Norme di Progetto

Sons Of SWE - Progetto Marvin

sonsofswe.swe@gmail.com

Informazioni sul documento

Versione	5.0.0
Redazione	Caldart Federico Cavallin Giovanni Thiella Eleonora Menegon Lorenzo Dalla Riva Giovanni Favero Andrea
Verifica	Favero Andrea Panozzo Stefano Cavallin Giovanni Eleonora Thiella
Approvazione	Cavallin Giovanni
Uso	Interno
Distribuzione	Vardanega Tullio Cardin Riccardo Gruppo Sons Of SWE

Descrizione

Documento contenente le norme di progetto che il gruppo Sons Of SWE seguirà durante tutte le fasi di realizzazione del prodotto Marvin.

Registro delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
5.0.0	2018-07-12	Approvazione	Federico Caldart	*Responsabile di Progetto*
4.1.0	2018-07-11	Verifica	Andrea Favero	*Verificatore*
4.0.1	2018-07-08	Incremento dei processi primari	Lorenzo Menegon	*Amministratore*
4.0.0	2018-06-06	Approvazione	Lorenzo Menegon	*Responsabile di Progetto*
3.2.0	2018-06-05	Verifica	Stefano Panozzo	*Verificatore*
3.1.2	2018-06-05	Incremento dei processi di codifica e progettazione	Eleonora Thiella	*Amministratore*
3.1.1	2018-06-04	Aggiunta di contenuti e della struttura organizzativa riguardante i processi	Eleonora Thiella	*Amministratore*
3.1.0	2018-05-19	Verifica	Giovanni Cavallin	*Verificatore*
3.0.1	2018-05-18	Revisione correttiva dei contenuti di seguito alle segnalazione del committente	Federico Caldart	*Amministratore*
3.0.0	2018-05-07	Approvazione	Eleonora Thiella	*Responsabile di Progetto*
2.1.0	2018-05-06	Verifica	Stefano Panozzo	*Verificatore*
2.0.3	2018-05-06	Aggiornata la lista degli strumenti utilizzati	Andrea Favero	*Analista*
2.0.2	2018-05-05	Aggiunta del *processo di validazione*	Giovanni Dalla Riva	*Amministratore*
2.0.1	2018-05-04	Incremento dei *processi di supporto*	Andrea Favero	*Amministratore*
2.0.0	2018-04-28	Approvazione	Giovanni Dalla Riva	*Responsabile di Progetto*
1.1.0	2018-04-28	Verifica	Giovanni Cavallin	*Verificatore*
1.0.1	2018-04-27	Revisione correttiva dei contenuti di seguito alle segnalazioni del committente	Federico Caldart	*Amministratore*
1.0.0	2018-03-12	Approvazione	Stefano Panozzo	*Responsabile di Progetto*
0.1.0	2018-03-12	Verifica	Andrea Favero	*Verificatore*
0.0.10	2018-03-09	Completata la stesura dei *processi organizzativi*	Lorenzo Menegon	*Amministratore*
0.0.9	2018-03-09	Completata la stesura dei *processi di supporto*	Giovanni Cavallin	*Amministratore*
0.0.8	2018-03-08	Completata le stesura della sezione sui *processi primari*	Eleonora Thiella	*Amministratore*
0.0.7	2018-03-20	Correzione per una formattazione standard per tutte le sezioni	Eleonora Thiella	*Amministratore*
0.0.6	2018-03-08	Correzione punteggiatura e ortografia per tutte le sezioni	Giovanni Cavallin	*Amministratore*
0.0.5	2018-03-07	Aggiornati strumenti per la sezione *processi organizzativi*	Federico Caldart	*Amministratore*

Versione	Data	Descrizione	Autore	Ruolo
0.0.4	2018-03-05	Creata la sezione riguardante i *processi organizzativi*	Federico Caldart	*Amministratore*
0.0.3	2018-03-04	Creata la sezione riguardante i *processi di supporto*	Giovanni Cavallin	*Amministratore*
0.0.2	2018-03-04	Creata sezione riguardante i *processi primari*	Eleonora Thiella	*Amministratore*
0.0.1	2018-03-03	Creato lo scheletro del documento e la sezione *Introduzione*	Giovanni Cavallin	*Amministratore*

Indice

1 Introduzione

1.1 Scopo del documento

In questo documento verranno trattate le norme interne alle quali i membri di *SonsOfSwe* dovranno obbligatoriamente sottostare. Ogni membro dovrà visionare il documento e seguirne le regole in esso contenute, per ottenere la massima *efficienza*$_{\mathrm{G}}$ ed *efficacia*$_{\mathrm{G}}$, mantenendo un certo grado di uniformità.
In questo documento verranno esposte le norme riguardanti:

- L'identificazione dei ruoli e delle relative mansioni che essi dovranno svolgere;
- Le modalità di lavoro durante le fasi di *progetto*$_{\mathrm{G}}$;
- Le interazioni tra i membri del gruppo e con le entità esterne;
- L'organizzazione e la cooperazione all'interno del *team*$_{\mathrm{G}}$;
- La modalità e le regole adottate per la stesura dei documenti;
- La definizione degli ambienti di sviluppo.

1.2 Contenuto del documento

Dal momento che la stesura di questo documento e le norme in esso contenute vengono redatte incrementalmente a seconda delle esigenze di progetto, la sua forma e il suo contenuto non saranno completi fino alla realizzazione del prodotto finale.

1.3 Scopo del prodotto

Lo scopo del prodotto è quello di realizzare un *prototipo*$_{\mathrm{G}}$ di Uniweb come una *ĐApp*$_{\mathrm{G}}$ in esecuzione su *Ethereum*$_{\mathrm{G}}$. I cinque attori principali che si rapportano con Marvin sono:

- Utente non autenticato;
- Università;
- Amministratore;
- Professore;
- Studente.

Il portale deve quindi permettere agli studenti di accedere alle informazioni riguardanti le loro carriere universitarie, di iscriversi agli esami, di accettare o rifiutare voti e di poter vedere il loro libretto universitario. Ai professori deve invece essere permesso di registrare i voti degli studenti. L'università ogni anno crea una serie di corsi di laurea rivolti a studenti, dove ognuno di essi comprende un elenco di esami disponibili per anno accademico. Ogni esame ha un argomento, un numero di crediti e un professore associato. Gli studenti si iscrivono ad un corso di laurea e tramite il libretto elettronico mantengono traccia ufficiale del progresso.

1.4 Glossario

Nel documento *Glossario_v3.0.0* i termini tecnici, gli acronimi e le abbreviazioni sono definiti in modo chiaro e conciso, in modo tale da evitare ambiguità e massimizzare la comprensione dei documenti.
I vocaboli presenti in esso saranno posti in corsivo e presenteranno una "G" maiuscola a pedice.

1.5 Riferimenti

1.5.1 Normativi

- **Regolamento del progetto didattico**: http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/P01.pdf;
- ***ISO*$_{G}$ 31-0**: http://en.wikipedia.org/wiki/ISOwiki/ISO_31;
- **ISO 8601**: http://it.wikipedia.org/wiki/ISO_8601;
- **ISO 12207-1995**: http://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf.

1.5.2 Informativi

- **Amministrazione di progetto**: http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/P06.pdf;
- *Specifiche UTF-8*$_{G}$: http://www.unicode.org/versions/Unicode6.1.0/ch03.pdf;
- **Slides del corso di Ingegneria del Software**: http://www.math.unipd.it/~tullio/IS-1/2017/

2 Processi primari

2.1 Processo di fornitura

2.1.1 Studio di Fattibilità

Si tratta del documento in cui vengono analizzati tutti i capitolati, valutandone pregi e difetti, allo scopo di scegliere quello di maggiore affinità per il gruppo. Dopo che il *Responsabile di Progetto* (il cui ruolo è descritto nel paragrafo 4.2.2.1) avrà riunito il team e discusso con esso di tutti i capitolati, gli *Analisti* avranno il compito di stilare lo *StudioDiFattibilità_v1.0.0* seguendo le considerazioni emerse.
Lo *StudioDiFattibilità_v1.0.0* sarà organizzato come segue:

- **Informazioni sul capitolato**: Vengono ricordati il nome del *progetto*$_{\text{G}}$, il *proponente*$_{\text{G}}$ e i *committenti*$_{\text{G}}$;
- **Descrizione**: Si riassume lo scopo del capitolato;
- **Dominio applicativo**: Si specifica il settore di utilizzo del prodotto finale;
- **Dominio tecnologico**: Si elencano le tecnologie che dovranno essere utilizzate nello sviluppo del capitolato;
- **Aspetti positivi**: Si elencano i motivi che il gruppo potrebbe considerare vantaggiosi;
- **Aspetti negativi**: Si elencano le potenziali criticità che il gruppo dovrà tenere in considerazione nella scelta finale;
- **Valutazione finale**: Si analizzano gli aspetti positivi e quelli negativi riscontrati e si motiva l'eventuale approvazione o esclusione.

2.1.2 Rapporti con il proponente

Una volta scelto il capitolato, si intende instaurare un rapporto quanto più costante e profittevole con la Red Babel allo scopo di:

- Stabilire un accordo in merito allo sviluppo, al mantenimento, al funzionamento e alla consegna del prodotto;
- Realizzare un prodotto che soddisfi totalmente i requisiti obbligatori concordati e quanto più possibile quelli desiderabili;
- Stimare i costi;
- Concordare la qualifica del prodotto.

2.1.3 Collaudo e consegna del prodotto

Una volta terminate le fasi di sviluppo, verifica e validazione si effettuerà il collaudo al fine di dimostrare che tutti i requisiti obbligatori e, possibilmente, anche alcuni dei requisiti opzionali siano stati soddisfatti.
In questa fase inoltre si dovrà dimostrare che l'esecuzione di tutti i test di validazione abbia dato un'esito positivo.
Il team consegnerà in ultima il prodotto finale su un supporto fisico ai committenti Tullio Vardanega e Riccardo Cardin.

2.2 Processo di sviluppo

2.2.1 Scopo del processo

Contiene tutte le attività svolte dal team, al fine di produrre il software finale.

2.2.2 Aspettative del processo

Le aspettative previste dal team per una efficace implementazione del processo sono le seguenti:

- realizzare un prodotto finale che soddisfi i testi di validazione e verifica, secondo quanto concordato con il proponente;
- fissare gli obiettivi di sviluppo;
- normare i vincoli tecnologici per lo sviluppo.

2.2.3 Descrizione

Secondo lo standard ISO/IEC 12207, il processo di sviluppo si divide nelle seguenti attività:

- Analisi dei requisiti;
- Progettazione;
- Codifica.

2.2.4 Attività

2.2.4.1 Analisi dei Requisiti

2.2.4.1.1 Scopo : Lo scopo dell'*AnalisiDeiRequisiti_v4.0.0*consiste nell'individuare i requisiti del progetto tramite le specifiche del capitolato, tramite le comunicazioni con il proponente e tramite le valutazioni effettuate durante le riunioni interne del gruppo.

2.2.4.1.2 Descrizione : Il risultato di tale attività dovrà essere stilato dagli *Analisti*, contenente la lista dei requisiti e dei casi d'uso. Essi dovranno inoltre fornire un diagramma conforme allo standard UML per ogni caso d'uso. Tale documento ha il fine di comprendere maggiormente le scelte di progettazione effettuate.

2.2.4.1.3 Classificazione dei Requisiti : i requisiti saranno classificati secondo la seguente codifica:

R[Importanza][Tipo][Codice]

dove:

- **Importanza** può assumere questi valori:
  - F: indica un requisito funzionale;
  - Q: indica un requisito di qualità;
  - P: indica un requisito prestazionale;
  - V: indica un requisito di vincolo.
- **Tipo** può assumere questi valori:
  - O: indica un requisito obbligatorio;

  - D: indica un requisito desiderabile;
  - F: indica un requisito facoltativo.
- **Codice** indica il codice identificativo del requisito, è univoco e deve essere indicato in forma gerarchica.

Per ogni requisito si dovrà inoltre indicare una breve descrizione e la fonte, che può essere una tra le seguenti:

- Capitolato: deriva direttamente dal testo del capitolato;
- Verbale: deriva da un incontro verbalizzato;
- Interno: deriva da discussioni interne al team.

2.2.4.1.4 Classificazione dei casi d'uso : i casi d'uso saranno classificati secondo la seguente codifica:

UC[Codice padre].[Codice identificativo]

dove:

- Codice padre: indica il codice del caso d'uso padre di quello in esame, se non è identificabile è da omettere;
- Codice identificativo: codice univoco e progressivo del caso d'uso in esame.

Per ogni caso d'uso saranno inoltre identificate le seguenti informazioni:

- **Attori**: indica gli attori coinvolti nel caso d'uso;
- **Descrizione**: chiara, precisa e concisa descrizione del caso d'uso;
- **Precondizione**: indica la situazione che deve essere vera prima dell'esecuzione del caso d'uso;
- **Flusso principale degli eventi**: descrizione composta dal flusso dei casi d'uso figli;
- **Postcondizione**: indica la situazione che deve essere vera dopo l'esecuzione del caso d'uso;
- **Estensioni**: indica quali sono tutte le estensioni, se presenti;
- **Generalizzazioni**: indica quali sono tutte le generalizzazioni, se presenti.

2.2.4.2 Progettazione

2.2.4.2.1 Scopo

L'attività di progettazione deve precedere la produzione del software ed ha il compito di soddisfare le peculiarità ed i bisogni individuati durante l'attività di *analisi dei requisiti*, e di realizzare al meglio i requisiti di qualità imposti dal committente. L'attività di progettazione si impone inoltre di generare una soluzione soddisfacente per tutti gli individui che fanno parte del progetto. Una volta compresi pienamente i requisiti del problema e approfondita la progettazione in parti abbastanza semplici, si forniranno le istruzioni necessarie ai *Progettisti* per sviluppare il prodotto finito.
I *Progettisti* (il cui ruolo è descritto nel paragrafo 4.2.2.4) dovranno delinerare i requisiti utili alla documentazione specifica e determinare le linee guida da seguire in funzione dei requisiti individuati dall'analisi.

2.2.4.2.2 Progettazione architetturale La progettazione permette di:

- Costruire l'architettura logica del prodotto;
- Ottimizzare l'utilizzo delle risorse;
- Garantire la qualità prefissata del prodotto;
- Organizzare le varie parti del progetto in modo da ottenere componenti singole e facili da implementare durante la successiva fase di codifica.

La progettazione si colloca nel periodo di *Progettazione e codifica* all'interno del *PianoDiProgetto_v4.0.0*; periodo all'interno del quale vengono individuati i design pattern e realizzata l'architettura del prodotto tramite l'utilizzo dei diagrammi delle classi e di sequenza. Il lavoro complessivo ottenuto in questo arco di tempo rappresenta la **Product Baseline**. Una volta sviluppata l'architettura generale del software dovranno inoltre essere documentati i seguenti task:

- Suddividere il prodotto in componenti;
- Definire il ruolo dei membri;
- Definire le interazioni tra i componenti;
- Assicurarsi che ogni requisito sia soddisfatto da almeno un componente;
- Realizzare e documentare i test d'integrazione, al fine di verificare il corretto funzionamento di più componenti tra loro;
-

2.2.4.2.3 Linee guida per la progettazione Dopo aver completato l'*AnalisiDeiRequisiti_v4.0.0* i *Progettisti* dovranno sottostare alle seguenti linee guida per lo sviluppo dell'architettura logica del sistema:

- Si dovrà puntare ad una progettazione chiara e di immediata comprensione;
- Le componenti progettate dovranno essere quanto più riutilizzabili e manutenibili;
- La complessità non dovrà mai essere intrattabile;
- I *Progettisti* dovranno rientrare nei costi e nelle risorse disponibili;
- I *Progettisti* dovranno descrivere i *design pattern*$_G$ che intendono utilizzare per la realizzazione dell'architettura, fornendone una breve descrizione e un diagramma;
- Si dovrà evitare l'utilizzo, ove possibile di classi astratte;
- Si dovrà evitare package vuoti, classi non utilizzate, e parametri senza tipo.

2.2.4.2.4 UML
Le tipologie di diagrammi *UML*$_G$ che verranno adoperate per analizzare, descrivere e specificare le scelte progettuali adottate saranno:

- **Diagrammi di classe**: illustrano una collezione di elementi che rappresenta un modello di classi e tipi, con i rispettivi contenuti e relazioni;
- **Diagrammi di *package*$_G$**: documentano le dipendenze tra le classi ed è utile per controllare la complessità strutturale in sistemi medio-grandi;
- **Diagrammi di attività**: modellano un processo e organizzano più entità in un sistema di azioni secondo un determinato flusso. I diagrammi delle attività sono un tipo particolare di *diagramma di stato*$_G$ che identifica la variazione di stato al verificarsi di alcune condizioni legate ad una o più entità;
- **Diagrammi di sequenza**: descrivono la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento. Sono diagrammi molto semplici, ma che permettono di capire se l'architettura creata viene eseguita.

2.2.4.2.5 Testing Per quanto riguarda l'attività di test, i *Progettisti* devono preoccuparsi di definire delle classi di verifica per accertare il funzionamento dei componenti. In alcuni casi non è quindi possibile fornire ai *Verificatori* degli strumenti automatici per l'esecuzione dei test. La progettazione delle classi per i test necessita di essere svolta senza sovrapporre lo stesso ruolo; chi crea una classe non può essere incaricato anche di testarla.

2.2.4.2.6 Requisiti per i progettisti I *Progettisti* sono responsabili delle attività di progettazione. Per un corretto svolgersi del loro lavoro, sono tenuti ad acquisire le seguenti conoscenze:

- Buona conoscenza generale del processo di sviluppo software;
- Estesa visione generale delle scelte progettuali da attuare in situazioni particolarmente problematiche;
- Discreta inventiva per trovare possibili soluzioni progettuali anche nel caso in cui non vi siano casi metodologici esplicitamente applicabili;
- Capacità di individuare le scelte più opportune e coerenti alle esigenze del team.

2.2.4.2.7 Obiettivi di progettazione La progettazione ha lo scopo di conseguire i seguenti obiettivi:

- Progettare un software con caratteristiche qualitative pari a quelle descritte nella fase di analisi dei requisiti;
- Progettare un software affinchè sia possibile effettuare modifiche nel caso in cui sia necessario, senza avere la necessità di mettere in discussione l'intera struttura del software;
- Progettare un software che sia modulare, ossia che ogni parte sia chiara e distinta ed abbia una determinata funzione;
- Soddisfare i requisiti fissati dal committente;
- Costruire un software robusto, ed in grado di gestire malfunzionamenti di fronte a situazioni critiche improvvise;
- Ridurre ove sia possibile gli sprechi di tempo e spazio, ragionando in termini di efficienza;
- Progettare un software sicuro per quanto riguarda possibili intrusioni e pericoli esterni;
- Rispettare il principio dell'incapsulazione, tramite un corretto utilizzo del *data hiding*$_{\text{G}}$.

2.2.4.3 Codifica In questa fase i *Programmatori* (il cui ruolo è descritto nel paragrafo 4.2.2.5), seguendo le norme delineate nella progettazione, devono realizzare il passaggio dalla fase di pianificazione all'effettiva realizzazione del prodotto. Le norme qui presenti serviranno come strumento per realizzare un codice uniforme e di alta qualità. Inoltre, per mantenerne la manutenibilità, dovrà essere realizzato in inglese. I *Programmatori* si dovranno attenere agli standard di codifica qui di seguito elencati.

2.2.4.3.1 Indentazione I blocchi di codice prodotti dovranno essere indentati secondo le seguenti norme:

- *Annidamento*: I blocchi annidati dovranno essere correttamente indentati, usando per ciascun livello di profondità un *Tab* (equivalente a quattro spazi) per allineare correttamente le sezioni di codice;
- *Capo linea*: Per andare a capo linea di una riga di codice è opportuno utilizzare due volte un *Tab* (equivalente a otto spazi), per allineare correttamente le sezioni di codice;
- *Profondità*: Nel caso in cui ci siano blocchi di codice annidati tra loro, si dovrà sviluppare il codice sul livello relativo al proprio blocco, tenendo conto delle regole precedentemente elencate.
- *Parentesizzazione*: I blocchi di codice andranno sempre chiusi all'interno di parentesi graffe, anche nel caso in cui il linguaggio non lo richieda esplicitamente, come per esempio nel caso dei controlli condizionali *if-else*;
- *Parentesizzazione 2*: Le parentesi graffe andranno collocate sulla stessa riga in cui inizia il blocco di codice.

2.2.4.3.2 Nomi

- Ogni elemento (classe, metodo, e variabile) deve avere un nome rappresentativo e pertinente alla funzione da esso svolta;
- Il nome di una classe deve sempre iniziare con la lettera maisucola;
- Si dovrà utilizzare la notazione *CamelCase*, ovvero la concatenazione di più parole, ognuna delle quali con lettera iniziale maiuscola. In caso di metodi e variabili la prima lettera dovrà essere minuscola, mentre per le classi maiuscola;
- Si potranno utilizzare singole lettere esclusivamente per identificare gli indici dei cicli;
- Saranno da evitare notazioni troppo simili tra di loro in significato e denominazione;
- Si dovranno evitare errori di ortografia.

2.2.4.3.3 Commenti

Sarà necessario che il codice contenga dei commenti esplicativi per facilitarne la comprensione.
In particolare, si dovranno seguire le seguenti linee guida:

- Quando si modifica codice già esistente, vanno aggiornati i relativi commenti;
- Evitare di aggiungere commenti sulla stessa riga di codice; la continua presenza di codice misto a testo può creare confusione e difficoltà di lettura;
- Evitare di scrivere commenti privi di senso logico, al fine di separare righe di codice; nel caso in cui insorgesse questa necessità si dovrà utilizzare spazi vuoti;
- Il fine dei commenti deve essere di chiarire il codice scritto; è quindi opportuno utilizzare frasi semplici, di senso compiuto e con una corretta punteggiatura;
- E' opportuno commentare tutte le parti di codice non chiare o particolarmente complicate;
- E' opportuno utilizzare i commenti nel caso in cui una parte di codice venga corretta in seguito a degli errori, al fine di tenere presenti errori frequenti e le possibili soluzioni da applicare.

2.2.4.3.4 Ricorsione La ricorsione dovrà essere evitata ove possibile, cercando di limitarne l'uso ai casi indispensabili. Bisognerà quindi utilizzare in primis l'approccio iterativo nel risolvere i problemi, con eccezione per i casi in cui l'utilizzo della ricorsione possa fare notevole differenza in termini di costo computazionale ed efficienza.

2.2.4.3.5 Formattazione La formattazione serve a facilitare la comprensione dell'organizzazione logica del codice. Per consentire agli sviluppatori di decifrare facilmente il codice si dovranno seguire le seguenti regole:

- Il rientro predefinito dovrà essere di un *Tab* per allineare le sezioni di codice;
- Prima e dopo ogni operatore dovrà esserci uno spazio;
- I blocchi saranno tra loro spaziati per una maggiore comprensione;
- Utilizzare spazi vuoti per definire la struttura del codice; in questo modo sarà possibile ottenere paragrafi separati tra loro più facilmente comprensibili;

2.2.4.3.6 Visibilità Le variabili vanno dichiarate nel blocco più interno che comprenda l'utilizzo della variabile stessa, al fine di ottenere codice di qualità migliore. E' quindi fortemente sconsigliato l'utilizzo di variabili globali.

2.2.4.3.7 Inizializzazione Le variabili devono sempre essere inizializzate quanto prima possibile. E' consigliato inizializzare una variabile ogni qual volta venga dichiarata. Nel caso in cui non si possa effettuare questa operazione, è consigliabile posticipare la dichiarazione al momento in cui si potrà effettuare.

2.2.4.3.8 Codice esterno Nel caso in cui si decida di utilizzare codice esterno sviluppato da terze parti, è consigliabile utilizzare lo stile e le norme di codifica degli autori di quel codice, al fine di non creare ambiguità stilistiche e facilitarne la comprensione.

2.2.4.3.9 Classificazione dei test È compito dei *Progettisti* lo sviluppo dei test di unità e integrazione e degli *Analisti* creare quelli sistema e accettazione al fine di verificare che tutti i requisiti individuati precedentemente siano stati soddisfatti. Ogni test deve essere codificato come segue:

T[X][Y]

dove:

- **X**: rappresenta il tipo di test, e può assumere i seguenti valori:
  - **U**: indica un test di unità;
  - **I**: indica un test di integrazione;
  - **S**: indica un test di sistema;
  - **R**: indica un test di regressione;
  - **V**: indica un test di validazione.
- **Y**: rappresenta il codice univoco di ogni test secondo la classificazione gerarchica.

2.2.5 Strumenti

Gli strumenti utilizzati durante la fase dei processi primari sono:

- **TexStudio**
  Il gruppo ha scelto *TexStudio* come editor multipiattaforma per comporre i documenti in *LaTeX*$_G$;
- **SWEgo**$_G$ [1]
  Il gruppo ha scelto *SWEgo* come database per la gestione dei casi d'uso, dei requisiti e del loro tracciamento per il documento *AnalisiDeiRequisiti_v4.0.0*. Dopo un'attenta analisi preliminare dello strumento il team ha deciso di utilizzarlo nonostante alcuni problemi rilevati nella generazione del codice LaTeX perché, permette una stesura standardizzata e sempre aggiornata di alcuni capitoli importanti del documento, a fronte di correzioni minori e applicabili in maniera seriale e regolamentata;
- **Astah**$_G$
  Il gruppo ha scelto l'utilizzo di *Astah* per la generazione dei diagrammi dei casi d'uso dell'*AnalisiDeiRequisiti_v4.0.0*;
- **Microsoft Excel 365**
  Il gruppo ha scelto di utilizzare *Microsoft Excel 365*$_G$ per la generazione di grafici e di tabelle da poter inserire all'interno del *PianoDiQualifica_v4.0.0* e *PianoDiProgetto_v4.0.0* perché di immediata realizzazione e visualizzazione;
- **Instagantt** [2]
  Instagantt è un servizio web fortemente legato ad Asana (vedi descrizione negli strumenti dei *processi organizzativi*), strumento con il quale si integra perfettamente, nonostante sia disponibile anche una versione *standalone*$_G$. Esso permette di creare diagrammi di Gantt e gestire in maniera semplificata la timeline e la struttura di un progetto.

[1] https://www.swego.it/
[2] https://instagantt.com/

- **Visual Studio Code** [3]
  Visual Studio Code è un editor di testo gratuito e multipiattaforma che supporta operazioni di debugging e di versionamento. Permette inoltre di installare vari pacchetti che consentono di personalizzarne l'usabilità e di supportare nuove tecnologie o linguaggi di programmazione. Il gruppo lo ha scelto come editor di testo per scrivere il codice in React e Redux.
- **Truffle** [4]
  Truffle è un ambiente di sviluppo e testing per Ethereum. Il gruppo ha deciso di utilizzarlo perché fornisce un aiuto nello sviluppare, pubblicare e testare gli smart contract.
- **Ganache** [5]
  Ganache è uno strumento legato a Truffle che consente di avere una blockchain virtuale locale utilizzabile per sviluppare contratti, applicazioni ed effettuare dei test. Permette anche di generare degli account di testing utilizzabili durante lo sviluppo.
- **Npm** [6]
  npm è un gestore di pacchetti per il linguaggio Javascript.
- **Remix**
  Remix è l'ambiente di sviluppo integrato ufficiale di Ethereum per il linguaggio di programmazione di smart contract *solidity*$_G$. Viene eseguito all'interno di un browser web, online all'indirizzo `https://remix.ethereum.org/` oppure in locale scaricando i relativi file sulla propria macchina.

[3] https://code.visualstudio.com/
[4] http://truffleframework.com/
[5] http://truffleframework.com/ganache/
[6] https://www.npmjs.com/

3 Processi di Supporto

3.1 Processo di documentazione

3.1.1 Scopo del processo

Lo scopo di questo processo consiste nell'illustrare le linee guida di tutta la documentazione che verrà prodotta riguardante il ciclo di vita del software.

3.1.2 Descrizione

In questa sezione del documento dovranno essere redatte tutte le norme utilizzate dal team per produrre una documentazione coerente e di qualità riguardo lo sviluppo del software.

3.1.3 Organizzazione

Per questo motivo, il team suddividerà i documenti in:

- **Documenti interni**
  Tutti quei documenti che saranno visionati da fornitori e committenti;
- **Documenti esterni**
  Tutti quei documenti che saranno visionati anche dai proponenti.

3.1.4 Ciclo di vita di un documento

Un documento passerà attraverso tre stati:

- **In lavorazione**: si tratta della fase di stesura del documento e non è consultabile;
- **Da verificare**: dopo che il documento è stato ultimato, passerà nelle mani del *Verificatore* (il cui ruolo è descritto nel paragrafo 4.2.2.6), che dovrà esaminarlo;
- **Approvato**: dopo la verifica, il documento dovrà essere approvato definitivamente dal *Responsabile di Progetto*.

Ogni documento sarà identificato con un flag alla fine del nome, distanziato con un underscore, in base allo stato in cui si trova. Per il primo si userà $\_L$, per il secondo $\_V$, per il terzo $\_A$.

3.1.5 Documenti finali ad uso interno

3.1.5.1 Studio di fattibilità (SdF)

Lo *StudioDiFattibilità_v1.0.0* ha lo scopo di raccogliere le informazioni salienti dei capitolati proposti, esprimendone gli aspetti positivi e le potenziali criticità che sono emerse durante il confronto col gruppo.

3.1.5.2 Norme di progetto (NdP)

Le *NormeDiProgetto_v4.0.0* contengono le regole che il team utilizzerà durante lo sviluppo del progetto.

3.1.5.3 Verbale interno (VI)

Il *VerbaleInterno_v1.0.0* servirà al gruppo per documentare le discussioni e le decisioni prese durante le riunioni. La denominazione dovrà essere come segue:

verbale_Tipo del verbale_Numero del verbale_Data del verbale

dove:

- **Tipo del verbale**: specifica se Interno (I) o Esterno (E);
- **Numero del verbale**: numero univoco identificativo del verbale;
- **Data del verbale**: identifica la data in cui la riunione si è svolta. Si utilizzerà il formato:

YYYY-MM-DD

Nella parte introduttiva del verbale verranno specificati:

- Data riunione;
- Ora inizio riunione;
- Ora fine riunione;
- Durata riunione;
- Luogo d'incontro;
- Oggetto di discussione;
- Moderatore;
- Segretario;
- Partecipanti.

3.1.6 Documenti finali ad uso esterno

3.1.6.1 Piano di Progetto (PdP)

Il *PianoDiProgetto_v4.0.0* contiene le indicazioni sulle scadenze temporali e fornisce un preventivo dei costi da presentare al proponente.
Vengono inoltre individuati i rischi ed analizzate le loro ricorrenze.
In questo documento vengono fatte emergere le *milestone*$_G$ legate ai punti critici e viene effettuata una *pianificazione*$_{\text{G}}$ con l'uso di *diagrammi di Gantt*$_{\text{G}}$.
In questo documento vengono fatte emergere le *milestone*$_G$ legate ai punti critici e viene effettuata una *pianificazione*$_{\text{G}}$ con l'uso di diagrammi di *Gantt*$_{\text{G}}$.

3.1.6.2 Piano di Qualifica (PdQ)

Il *PianoDiQualifica_v4.0.0* deve fornire ai membri del gruppo tutte le informazioni con cui poter soddisfare gli obiettivi di qualità.

3.1.6.3 Analisi dei Requisiti (AdR)

L'*AnalisiDeiRequisiti_v4.0.0* descrive gli *attori*$_{\text{G}}$ del sistema, individua i *casi d'uso*$_{\text{G}}$ a partire dai *requisiti*$_{\text{G}}$ e fornisce una visione chiara ai *Progettisti* sul problema da trattare.

3.1.6.4 Glossario (G)

Nel documento *Glossario_v3.0.0* i termini tecnici, gli acronimi e le abbreviazioni sono definiti in modo chiaro e conciso, in modo tale da evitare ambiguità e massimizzare la comprensione dei documenti.

3.1.6.5 Manuale Utente (MU)

Il *ManualeUtente_v1.0.0* è un manuale pensato per aiutare l'utente ad utilizzare il prodotto, viene incrementato durante lo sviluppo di quest'ultimo. Deve avere un approccio incentrato sulle funzionalità che il prodotto offre.

3.1.6.6 Manuale Sviluppatore (MS)
Il *ManualeSviluppatore_v1.0.0* è un manuale per aiutare lo sviluppatore nella manutenzione e nell'incremento delle funzionalità del prodotto.

3.1.6.7 Verbale Esterno (VE)
Il *VerbaleEsterno_v1.0.0* è un documento in cui si tiene traccia delle discussioni del team con i committenti ed i proponenti. Come struttura ricalca quella del *VerbaleInterno_v1.0.0*.

3.1.7 Struttura del documento

3.1.7.1 Prima pagina La prima pagina di ogni documento sarà così strutturata:

- Logo;
- Nome del documento;
- Nome del gruppo - Nome del progetto;
- Email del gruppo;
- Informazioni sul documento:
  - Versione del documento;
  - Redazione;
  - *Verifica*$_{\mathrm{G}}$;
  - Approvazione;
  - *Uso*$_{\mathrm{G}}$;
  - Distribuzione.
- Descrizione del documento.

3.1.7.2 Registro delle modifiche
In seconda pagina il documento conterrà il registro delle modifiche che traccerà le modifiche apportate al documento. Sarà organizzato in una tabella che conterrà le seguenti colonne:

- Versione: indica la versione del documento;
- Data: indica la data in cui il documento è stato modificato;
- Descrizione: descrive la modifica effettuata nella relativa versione;
- Autore: indica il nome della persona che ha effettuato la modifica;
- Ruolo: indica il ruolo dell'autore.

3.1.7.3 Indice
Dopo il registro delle modifiche, il documento sarà correlato da un indice di tutte le sezioni. In alcuni documenti, se necessario, sarà aggiunto anche l'indice delle immagini, delle tabelle e dei riferimenti.

3.1.7.4 Formattazione generale della pagina
Ogni pagina del documento, fatta eccezione per la prima, conterrà una intestazione ed un pié di pagina.
L'intestazione presenterà a sinistra il logo e a destra l'email del gruppo.
Nel pié di pagina saranno presenti a sinistra il nome del documento susseguito dal nome del gruppo e, a destra il numero della pagina.

3.1.8 Norme tipografiche

Tutti i documenti dovranno sottostare alle seguenti norme tipografiche ed ortografiche.

3.1.8.1 Formati

- **Data**: il formato della data seguirà quello esplicato nell'*ISO*$_{\text{G}}$ *8601:2004*$_{\text{G}}$, quindi sarà:

  YYYY-MM-DD

  dove i simboli stanno per:
  - YYYY: anno;
  - MM: mese;
  - DD: giorno.
- **Orario**: ci si atterrà allo standard europeo delle 24 ore:

  hh:mm

  dove i simboli stanno per:
  - hh: ore;
  - mm: minuti.

3.1.8.2 Composizione del testo

- **Elenchi puntati**: ogni punto dell'elenco deve terminare con "*;*" tranne l'ultimo che deve terminare con ".". La prima parola deve iniziare con unla lettera maiuscola;
- **Glossario**: il pedice "$_{\text{G}}$" verrà utilizzato in corrispondenza di vocaboli presenti nel Glossario_v3.0.0.

3.1.8.3 Stili di testo

- **Grassetto**: il grassetto deve essere utilizzato per evidenziare parole particolarmente importanti negli elenchi puntati o nelle frasi;
- **Corsivo**: il corsivo deve essere utilizzato con i seguenti termini: situazioni:
  - Ruoli;
  - Documenti;
  - Stati del documento;
  - Citazioni;
  - Glossario;
  - Nomi di file.
- **Maiuscolo**: il maiuscolo deve essere utilizzato solamente per gli acronimi.

3.1.8.4 Sintassi

Per la stesura dei documenti i membri del team adotteranno la terza persona singolare.

3.1.8.5 Sigle

- **AdR**: Analisi dei Requisiti;
- **PdP**: Piano di Progetto;
- **NdP**: Norme di Progetto;
- **SdF**: Studio di Fattibilità;
- **PdQ**: Piano di Qualifica;
- **LdP**: Lettera di Presentazione;
- **G**: Glossario;
- **TB**: Technology Baseline;
- **PB**: Product Baseline;
- **MU**: Manuale Utente;
- **MS**: Manuale Sviluppatore;
- **RR**: Revisione dei Requisiti;
- **RP**: Revisione di Progettazione;
- **RQ**: Revisione di Qualifica;
- **RA**: Revisione di Accettazione.

3.1.9 Nome del file *.pdf*$_G$

Ogni documento sarà generato come file con estensione .pdf ed avrà un nome che rispetti la seguente convenzione:

NomeFile_versione.pdf

Il *NomeFile* seguirà la notazione CamelCase già descritta, invece la *versione* seguirà lo standard descritto qui di seguito.

3.1.10 Struttura dei file in LaTeX

Lo scheletro dei file necessari per generare un file .pdf di un documento è così strutturato:

- *sos.sty*: contiene tutti i package e le macro che possono trovare utilizzo in tutti i documenti. Contiene anche anche la macro che genera la copertina della prima pagina e che viene invocata dal file *main.tex*;
- *main.tex*: Include il template e tutte le sezioni che compongono il documento. Il file non è monolitico perché include le sezioni (che quindi sono dei file *.tex* che hanno vita propria), in questo modo si cerca di puntare all'assenza di conflitti durante modifiche contemporanee a più parti dello stesso documento e, di facilitare la revisione di singole parti del documento;
- *comandi.tex*: contiene delle macro specifiche per il documento che si sta creando;
- *"img" folder*: contiene il logo del gruppo e le altre immagini che sono incluse nel documento.

3.1.11 Versionamento

Il *versionamento*$_G$ permette a ciascun membro del team di condividere il lavoro nello spazio comune e di lavorare su vecchi e nuovi *Configuration Item*$_G$ senza rischio di sovrascritture accidentali.
Avrà questa forma:

vX.Y.Z

dove:

- **X**:
  - Inizia da 0;
  - Viene incrementato solo nel caso in cui il documento venga rilasciato pubblicamente.
- **Y**:
  - Inizia da 0;
  - Viene incrementato dal *Responsabile di Progetto* una volta approvato il documento;
  - Quando viene incrementato, *Z* viene riportato a 0.
- **Z**:
  - Alla creazione parte da 1;
  - Viene incrementato dal redattore del documento ogni volta che questo è viene modificato.
  - Quando vengono incrementati X o Y viene riportato a 0.

3.1.12 Strumenti

3.1.12.1 LaTeX

Il team ha scelto di usufruire del linguaggio LaTeX per questi motivi:

- Permette un versionamento più semplice e compatibile con *Git*$_G$;
- Evita i conflitti che si possono creare usando software differenti (per esempio OpenOffice e Microsoft Word).

3.1.13 Gestione del *repository*$_G$

Per tenere traccia di tutte le modifiche apportate ai documenti ed al codice che sono stati prodotti, il team ha deciso di utilizzare *GitHub*, creando una *repository*$_G$ dedicata. Si seguiranno le norme descritte qui di seguito.

3.1.13.1 Struttura

- Il *master*$_G$ è annidato nella repository *origin*$_G$ e deve contenere solo il *template*$_G$ per la scrittura di tutti i documenti;
- Ogni *branch*$_G$ annidato sul master deve corrispondere solo alle *revisioni*$_G$;
- Ogni branch annidato in branch relativi alle revisioni contiene dei documenti che le riguardano;
- Ogni membro del gruppo potrà modificare i file di ogni documento o codice, o direttamente sul branch corrispondente oppure creandone uno parallelo a sua discrezione a meno di direttive diverse da parte dei *Progettisti*.

3.1.13.2 Tipi di file Tutti i file pertinenti verranno caricati nel repository e saranno sottoposti a versionamento. Verranno tuttavia ignorati, quindi inseriti all'interno del file *.gitignore*$_G$, tutti quelle quei file generati automaticamente durante le varie *build*$_G$ e che hanno estensioni indesiderate (ad esempio *.log*, *.out*).

3.1.13.3 Norme sui *commit*$_G$

I membri del team dovranno registrare le modifiche apportate alla repository tramite commit. Questo genererà una *pull request*$_G$ che il responsabile dovrà approvare prima di unire le modifiche al branch di riferimento.

3.2 Processo di verifica

3.2.1 Scopo del processo

Lo scopo del processo di verifica consiste nel controllare e correggere eventuali errori commessi durante il compimento delle attività dei processi svolti.

3.2.2 Descrizione

Il processo di verifica deve essere continuo, questo serve ad evitare che eventuali errori arrivino fino alla fase di validazione. Questa attività, che viene svolta in corso d'opera, deve essere:

- Tempestiva, cioè il dato deve esserci quando serve;
- Accurata, devono essere evitate scorrettezze;
- Non intrusiva, non deve interrompere alcuna attività durante la sua esecuzione.

3.2.3 Qualità

Nel documento *PianoDiQualifica_v4.0.0* si devono descrivere le tecniche usate dai membri del gruppo con lo scopo di garantire la qualità del prodotto, in particolare:

- tutti gli standard adottati e che si occupano della qualità dei processi devono essere citati;
- devono essere presentati tutti i processi rilevanti di tali standard.

Inoltre per ogni processo si devono descrivere:

- gli obbiettivi di qualità che derivano da esso;
- le metodologie e le strategie considerate di utilità per raggiungerne gli obbiettivi;
- le metriche da utilizzare per poter effettuare delle misurazioni oggettive e qualitative.

3.2.3.1 Metriche per il processo

In questa sezione verranno descritte le metriche che verranno utilizzate per garantire la qualità dei processi, comprese le metriche relative alla qualità del codice prodotto.

3.2.3.1.1 Compiti assegnati

Tale metrica verrà utilizzata per controllare che la pianificazione di progetto sia stata eseguita correttamente, verificando che tutti i compiti che compongono ogni attività siano distribuiti ed assegnati ai membri del team. Verrà calcolata come segue:

$$\text{SV} = \frac{\text{Numero compiti assegnati}}{\text{Numero compiti totali}} * 100$$

3.2.3.1.2 Schedule Variance - SV

La Schedule Variance è un indicatore che permette di capire se un processo è il linea, in anticipo o in ritardo con la schedulazione temporale indicata del *PianoDiProgetto_v4.0.0*. Viene calcolata come percentuale di tempo speso, considerando le date di inizio e di fine previste e la data di completamento effettiva:

$$\text{SV} = \frac{\text{Data di completamento effettiva - Data di completamento pianificata}}{\text{Data di completamento pianificata - Data di inizio pianificata}} * 100$$

I risultati possibili sono tre:

- SV > 0, che indica un ritardo sui tempi pianificati;
- SV $= 0$, che indica l'essere in linea con i tempi pianificati;
- SV < 0, che indica un anticipo sui tempi pianificati.

3.2.3.1.3 Cost Variance - CV

La Cost Variance è una metrica che permette di capire se i costi effettivi siano in linea o meno con i costi pianificati nel *PianoDiProgetto_v4.0.0*. Viene calcolata come percentuale di costo utilizzando BCWP (Budgeted Cost of Work Performed), cioè il valore delle attività svolte fino al momento del calcolo e ACWP (Actual Cost of Work Performed).

$$\text{CV} = \frac{\text{BCWP} - \text{ACWP}}{BCWP} * 100$$

I risultati possibili sono tre:

- CV > 0, che indica che il progetto sta producendo con un minor costo rispetto a quanto pianificato;
- CV $= 0$, che indica l'essere in linea con i costi preventivati;
- CV < 0, che indica che il progetto sta producendo con un costo maggiore rispetto a quello pianificato.

3.2.3.1.4 Rischi non preventivati

Tale metrica verrà utilizzata per evidenziare il numero di rischi che non sono stati preventivati all'inizio di ogni fase di progetto e che hanno dunque fatto sorgere problemi inaspettati. Viene calcolato come indice numerico, il quale aumenta con il presentarsi di un problema non preventivato.

3.2.3.1.5 SFIN - Structural fan in

La structural fan in è una metrica che verrà utilizzata insieme alla successiva per misurare il livello di relazione tra diversi elementi del software. In particolare, un risultato di SFIN alto indicherà che la procedura sulla quale è calcolata è largamente utilizzata dal sistema e dunque c'è molto riuso del suo codice. Data una procedura, viene calcolata come segue:

$$\text{SFIN} = numerodicomponenticheutilizzanolaprocedura.$$

3.2.3.1.6 SFOUT - Structural fan out

La structural fan out è una metrica che verrà utilizzata insieme alla precedente per misurare il livello di relazione tra diversi elementi del software. In particolare, un risultato di SFOUT alto indicherà un codice fortemente accoppiato, dunque probabilmente molto complesso da eseguire e testare. Data una procedura o un componente, viene calcolata come segue:

$$\text{SFOUT} = numerodicomponentiesterneutilizzatedallaproceduraodalmodulo.$$

3.2.3.1.7 Complessità ciclomatica

La complessità ciclomatica è una metrica utilizzata per misurare la complessità di un software attraverso la valutazione dei suoi metodi, classi e algoritmi. Essa è calcolata utilizzando il grafo di flusso: in esso i nodi corrispondono a gruppi indivisibili di istruzioni, mentre gli archi connettono due nodi se le istruzioni del secondo possono essere eseguite immediatamente dopo quelle del primo. Questa misurazione sarà utile nella fase di sviluppo per limitare la complessità delle singole parti del software e nella fase di test per capire quanti test diversi saranno necessari per testare adeguatamente il codice. La misurazione si baserà su un indice numerico intero: valori troppo alti indicano un'eccessiva complessità del codice con conseguente scarsa manutenibilità, mentre valori troppo bassi potrebbero indicare una scarsa efficienza.

3.2.3.1.8 Commenti per linee di codice

Attraverso tale metrica si valuterà il rapporto commenti/linee di codice: una percentuale abbastanza alta di commenti aiuterà la comprensione del sorgente. Verrà misurata come segue:

$$\text{CxSLOC} = \frac{\text{Numero linee di commento}}{\text{Numero } SLOC_{\text{G}}} * 100$$

3.2.3.1.9 Parametri per metodo

Tale metrica si basa sul numero di parametri formali dei $metodi_{\text{G}}$ per valutare la complessità del codice: un numero elevato potrebbe infatti indicare un livello di complessità troppo alto per i singoli metodi.

3.2.3.1.10 Linee di codice per metodo

Tale metrica verrà utilizzata congiuntamente alla precedente (Parametri per metodo) per valutare il grado di complessità di un metodo: controllando il numero di $statement_{\text{G}}$ per ogni metodo è possibile facilitarne comprensione e verifica, spingendo verso una modularizzazione del codice il più ampia possibile. Questa metrica sarà fortemente influenzata dall'esperienza che il team guadagnerà durante lo sviluppo del progetto, motivo per cui i valori che seguono saranno indicativi e molto probabilmente modificati in futuro.

3.2.3.1.11 Copertura del codice

Tale metrica è orientata alla valutazione della qualità dei test; essa misura infatti la capacità di coprire mediante test gli statement del codice, attraverso il loro conteggio in percentuale, al fine di fornire dei test che assicurino una valutazione del software il più affidabile possibile. Verrà calcolata come segue:

$$\text{Copertura} = \frac{\text{Numero di statement testati}}{\text{Numero di statement totali}} * 100$$

3.2.3.1.12 Copertura dei branch

Tale metrica verrà utilizzata congiuntamente alla precedente (Copertura del codice) per valutare la qualità dei test. Essa indicherà la capacità dei test di valutare il maggior numero possibile di rami decisionali del grafo di flusso del software. Verrà calcolata come segue:

$$\text{Copertura}_{\text{branch}} = \frac{\text{Numero di rami raggiunti}}{\text{Numero di rami totali}} * 100$$

3.2.3.2 Metriche per i documenti In questa sezione vengono descritte le metriche che verranno utilizzate nel processo di verifica dei documenti prodotti.

3.2.3.2.1 Errori ortografici

Questa è la metrica che serve ad esprimere un giudizio di correttezza ortografica riguardo il documento prodotto. Gli errori saranno individuati secondo le seguenti modalità: il primo controllo, per quanto riguarda i documenti in lingua inglese, avverrà a tempo di stesura del documento tramite lo strumento di autocorrezione dell'ambiente *"TexStudio"*, mentre il secondo controllo avverrà dopo aver terminato la prima redazione del documento stesso. Esso avverrà tramite due verifiche manuali del *Verificatore*: una temporanea dopo aver finito la stesura del documento ed una definitiva prima dell'approvazione finale del documento stesso.
Formula:

Errori ortografici = numero di errori totali riscontrati dopo la verifica manuale definitiva

3.2.3.2.2 Indice Gulpease

L'Indice Gulpease è un indice di leggibilità di un testo tarato sulla lingua italiana, con il vantaggio rispetto ad altri indici di utilizzare la lunghezza delle parole in lettere anzichè in sillabe, semplificandone il calcolo automatico. L'indice utilizza due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere.
La formula per il suo calcolo è la seguente:

$$\text{Indice Gulpease} = 89 + \frac{300 * (\text{numero delle frasi}) - 10 * (\text{numero delle lettere})}{\text{numero delle parole}}$$

I risultati sono compresi tra 0 e 100, dove il valore "100" indica la leggibilità più alta e "0" la leggibilità più bassa. In generale risulta che testi con un indice

- Inferiori a 80 sono difficili da leggere per chi ha la licenza elementare;
- Inferiore a 60 sono difficili da leggere per chi ha la licenza media;
- Inferiore a 40 sono difficili da leggere per chi ha un diploma superiore.

3.2.3.2.3 Errori contenutistici

Questa è la metrica necessaria per esprimere la correttezza del contenuto di un documento. È importante verificare che i concetti trattati siano corretti e coerenti con quanto prefissato. Il valore ottenuto da questa metrica rappresenta il numero di errori concettuali riscontrati dal *Verificatore* durante la verifica definitiva del documento.
La formula utilizzata per il calcolo degli errori è la seguente:

Errori concettuali = numero di errori totali riscontrati dopo la verifica manuale definitiva

3.2.3.2.4 Struttura del documento

Viene utilizzata questa unità di misura per verificare quanto un documento sia attinente alle regole strutturali descritte nel documento *NormeDiProgetto_v4.0.0*. La metrica si basa sul numero di errori segnalati dal *Verificatore* durante la verifica definitiva del documento.
La formula utilizzata per il calcolo degli errori è la seguente:

Errori di forma = numero di errori totali riscontrati dopo la verifica manuale definitiva

3.2.3.3 Metriche per il prodotto software In questa sezione si descrivono le metriche che verranno usate dal gruppo per verificare e garantire la qualità dei prodotti software durante il periodo del progetto. Si sottolinea il fatto che questa sarà solo una prima stesura delle metriche e sarà raffinata nel corso delle varie revisioni, facendo frutto dell'esperienza che verrà acquisita negli intervalli di lavoro tra esse.

3.2.3.3.1 Requisiti soddisfatti

Tale metrica verrà utilizzata per valutare la funzionalità del software prodotto attraverso una misurazione quantificativa dei requisiti soddisfatti; verranno effettuate due misurazioni differenti, una per i soli requisiti obbligatori e una per tutti.

Requisiti obbligatori

$$\text{ROS} = \frac{\text{numero requisiti obbligatori soddisfatti}}{\text{numero totale requisiti obbligatori}}$$

Requisiti obbligatori e facoltativi

$$\text{ROFS} = \frac{\text{numero requisiti obbligatori soddisfatti} + \text{numero requisiti facoltativi soddisfatti}}{\text{numero totale requisiti}}$$

3.2.3.3.2 Successo dei test

Tale metrica verrà utilizzata per valutare in parte il livello di affidabilità del prodotto software tramite il calcolo della percentuale di test aventi successo nella fase di verifica.

$$\text{Successo dei test} = \frac{\text{numero test aventi successo}}{\text{numero totale dei test effettuati}} * 100$$

3.2.3.3.3 Tempo di risposta

Tale metrica verrà utilizzata per valutare l'efficienza del prodotto basandosi sul tempo medio che intercorrà tra la richiesta di una certa funzionalità da parte dell'utente e la risposta del software. Con *tempo medio* si intende la media tra i tempi medi di risposta di tutte le funzionalità: ognuna di esse dovrà essere testata almeno 5 volte ed in condizioni quanto più differenti.

$$\text{T}_{\text{rispostaF}} = \frac{\sum_{k=1}^{5} T_{\text{test}} k}{5}$$

$$\text{T}_{\text{rispostaTOT}} = \frac{\sum_{k=1}^{n} T_{\text{rispostaF}} k}{n}$$

3.2.3.3.4 Validazione pagine web

Tale metrica verrà usata come tentativo di applicare una metrica oggettiva e misurabile per valutare l'usabilità del prodotto finale; si è usata la parola "tentativo" poichè in effetti l'usabilità e l'accessibilità di un sito web sono due cose distinte, anche se affini: pagine web con contenuto inaccessibile saranno sicuramente poco usabili. Valutare l'accessibilità attraverso l'analisi del codice prodotto permetterà dunque di fornire una base allo sviluppo di pagine usabili. W3C offre uno strumento per valutare le pagine $HTML_{\text{G}}$, come dichiarato nelle *NormeDiProgetto_v4.0.0*: esso riporta il numero e il tipo di errori trovati nel documento in esame.

3.2.3.4 Verifica dei documenti Il *Responsabile di Progetto* Ha il compito di dare inizio al processo di verifica della documentazione prodotta, assegnando i corrispettivi compiti ai *Verificatori*. Questi dovranno verificare che siano state rispettate in maniera rigida le seguenti regole, e in caso contrario segnalarlo per una successiva correzione:

- Utilizzo di una sintassi corretta e semplice;
- Utilizzo di brevi periodi, per evitare frasi troppo complesse;
- Un corretto annidamento della struttura del documento, evitando gerarchie troppo profonde o estese;
- Rispetto di tutte le norme precedentemente elencate in questo documento.

Per tutta la durata del progetto il team avrà inoltre l'obbligo di utilizzare il registro delle modifiche ogni qual volta deciderà di alterare il contenuto di un documento, al fine di tenere traccia dei cambiamenti effettuati e degli errori commessi. Un *Verificatore*, per un corretto svolgimento della sua attività, ha il compito di svolgere i task descritti nei paragrafi successivi secondo il seguente ordine:

- Attività manuale di verifica;

- Attività automatica di verifica;
- Resoconto delle attività di verifica.

3.2.3.4.1 Attività manuale di verifica I *Verificatori* avranno il compito di eseguire l'attività manuale di verifica sui documenti in maniera attenta e minuziosa, accompagnando l'attività stessa da un elenco contenente il resoconto degli errori riscontrati secondo tutte le metriche presentate nelle *NormeDiProgetto_v4.0.0* e discusse nel *PianoDiQualifica_v4.0.0*.

3.2.3.4.2 Attività automatica di verifica Per ottenere una seconda verifica sui documenti prodotti, andando a consolidare già la prima verifica manuale effettuata, si è deciso di svolgere una verifica automatizzata tramite l'utilizzo del correttore automatico dell'editor di testo *TeXstudio*, al fine di individuare eventuali errori tralasciati per motivi di distrazione.

3.2.3.4.3 Resoconto dell'attività di verifica Al termine dello svolgimento dell'attività di verifica, il *Verificatore* è tenuto a consegnare al *Responsabile di Progetto*, il resoconto finale contenente tutti gli errori riscontrati durante lo svolgimento dell'attività, in modo che possano venire effettuate le opportune correzioni e discussi eventuali cambiamenti da entrambi i membri. Al termine della correzione dovrà quindi essere aggiornato il registro delle modifiche per tenere traccia dei cambiamenti effettuati e degli errori precedentemente commessi. Tutti gli errori riscontrati durante l'ultima attività di verifica, verranno inoltre impiegati in seguito per calcolare le metriche riguardanti gli obiettivi di qualità prefissati per i documenti redatti, e pubblicati nelle appendici del *PianoDiQualifica_v4.0.0*.

3.2.4 Analisi

3.2.4.1 Analisi statica

Studia il codice sorgente e la documentazione e controlla che essi seguano le norme. Non richiede l'esecuzione del prodotto software in nessuna sua parte, ma si limita all'osservazione. Questo processo può essere visto come una verifica dinamica del comportamento del programma su un insieme finito di casi selezionati nel dominio di tutte le esecuzioni possibili. Ciascun caso di prova, specifica i valori in ingresso e lo stato iniziale del sistema e deve produrre un esito decidibile, verificato rispetto ad un comportamento atteso.
Si può declinare in due maniere:

- ***Walkthrough***$_{\text{G}}$
  Questa tecnica di analisi deve essere effettuata nella fase iniziale dello sviluppo del prodotto per trovare eventuali errori che possano essere riscontrati. Non sapendo che tipo di errori cercare, si effettua una lettura a largo spettro. Quando i *Verificatori* avranno stilato una *checklist*$_{\text{G}}$ di tutti gli errori più comuni, si passerà alla fase di *Inspection*$_{\text{G}}$. Essendo un'analisi continua, la checklist tenderà ad aumentare costantemente di dimensione, rendendo più efficace ed efficiente l'Inspection.
- **Inspection**
  Basandosi sulla checklist redatta durante la Walkthrough, i *Verificatori* analizzeranno tutto il prodotto cercando, di volta in volta e in modo mirato, tutti gli errori ricorrenti in essa contenuti.

3.2.4.2 Analisi dinamica

L'analisi dinamica, al contrario di quella statica, richiede l'esecuzione del codice. Viene effettuata tramite test ed è coinvolta sia nel processo di verifica che in quello di validazione. I test verranno descritti in un capitolo a parte.

3.2.5 Test

Il testing è una parte essenziale del processo di verifica: produce una misura della qualità del sistema aumentandone il valore, identificandone e rimuovendone i difetti. Il suo inizio non va differito al termine delle

attività di codifica e le sue esigenze devono essere tenute in conto nella progettazione del sistema. Tutti i test devono essere rieseguibili e devono sempre produrre lo stesso esito. Devono essere eseguiti in condizioni controllate, diventando così deterministici.
Di seguito vengono descritte le tipologie di test, assieme alla sintassi relativa al modo in cui sono identificati all'interno del *PianoDiQualifica_v4.0.0*.

3.2.5.1 Test di unità (TU)

Il loro obiettivo è quello di verificare la correttezza del codice *as implemented*, ovvero puntando a controllare il codice in maniera microscopica.
La responsabilità della loro realizzazione è del *Programmatore* per le unità più semplici, e di un *Verificatore* indipendente per le altre.
Le risorse consumate da questo tipo di test sono molto poche perché, sono coinvolte piccole parti di codice che hanno basso accoppiamento le une con le altre. I test di unità sono classificati come segue:

TU[codice identificativo]

dove il codice identificativo è numerico, incrementale (iniziando da 1) ed univoco per ogni test.

3.2.5.2 Test di integrazione (TI)

Questi test verificano non solo il corretto comportamento di ogni singolo oggetto, ma anche le relazioni con gli altri componenti dell'applicazione.
Possono rilevare i seguenti problemi:

- Errori residui nella realizzazione dei componenti;
- Modifica delle interfacce o cambiamenti nei requisiti;
- Riuso di componenti dal comportamento oscuro o inadatto;
- Integrazione con altre applicazioni non bene conosciute.

Un test di questo tipo consuma molte risorse dato che, la grandezza del codice da controllare è significativa ed il grado di accoppiamento tra le varie parti da testare è massimo. L'utilizzo di un test di questo tipo deve quindi essere ponderato e, in generale, da non preferire a quello di unità. I test di integrazione sono classificati come segue:

TI[codice identificativo]

dove il codice identificativo è numerico, incrementale (iniziando da 1) ed univoco per ogni test.

3.2.5.3 Test di sistema (TS)

I test di sistema possono essere visti come un'attività interna del fornitore per accertare la copertura dei requisiti software.
Questo tipo di test richiede molte risorse e, in generale, non va utilizzato per testare unità singole. I test di sistema sono classificati come segue:

TS[codice requisito]

dove il codice requisito identifica il codice univoco associato ad ogni requisito descritto nel documento *AnalisiDeiRequisiti_v4.0.0*.

3.2.5.4 Test di regressione (TR)

I test di regressione sono l'insieme dei TU e TI necessari ad accertare che la modifica di una parte del prodotto non causi errori nelle altre parti che hanno relazioni con essa.
Un test di questo tipo consuma tante più risorse quanto la parte modificata è accoppiata con le altre, anche perché comporta la ripetizione di test già previsti ed effettuati per ogni parte che non è stata modificata.

3.2.5.5 Test di validazione (TV)

Il test di validazione è un'attività supervisionata dal committente e dal proponente come dimostrazione di conformità del prodotto sulla base di casi di prova specificati o implicati dal contratto. Alla validazione segue il rilascio del prodotto con eventuale garanzia e la fine della *commessa*$_G$, con eventuale manutenzione. I test di validazione sono classificati come segue:

TV[codice requisito][Y]

dove il codice requisito identifica il codice univoco associato ad ogni requisito descritto nel documento *AnalisiDeiRequisiti_v4.0.0*, mentre [Y] è una lettera che viene associata ai test che si riferiscono ad uno stesso requisito secondo una profondità gerarchica. Nel caso in cui non ci siano più test che si riferiscono allo stesso requisito [Y] può essere non presente.

3.2.6 Strumenti

3.2.6.1 Strumenti per l'analisi statica

- **TexStudio**$_G$[7]
  Questo editor include un correttore ortografico automatico che sarà utilizzato per verificare la corretta sintassi dei manuali in lingua inglese da fornire alla Red Babel. Tuttavia tale strumento non è preciso nell'individuazione degli errori più sottili, quindi sarà comunque necessaria un'analisi più approfondita da parte dei *Verificatori*;
- **Indice *Gulpease***$_G$
  Per i test di leggibilità il team ricorrerà al calcolo dell'indice Gulpease;
- ***W3C Markup Validation Service***$_G$[8]
  È uno strumento per la validazione rispetto agli standard dei documenti *HTML*$_G$ e *xHTML*$_G$;
- **Microsoft Excel**$_G$
  Tale programma verrà utilizzato per calcolare alcune metriche di qualità e fornirne i risultati in un formato grafico nel *PianoDiQualifica_v4.0.0*.

3.3 Validazione

Il processo di validazione ha i seguenti obiettivi:

- verificare che il prodotto finale sia conforme con quanto specificato;
- verificare che il prodotto finale sia in grado di minimizzare gli effetti degli errori commessi.

3.3.1 Ruoli

I ruoli per il processo di validazione saranno divisi nel seguente modo:

- i *Verificatori* avranno il compito di eseguire i test, tracciando i risultati e riportandone la valutazione;
- il *Responsabile di Progetto* avrà il compito di revisionare i risultati dei test effettuati, e decidere se convalidarli o ripeterli. Egli sarà inoltre incaricato di fornire al committente i risultati dei test effettuati, accompagnati da una documentazione per poterli svolgere in maniera indipendente.

3.3.2 Procedure di validazione

L'intera procedura di validazione dovrà quindi comprendere i seguenti punti:

- La verifica da parte dei *Verificatori* sul prodotto finale, ed il tracciamento dei risultati ottenuti;

[7] https://www.texstudio.org/
[8] https://validator.w3.org/

- L'analisi dei risultati da parte del *Responsabile di Progetto*, con verdetto finale di accettazione o ripetizione dei test;
- La consegna da parte del *Responsabile di Progetto* dei risultati accettati al proponente, informandolo dettagliatamente sulle modalità di validazione.

4 Processi organizzativi

4.1 Processo di coordinamento

4.1.1 Scopo del processo

Lo scopo di questo processo è di illustrare le norme per la produzione del documento *PianoDiProgetto_v4.0.0*, che conterrà tutte le informazioni per gestire e pianificare i ruoli e le rispettive attività.

4.1.2 Descrizione

In questa sezione del documento dovranno essere redatte tutte le norme utilizzate dal team per quanto riguarda l'organizzazione del lavoro tra i vari membri del gruppo.

4.1.3 Comunicazioni

4.1.3.1 Comunicazioni interne Per gestire le comunicazini interne si è riscontrata la necessità di distinguere la modalità di presentazione delle informazioni in:

- **Comunicazione formale**: questa modalità viene usata per le discussioni ufficiali ed inerenti a gestione e sviluppo del progetto;
- **Comunicazione informale**: questa modalità viene usata per scambiare informazioni non ufficiali e discutere riguardo le attività di progetto.

Si è deciso di utilizzare i seguenti strumenti:

- ***Telegram***$_{\mathrm{G}}$;
- ***Slack***$_{\mathrm{G}}$.

4.1.3.2 Comunicazioni esterne

- **Email**: è stato creato l'indirizzo di posta elettronica: **sonsofswe.swe@gmail.com**

  L'utilizzo di tale casella di posta è affidato unicamente al *Responsabile di Progetto*, per la gestione delle relazioni con il committente ed il proponente del progetto.

  Le email dovranno avere la seguente forma:

  - **Oggetto**: l'oggetto dev'essere chiaro e conciso, per riconoscere e distinguere facilmente tra loro le email;
  - **Apertura**: ogni email deve iniziare con un aggettivo di circostanza seguito dal titolo e dal nome del destinatario, oppure con un saluto formale e terminare con una virgola;
  - **Corpo**: nel corpo, che deve essere breve ed esaustivo, verranno spiegate le ragioni per cui si sta scrivendo l'email;
  - **Allegati**: è possibile l'invio di allegati, su richiesta del proponente o del committente;
  - **Chiusura**: la chiusura deve essere separata dal corpo con il doppio ritorno a capo e prevede un congedo formale e la firma del mittente.
- **Slack**: verrà utilizzato, su richiesta del committente, per lo scambio di informazioni in maniera ufficiosa.

4.1.4 Riunioni

4.1.4.1 Obiettivi

Le riunioni sono di fondamentale importanza per la gestione di un progetto. È indispensabile che i membri del team abbiano modo di confrontarsi tra loro al fine di risolvere i problemi esistenti e generare nuove idee. Una riunione è inoltre un ottimo mezzo per creare armonia e consolidare i rapporti all'interno del gruppo. Le riunioni con il proponente/i o committente/i avranno invece lo scopo di condividere gli obiettivi raggiunti e di confrontarsi nel caso in cui si presenti la necessità di colmare lacune.

4.1.4.2 Riunioni interne

Per un produttivo svolgimento delle riunioni interne, verranno rispettati i seguenti ruoli:

- **Moderatore**: il *Responsabile di Progetto* ha il dovere di convocare il team alle riunioni interne quando necessario per un confronto tra i vari membri; durante gli incontri è richiesto che tutti i membri siano presenti, salvo eccezioni precedentemente segnalate. Il *Responsabile di Progetto* avrà quindi l'obbligo di seguire l'ordine del giorno riguardo agli argomenti da affrontare e di trovare un consenso unanime riguardo le decisioni da prendere. Per svolgere il prorio compito in maniera adeguata, il moderatore è tenuto ad avere un atteggiamento autorevole, flessibile e diligente;
- **Segretario**$_G$: il *Segretario* ha il compito di stendere una prima minuta dell'incontro, controllare che venga seguito punto per punto l'ordine del giorno e redigere la versione finale del *verbale*$_G$. Conclusa una riunione, il *Segretario* avrà inoltre il compito di fornire una copia del verbale ad ogni membro del team, comprendente un resoconto delle decisioni prese e degli obiettivi prefissati;
- **Partecipanti**: i partecipanti sono tenuti a presenziare puntualmente qualora venga convocata una riunione da parte del *Responsabile di Progetto*. Nel caso in cui un membro sia impossibilitato a partecipare è invitato a comunicarlo tempestivamente al *Responsabile di Progetto*, al fine di accordarsi sul come procedere. È richiesto ai partecipanti di tenere un comportamento diligente e responsabile per un corretto svolgimento dell'assemblea.

4.1.4.2.1 Descrizione

Le riunioni dovranno avere cadenza settimanale, per fare il punto della situazione riguardo gli obettivi prefissati e le difficoltà incontrate. Il *Responsabile di Progetto*, dovrà informare i vari membri della riunione tramite email e questi saranno tenuti a rispondere confermando la presenza o motivando l'assenza. Una riunione verrà effettivamente considerata valida solo nel caso in cui siano presenti almeno la metà dei membri convocati. Nel caso contrario, sarà proposta una seconda data all'interno della settimana per ripetere e validare l'incontro. Le riunioni dovranno concentrarsi su quattro punti cardine:

- Informare i membri riguardo le novità;
- Valutare il lavoro precedentemente fatto;
- Prendere decisioni collettive riguardo eventuali problematiche;
- Progettare il percorso per raggiungere gli obiettivi prefissati entro le tempistiche stabilite.

Al termine di ogni riunione verrà quindi redatto il corrispondente verbale ed inviato per mail a tutti i membri del team.

4.1.4.3 Riunioni esterne

4.1.4.3.1 Descrizione Le riunioni con il proponente hanno la funzione di supportare il gruppo di progetto e di consolidare il rapporto tra entrambe le parti. Durante le riunioni, il gruppo avrà l'impegno di condividere gli obiettivi raggiunti e le eventuali problematiche incontrate. Il *Responsabile di Progetto* avrà il dovere di convocare le riunioni quando necessario, anche confrontandosi in base alle esigenze dei membri del team. Il segretario scelto invece avrà il compito di verbalizzare quanto emerso dalla discussione, e le possibili richieste di cambiamento e/o correzione emesse dal proponente.

4.2 Processo di pianificazione

4.2.1 Descrizione

Lo sviluppo di un progetto prevede la cooperazione di diversi ruoli. Per una comprensione unanime del progetto, ogni membro del team dovrà esercitare obbligatoriamente tutti i ruoli previsti, durante periodi temporali differenti, che saranno di seguito elencati. Nel caso in cui sorga la necessità, un membro potrà ricoprire più di un ruolo contemporaneamente, ma solo nel caso in cui non si tratti di redattore e verificatore. Questo caso in particolare è considerato un controsenso, in quanto la figura del verificatore risulterebbe corrotta nell'analisi dei propri documenti. Inoltre, ogni ruolo avrà incarichi diversi, i quali verranno gestiti ed assegnati dal *Responsabile di Progetto* con l'ausilio di opportuni strumenti, in modo da garantire monitoraggio, controllo e revisione del progetto per tutta la sua durata.

4.2.2 Ruoli

4.2.2.1 Responsabile Il *Responsabile di Progetto* è colui che detiene la responsabilità sul lavoro svolto dal team, mantiene i contatti esterni e presenta al committente il progetto finale. Egli possiede il potere decisionale e si fa carico dei seguenti oneri:

- Pianificazione, coordinamento e controllo delle attività;
- Gestione e controllo delle risorse;
- Analisi e gestione dei rischi;
- Approvazione della documentazione;
- Contatti con gli enti esterni.

Di conseguenza il *Responsabile di Progetto* ha il compito di assicurarsi che le attività di verifica e validazione vengano svolte in riferimento alle *NormeDiProgetto_v4.0.0*, di redigere l'$Organigramma_G$, di garantire che vengano rispettati i ruoli assegnati all'interno del *PianoDiProgetto_v4.0.0*, e di garantire che non vi siano conflitti tra redattori e verificatori.

4.2.2.2 Analista L'*Analista* si occupa di capire il problema da affrontare, ascoltando le richieste del committente; è un individuo esperto, in grado di capire il dominio e la complessità del problema. Le sue mansioni principali sono:

- Analizzare le qualità e i servizi che dovrà offrire il prodotto finale;
- Valutare la fattibilità del progetto, riportando tali valutazioni nel *StudioDiFattibilità_v1.0.0*;
- Redigere l'*AnalisiDeiRequisiti_v4.0.0*, in cui verranno indicati tutti i requisiti del progetto individuati.

4.2.2.3 Amministratore L'*Amministratore* è responsabile della gestione dell'ambiente di lavoro; deve offrire al gruppo più facilitazioni e automazioni possibili al fine di incrementare l'operatività e l'efficienza. I suoi doveri sono:

- Gestione della documentazione di progetto;
- Controllo di versioni e configurazioni;
- Ricerca e gestione di strumenti di supporto che facilitino l'operato del gruppo;
- Redazione delle *NormeDiProgetto_v4.0.0* e supporto alla redazione del *PianoDiProgetto_v4.0.0*.

4.2.2.4 Progettista Il *Progettista* ha il compito di gestire la progettazione vera e propria, sfruttando le sue competenze e conoscenze in ambiti tecnico e tecnologico; il suo ruolo è direttamente collegato a quello dell'*Analista*, in quanto deve capire e spiegare come risolvere i problemi identificati precedentemente dagli *Analisti*. Ha il compito di:

- Influenzare le scelte tecniche e tecnologiche, per:
  - Orientare il gruppo all'utilizzo di strumenti e servizi il più possibile efficienti;
  - Effettuare scelte progettuali atte a garantire la manutenibilità e la modularità del prodotto finale.
- Sviluppare l'architettura del prodotto software e i relativi documenti informativi e redigere la parte programmatica del *PianoDiQualifica_v4.0.0*.

4.2.2.5 Programmatore Il *Programmatore* avrà il compito di codificare e mantenere il codice prodotto, implementando le soluzioni proposte dal *Progettista*: deve perciò avere alte competenze tecniche. I suoi compiti sono:

- Implementare in maniera rigorosa quanto richiesto dai *Progettisti*;
- Implementare componenti aggiuntive allo scopo di creare strumenti di test e verifica del prodotto;
- Redigere eventuali *ManualeUtente_v1.0.0* e *ManualeSviluppatore_v1.0.0*.

4.2.2.6 Verificatore Il *Verificatore* sarà responsabile della verifica, vale a dire del continuo controllo del prodotto e della documentazione: sarà attivo per tutta la durata del progetto ed agirà sfruttando le proprie capacità di giudizio, esperienza e competenza. I suoi compiti sono:

- Accertarsi del rispetto delle *NormeDiProgetto_v4.0.0* e della conformità rispetto al *PianoDiQualifica_v4.0.0*;
- Redigere il *PianoDiQualifica_v4.0.0*.

4.2.3 Ticketing

Al fine di permettere una migliore gestione del lavoro interno al gruppo, verrà utilizzato lo strumento **Asana**, utile a creare ed assegnare $Task_{G}$; in questo modo il *Responsabile di Progetto* sarà in grado di tenere costantemente sotto controllo l'avanzamento delle attività di progetto. Complementariamente verrà usato **InstaGantt**, strumento che permette di creare diagrammi di Gantt basandosi su ciò che viene dichiarato in Asana.

4.2.3.1 Procedura di assegnazione

L'assegnazione di un Task coinciderà con la sua creazione e si basa sulla seguente sequenza di passi:

- Assegnazione di un titolo al task;
- Assegnazione del task stesso al/ai membro/i designato/i;
- Aggiunta di una breve ma concisa descrizione del compito e dei suoi obiettivi finali;
- Inserimento delle date di inizio e fine;
- Impostazione dello stato del ticket ad "Aperto".

4.2.3.2 Possibile stato di un ticket

Un ticket può essere nei seguenti stati:

- Aperto;
- In elaborazione;

- Sospeso;
- Completato/Risolto.

4.3 Processo dell'infrastruttura

4.3.1 Ambienti di sviluppo

Ogni componente del gruppo avrà la possibilità di utilizzare il sistema operativo che più gli aggrada, a patto che i servizi offerti siano gli stessi. In particolare, verranno utilizzati:

- Windows 10 Pro x64;
- Windows 10 Home x64;
- Windows 10 Education x64;
- Ubuntu 17.04 x64;
- Ubuntu Mate 16.04 x64;
- OSX El Capitan versione 10.11.6.

4.3.2 Strumenti

Gli strumenti utilizzati durante la fase dei processi organizzativi sono:

- **Telegram** [9]
  Telegram è un servizio di messaggistica istantanea multipiattaforma;
- **Slack** [10]
  Slack è un servizio di comunicazione multipiattaforma pensato per facilitare il lavoro di gruppo tramite canali distinti. Fornisce inoltre la possibiltà di integrare servizi esterni, come Github e Google Drive;
- **Google Drive** [11]
  Google Drive è un servizio di memorizzazione ed archiviazione online fornito da Google basato sul *Cloud*$_G$. Esso verrà utilizzato dal gruppo per lo scambio di documenti ed informazioni di supporto allo sviluppo del progetto ma non soggetti al versionamento;
- **Fogli Google** [12]
  Fogli Google è un servizio offerto da Google per la creazione di fogli di lavoro online modificabili in contemporanea da chiunque ne abbia accesso. Il gruppo utilizzerà questo strumento per il rendiconto delle ore di lavoro;
- **Github** [13]
  Github è un' implementazione dello strumento di controllo di versione Git ed offre un servizio di *hosting*$_G$ per progetti software. Il gruppo utilizzerà un repository comune denominato *Marvin* avente come proprietario il profilo *SOS-SonsOfSwe* le cui credenziali, saranno rese note a tutti i componenti; ognuno potrà inoltre apportare le proprie modifiche direttamente dal proprio account personale;
- **Asana** [14]
  Asana è un servizio web utile a migliorare la collaborazione all'interno di un gruppo di lavoro, dando la possibilità di gestire i task di ogni componente del team online. In esso è possibile:
  - Suddividere il progetto sezioni, in modo da dividere gli ambiti di lavoro;

[9] https://telegram.org/
[10] https://slack.com/
[11] https://www.google.com/drive/
[12] https://www.google.it/intl/it/sheets/about/
[13] https://github.com/
[14] https://asana.com/

- Suddividere le sezioni in task veri e propri;
- Ipostare per ogni task scadenze diverse;
- Impostare dipendenze tra le parti, in modo che un task non possa essere completato se uno o più task, da cui *dipende*, non sono stati completati in precedenza;
- Suddividere i task in sotto-task, qualora il compito risulti troppo complesso e necessiti dunque un'ulteriore modularizzazione;
- Avere un calendario delle scadenze sempre aggiornato;
- Avere una completa visione della distribuzione del carico di lavoro all'interno del team.

Per gruppi di studenti, Asana offre la propria versione premium gratuitamente.