applicando al prodotto$_G$ soluzioni note ed ottimizzate;
- effettuare 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:

- implementazione precisa e scrupolosa delle soluzioni generate dal Progettista;
- scrittura di codice sorgente altamente e facilmente manutenibile;
- versionamento e documentazione del codice prodotto$_G$;
- realizzazione di 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:

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

4.1.3 Procedure

4.1.3.1 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.

4.1.3.2 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 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 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.

4.1.4 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:

4.1.4.1 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.

4.1.4.2 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.

4.1.4.3 Gmail

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

4.1.4.4 Slack

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

4.1.4.5 Zoom

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

4.1.4.6 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 appunto quello di fornire ai diversi membri del gruppo una sufficiente formazione.
In particolare, oltre al materiale indicato nella sotto sezione *Riferimenti Informativi*, 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 quello dell'apprendimento dall'operato altrui, facendo si che le proprie conoscenze possano essere arricchite da quelle di altre persone maggiormente preparate in specifici argomenti.