> CommandlineTeam _

Commandline Team - Progetto "TuTourSelf"

Norme di Progetto

Versione	3.0.0
Approvazione	Alberto Battistini
Redazione	Daniele Penazzo Michele Tagliabue Giulia Corò
Verifica	Marco Giollo Nicola Agostini
Stato	Approvato
Uso	Interno
Destinato a	Commandline Team Prof. Tullio Vardanega Prof. Riccardo Cardin

Descrizione
Questo documento descrive regole, strumenti e convenzioni adottate da Commandline Team durante la realizzazione del progetto "TuTourSelf".

commandlineteam@gmail.com

Contenuto

Lista delle Immagini

Lista delle Tabelle

Changelog

Versione	Data	Nominativo	Ruolo	Descrizione
3.0.0	2018-05-07	Alberto Battistini	Responsabile	Approvazione documento
2.1.0	2018-05-06	Nicola Agostini	Verificatore	Verifica aggiunte §1.1 §3.4.4.1
2.0.2	2018-03-31	Giulia Corò	Amministratore	Espanse metriche per il software §3.4.4.1
2.0.1	2018-03-21	Giulia Corò	Amministratore	Fix §1.1 e aggiunti range di accettazione ed ottimali metriche in §3.4.4.1
2.0.0	2018-03-11	Giulia Corò	Responsabile	Approvazione
1.2.0	2018-03-10	Giovanni Motterle	Verificatore	Verifica sezioni rimanenti
1.1.1	2018-03-10	Giulia Corò	Responsabile	Aggiunti alcuni grafici ed immagini al documento
1.1.0	2018-03-08	Nicola Agostini	Verificatore	Verifica aggiunte dopo versione 1.0.0: §3.3, §3.2.4.1, §3.2, §3.4.4.1, §3.4.4.2, §4.2
1.0.7	2018-03-08	Daniele Penazzo	Amministratore	Inserimento §3.2.4.1 e anomalie di codifica
1.0.6	2018-03-01	Daniele Penazzo	Amministratore	Fix 2.1.3, 2.2.4.2, 2.2.4.3
1.0.5	2018-02-23	Daniele Penazzo	Amministratore	Inserimento §4.2
1.0.4	2018-02-22	Giulia Corò	Responsabile	Aggiunta sezione §3.2, ampliate metriche processi in §3.4.4.1, aggiunta checklist inspection in §3.4.4.2.
1.0.3	2018-02-20	Daniele Penazzo	Amministratore	Inserimento nuovi riferimenti informativi, correzione norme riguardanti AR
1.0.2	2018-02-17	Giulia Corò	Responsabile	Aggiunta sezione §3.3.
1.0.1	2018-02-16	Giulia Corò	Responsabile	Fix struttura documento ed introduzione
1.0.0	2018-01-10	Michele Tagliabue	Responsabile	Approvazione
0.2.0	2018-01-02	Alberto Battistini, Marco Giollo	Verificatori	Verifica documento
0.1.2	2017-12-30	Michele Tagliabue	Responsabile	Tolto riferimento a licenza open source, sostituito gitlab con trello in §4.1.6.3
0.1.1	2017-12-27	Michele Tagliabue	Responsabile	Aggiunta sezione relativa a Slack §4.1.6.5
0.1.0	2017-12-23	Marco Giollo	Verificatore	Verifica del documento
0.0.19	2017-12-15	Giulia Corò	Amministratore	Completata sezione "Processi di Supporto" §3
0.0.18	2017-12-11	Giulia Corò	Amministratore	Avanzamento stesura "Processi di Supporto" §3
0.0.17	2017-12-10	Michele Tagliabue	Amministratore	Completata prima stesura sezione "Processi Organizzativi" §4
0.0.16	2017-12-10	Michele Tagliabue	Amministratore	Avanzamento stesura "Processi Organizzativi" §4
0.0.15	2017-12-08	Giulia Corò	Amministratore	Avanzamento stesura "Processi di Supporto" §3
0.0.14	2017-12-07	Michele Tagliabue	Amministratore	Avanzamento stesura "processi organizzativi" §4
0.0.13	2017-12-07	Daniele Penazzo	Responsabile	Fine dei processi primari §2
0.0.12	2017-12-05	Daniele Penazzo	Responsabile	Avanzamenti nei processi primari §2.
0.0.11	2017-12-05	Giulia Corò	Amministratore	Avanzamento stesura "Processi di Supporto" §3
0.0.10	2017-12-03	Michele Tagliabue	Amministratore	Avanzamento stesura "Processi organizzativi" §4
0.0.9	2017-12-02	Giulia Corò	Amministratore	Avanzamento stesura "Processi di Supporto" §3
0.0.8	2017-12-01	Daniele Penazzo	Responsabile	Aggiornamento, correzione e completamento sezione 'Processi Primari' §2
0.0.7	2017-11-29	Giulia Corò	Amministratore	Inizio stesura sezione "Processi di Supporto" §3

0.0.6	2017-11-28	Michele Tagliabue	Amministratore	Iniziata stesura "Processi organizzativi" §4.
0.0.5	2017-11-24	Daniele Penazzo	Responsabile	Scrittura attività "Studio di Fattibilità"
0.0.4	2017-11-23	Michele Tagliabue	Amministratore	Aggiunta sezione glossario
0.0.3	2017-11-23	Michele Tagliabue	Amministratore	Scrittura introduzione, scopo del documento (§1.1) e del prodotto (§1.2)
0.0.2	2017-11-23	Daniele Penazzo	Responsabile	Inserimento riferimenti normativi ed informativi
0.0.1	2017-11-22	Daniele Penazzo	Responsabile	Creazione struttura e template

Tabella 1: Changelog di questo documento

1 Introduzione

1.1 Scopo del documento

Questo documento si propone di definire delle regole che i membri del gruppo *CommandLine Team* sono tenuti a rispettare durante tutto lo svolgimento del progetto *TuTourSelf*, al fine di garantire quanto più possibile l'uniformità del materiale prodotto.
Le norme presenti in questo documento verranno prodotte incrementalmente, al progressivo maturare delle esigenze di progetto e delle attività di progetto: il documento allo stato corrente non è quindi da considerarsi completo, poiché verranno prima normati gli aspetti più impellenti e ricorrenti, e solo in seguito quelli che interverranno più avanti, sempre garantendo che ogni attività da svolgere sia stata precedentemente normata. Il documento sarà perciò aggiornato e sottoposto a più revisioni.

1.2 Scopo del prodotto

Il capitolato si prefigge di creare una piattaforma per facilitare comunicazione ed eventuale successivo accordo$_G$ tra artisti indipendenti e locali.
In particolare, il prodotto si prefigge di:

- Creare un database centralizzato di artisti/gruppi, che possa inglobare collegamenti con eventuali altri profili di ciascun artista/gruppo (es. sito web personale, Youtube, Bandcamp, Pinterest, varie pagine social);
- Creare un database centralizzato di locali convenzionati, che puntualizzi la tipologia di artista richiesta e la strumentazione messa a disposizione dal locale stesso
- Creare un'applicazione web (che poi in futuro possa essere facilmente trasformata in applicazione mobile) che permetta **facilmente** di mettere in contatto artisti e locali (tenendo conto della compatibilità tra le caratteristiche dell' artista e quelle del locale), consentendo di giungere ad un accordo tra le due parti. L'accordo, tra le altre cose, deve definire la data dell'esibizione, la durata dello show e il compenso concordato;
- Creare un sistema che permetta ad artisti, locali e pubblico di rilasciare feedback in base a parametri prestabiliti (comportamento, professionalità, quantità di pubblico, incasso ecc.);
- Creare un modulo che permetta al gestore del locale di pagare facilmente il cachet all'artista/gruppo.

1.3 Ambiguità e glossario

Nel caso in cui, all'interno dei documenti, vengano utilizzati termini e/o abbreviazioni fraintendibili e/o ambigui, essi saranno marchiati con il pedice$_G$ e inseriti nel glossario.

1.4 Riferimenti

1.4.1 Riferimenti Normativi

- **ISO 8601** - "Data elements and interchange formats – Information interchange – Representation of dates and times"
  `https://en.wikipedia.org/wiki/ISO_8601` (Ultima visita: 2017-11-23);
- **ISO/IEC 12207** - "Systems and Software Engineering - Software life cycle processes"
  `https://en.wikipedia.org/wiki/ISO/IEC_12207` (Ultima visita: 2017-11-23);
- **Testo del Capitolato**
  `http://www.math.unipd.it/~tullio/IS-1/2017/Progetto/C8.pdf` (Ultima visita: 2017-11-23).

1.4.2 Riferimenti Informativi

- Guide e documentazione LaTeX su ShareLatex
  `https://www.sharelatex.com/learn/` (Ultima visita: 2017-11-23);
- Guide e documentazione LaTeX su WikiBooks
  `https://en.wikibooks.org/wiki/LaTeX` (Ultima visita: 2017-11-23);

- Slides Presentazione del Capitolato 8
  `http://www.math.unipd.it/~tullio/IS-1/2017/Progetto/C8p.pdf` (Ultima visita: 2017-11-23);
- Lista di risorse "Awesome" su GitHub
  `https://github.com/sindresorhus/awesome`;
- Corsi interattivi gratuiti di JavaScript e ReactJS - Codecademy
  `https://www.codecademy.com/catalog/language/javascript`;
- Libri offerti dall'iniziativa "Free Learning" di Packt
  `https://www.packtpub.com/packt/offers/free-learning/` (Ultima Visita: 2017-12-05)
  - "Mastering GIT" di Jakub Narębski - Offerto gratuitamente il 2017-11-05;
  - "Mastering React" di Adam Horton e Ryan Vice - Offerto gratuitamente il 2017-11-14;
  - "Node.js 6.x Blueprints" di Fernando Monteiro - Offerto gratuitamente il 2017-12-12-15;
  - "Node.js Design Patterns - Second Edition" di Mario Casciaro e Luciano Mammino - Offerto gratuitamente il 2017-11-24;
  - "Mastering React Native" di Eric Masiello e Jacob Friedmann - Offerto gratuitamente il 2017-11-28;
  - "AWS Administration - The Definitive Guide" di Yohan Wadia - Offerto Gratuitamente il 2017-12-01.
- Libri offerti dall'iniziativa "Free Ebook" di Packt
  - "Instant MongoDB" di Amol Nayak
    `https://www.packtpub.com/packt/free-ebook/mongoDB-starter-guide`;
  - "What you need to know about node.js" di Bruno Joseph Dmello
    `https://www.packtpub.com/packt/free-ebook/what-you-need-know-about-nodejs`;
- *Guida agli script Automatizzati v1.0.0.*
- Motore di ricerca di alternative software
  `https://alternativeto.org`

2 Processi Primari

2.1 Fornitura

2.1.1 Scopo

In questa sezione vengono trattate le norme che i membri di Commandline Team sono tenuti a rispettare al fine di proporsi e divenire fornitori nei confronti della proponente TuTourSelf S.r.l. e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin nell'ambito della progettazione, sviluppo e consegna del prodotto TuTourSelf.

2.1.2 Descrizione

Durante l'intero progetto si vuole instaurare un costante rapporto con TuTourSelf S.r.l. a fine di:

- Determinare i bisogni del proponente e gli aspetti chiave per soddisfarli;
- Stabilire scelte riguardanti definizione e realizzazione del prodotto;
- Fare scelte volte alla definizione e miglior esecuzione dei processi;
- Fare una stima dei costi;
- Accordarsi sulla qualifica di prodotto;
- Studiare e determinare nuove opportunità di miglioramento dell'idea di prodotto.

2.1.3 Materiale Fornito

Di seguito si elencano i documenti ed il materiale forniti da parte di Commandline Team alla proponente TuTourSelf S.r.l. ed ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin, in modo da assicurare la massima trasparenza possibile per quanto concerne le attività di:

Pianificazione, consegna e completamento del progetto: descritte all'interno del *Piano di Progetto v3.0.0*;

Analisi: l'analisi dei casi d'uso e dei requisiti effettuate dal gruppo sono contenute all'interno del documento *Analisi dei Requisiti v3.0.0*;

Progettazione dell'architettura: una panoramica generale, ad alto livello, dell'applicazione e delle tecnologie adottate è attuata nelle *Proof of Concept*$_G$ della *Technology Baseline*$_G$;

Progettazione di Dettaglio: l'insieme di classi, metodi, attributi e scelte implementative è descritto all'interno del documento tecnico di *Product Baseline*$_G$;

Verifica e Validazione: Le modalità di verifica e validazione del prodotto sono descritte all'interno del *Piano di Qualifica v3.0.0*;

Garanzia della qualità dei processi e del prodotto: Le modalità con cui si garantisce qualità dei processi e del prodotto sono definite nel sopra citato *Piano di Qualifica v3.0.0*.

2.1.4 Attività

2.1.4.1 Studio di Fattibilità

L'organizzazione di riunioni tra i membri del gruppo è compito del *Responsabile*$_G$ di progetto, queste riunioni avranno il fine di consentire lo scambio di opinioni sui capitolati proposti. Il documento sarà redatto dall'*Analista*$_G$ sulla base dei seguenti punti:

Dominio applicativo e tecnologico: si valuta il capitolato considerandone l'obiettivo finale, le tecnologie e le conoscenze richieste riguardo a tali tecnologie. Si valutano inoltre anche eventuali esperienze pregresse con problematiche simili a quelle proposte;

Rapporto tra costi e benefici: si analizzano il numero di requisiti obbligatori, il costo rispetto ai risultati previsti, oltre all'interesse generale dei membri nei confronti delle tematiche proposte dal capitolato;

Identificazione dei rischi: si analizzano ed identificano i punti critici della realizzazione, come mancanza di conoscenze adeguate o problematiche nell'individuazione dei requisiti. Sono, inoltre, analizzate possibili problematiche che possono sorgere in corso d'opera.

Questa attività è culminata nella stesura del documento *Studio Di Fattibilità v1.0.0*

2.1.4.2 Piano di Progetto

Il *Responsabile*, con l'aiuto degli *Amministratori*, dovrà occuparsi di redigere un piano da seguire nella realizzazione del progetto. Tale documento conterrà:

Analisi dei Rischi: dove si analizzano in dettaglio le criticità ed i rischi che potrebbero insorgere nello svilupparsi del progetto, i modi per affrontarli, stimando probabilità ed impatto ad essi associato;

Pianificazione: dove si pianificano le attività da svolgere nel corso del progetto, fornendo deadline$_G$ ben precise;

Stima preventiva e consuntivo: sulla base di quanto pianificato, si va a stimare la quantità di lavoro necessaria per portare a termine ogni fase, proponendo così una stima preventiva del costo totale del progetto.
Alla fine di ogni attività si redigerà un consuntivo atto a tracciare l'andamento reale rispetto alle stime fatte.

2.1.4.3 Piano di Qualifica

I *Verificatori*$_G$ dovranno definire una strategia da adottare per la verifica$_G$ e la validazione$_G$ del materiale prodotto dal gruppo. Questa strategia sarà descritta nel *Piano di Qualifica v3.0.0*, che avrà il seguente contenuto:

Visione Generale: dove si vanno a stabilire responsabilità, organizzazione e risorse coinvolte nei processi di verifica e validazione;

Obbiettivi: definizione degli obbiettivi di qualità di processo e prodotto (documenti e software);

Metriche: ogni metrica è associata ad un obbiettivo e permette di ottenere una valutazione quantitativa della qualità;

Resoconto delle attività di verifica: dove alla fine di ogni attività sono riportate le metriche calcolate ed un resoconto sulla verifica di tale attività.

Piano di collaudo: dove si definiscono in dettaglio i metodi di collaudo del prodotto realizzato;

2.1.4.4 Collaudo e consegna del progetto

Con l'avvicinarsi della data di collaudo, si cercherà quanto più possibile di instaurare un rapporto costante con i Referenti della proponente TuTourSelf S.r.l., ai quali saranno sottoposti prototipi del prodotto, con il fine di ricevere feedback migliorativi sullo stesso.
La consegna avverrà in luogo e data da decidersi.
Verranno forniti alla Proponente TuTourSelf S.r.l. ed ai Committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin:

- Codice Sorgente;
- Documentazione relativa al prodotto, come specificato in § 2.1.3, a cui si aggiungono:
  - *Glossario v3.0.0*: Allo scopo di facilitare la comprensione degli altri documenti;
  - *Manuale Utente (da redarre)*: che illustra il processo di installazione e le funzionalità del prodotto;
  - *Manuale dello Sviluppatore (da redarre)*: allo scopo di semplificare la contribuzione al progetto.

2.1.4.5 Completamento del Progetto

Salvo diversi accordi, a seguito della consegna del prodotto e del relativo collaudo, il progetto si considera concluso. In caso di necessità si continuerà a fornire supporto alla proponente TuTourSelf S.r.l. circa il prodotto sviluppato, e l'applicazione continuerà ad essere sviluppata in accordo con la proponente.

2.2 Sviluppo

2.2.1 Scopo

Questo processo contiene tutte le attività e i compiti svolti durante la produzione del prodotto software finale.

2.2.2 Obiettivi

Allo scopo di implementare il processo di sviluppo in modo corretto, Commandline Team ha i seguenti obiettivi:

- Realizzare un prodotto software conforme a quanto richiesto dal proponente;
- Fare in modo che tale prodotto soddisfi i test di verifica;
- Avere un prodotto finale che soddisfi i test di validazione;
- Fissare i vincoli tecnologici;
- Fissare i vincoli di design;
- Fissare gli obiettivi di sviluppo.

2.2.3 Descrizione

Il processo di sviluppo si svolgerà secondo lo standard ISO/IEC 12207-1995, quindi le attività svolte saranno:

- Studio del dominio, del capitolato ed analisi dei requisiti;
- Progettazione del sistema software;
- Codifica.

2.2.4 Attività

2.2.4.1 Analisi dei Requisiti

Scopo È compito degli Analisti redigere il documento di Analisi dei Requisiti, al fine di:

- Individuare ed elencare, senza ambiguità, le funzionalità ed i requisiti concordati col proponente;
- Fornire ai Progettisti delle linee guida precise ed affidabili;
- Fornire ai Verificatori dei riferimenti su cui basare l'analisi statica della progettazione;
- Fare una stima dei costi;
- Semplificare le revisioni del codice.

Il risultato di quest'attività è un accordo vincolante tra il proponente TuTourSelf S.r.l. ed il gruppo Commandline Team riguardo le funzionalità ed i requisiti del prodotto finale, descritto nel documento *Analisi dei Requisiti v3.0.0*e che rispettano le norme introdotte in § 3.1.7.3.

Descrizione Durante l'analisi dei requisiti, il gruppo analizza le fonti e consulta il proponente allo scopo di individuare i requisiti ed i casi d'uso.

Classificazione dei Requisiti È compito degli analisti redigere una lista di requisiti che sono emersi durante la fase di analisi.
I requisiti possono essere ricavati da varie fonti, tra le quali possiamo trovare:

Capitolato Il requisito è emerso da un'analisi del documento fornito dalla Proponente TuTourSelf S.r.l. e dalla presentazione mostrata;

Verbali Esterni Il requisito è stato discusso e contrattato durante un incontro con il Referente.

Casi D'uso Il requisito è emerso dall'analisi di uno o più casi d'uso;

Studio del Dominio Il requisito è emerso dallo studio del dominio applicativo;

Tracciabilità Interna Il requisito è emerso da discussioni ed incontri tra gli Analisti del gruppo.

Inoltre I requisiti dovranno essere classificati secondo la seguente notazione:

R[Tipo][Importanza]-[Codice Identificativo]

Dove:

Tipo Rappresenta la tipologia del requisito, può assumere uno dei seguenti valori:

- **F** Requisito di Funzionalità;
- **P** Requisito Prestazionale;
- **Q** Requisito di Qualità.
- **V** Requisito di Vincolo.

Importanza Indica il livello di importanza associato al requisito, come indicato di seguito:

- **0** Requisito Critico - il cui soddisfacimento è assolutamente necessario per garantire le funzioni base del sistema;
- **1** Requisito Desiderabile - un requisito il cui soddisfacimento porta ad una maggiore completezza del sistema, ma il mancato soddisfacimento non pregiudica alcuna funzionalità base;
- **2** Requisito Facoltativo - requisito che se soddisfatto renderebbe il sistema più completo, ma può portare ad un aumento dei costi;
- **3** Requisito Aggiuntivo - un requisito che aggiunge funzionalità di contorno al sistema, non legate alle funzionalità base ma comunque in forte relazione con il dominio applicativo.

Codice Identificativo Un identificatore univoco del requisito, nel formato *[Padre].[Figlio].*

Il codice stabilito secondo la convenzione appena vista, una volta associato ad un requisito, è immutabile.
Inoltre ciascun requisito dovrà riportare le fonti da cui è stato ricavato o dedotto, le relazioni di dipendenza con altri requisiti ed una breve descrizione.

ID	Descrizione	Fonti
RF0-1	Il cliente deve essere in grado di accedere alla pagina di un evento	Capitolato UC1
RF0-1.1	Il cliente deve poter rilasciare un feedback per un evento	Capitolato UC2, UC3

Tabella 2: Esempio di Elenco dei requisiti

Classificazione dei Casi D'uso Altro compito appannaggio degli Analisti è l'identificazione dei casi d'uso, elencandoli dal più generale al più dettagliato.
La classificazione dei casi d'uso avviene secondo la seguente convenzione:

UC-[Codice Gerarchia].[Codice Identificativo]*

Dove:

Codice Gerarchia identifica il caso d'uso generico che ha generato il caso d'uso che si sta esaminando. Se non esiste, deve essere tralasciato. Possono essere presenti anche più codici;

Codice Identificativo identifica univocamente il caso d'uso all'interno della propria gerarchia.

Entrambi i codici sono solo numerici.
I casi d'uso sono identificati secondo la seguente struttura:

ID: Il codice del caso d'uso, in accordo con la specifica appena vista;

Titolo: Specifica il titolo del caso d'uso;

Attori Principali: Indica gli attori principali del caso d'uso;

Attori Secondari: Indica gli attori secondari del caso d'uso;

Pre-Condizione: Indica le condizioni che sono considerate vere prima del verificarsi degli eventi del caso d'uso;

Post-Condizione: Indica le condizioni che devono essere vere dopo il verificarsi degli eventi del caso d'uso;

Scenario Principale: Rappresenta, tramite una lista numerata, il flusso degli eventi. Per ogni evento vanno specificati:

- Titolo;
- Descrizione;
- Attori coinvolti nell'evento;
- Casi D'uso generati dall'evento.

Inclusioni Usate per evitare di descrivere lo stesso flusso di eventi molteplici volte, usando un caso a parte per descrivere il comportamento comune;

Estensioni: Casi d'uso che non fanno parte del flusso principale degli eventi.

Alcuni casi d'uso possono essere associati ad un *Diagramma UML2*$_{\text{G}}$ dei casi d'uso, che riporterà lo stesso codice e titolo.

Qualità dei Requisiti La specifica dei requisiti deve avere le seguenti qualità essenziali:

Completezza Ogni funzionalità richiesta al software ed il proprio comportamento rispetto agli input è dettagliatamente specificato;

Correttezza Ogni requisito specificato è realmente richiesto e/o necessario all'utenza del prodotto finale;

Verificabilità È possibile verificare che il sistema vada a realizzare ogni requisito;

Consistenza Non vi è contraddizione tra requisiti

Non Ambiguità Ogni requisito ha un'interpretazione formale univoca;

Modificabilità Lo stile e la struttura dei requisiti sono modificabili preservando consistenza e completezza;

Tracciabilità L'origine dei requisiti è chiara e tale origine può essere referenziata nel futuro.

Tracciamento e Strumenti

- **PragmaDB**: Per il tracciamento dei casi d'uso e dei requisiti sarà usato il software "PragmaDB", installato su una macchina messa a disposizione da un membro del gruppo all'indirizzo `http://tagliabuemichele.homepc.it/PragmaDB/`;
- **StarUML**: Per la realizzazione dei diagrammi UML 2.x sarà usato il software StarUML, reperibile presso `http://staruml.io/`.

2.2.4.2 Progettazione

Scopo Questa attività, dati i requisiti specificati nel documento di *Analisi dei Requisiti v3.0.0*, definisce le caratteristiche essenziali del software richiesto. L'attività di progettazione ha come scopo la realizzazione dell'architettura del sistema, che sarà attuata nei Proof of Concept della Technology Baseline, oltre che descritta nel documento tecnico allegato alla Product Baseline. Questa fase permette inoltre di:

- Garantire la qualità del prodotto sviluppato, perseguendo il principio di *correttezza per costruzione*
- Organizzare e ripartire i compiti di implementazione
- Ottimizzare l'uso di risorse.

Descrizione Precedentemente alla realizzazione dell'architettura si dovranno definire le tecnologie da utilizzare, studiandone approfonditamente vantaggi e criticità, producendo delle Proof of Concept. Durante l'attività di progettazione il gruppo dovrà realizzare l'architettura del sistema rispettando i vincoli stabiliti con il proponente.

Diagrammi UML 2.x Questa attività farà uso di diagrammi UML 2.x, precisamente:

Diagrammi delle classi: Definiscono relazioni, classi, attributi, tipi e metodi; indipendentemente dal linguaggio di programmazione.

Diagramma dei package: Mostrano raggruppamenti di classi con fini collegati e che necessitano di essere riusate assieme.

Diagrammi delle attività: Servono a descrivere interazioni fra oggetti che implementano collettivamente un comportamento, mostrando sequenze di azioni attraverso scelte definite.

Diagrammi di Sequenza: Descrivono il flusso di operazioni di un'attività, mostrandone la logica procedurale

Design Patterns$_G$ Successivamente all'attività di Analisi, sarà compito dei progettisti adottare soluzioni a problemi ricorrenti e riportarle nel documento di *Technology Baseline*.
Ogni Design Pattern dovrà riportare un diagramma che lo illustri, oltre ad una descrizione testuale che specifichi l'utilità all'interno dell'architettura e la sua applicazione.

Test È Compito dei progettisti definire opportuni test. Assieme a tali test dovranno essere previste anche delle classi utili ad individuare anomalie ed errori.
I test progettati dovranno rispettare la nomenclatura specificata nel § 3.4.4.2

Qualità dell'architettura Al fine di portare alla Proponente un prodotto di alta qualità, l'architettura definita dovrà attenersi ai seguenti principi:

Comprensibilità : l'architettura dovrà essere capita dagli stakeholders$_G$, inoltre dovrà essere tracciabile rispetto ai requisiti;

Semplicità : l'architettura dovrà prediligere la semplicità, contenendo solo quanto necessario;

Sicurezza : l'architettura dovrà essere sicura da intrusioni e malfunzionamenti;

Incapsulazione : l'architettura dovrà seguire i principi dell'*information hiding*;

Ridotto Accoppiamento : non vi dovranno essere dipendenze non desiderabili all'interno dell'architettura;

Modularità : l'architettura dovrà essere divisa in parti distinte, senza sovrapposizioni di funzionalità;

Manutenibilità : Le operazioni di manutenzione devono essere possibili in maniera semplice;

Efficienza : l'architettura dovrà soddisfare tutti i requisiti in modo da minimizzare gli sprechi di tempo e spazio;

Riusabilità : l'architettura deve essere strutturata in modo da permettere il riutilizzo di alcune parti;

Robustezza : l'architettura dovrà rimanere operativa di fronte a situazioni erronee impreviste;

Disponibilità : l'architettura deve necessitare di tempi ridotti per la manutenzione, in modo da garantire un servizio quanto più continuo possibile;

Coesione : Le parti dell'architettura dovranno essere raggruppate secondo l'obiettivo a cui concorrono;

Flessibilità : l'architettura dovrà permettere modifiche a costi contenuti, in caso di variazione dei requisiti;

Sufficienza : l'architettura dovrà soddisfare i requisiti definiti nel documento *Analisi dei Requisiti v3.0.0*;

Affidabilità : l'architettura dovrà garantire che i servizi esposti siano sempre disponibili, cioè dovrà svolgere i propri compiti quando usata.

Linee Guida di Progettazione Al fine di implementare le qualità appena descritte, si chiede ai *Progettisti* di seguire le seguenti linee guida:

- Evitare package vuoti, classi e parametri inutilizzati e parametri senza tipo;
- Evitare le dipendenze circolari$_G$;
- Evitare modifiche a servizi offerti da librerie esterne;
- Usare classi astratte quando possibile;
- Usare nomi significativi per classi e per i metodi in esse contenuti;
- Assegnare ad ogni metodo ed attributo la visibilità più stretta che ne permetta il funzionamento.

2.2.4.3 Codifica

Scopo: Questa attività ha come scopo la realizzazione effettiva del prodotto richiesto. In questa fase si concretizza la soluzione attraverso il processo di programmazione, in modo da ottenere il prodotto software finale.

Aspettative: L'obiettivo di questa attività è la creazione di un prodotto software che sia conforme con quanto è stato contrattato con il proponente.

Descrizione: La codifica sarà vincolata dalle indicazioni nell'allegato tecnico di Product Baseline. Inoltre la scrittura del codice sorgente dovrà rispettare gli obiettivi di qualità definiti all'interno del documento *Piano di Qualifica v3.0.0* allo scopo di garantire un prodotto manutenibile e di alta qualità.

Lingua Utilizzata: Nel codice, i commenti ed i nomi di classi e metodi saranno scritti in lingua inglese, in modo da rendere possibile il futuro intervento da parte di programmatori non italiani.

Linguaggi di Codifica Utilizzati: Durante la codifica si faranno uso dei seguenti linguaggi:

JavaScript: Un linguaggio di Scripting lato client molto conosciuto nell'ambito della programmazione web.

Nomenclatura dei Files: I nomi dei file devono essere brevi ma descrittivi, non possono contenere spazi (Che possono essere comunque sostituiti con un carattere underscore$_G$) o caratteri speciali diversi da underscore.
Se il file figura una serie di metodi di utilità indipendenti dalle istanze di classe, questo dovrebbe avere suffisso "util", ad esempio "vectorutil.js". I nomi dei file devono essere scritti tutti in formato *lowercase*$_G$.

Struttura del repository: Per la struttura del repository di codice si rimanda alla sezione §3.2.4.1.

Lunghezza delle linee di codice: Allo scopo di semplificare il confronto di listati di codice affiancati, le linee di codice non dovranno superare i 79 caratteri.
Gli IDE mettono a disposizione nella barra di stato un contatore di "columns", una funzione di spezzamento automatico delle linee di codice oppure evidenziano l'ottantesima colonna.

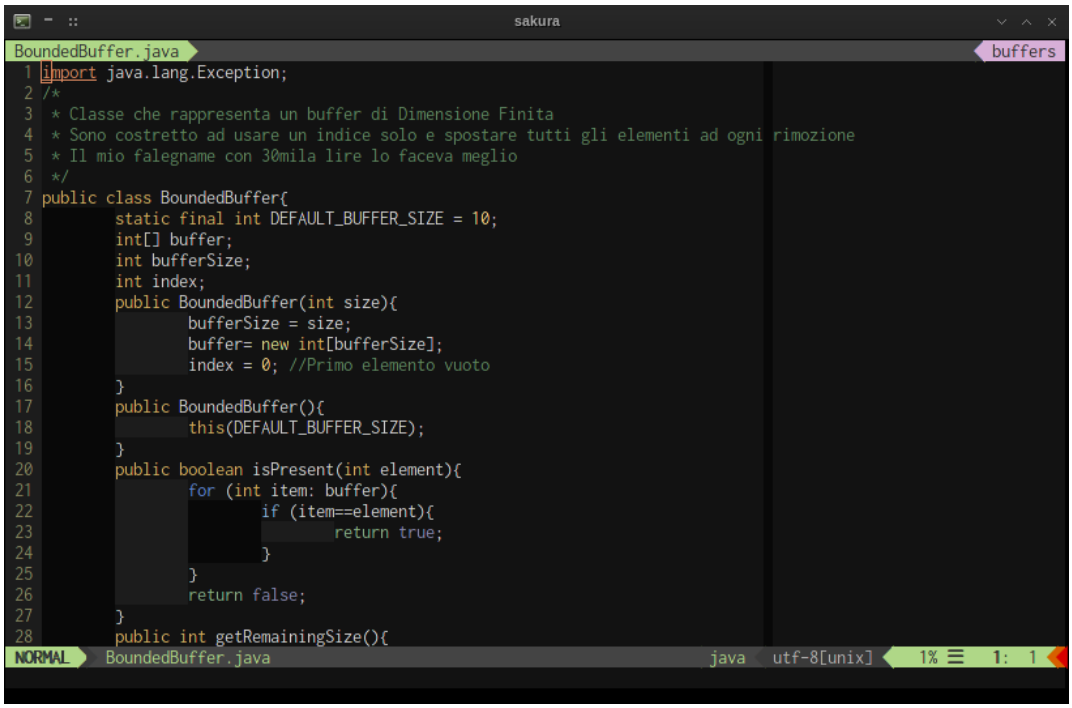

Immagine 1: Esempio di editor che evidenzia l'ottantesima colonna (NeoVim)

Indentazione e parentesi: Il codice dovrà essere indentato ogniqualvolta si viene a generare un nuovo scope, in modo da rendere il codice più leggibile.
In caso di costrutti condizionali molto brevi è possibile usare costrutti inline, purché questo non vada a pregiudicare la leggibilità e la comprensibilità del codice.
Il codice dovrà essere sempre indentato con **quattro** spazi rispetto al livello di indentazione precedente, e non facendo uso di tabulazioni.
I blocchi di codice sono delimitati da parentesi graffe, con la parentesi di apertura scritta in linea, mentre quella di chiusura deve essere scritta a capo dell'ultimo statement$_G$ del blocco.
In caso si debba spezzare una riga di codice in mezzo ad una chiamata a funzione, si dovranno allineare le nuove righe con il primo carattere all'interno delle parentesi.
Ad esempio:

```
String i = someReallyLongExpression(that,
                                    would,
                                    not,
                                    fit)
    someOtherStuff()
```

Se questo non è possibile, la linea dovrà essere indentata di 8 spazi ed i parametri dovranno essere allineati verticalmente:

```
String i = someReallyLongExpression(
        that,
        would,
        not,
```

```
        fit)
    someOtherStuff()
```

Nel caso invece non sia possibile definire le modalità di ritorno a capo, la linea dovrà essere indentata di 8 spazi:

```
String someReallyReallyReallyLongName =
        someReallyLongExpression(that, would,
                                 not, fit)
    someOtherStuff()
```

Spaziature: Le classi dovranno essere inserite ognuna in un file diverso.
Tra due metodi di una classe deve essere presente una riga di codice vuota.
Il primo metodo dovrà essere separato dagli import da due righe di codice vuote.
Tra due metodi non legati ad una classe vi devono essere due righe di codice vuote.
Per rendere più chiaro il codice, si dovranno inserire spazi intorno agli operatori (assegnazione, somma, sottrazione, ...), oltre ad inserire uno spazio dopo l'operatore virgola.

Dimensione e funzionalità di metodi e classi: Le classi dovranno essere quanto più possibile autocontenute e con una singola funzionalità.
Classi e metodi devono essere naturalmente brevi, e non dovrebbero essere artificiosamente accorciate, dato che questo ne limiterebbe la leggibilità.
Se costretto a scegliere tra brevità e leggibilità, il programmatore dovrebbe scegliere la leggibilità.

Nomi di variabili, classi e metodi: Variabili, classi e metodi dovrebbero avere nomi brevi ma chiari, anche al di fuori del contesto del progetto.
Abbreviazioni nei nomi sono consentite, se il minor numero di caratteri scritti migliora la comprensibilità del codice.
Le variabili globali vanno scritte in stile UPPERCASE$_G$ e circondate da due caratteri di sottolineatura (ad esempio __GLOBAL__)
Le costanti vanno scritte in solo stile UPPERCASE, senza caratteri di sottolineatura.
Le variabili di classe vanno scritte in mixedCase$_G$.
I nomi di classe vanno scritti in CamelCase$_G$.
I nomi di metodi vanno scritti in mixedCase.
I nomi ortograficamente errati o non coerenti con il contesto dovrebbero essere evitati.

Intestazioni e commenti: Ogni file contenente del codice deve iniziare con la seguente intestazione:

```
/*
 * FileName: Nome del File
 * Version: Versione del File
 * Type: Tipo del File
 * Date: Data di creazione
 * Authors:
 * - Lista degli autori
 * E-mail: commandlineteam@gmail.com
 *
 * License: licenza usata
 *
 * Description:
 * Breve descrizione del file
 *
 * Changelog:
```

```
 * Date       | Author       | Description
 * -----------+--------------+-----------------
 */
```

Ogni metodo dove essere preceduto da questo commento:

```
/*
 * @method Nome Metodo
 * Descrizione del metodo (Markdown supportato)
 *
 * @param {Tipo} [nome="valore di default"] Descrizione
 * @param {Tipo} [nome] Descrizione
 * @return {Tipo} Descrizione
 */
```

Nel caso il metodo fosse un costruttore, questo avrà nome "constructor", all'interno dell'intestazione, per compatibilità con JSDoc.
Ogni classe deve essere preceduta da questa intestazione:

```
/*
 * @class Nome Classe
 * @extends Nome Classe Padre
 * @abstract (se astratta)
 *
 * Descrizione della classe
 */
```

I commenti inline sono consentiti, in quantità molto limitata. Si dà preferenza ad i commenti presenti su una propria riga.

Annotazioni nei commenti: Allo scopo di facilitare l'identificazione di parti di codice da revisionare e riparare in maniera automatizzata da parte di IDE, si farà uso di appositi tag_G:

TODO: Rappresenta un requisito o una specifica non ancora implementata

FIXME: Rappresenta una parte di codice che necessita di revisione, solitamente a causa di Bug_G

TESTME: Rappresenta una parte di codice non coperta da test automatizzati.

Tali tag devono anche riportare una breve descrizione del problema.
Se il tag deve essere applicato ad una sola riga, è preferibile inserirlo in un commento a sè stante, sopra la riga in oggetto.
Se il tag deve essere applicato ad un intero metodo o classe, questo deve essere riportato all'interno del commento precedente il metodo o la classe in oggetto.

Versionamento: La versione del codice viene inserita all'inizio del file, nell'intestazione e rispetterà il seguente formato:

$$X.Y$$

Dove X rappresenta un avanzamento della versione stabile. Un aumento di questo valore porta l'indice Y a zero.
Y rappresenta una modifica parziale, corrispondente ad una modifica rilevante come ad esempio l'aggiunta di uno statement.

Minificazione del codice: Al fine di mantenere la comprensibilità del codice sorgente, è assolutamente vietato far uso di minificatori di codice durante lo svolgimento del progetto.

Ricorsione: È richiesto di evitare quanto più possibile l'uso di codice ricorsivo, a meno che questo non renda molto più chiaro e comprensibile il codice o comunque che sia fornita una valida ragione nei commenti.

2.2.5 Strumenti

2.2.5.1 JSDoc

Per la generazione automatica della documentazione in stile JavaDoc, sarà usato il software JSDoc, reperibile all'indirizzo `http://usejsdoc.org`.

2.2.5.2 Integrazione Continua$_G$

Per la gestione delle build e dei test automatici, sarà usato il sistema di Integrazione Continua interno a GitLab.

3 Processi di Supporto

3.1 Documentazione

3.1.1 Scopo

Il processo di documentazione include tutti i dettagli su come deve essere redatta la documentazione durante tutto il ciclo di vita del software.

3.1.2 Aspettative

Le aspettative riguardo l'implementazione di questo processo sono:

- Avere una visione precisa della documentazione che va prodotta durante il ciclo di vita del software;
- L'individuazione di una serie di norme che dovranno essere rispettate da tutti i membri del gruppo, al fine di produrre documentazione formale, coerente e valida.

3.1.3 Descrizione

Questa sezione riguarda i processi di documentazione, ed include nel dettaglio tutte le norme che sono state adottate per la stesura, verifica, approvazione e successiva manutenzione della documentazione ufficiale. Tali norme sono tassative, e dovranno essere adottate per tutti i documenti formali redatti dal gruppo *CommandLine Team*.

3.1.4 Procedure

Per la stesura di tutta la documentazione il gruppo ha adottato il linguaggio LaTeX.

3.1.4.1 Approvazione dei documenti

Il *Responsabile di Progetto* è colui che si occupa dell'approvazione dei documenti. Ogni qualvolta sia stata completata la stesura completa di un documento, o di una sua unità, il Responsabile incarica i Verificatori di controllarne il contenuto e la forma. Nel caso i Verificatori trovino degli errori, dovranno seguire la procedura descritta in 3.4.5.2. Al termine dell'attività di verifica, l'approvazione spetterà al responsabile. In caso di non approvazione, il Responsabile dovrà comunicare le motivazioni della sua scelta, esplicitando le modifiche che dovranno essere apportate, ed il ciclo verrà ripetuto fino a che il Responsabile non approvi il documento. Una volta approvato dal Responsabile di Progetto, un documento potrà considerarsi come documento formale.

3.1.5 Classificazione dei documenti

3.1.5.1 Materiale Grezzo

Vengono classificati "materiale grezzo" tutte le versioni dei documenti che non sono state approvate dal Responsabile di Progetto. In quanto tali sono considerate soltanto ad uso interno, e *non* devono essere distribuite all'esterno del gruppo.

3.1.5.2 Documenti Formali

Un documento può considerarsi formale dopo che è stato approvato dal Responsabile di Progetto. Per arrivare a tale stato il documento dovrà quindi aver superato la verifica e la validazione. I documenti formali possono essere:

- **Interni:** documenti riguardanti l'organizzazione ed il way of working$_G$ del gruppo. La consultazione di questi documenti è limitata ai componenti del gruppo.
- **Esterni:** documenti destinati alla consultazione da parte di persone esterne al gruppo. Soltanto i documenti formali possono essere documenti esterni.

3.1.5.3 Verbali

I verbali sono documenti redatti in occasione di incontri interni al gruppo (Verbali Interni), oppure con altre entità esterne (Verbali Esterni), e consistono in una descrizione delle attività svolte durante l'incontro. I verbali non subiscono modifiche successive alla redazione, perciò non prevedono versionamento$_G$. Ogni verbale deve comunque essere sottoposto a verifica, ed approvato dal Responsabile di Progetto. In ogni verbale sarà presente un primo paragrafo *Informazioni Generali*, nel quale dovrà figurare le sezione *Informazioni sull'incontro* con indicate le seguenti informazioni, nel seguente ordine e formato:

- **Luogo:** Città (Provincia), Via, Sede;
- **Data:** YYYY-MM-DD;
- **Ora inizio:** HH:MM (formato 24h);
- **Ora fine:** HH:MM (formato 24h);
- **Durata:** in minuti;
- **Partecipanti del Gruppo**;
- **Partecipanti Esterni**.

In questo primo paragrafo sarà inoltre presente la sezione *Argomenti trattati*, dove saranno elencati in breve tutti gli argomenti che sono stati toccati durante l'incontro. Segue poi un secondo paragrafo *Riunione in dettaglio*, che esplorerà più nel dettaglio le attività svolte. Le sezioni di questo paragrafo potranno variare a seconda del tipo di attività svolte all'incontro. I verbali possono essere:

- **Verbali Interni:** nel caso l'incontro abbia visto la partecipazione di soli membri appartenenti al gruppo;
- **Verbali Esterni:** nel caso siano stati presenti all'incontro uno o più partecipanti esterni. In questo caso sarà cruciale elencare nel dettaglio tutte le questioni emerse dall'incontro frontale con i proponenti.

3.1.6 Struttura della documentazione

3.1.6.1 Nomenclatura e Versionamento dei Documenti

La nomenclatura di ogni documento deve rispettare un formato comune, il più possibile chiaro, affinché vengano identificati facilmente nome e versione del documento. Il gruppo ha scelto di utilizzare la seguente nomenclatura:

Nome Del Documento vX_Y_Z

All'interno del nome, *vX_Y_Z* indica la versione del documento nel seguente modo:

- **X**: è l'indice di versione principale. Inizia da 0 e indica il numero di pubblicazioni del documento, ed è incrementato esclusivamente dal Responsabile di Progetto in seguito alla sua approvazione finale. L'incrememento di tale indice va ad azzerare gli altri indici Y e Z;
- **Y**: indica il numero di verifiche effettuate dopo una approvazione. Inizia da 0 ed è incrementato dai Verificatori. L'incremento di questo indice azzera l'indice Z;
- **Z**: indica il numero di aggiornamenti minori al documento in seguito ad ogni verifica o approvazione. Inizia da 0 e viene incrementato ogni volta che uno degli incaricati alla stesura del documento effettua una modifica.

Ogni modifica al numero di versione del documento sarà riflessa nel *Changelog*, il quale deve essere obbligatoriamente presente alla fine di ogni documento.

3.1.6.2 Template

Il gruppo ha creato un template LaTeX per uniformare la struttura e lo stile di formattazione dei documenti, in modo che i membri del gruppo possano concentrarsi sulla stesura del contenuto piuttosto che sull'aspetto del documento.

3.1.6.3 Struttura del documento

Frontespizio

Il frontespizio di ogni documento è così strutturato:

- **Logo del Gruppo**: centrato orizzontalmente in alto;
- **Gruppo e Progetto**: nome del gruppo e nome del progetto, posti subito sotto al logo e centrati orizzontalmente;
- **Titolo del Documento**: il nome del documento in questione, centrato orizzontalmente e ben visibile;

- **Tabella descrittiva**: posta dopo il titolo, centrata orizzontalmente e contenente le seguenti informazioni:
  - **Versione**: con l'esclusione dei Verbali, che non prevedono versionamento, tutti gli altri documenti avranno indicato il numero di versione corrente;
  - **Approvazione**: nome e cognome del membro del gruppo Responsabile per il documento, che si è quindi occupato dell'approvazione;
  - **Redazione**: nome e cognome dei membri del gruppo che si sono occupati della redazione del documento;
  - **Verifica**: nome dei membri del gruppo che si sono occupati della verifica del documento;
  - **Stato**: stato corrente del documento;
  - **Uso**: uso a cui viene destinato il documento: interno o esterno;
  - **Destinato a**: entità destinatarie del documento.
- **Descrizione**: descrizione molto sintetica del contenuto del documento, centrata orizzontalmente;
- **Email di contatto**: indirizzo di posta elettronica del gruppo, per qualsiasi comunicazione. Posto in fondo alla pagina e centrato orizzontalmente.

Indice dei Contenuti

Ogni documento avrà un indice dei contenuti che ne faciliti la consultazione, posto a seguito della prima pagina. Questo permetterà una lettura non necessariamente sequenziale del documento, bensì ipertestuale$_G$.
La numerazione delle sezioni partirà sempre da 1, e ciascuna sottosezione sarà separata dalla sezione padre tramite un punto, e la numerazione dovrà a sua volta ripartire da 1. Inoltre, in alcuni casi nei quali il documento si prestava ad essere suddiviso in poche ed estese macrosezioni (come ad esempio il documento corrente) si è scelto di catalogare le parti con un indice in numero romano, che comincerà sempre da I.

Contenuto del documento

Il contenuto del documento sarà disposto in ogni pagina rispettando i margini orizzontali e verticali previsti dal template LaTeX. Tutte le pagine, ad eccezione della prima, devono contenere intestazione e piè di pagina.

Intestazioni e piè di pagina

Ad eccezione della prima, tutte le pagine devono contenere intestazione e piè di pagina. L'intestazione è così strutturata:

- Logo del gruppo posto a sinistra;
- Titolo del documento posto a destra.

Il piè di pagina è così strutturato:

- Titolo della sezione presente nella pagina;
- Numero della pagina corrente a destra.

Changelog

Per un corretto versionamento della documentazione, ogni documento dovrà contenere una tabella di Changelog, posta in prima pagina, contenente la storia di tutte le modifiche effettuate. Ogni riga corrisponderà ad una versione nel registro delle modifiche. Il numero di versione è sottoposto a un formalismo preciso, già enunciato in § 3.1.6.1.

3.1.6.4 Norme Stilistiche

Stile del testo

- **Glossario**: ogni parola contenuta nel *Glossario v3.0.0* sarà segnalata, alla sua prima apparizione nel documento, con una G maiuscola a pedice:

  $$\text{Parola}_{\text{G}}$$

- **Grassetto**: viene applicato ai titoli e agli elementi di un elenco puntato che riassumono il contenuto di tale voce;
- **Corsivo**: il corsivo viene utilizzato per:
  - riferimenti ad altri documenti;
  - citazioni;
  - parole particolari poco usate;
  - parole sulle quali si vuole porre enfasi.
- **Maiuscolo**: le uniche parole che è consentito scrivere totalmente in maiuscolo sono gli acronimi.

Elenchi Puntati

Gli elenchi puntati vengono utilizzati per esprimere in modo preciso e chiaro un concetto, evitando così frasi troppo lunghe o discorsive. Ogni voce di un elenco puntato è rappresentata graficamente con un pallino per il primo livello, con una lineetta per il secondo livello, ed un asterisco per il terzo. Ogni voce di un elenco puntato terminerà con un punto e virgola, ad eccezione dell'ultima che terminerà con un punto. Una voce che viene ulteriormente espansa in un altro elenco puntato terminerà invece con i due punti.

- elemento di primo livello espanso:
  - elemento di secondo livello espanso:
    - elemento di terzo livello;
    - ultimo elemento di terzo livello.

Formati Comuni

Per le seguenti tipologie di concetti vengono utilizzati i seguenti formalismi:

- **Date**: per le date si è scelto di usare il seguente formato:

  $$\text{YYYY-MM-DD}$$

  - **YYYY**: dove Y sta per *year*, rappresenta l'anno utilizzando quattro cifre;
  - **MM**: dove M sta per *month*, rappresenta il mese utilizzando due cifre;
  - **DD**: dove D sta per *day*, rappresenta il giorno utilizzando due cifre.
- **Orari**: per gli orari si è scelto di usare il seguente formato:

  $$\text{HH:MM}$$

  - **H**: dove H sta per *hour*, rappresenta l'ora in formato 24 ore, può quindi assumere valori tra 0 e 23;
  - **M**: dove M sta per *minute*, rappresenta i minuti e può assumere valori tra 0 e 59.
- **Durate**: per le durate si è scelto di rappresentarle sempre con numeri interi, in minuti oppure ore (a seconda dell'effettiva quantità);
- **Ruoli di progetto**: ogni nome di ruolo di progetto viene scritto con la lettera iniziale maiuscola;
- **Nomi di documenti**: ogni nome di documento viene scritto in corsivo e con la lettera iniziale di ogni parola in maiuscolo, eccezion fatta per articoli o preposizioni;
- **Nomi propri**: ogni nome proprio di persona andrà chiaramente scritto con le iniziali maiuscole, e nella forma *Nome Cognome*.

Sigle
E' previsto l'utilizzo delle seguenti sigle:

- **RR**: Revisione dei requisiti;
- **RP**: Revisione di progettazione;
- **RQ**: Revisione di qualifica;
- **RA**: Revisione di accettazione;
- **NdP**: Norme di Progetto;
- **SdF**: Studio di Fattibilità;
- **PdQ**: Piano di Qualifica;
- **PdP**: Piano di Progetto;
- **AR**: Analisi dei Requisiti;
- **VI**: Verbale Interno;
- **VE**: Verbale Esterno;

Tabelle
Ogni tabella dovrà essere centrata orizzontalmente nella pagina e dovrà presentare sotto di essa una didascalia, nella quale dovrà comparire il numero della tabella, che sarà incrementale in tutto il documento così da facilitarne la tracciabilità, ed inoltre una breve descrizione del contenuto della tabella stessa.

Immagini
Ogni immagine dovrà essere centrata orizzontalmente nella pagina e dovrà presentare sotto di essa una didascalia, simile a quella usata per le tabelle, con numero identificativo e breve descrizione. L'immagine dovrà avere un distacco netto dal testo sovrastante e sottostante per migliorare la leggibilità dello stesso.
I diagrammi UML saranno inseriti nei documenti come immagini.

3.1.7 Documenti da consegnare

Di seguito sono elencati i documenti che il gruppo *CommandLine Team* si impegna a consegnare in occasione della RR.

3.1.7.1 Studio di fattibilità

- **Classificazione**: Interno;
- **Distribuzione**: Gruppo e committente;
- **Contenuto**: Lo *Studio Di Fattibilità v1.0.0* contiene l'analisi di tutti i capitolati svolta dal gruppo *CommandLine Team*. Verranno in esso elencati gli aspetti positivi e negativi emersi di ogni capitolato, e le motivazioni che hanno portato il gruppo alla scelta o meno di ognuno di essi.

3.1.7.2 Norme di Progetto

- **Classificazione**: Interno;
- **Distribuzione**: Gruppo e committente;
- **Contenuto**: Nelle *Norme di Progetto v3.0.0* verranno elencate tutte le norme, le convenzioni e gli strumenti che il gruppo *CommandLine Team* ha scelto di adottare durante tutto lo svolgimento del progetto. Tali norme sono assolute e dovranno essere rispettate da tutti i componenti del gruppo.

3.1.7.3 Analisi dei Requisiti

- **Classificazione**: Esterno;
- **Distribuzione**: Gruppo, committente e proponente;
- **Contenuto**: Il documento di *Analisi dei Requisiti v3.0.0* si prefigge lo scopo di fornire un'analisi dei requisiti del progetto, ed una loro catalogazione. I requisiti dovranno essere tracciabili ed identificati univocamente da un codice che, una volta associato ad un requisito, sarà immutabile. Il codice identificativo di un requisito si presenta nel seguente formato:

  R[Tipo][Importanza]-[Codice Identificativo]

  - **Tipo**: Rappresenta la tipologia del requisito, può assumere uno dei seguenti valori:
    - **F**: Requisito di Funzionalità
    - **P**: Requisito Prestazionale
    - **Q**:Requisito di Qualità
    - **V**: Requisito di Vincolo
  - **Importanza**: Indica il livello di importanza associato al requisito, come indicato di seguito:
    - **0**: Requisito Critico - il cui soddisfacimento è assolutamente necessario per garantire le funzioni base del sistema
    - **1**: Requisito Desiderabile - un requisito il cui soddisfacimento porta ad una maggiore completezza del sistema, ma il mancato soddisfacimento non pregiudica alcuna funzionalità base
    - **2**: Requisito Facoltativo - requisito che se soddisfatto renderebbe il sistema più completo, ma può portare ad un aumento dei costi
    - **3**: Requisito Aggiuntivo - un requisito che aggiunge funzionalità di contorno al sistema, non legate alle funzionalità base ma comunque in forte relazione con il dominio applicativo.
  - **Codice Identificativo**: Un identificatore univoco del requisito, nel formato [Padre].[Figlio]

  Inoltre ciascun requisito dovrà riportare le fonti da cui è stato ricavato o dedotto, le relazioni di dipendenza con altri requisiti ed una breve descrizione.

3.1.7.4 Piano di Progetto

- **Classificazione**: Esterno;
- **Distribuzione**: Gruppo, committente e proponente;
- **Contenuto**: Il *Piano di Progetto v3.0.0* descrive come il gruppo *CommandLine Team* ha impiegato le risorse umane nelle varie attività di progetto, e si prefigge inoltre di fare una buona pianificazione delle attività future previste per la realizzazione del prodotto richiesto dal progetto. È desiderabile che tale pianificazione sia verosimile e venga il più possibile rispettata.

3.1.7.5 Piano di Qualifica

- **Classificazione**: Esterno;
- **Distribuzione**: Gruppo, committente e proponente;
- **Contenuto**: Il *Piano di Qualifica v3.0.0* si occupa di descrivere il modo in cui il gruppo *CommandLine Team* ha deciso di perseguire la qualità del prodotto. In esso saranno dichiarati gli standard di qualità adottati e le metriche usate per la verifica del prodotto.

3.1.7.6 Glossario

- **Classificazione**: Esterno;
- **Distribuzione**: Gruppo, committente e proponente;
- **Contenuto**: Il *Glossario v3.0.0* ha lo scopo di rendere il significato di tutti i termini usati nella documentazione non ambiguo e comprensibile a tutti i destinatari. In esso dovranno essere elencati in ordine alfabetico tutti i termini non di uso comune, seguiti da una sintetica ma esplicativa definizione del loro significato. Tutti i termini presenti nel glossario verranno indicati nel documento con una G a pedice della loro prima apparizione in ciascun documento.

3.1.7.7 Verbali

- **Classificazione**: Interni ed esterni;
- **Distribuzione**: Gruppo e committente per VI / Gruppo, committente e proponente per VE;
- **Contenuto**: In ogni verbale sarà contenuto un sunto della riunione svolta. La struttura del verbale dovrà rispettare le norme definite in § 3.1.5.3.

3.1.8 Strumenti

3.1.8.1 LaTeX

Per la stesura di tutta la documentazione il gruppo ha scelto il linguaggio LaTeX poichè offre i seguenti vantaggi:

- Permette la separazione del contenuto dalla formattazione grazie all'uso di un template;
- Permette di creare documenti formali divisi in sezioni molto velocemente;
- È molto adatto a documenti che subiranno successive modifiche con aggiunta o rimozioni di parte di essi, grazie all'utilizzo della numerazione automatica delle sezioni;
- Permette una buona personalizzazione del documento grazie all'elevato numero di librerie.

Per la stesura dei documenti in LaTeX non è stato imposto ai componenti del gruppo un editor$_G$unico, al fine di permettere ai membri del gruppo di utilizzare l'editor che meglio conoscono o che ritengono più adatto, evitando così una potenziale perdita di tempo derivata dal dover apprendere l'utilizzo di nuovi software, permettendo ai membri di concentrarsi più sul contenuto effettivo del documento che sullo strumento utilizzato.
Di seguito vengono elencati tutti i diversi strumenti usati.

Texmaker

Texmaker è un editor multipiattaforma LaTeX open-source$_G$ con un visualizzatore di PDF integrato. Include inoltre supporto unicode, correttore ortografico, suggerimenti di auto-completamento, code folding$_G$.

`http://www.xm1math.net/texmaker/`

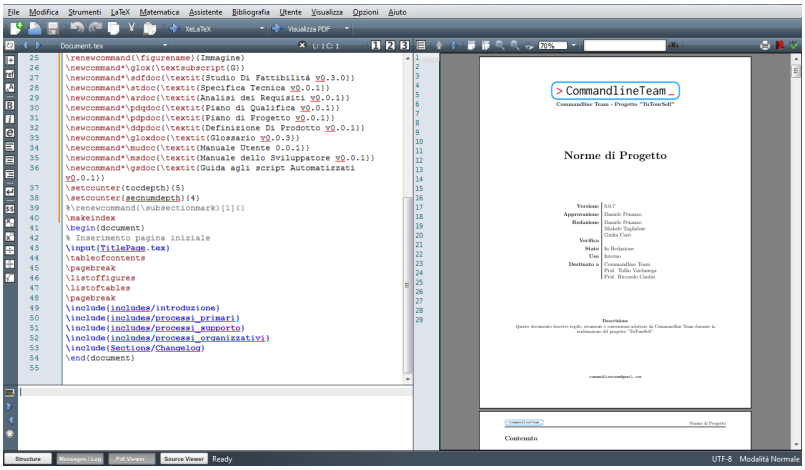

Immagine 2: Interfaccia di TexMaker per Windows.

TeXstudio
TeXstudio è un editor multipiattaforma LaTeX open-source. Originariamente chiamato TexMakerX, TeXstudio era inizialmente una variazione di Texmaker che provava ad espanderlo con caratteristiche aggiuntive, mantenendo invece intatta la sua percezione visiva. Le caratteristiche di TeXstudio includono un correttore ortografico interattivo, code folding, ed evidenziatore della sintassi.

`https://www.texstudio.org`

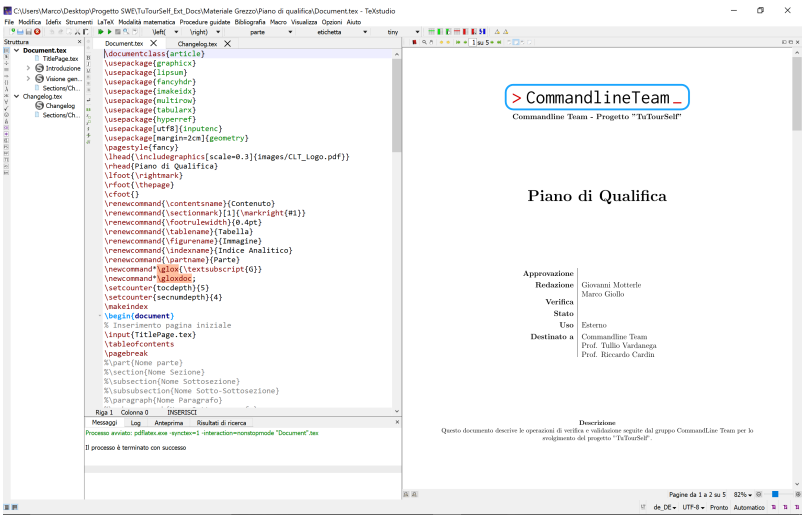

Immagine 3: Interfaccia di TeXStudio per Windows.

Neovim Neovim è un editor di testo gratuito ed open-source, derivato da Vim, che si concentra su estensibilità e riusabilità.

`https://neovim.io`

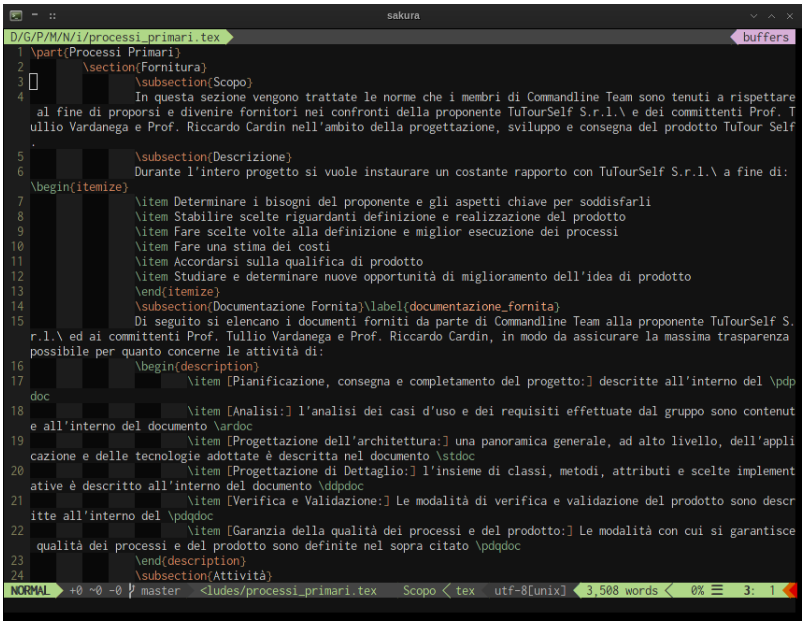

Immagine 4: Interfaccia di NeoVim.

3.1.8.2 StarUML

Per la realizzazione di diagrammi illustrativi per i documenti viene utilizzato StarUML. Questo è distribuito sotto licenza proprietaria, ma il gruppo ha deciso di usare una versione di valutazione che permette comunque l'uso delle funzionalità core$_G$.

`http://staruml.io/`

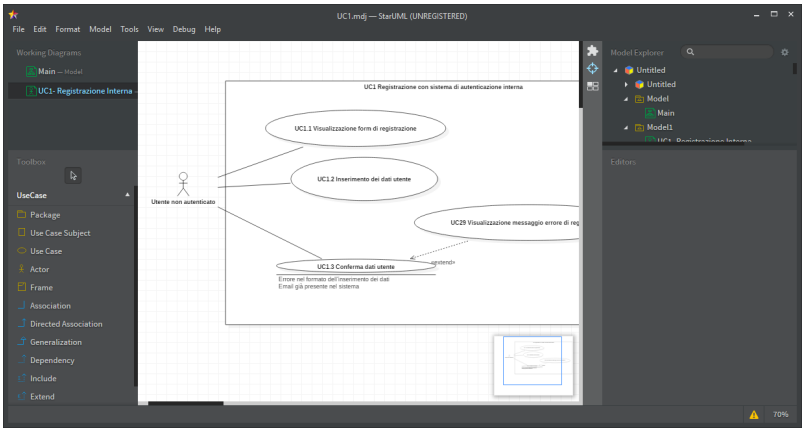

Immagine 5: Interfaccia di StarUML.

3.1.8.3 Script per l'automazione

A supporto del processo di documentazione, sono stati creati appositamente una serie di script$_G$ atti a velocizzare alcune fasi di quest'ultima. Una guida completa all'utilizzo dei seguenti script può essere consultata nel documento interno *Guida Script*.

- **The BookKeeper**: uno script scritto in Python 3 per poter inserire velocemente una nuova voce nel changelog$_G$di un documento;
- **Glossarizer**: uno script in Python 3 che permette di inserire una nuova voce di glossario, rilevando automaticamente il file corretto in cui inserire la voce, convertendo la definizione in codice LaTeX, ed inserendola in ordine alfabetico all'interno del file;

- **Documenter**: uno script Python 3 che permette di inizializzare, ricompilare documenti e ripulire la cartella dai file temporanei creati da LaTeX.

3.2 Gestione della Configurazione

3.2.1 Scopo

Questo processo ha lo scopo di identificare, registrare, analizzare e verificare tutti i cambiamenti effettuati su documentazione e software.

3.2.2 Aspettative

Una corretta implementazione di questo processo permette di:

- Avere delle norme efficaci per l'identificazione ed il tracciamento delle modifiche alla documentazione e al software;
- Avere a disposizione degli strumenti efficienti per l'analisi e la verifica dei cambiamenti effettuati a documentazione e software in qualsiasi momento.

3.2.3 Descrizione

Il processo di gestione della configurazione prevede attività e compiti relativi all'applicazione di procedure tecniche ed amministrative per assicurare la completezza, la consistenza e la correttezza della documentazione e del software.

3.2.4 Attività

3.2.4.1 Versionamento

Repository Come strumento di versionamento e salvataggio dei dati si è scelto di utilizzare diversi repository$_G$ su *GitLab*$_G$. La creazione dei vari repository è compito dell'Amministratore di progetto. Lo stesso *Amministratore* è incaricato di far sì che tutti i membri del gruppo possano accedere a tali repository.

E' previsto l'utilizzo di un repository di nome *TuTourSelf_Ext_Docs* che contiene tutta la documentazione del progetto, e di un repository di nome *TuTourSelf* che ne contiene il codice software.

Struttura dei Repository di gestione E' compito dei membri del gruppo rispettare e quindi mantenere la struttura dei repository, rispettando le organizzazioni qui di seguito elencate.

- Per quanto riguarda il repository *TuTourSelf_Ext_Docs* si ha la seguente struttura:
  - **Documenti formali**: cartella nella quale si trovano i file PDF di ciascun documento già approvato.
  - **Materiale Grezzo**: cartella nella quale si trovano i file LaTeX e i rispettivi PDF di tutti i documenti in fase di redazione.
- Per quanto riguarda il repository per il codice *TuTourSelf* si ha la seguente struttura:
  - **src**: cartella nella quale si trova il codice sorgente del prodotto;
  - **docs**: cartella contenente la documentazione generata automaticamente da JSDoc;
  - **config**: cartella contenente la configurazione della pipeline di Continuous Integration, deve contenere almeno i seguenti file e cartelle:
    - **eslint_config.js**: contenente la configurazione di base dello strumento di analisi statica ESlint;
    - **eslintignore**: contenente i percorsi che non devono essere analizzati da ESlint;
    - **jsdoc_config.json**: contenente la configurazione di base del generatore automatico di documentazione JSDoc;
    - **ci-scripts**: contenente gli script di notifica dello stato della pipeline.
  - **tests**: cartella contenente tutti i test che dovranno essere eseguiti da Mocha;
  - **.gitignore**: file contenente tutti i file e percorsi che devono essere ignorati da Git;

- **package.json**: file contenente tutti i moduli e le dipendenze necessarie per far funzionare il prodotto;
- **.gitlab-ci.yml**: file contenente le istruzioni per l'esecuzione della pipeline di continuous integration;
- **README.md**: file contenente una piccola descrizione del progetto e con eventuali collegamenti ai risultati della pipeline, come Code Coverage e stato della pipeline stessa;

Tipi di files e .gitignore All'interno delle cartelle contenenti i vari documenti saranno presenti solo file sorgente in formato *.tex*, file compilati in formato *.pdf* e file di immagini in formato *.png* e *.jpg*.
I file temporanei prodotti dalla compilazione LaTeXsono stati inclusi al file *.gitignore*, pertanto verranno ignorati da Git e quindi non caricati.

Norme sui commit Ogni modifica al repository (dunque ogni *commit*$_G$) deve essere accompagnata da una sensata descrizione, che permetta facilmente di capire a tutti i membri del gruppo che cosa quella modifica riguarda e che file coinvolge.

Comandi basilari Git Di seguito è riportata una breve introduzione ai comandi fondamentali per l'uso di Git$_G$ all'interno del progetto. Escluso il comando per la creazione della copia locale, i comandi sono da eseguirsi all'interno della cartella contenente il repository.

- *git clone indirizzo_repository nome_cartella*: Crea la copia locale del repository. Si può evitare di specificare il nome della cartella di destinazione nel qual caso il repository verrà creato nella locazione in cui viene eseguito il comando;
- *git pull*: Aggiorna sovrascrivendo il repository locale con quello remoto;
- *git status*: Mostra lo stato attuale del repository locale con i file modificati, tracciati e non tracciati da Git;
- *git add nome_file*: Aggiunge il file alla lista dei file tracciati da Git, si può usare con il simbolo di wildcard "*". E' da usare sia per aggiungere nuovi file sia per aggiungere le modifiche apportate ad un file , altrimenti non verranno applicate;
- *git rm nome_file*: Rimuove un file in modo che eseguendo una commit delle modifiche venga poi rimosso anche dal repository;
- *git commit -m "descrizione modifica"*: Fa la commit delle modifiche effettuate, salvandole sul repository locale. Viene eseguito solo sui file di cui si è eseguito il comando di *add*. La descrizione è obbligatoria e deve spiegare la modifica che viene salvata con la commit;
- *git branch*: Permette di visualizzare i branch presenti nel repository locale;
- *git checkout*: Permette di cambiare il branch attivo nel repository locale, tutti i comandi eseguiti hanno effetto sul branch attivo;
- *git fetch*: Scarica la versione aggiornata dal repository ma non esegue automaticamente il merge;
- *git merge*: Effettua l'unione del repository remoto, scaricato con il comando *fetch*, con il repository locale;
- *git push*: Aggiorna sovrascrivendo il repository remoto con quello locale.

3.2.4.2 Controllo di configurazione

Al fine di identificare e tracciare le richieste di cambiamento, analizzare e valutare le modifiche effettuate, viene usato il sistema di ticketing di Trello$_G$. La suddivisione del lavoro in *task*$_G$ spetta al Responsabile di progetto, e viene effettuata facendo appunto uso di Trello: egli dovrà quindi creare le varie schede$_G$ e assegnarle ai vari membri del gruppo.
Una scheda riguardante un *ticket* (o *issue*) avrà le seguenti proprietà:

- Titolo;
- Breve descrizione facoltativa;
- Assegnatari;

- Data di scadenza;
- Eventuali etichette, che permettono un'efficace catalogazione degli stessi;
- Una checklist, che permetterà di dividere il task in ulteriori azioni elementari, per una migliore gestione;
- Eventuali commenti, per segnalare difficoltà riscontrate durante lo svolgimento di tale compito.

Una volta creato il ticket e assegnato ad un membro del gruppo, egli riceverà una mail di avviso con tutti i dettagli del *ticket* appena assegnatogli.
Una volta risolto, il ticket dovrà essere marcato come tale dalla persona che lo ha risolto. Una commit su Git potrà far riferimento ad un certo ticket. In questo modo si potranno tracciare tutti i cambiamenti effettuati.

3.2.4.3 Stato della configurazione

A seguito di ciascuna revisione, l'ultima commit effettuata verrà marcata con una tag_G di versione e costituirà la $baseline_G$ di partenza per il progredire del lavoro verso la prossima $milestone_G$, secondo i tempi e modi definiti nel *Piano di Progetto v3.0.0*. L'esito di ciascuna revisione, e le correzioni da apportare andranno riportate in appendice nel *Piano di Qualifica v3.0.0*.

3.2.4.4 Gestione dei rilasci e delle consegne

I rilasci e la consegna di documentazione e prodotti software devono essere sottoposti a rigido controllo. La copia principale e costantemente aggiornata della documentazione verrà mantenuta all'interno dell'apposito repository su Gitlab, mentre al committente prima di ogni revisione verrà consegnato, secondo le tempistiche stabilite dal *Piano di Progetto v3.0.0*, un archivio contenente tutta la documentazione tecnica di interesse in formato ".pdf". I documenti saranno accompagnati da una Lettera di Presentazione anch'essa inclusa nell'archivio e il link per il reperimento di tali file verrà comunicato tempestivamente al committente tramite email. E' assolutamente da evitare la consegna della documentazione come allegato di posta elettronica, in quando poco fruibile, scomodo e difficilmente manipolabile.

3.3 Assicurazione della Qualità

3.3.1 Scopo

Questo processo ha lo scopo di prevenire il presentarsi dei problemi, di individuarli quando si presentano, di identificarne le cause e di trovarvi rimedio. In questo modo si potrà garantire la qualità dei processi e dei prodotti durante il ciclo di vita del software.

3.3.2 Aspettative

Una corretta implementazione di questo processo permette di:

- Determinare le politiche per la qualità, gli obiettivi e le responsabilità;
- Implementare le politiche di qualità stabilite, attraverso pianificazione, controllo e miglioramento della qualità.

3.3.3 Descrizione

L'assicurazione della qualità riguarda sia i prodotti che i processi:

- Il prodotto deve essere conforme ai requisiti, definiti nel documento di Analisi dei Requisiti, ed aderire al Piano di Progetto, al fine di poter possedere gli specificati attributi di qualità;
- Il processo, oltre ad essere conforme a quanto specificato ed aderire alla pianificazione specificata nel Piano di Progetto , deve rispettare gli standard utilizzati per la sua implementazione.

Le attività riguardanti l'assicurazione della qualità che vengono di seguito presentate sono inoltre descritte nel dettaglio nel documento *Piano di Qualifica v3.0.0*, dove ne viene illustrata anche la concreta attuazione.

3.3.4 Attività

3.3.4.1 Assicurazione della Qualità di Processo

Descrizione
La qualità dei processi è un fattore fondamentale durante lo svolgimento del progetto. Al fine di assicurarla, il gruppo *CommandLine Team* utilizza gli standard ISO/IEC 15504, denominato $Spice_G$, e lo standard $PDCA_G$ (Plan-Do-Check-Act).

Standard ISO/IEC 15504
Questo standard permette di valutare e classificare il livello di maturità dei processi e di verificare l'adeguatezza secondo il relativo obiettivo. Il livello di maturità viene valutato secondo gli attributi dei processi, ed è definito secondo questa scala:

- **Level 0 - Incompleto**: il processo non è implementato o non raggiunge i suoi obiettivi;
- **Level 1 - Eseguito**: il processo è implementato e raggiunge i suoi obiettivi. Misurato secondo:
  - Performance: capacità di ottenere risultati identificabili.
- **Level 2 - Gestito**: il processo agisce in base ad una pianificazione ed ogni sua azione è tracciata. Misurato secondo:
  - Gestione delle performance: capacità di elaborare un prodotto coerente con gli obiettivi attesi;
  - Gestione delle performance: capacità di elaborare un prodotto documentato, controllato e verificato.
- **Level 3 - Definito**: il processo agisce in base a linee guida uniformi nell'intera organizzazione. Misurato secondo:
  - Definizione: capacità di elaborare un prodotto seguendo gli standard preposti;
  - Risorse: capacità di sfruttare le risorse a disposizione così da venir attuato al meglio.
- **Level 4 - Predicibile**: il processo agisce entro certi limiti. Misurato secondo:
  - Misurazioni: capacità di sfruttare le misure ricavate durante l'esecuzione così da raggiungere i propri obiettivi;
  - Controllo: capacità di sfruttare le misure ricavate durante l'esecuzione così da migliorarsi e correggersi, se necessario.
- **Level 5 - Ottimizzato**: il processo viene misurato e quindi ottimizzato. Misurato secondo:
  - Cambiamenti: capacità di supportare cambiamenti strutturali e di esecuzione;
  - Miglioramento continuo: capacità di sfruttare i cambiamenti strutturali e di esecuzione così da migliorarsi continuamente nel raggiungimento dei propri obiettivi.

Ogni attributo di un processo viene valutato in una scala metrica di quattro unità:

- N - Non posseduto [0 - 15%]
- P - Parzialmente posseduto]15% - 50%]
- L - Largamente posseduto]50% - 85%]
- F - Completamente posseduto]85% - 100%]

PDCA Sulla base delle metriche di maturità precedentemente descritte, viene applicata una strategia di miglioramento continuo della qualità dei processi di sviluppo utilizzando il modello PDCA, noto anche come Ciclo di Deming. In questo modo il team cerca di ottimizzare l'uso delle risorse durante l'intero ciclo di vita del prodotto puntando ad un risultato di qualità. Questo modello si suddivide in quattro iterazioni, e assicura un miglioramento progressivo ad ogni ciclo. Nello specifico:

- **Plan**: vengono stabiliti obiettivi e processi necessari per raggiungere i risultati aspettati;
- **Do**: implementazione del punto precedente ed attuazione dei processi, il tutto finalizzato alla creazione del prodotto;

- **Check**: vengono comparati i risultati ottenuti con quelli attesi, e raccolti in grafici e tabelle per uno studio approfondito del traguardo raggiunto. Se si sono raggiunti gli obiettivi preposti si può passare alla fase successiva, altrimenti è necessario ripetere il ciclo PDCA tenendo conto delle cause che hanno contribuito al fallimento;
- **Act**: vengono attuate azioni di aggiustamento e correzione. La soluzione individuata diventa la nuova baseline, e si può quindi ripetere l'intero ciclo.

Le operazioni di *Do* e *Check* sono di competenza dei Verificatori, che ne avranno completa responsabilità e potranno riferire agli Amministratori per implementazioni software e integrazione norme. Le attività di *Act* e *Plan* dovranno essere invece discusse in presenza del Responsabile di progetto, che ne deve approvare le decisioni, secondo budget, ore persona impiegate e utilità.

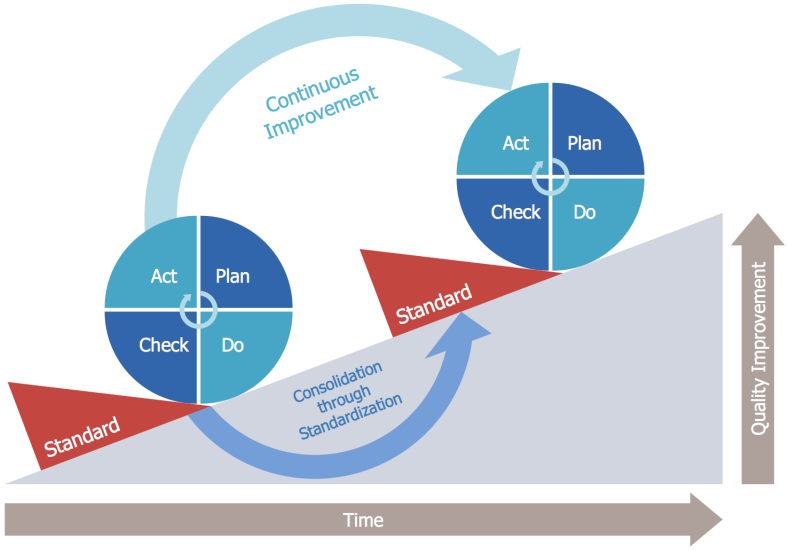

Immagine 6: Principio del miglioramento continuo secondo PDCA

3.3.4.2 Assicurazione della Qualità di Prodotto

Descrizione

Questa attività ha lo scopo di assicurare che il prodotto rispetti le caratteristiche di qualità concordate con il committente. La qualità del prodotto non deve essere assicurata solo al completamento del prodotto, ma anche durante tutto il suo periodo di sviluppo.

Strategie

Al fine di assicurare la qualità del software, ai Verificatori viene assegnato il compito di:

- Controllare che il sistema di testing e verifica rispettino i parametri descritti nel *Piano di Qualifica* e nelle *Norme di Progetto*. Questo deve essere fatto di seguito al rilascio di ogni nuova versione dei documenti sopracitati, e ogniqualvolta avvenga una modifica nella configurazione di sistema;
- Controllare periodicamente che il versionamento avvenga secondo le specifiche definite nelle *Norme di progetto*, e che i relativi strumenti vengano utilizzati correttamente;
- Avvisare il Responsabile di Progetto nel caso uno dei fattori precedentemente descritti non fosse regolare, affinché vi venga posto rimedio al più presto. Il Verificatore stesso non deve, e non ha le competenze necessarie, per porre rimedio da solo all'attività mal gestita.

3.4 Verifica

3.4.1 Scopo

Il processo di verifica si occupa di accertare che non vengano prodotti errori a seguito dell'esecuzione delle attività da parte dei membri del gruppo.

3.4.2 Aspettative

Le aspettative derivanti dalla corretta implementazione di tale processo sono:

- L'individuazione di una procedura di verifica da seguire;
- Dei criteri fissati per la verifica del prodotto;
- Una catalogazione dei difetti che devono essere corretti.

3.4.3 Descrizione

In questa sezione verrà descritto il processo di verifica. Esso è composto da due attività distinte:

- **Analisi**: l'analisi e successiva esecuzione del codice sorgente. Viene effettuata tramite analisi statica ed analisi dinamica;
- **Test**: tutti i test che vengono eseguiti sul prodotto software.

3.4.4 Attività

3.4.4.1 Metriche

Per garantire la qualità del lavoro, i Verificatori hanno stabilito delle metriche, delle quali vengono riportati gli obbiettivi nel *Piano di Qualifica v3.0.0*. Tali metriche devono rispettare la seguente notazione:

M[Ambito][Codice Identificativo]

Dove:

- **Ambito**: indica se la metrica si riferisce a processi, prodotto documento oppure prodotto software, e può assumere i seguenti valori:
  - **PC**: per indicare una metrica per il processo;
  - **PD**: per indicare una metrica per il documento;
  - **PS**: per indicare una metrica per il software;
- **Codice Identificativo**: indica in codice univoco della metrica, formato da un intero incrementale a partire da 1.

Metriche per i processi

- **MPC1 - ISO/IEC 15504 (SPICE)**: Lo standard ISO/IEC 15504, anche conosciuto come SPICE (Software Process Improvement Capability Determination) è lo standard di riferimento per valutare oggettivamente la qualità dei processi software, con il fine di migliorarli e permette di misurare indipendentemente la capacità di ogni processo tramite degli attributi, studiando il range di risultati che si ottengono eseguendolo. Tale metrica viene descritta nel dettaglio nella sezione §3.3.4.1 di questo documento e nel documento *Piano di Qualifica v3.0.0*.
  - Range di accettazione: livello 2
  - Range ottimale: $\geq$ livello 4
- **MPC2 - Requirement Stability Index (RSI)**: Indica la percentuale dei requisiti rimasti invariati nel tempo. Un valore elevato di tale metrica indica un'attività di analisi attenta e corretta. Viene calcolata tramite la seguente formula:

$$RSI[\%] = 1 - \frac{\#requisiti\_aggiunti + \#requisiti\_tolti + \#requisiti\_modificati}{\#requisiti\_totali\_iniziali}$$

  - Range di accettazione: ≥ 0.6
  - Range ottimale: 1

- **MPC3 - Errori frequenti nella documentazione**: Indica il numero di errori frequenti, descritti nella tabella §3.4.4.2, individuati durante una verifica tramite Inspection. Un riscontro elevato indica poca cura nella stesura dei documenti.
  - Range di accettazione: ≤ 6
  - Range ottimale: 0

- **MPC4 - Violazioni dello stile di codifica**: Indica il numero di occorrenze all'interno del codice di violazioni dello stile di codifica, normato secondo quanto scritto nella sezione §2.2.4.3. Un riscontro elevato indica poca cura nel processo di codifica e, di conseguenza, codice poco affidabile.
  - Range di accettazione: ≤ 10
  - Range ottimale: 0

- **MPC5 - Test di unità implementati**: Indica la percentuale di test di unità effettivamente implementati, rispetto a quelli previsti. Un valore elevato di tale metrica indica un livello soddisfacente di test sulle componenti base del sistema. Viene calcolata tramite la seguente formula:

$$TUI[\%] = \frac{\#test\_unita\_implementati}{\#test\_unita\_totali}$$

  - Range di accettazione: $\geq 90\%$
  - Range ottimale: 100%

- **MPC6 - Test di integrazione implementati**: Indica la percentuale di test di integrazione effettivamente implementati, rispetto a quelli previsti. Un valore elevato di tale metrica indica un livello soddisfacente di test per le interazioni tra le varie componenti del sistema. Viene calcolata tramite la seguente formula:

$$TII[\%] = \frac{\#test\_integrazione\_implementati}{\#test\_integrazione\_totali}$$

  - Range di accettazione: $\geq 90\%$
  - Range ottimale: 100%

- **MPC7 - Test di sistema implementati**: Indica la percentuale di test di sistema effettivamente implementati, rispetto a quelli previsti. Un valore elevato di tale metrica indica un buon funzionamento delle funzionalità di sistema. Viene calcolata tramite la seguente formula:

$$TSI[\%] = \frac{\#test\_sistema\_implementati}{\#test\_sistema\_totali}$$

  - Range di accettazione: $\geq 90\%$
  - Range ottimale:100%

- **MPC8 - Test di accettazione implementati**: Indica la percentuale di test di validazione effettivamente implementati, rispetto a quelli previsti. Un valore elevato di tale metrica indica il soddisfacimento delle aspettative del proponente. Viene calcolata tramite la seguente formula:

$$TAI[\%] = \frac{\#test\_accettazione\_implementati}{\#test\_accettazione\_totali}$$

  - Range di accettazione: $\geq 90\%$
  - Range ottimale: 100%

Metriche per i prodotti

Metriche per la Documentazione

- **MPD1 - Indice Gulpease**: L'Indice Gulpease è un indice di leggibilità di un testo tarato sulla lingua italiana. Rispetto ad altri ha il vantaggio di utilizzare la lunghezza delle parole in lettere anziché in sillabe, semplificandone il calcolo automatico. La formula per il suo calcolo è la seguente:

$$89 + \frac{300 * numero\_frasi - 10 * numero\_lettere}{numero\_parole}$$

  I risultati sono compresi tra 0 e 100, dove un valore più alto indica leggibilità più alta, in generale i testi con un valore inferiore 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 e inferiore a 40 sono difficili da leggere per chi ha un diploma superiore
  * Range di accettazione: ≥ 40
  * Range ottimale: ≥ 60
- **MPD2 - Errori ortografici corretti**: Questa metrica indicherà il numero di errori rilevati tramite GNU Aspell$_G$ e successivamente corretti. I documenti non devono presentare errori ortografici o grammaticali, a tal fine tutti gli errori individuati dovranno essere tempestivamente corretti
  * Range di accettazione: 100% corretti
  * Range ottimale: 100% corretti

Metriche per il Software

- **MPS1 - Copertura requisiti obbligatori**: Permette di monitorare in ogni istante la percentuale di requisiti obbligatori soddisfatti, e viene calcolata con la seguente formula:

$$\frac{(\#requisiti\_obbligatori\_soddisfatti)}{(\#requisiti\_obbligatori)}$$

  * Range di accettazione: 100%
  * Range ottimale: 100%
- **MPS2 - Copertura requisiti desiderabili**: Permette di monitorare in ogni istante la percentuale di requisiti desiderabili soddisfatti, e viene calcolata con la seguente formula:

$$\frac{(\#requisiti\_desiderabili\_soddisfatti)}{(\#requisiti\_desiderabili)}$$

  * Range di accettazione: 60%
  * Range ottimale: $\geq$80%
- **MPS3 - Linee di codice coperte dai test**: Indica la percentuale di istruzioni che vengono eseguite durante i test rispetto al totale. Maggiore è la percentuale testata, maggiore sarà la possibilità che eventuali errori vengano individuati e risolti. Il valore ottimale desiderato è corrispondente ad almeno il 70% di copertura, e viene calcolato con la seguente formula:

$$\frac{(\#istruzioni\_eseguite)}{(\#istruzioni\_totali)}$$

  * Range di accettazione: 50%
  * Range ottimale: $\geq$ 70%
- **MPS4 - Percentuale di superamento test**: Indica la percentuale di test superati correttamente alla fine delle attività di verifica, rispetto al totale dei test eseguiti. Un valore ottimale dovrebbe corrispondere al 100%. Viene calcolata con la seguente formula:

$$\frac{(\#test\_superati)}{(\#test\_eseguiti)}$$

  - Range di accettazione: 85%
  - Range ottimale: 100%
- **MPS5 - Numero di parametri per metodo**: Calcola il numero di parametri associati ad ogni metodo. Un numero troppo elevato di parametri è indice di scarsa manutenibilità del codice.
  - Range di accettazione: ≤ 5
  - Range ottimale: ≤ 3
- **MPS6 - Linee di codice per metodo**: Calcola il numero di linee di codice per ogni metodo, non conteggiando i commenti e le linee vuote. Un valore elevato di tale metrica può indicare un'eccessiva complessità del metodo.
  - Range di accettazione: ≤ 70
  - Range ottimale: ≤ 30
- **MPS7 - Rapporto tra linee di commento e linee di codice**: È il rapporto tra le linee di commento e le linee di codice, escludendo le righe vuote. Questo rapporto aiuta a stimare la manutenibilità del codice. Un rapporto troppo basso indica una carenza di informazioni necessarie alla comprensione del codice scritto. Viene calcolato come:
$$\frac{(\#linee\_commento)}{(\#linee\_codice)}$$
  - Range di accettazione: ≥ 0.25
  - Range ottimale: ≥ 0.3
- **MPS8 - Numero di metodi per classe**: Rappresenta il numero di metodi di una classe. Se una classe ha un numero elevato di metodi probabilmente viola i principi SOLID_G, soprattutto quello della Single Responsibility, per il quale ogni classe deve assolvere ad un solo compito. In caso la classe presenti un elevato numero di metodi, sarà preferibile suddividerla in più classi
  - Range di accettazione: ≤ 15
  - Range ottimale: ≤ 5
- **MPS9 - Numero di attributi per classe**: Considera il numero totale di campi dati presenti all'interno di una classe, escludendo i campi statici ereditati. Questa metrica è utile per comprendere i gradi di comprensibilità e manutenibilità del codice. In presenza di un valore troppo alto può essere utile scomporre la classe in più classi
  - Range di accettazione: ≤ 10
  - Range ottimale: ≤ 3
- **MPS10 - Accoppiamento tra le classi**: Calcola il numero di classi con cui ogni classe è accoppiata. Una classe si dice accoppiata con un'altra se è presente una dipendenza tra le due, indipendentemente dal verso della dipendenza. Un valore elevato di questa metrica non è desiderabile poiché evidenzia una bassa modularità del codice
  - Range di accettazione: ≤ 15
  - Range ottimale: ≤ 12
- **MPS11 - Instabilità dei package**: Questa metrica è stata proposta da Robert C. Martin per stimare l'instabilità dei vari package che compongono un sistema. Il livello di stabilità indica la possibilità di apportare modifiche ad un package senza che questo influenzi il funzionamento dell'intero sistema, causando una serie di correzioni a catena. Il valore di instabilità è calcolato secondo la seguente formula:
$$Instability[\%] = \frac{(SFOUT)}{(SFIN + SFOUT)}$$

  Dove
  - **Accoppiamento afferente (SFIN)**: rappresenta il grado di utilità delle classi di un package verso classi esterne ad esso. Un valore troppo basso potrebbe essere indice di scarsa utilità, dato che poche delle sue funzionalità sono usate all'esterno. Un valore troppo elevato, al contrario, può indicare un livello di dipendenza pericoloso del package in esame. Ciò può portare ad effetti indesiderati nelle classi esterne in caso di modifiche. Nonostante ciò, un valore elevato non è necessariamente indice di un errore di progettazione ma potrebbe indicare semplicemente la criticità del package in esame;

  - ∗ **Accoppiamento efferente (SFOUT)**: rappresenta il grado di dipendenza delle classi di un package verso classi di package esterni. Un valore basso garantisce la stabilità del package indipendentemente dalle possibili modifiche al resto del sistema.
  - ∗ Range di accettazione: $\leq 90\%$
  - ∗ Range ottimale: $\leq 50\%$
- **MPS12 - Complessità ciclomatica media**: Usata per stimare la complessità di funzioni, moduli, metodi o classi di un programma. Questo valore rappresenta quanto complesso è un programma tramite la misura del numero di cammini linearmente indipendenti che attraversano il grafo di controllo di flusso. In tale grafo, i nodi rappresentano gruppi indivisibili di istruzioni. Un arco connette due nodi se le istruzioni di uno dei nodi possono essere eseguite direttamente dopo l'esecuzione delle istruzioni dell'altro nodo. Un valore troppo elevato porta ad un'eccessiva complessità del codice. Al contrario, un valore ridotto potrebbe indicare una scarsa efficienza. La complessità ciclomatica è calcolata con la seguente formula:

  $$V(G) = E - N + 2P$$

  Dove, rappresentando un programma con il grafo di controllo del flusso:
  - ∗ **N**: rappresenta il numero di nodi;
  - ∗ **E**: rappresenta il numero di archi;
  - ∗ **P**: rappresenta il numero di componenti connesse.
  - ∗ Range di accettazione: ≤ 20
  - ∗ Range ottimale: ≤ 10

Strumenti

Per il calcolo dell'indice di Gulpease, è stato utilizzato uno strumento online, disponibile al seguente indirizzo:

`https://farfalla-project.org/readability_static/`

Al fine di semplificare l'estrazione del testo da sottoporre al calcolo dell'indice Gulpease è stato usato il programma opendetex, i cui sorgenti sono disponibili al seguente indirizzo:

`https://github.com/pkubowicz/opendetex`

3.4.4.2 Analisi

Analisi Statica

L'analisi statica è una tecnica che permette di individuare anomalie ed errori all'interno di documentazione e codice sorgente per tutto il loro ciclo di vita. Può essere applicata tramite due distinti metodi:

- **Walkthrough**: questa tecnica consiste nel fare una ricerca "a pettine", ossia eseguire un controllo dell'intera parte da analizzare secondo una disposizione parallela. Questa è un'attività onerosa che richiede la cooperazione di più persone, e perciò non è una tecnica efficiente. Verrà tuttavia usata principalmente durante la prima fase di progetto, quando la maggior parte dei membri non avrà piena conoscenza e padronanza delle Norme di Progetto e del Piano di Qualifica. Utilizzando questa tecnica è inoltre possibile stilare una lista di controllo contenente gli errori più comuni, che sarà utile per una seguente attività di inspection;
- **Inspection**: questa tecnica consiste in una lettura mirata e strutturata, volta a localizzare gli errori più comuni segnalati nella lista di controllo, con il minor costo possibile. Questa lista di controllo verrà progressivamente estesa grazie all'esperienza acquisita, rendendo l'inspection sempre più efficace. Normalmente questa tecnica è svolta da una persona singola, ciò la rende inoltre una tecnica efficiente.
  Di seguito viene riportata la lista degli errori frequenti che dovranno essere controllati durante l'attività di inspection, e che verrà aggiornata con il progredire del progetto.

  Anomalie comuni riscontrate nella documentazione

Anomalia	Descrizione
Elenchi puntati non conformi alle norme	Gli elenchi puntati non rispettano le norme, in particolare la terminazione con punto e virgola di ogni voce, e con punto di quella finale.
Posizionamento delle informazioni all'interno dei documenti	Le sezioni dei documenti vengono inserite nella parte sbagliata del documento, oppure nel documento sbagliato.
Non conformità delle lettere maiuscole	Nei titoli e nella stesura del testo tutti i membri devono mantenere un'uniformità nell'uso delle maiuscole, per creare coerenza nel documento.
Frontespizio del documento non aggiornato nelle informazioni	Le informazioni sullo stato del documento presenti nel frontespizio non vengono aggiornate all'approvazione del documento stesso.
Tracciamento azioni nel changelog	Tutte le modifiche fatte al documento devono essere tracciate nel changelog.
Sintesi nel changelog	Per motivi di sintesi, il luogo di modifica dovrà essere indicato nel changelog con un riferimento alla sezione coinvolta.

Tabella 3: Lista anomalie comuni riscontrate nella documentazione

Anomalie comuni riscontrate nel codice

Anomalia	Descrizione
Indentazioni Irregolari	Le indentazioni del codice risultano irregolari, con un numero inconsistente di spazi oppure facendo uso di tabulazioni invece che di spazi. Le indentazioni dovrebbero seguire le norme descritte in sezione §2.2.4.3
Mancanza di spazi attorno agli operatori	Attorno ad ogni operatore dovrebbero essere presenti spazi in modo da migliorarne la leggibilità, questa norma non è sempre stata rispettata.
Lunghezza delle righe di codice	Le righe di codice dovrebbero avere una lunghezza massima di 79 caratteri allo scopo di facilitare la lettura di sorgenti affiancati, non sempre questa norma è stata rispettata.

Tabella 4: Lista anomalie comuni riscontrate nel codice

Analisi Dinamica

L'analisi dinamica è una tecnica di analisi del prodotto software che ne richiede l'esecuzione. Viene eseguita ogni qualvolta che si è terminata una parte di prodotto tramite dei test di unità, volti a verificare il corretto funzionamento del prodotto e a permettere l'identificazione di anomalie.
I test devono essere ripetibili e deterministici, ossia date le stesse *precondizioni*$_G$ e lo stesso input, l'output deve essere sempre lo stesso. Per ogni test devono dunque essere definiti i seguenti parametri:

- **Ambiente**: il sistema hardware e software sul quale viene eseguito il test di prodotto;
- **Stato iniziale**: lo stato iniziale dal quale il test viene eseguito;
- **Input**: l'input che viene inserito;
- **Output**: l'output atteso;
- **Istruzioni aggiuntive**: ulteriori informazioni utili, quali istruzioni di esecuzione del test, istruzioni per l'interpretazione dell'output, eccetera.

3.4.5 Procedure

3.4.5.1 Controllo qualità

Il controllo della qualità, sia per quanto riguarda i processi che per i prodotti, è garantito dal rispetto delle Norme di Progetto e dai processi di Verifica e Validazione, secondo quanto stabilito nella sezione §3.3.

3.4.5.2 Gestione delle anomalie

Il Verificatore, alla chiusura di ogni issue$_G$, dovrà procedere alla verifica dell'operato. Egli dovrà inserire all'interno di un elenco ciascuna anomalia o errore rilevato e riportarlo al Responsabile di Progetto, il quale, tramite l'apertura di issue$_G$, incaricherà il redattore di quella porzione di codice o di documento di correggerli.

3.4.5.3 Gestione delle modifiche

Qualora il Verificatore individuerà parti di documentazione o codice che ritiene necessitino di una modifica, dovrà inoltrare al Responsabile di progetto la richiesta di modifica. Il Responsabile controllerà che ci sia l'effettiva necessità di una correzione e, in caso positivo, si occuperà di aprire ed assegnare gli issues per la modifica a chi ritiene più opportuno. A modifica effettuata, il Verificatore dovrà constatare che sia avvenuta con successo.
La richiesta di modifica che il Verificatore presenterà al Responsabile dovrà avere la seguente struttura:

- **Autore**: nome e cognome del Verificatore che inoltra la richiesta;
- **Documento o file**: indicazione di quale documento richiede una modifica;
- **Motivazione**: spiegazione del motivo per cui si ritiene necessaria la modifica.

Per motivi di tracciabilità, viene aggiunto dal Responsabile se la richiesta è stata accettata o meno, ed in caso negativo ne vengono indicate le motivazioni.
Dopo ogni correzione, questo ciclo di verifica viene ripetuto fino a che il documento è considerato del tutto corretto e valido dai Verificatori.

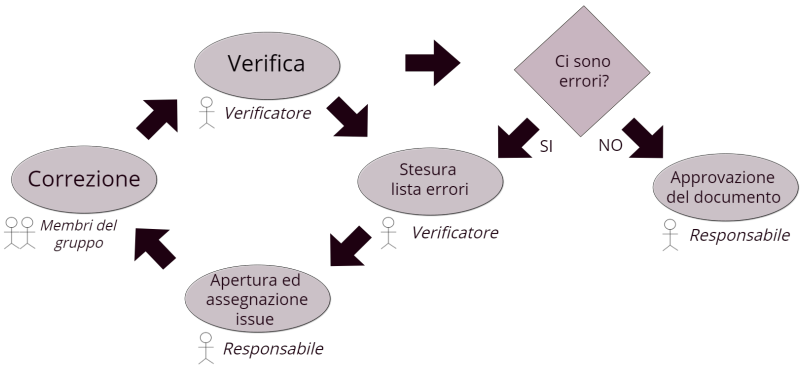

Immagine 7: Diagramma riassuntivo del ciclo di verifica

3.4.6 Strumenti

3.4.6.1 Controllo ortografico

- **Texmaker e TexStudio**: Per il controllo ortografico viene innanzitutto utilizzata la funzione di correzione in tempo reale integrata negli editor LaTeX usati quali Texmaker e TeXstudio, che permette la visualizzazione immediata degli errori grazie alla sottolineatura in rosso delle parole non corrette secondo la lingua italiana;
- **GNU Aspell**: È inoltre utilizzato GNU Aspell per controllare e segnalare errori ortografici, di digitazione e di sillabazione all'interno dei documenti.

`http://aspell.net`

3.4.6.2 Analisi Statica

Per l'esecuzione dell'analisi statica del codice vengono utilizzati i seguenti strumenti:

- **W3C$_G$ HTML Validator**: Per la validazione del linguaggio di markup HTML5 viene utilizzato lo strumento fornito dal W3C;

`https://validator.w3.org`

- **W3C CSS Validator**: Per la validazione dei fogli di stile CSS3, viene utilizzato lo strumento del W3C;

https://jigsaw.w3.org/css-validator/

- **EsLint**: ESLint è uno strumento per identificare errori di pattern nel codice JavaScript, con l'obiettivo di rendere il codice più consistente ed evitare bug;

https://eslint.org

3.4.6.3 Analisi Dinamica

Per l'esecuzione dell'analisi dinamica vengono utilizzati i seguenti strumenti:

- **Mocha**: Mocha è un framework ricco di funzionalità per l'esecuzione di test JavaScript, che esegue su Node.js e sul browser. Permette l'esecuzione di test asincroni e in serie, consentendo segnalazioni dei risultati flessibili ed accurati; correlando inoltre le eccezioni non catturate con i relativi test;

https://mochajs.org

- **Unit JS**: Unit.js è una libreria per JavaScript, che esegue su Node.js e sul browser. Può essere usata con ogni framework esecutore di test e test di unità come Mocha, Jasmine, Karma, QUnit eccetera. Unit.js supporta l'integrazione delle dipendenze ed è estensibile tramite un sistema di plugin di semplice uso.

http://unitjs.com

3.4.6.4 Documentazione di supporto al codice

- **JSDoc**: Per la creazione di documentazione di supporto al codice in stile JavaDoc, verrà usato JSDoc, un linguaggio di markup usato per commentare il codice sorgente JavaScript. Usando JSDoc, è possibile aggiungere annotazioni che descrivono l'interfaccia del codice che si sta scrivendo, le quali saranno processate da vari strumenti per produrre documentazione in formati accessibili come HTML e Rich Text Format.

http://usejsdoc.org/

3.5 Validazione

3.5.1 Scopo

Il processo di validazione serve ad accertare che il prodotto finale sia conforme a quanto pianificato. Tale processo viene effettuato soltanto quando, dopo aver eseguito molteplici verifiche sul prodotto, si è abbastanza certi che sia buono e completo, poichè non è desiderabile ottenere dalla validazione un risultato negativo.

3.5.2 Aspettative

Una corretta implementazione del processo di validazione permette di individuare:

- Una procedura di validazione;
- I criteri da prendere in esame per la validazione del prodotto;
- La conformità del prodotto finito.

3.5.3 Descrizione

L'attività di validazione consiste nel testare il prodotto, analizzare i risultati del test ed accertarsi che il prodotto soddisfi le caratteristiche e le aspettative per le quali è stato sviluppato. I test devono essere eseguiti e documentati secondo le norme elencate in § 3.5.5.1.

3.5.4 Attività

3.5.4.1 Analisi dinamica

I Verificatori hanno il compito di rieseguire tutti i test, ponendo attenzione ai risultati, in particolare per i test di validazione. Il Responsabile dovrà analizzare i risultati ottenuti e decidere in caso se è opportuno rieseguire ulteriormente i test.

3.5.5 Procedure

3.5.5.1 Test

Test di unità

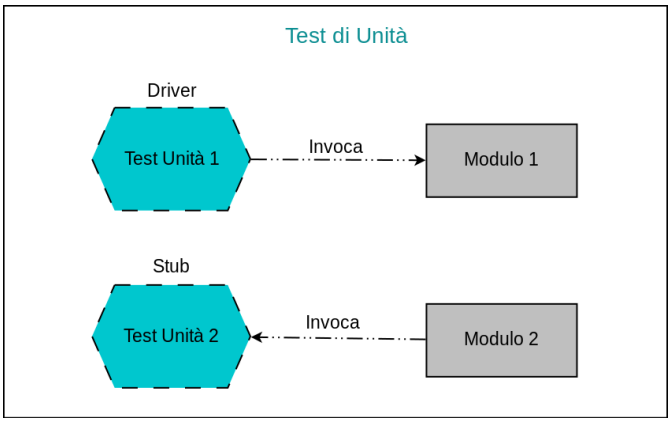

Immagine 8: Diagramma test di unità

Il test di unità si pone come obbiettivo l'isolare dal resto del codice la parte più piccola di software testabile, chiamata unità, per vedere se funziona come previsto oppure presenta anomalie. È molto importante sottoporre ogni unità a questo tipo di test prima che essa venga integrata in modulo per l'esecuzione del test delle interfacce tra i diversi moduli.
L'approccio più comune al test di unità necessita della preparazione di parti di software create ad hoc chiamate *driver*$_G$ e *stub*$_G$: il driver servirà a simulare un'unità chiamante, mentre lo stub simulerà un'unità chiamata.

Test di integrazione

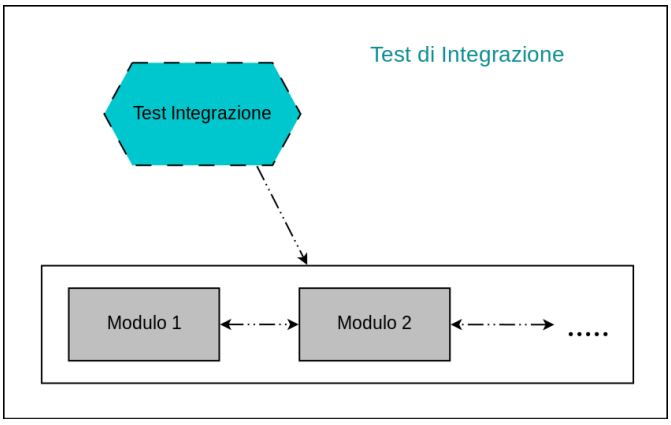

Immagine 9: Diagramma test di integrazione

Il test di integrazione rappresenta l'estensione logica del test di unità, e nella sua forma più semplice comprende la combinazione di due unità sottoposte a test in un'unico componente ed il test dell'interfaccia presente tra le due. Questo genere di test verifica quindi non solo il corretto comportamento di ogni singolo oggetto, ma anche le relazioni con gli altri componenti dell'applicazione. A partire da questo concetto applicato a singole unità, si estenderà progressivamente a testare moduli di un gruppo con quelli di altri gruppi.
Il testing a livello di integrazione è un'attività continuativa. Tranne i casi di software molto piccoli e semplici, le

strategie di test di integrazione sistematiche ed incrementali sono da preferire rispetto alla strategia di mettere tutti i componenti insieme nello stesso momento, in quello che viene chiamato *"big bang" testing*$_G$.

Test di sistema

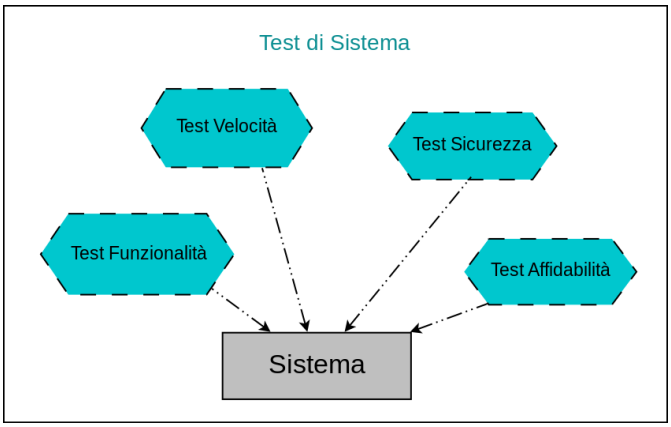

Immagine 10: Diagramma test di sistema

Il testing a livello di sistema si preoccupa del comportamento di un sistema nel suo complesso. La maggior parte degli errori dovrebbe essere già stato identificato durante il testing unitario e di integrazione. Il test di sistema viene di solito considerato appropriato per verificare il sistema anche rispetto ai requisiti non funzionali, come quelli di sicurezza, velocità, accuratezza ed affidabilità. A questo livello vengono anche testate le interfacce esterne nei confronti di altre applicazioni, componenti standard, dispositivi hardware e ambiente operativo.
Il superamento dei test di sistema sancisce la validazione del prodotto software, giunto ormai ad una versione definitiva.

Test di regressione

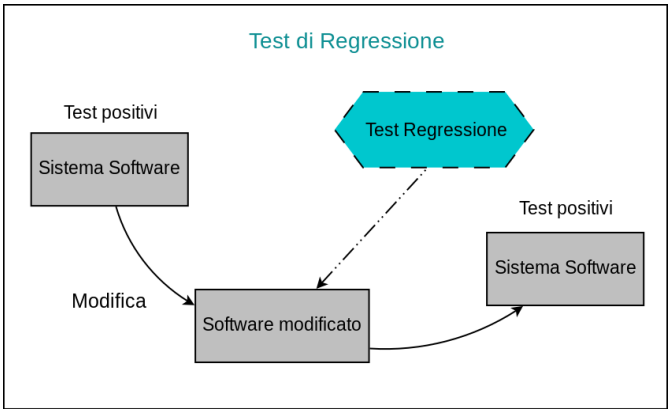

Immagine 11: Diagramma test di regressione

Il test di regressione consiste nell'eseguire nuovamente una selezione di test su un sistema o un componente modificato, per verificare che le modifiche non abbiano provocato effetti indesiderati. In pratica, l'idea è di dimostrare che il software che aveva passato i test prima delle modifiche continua a farlo anche dopo. Il test di regressione può essere effettuato ad ognuno dei livelli di test (unitario, di integrazione, di sistema).

Test di accettazione

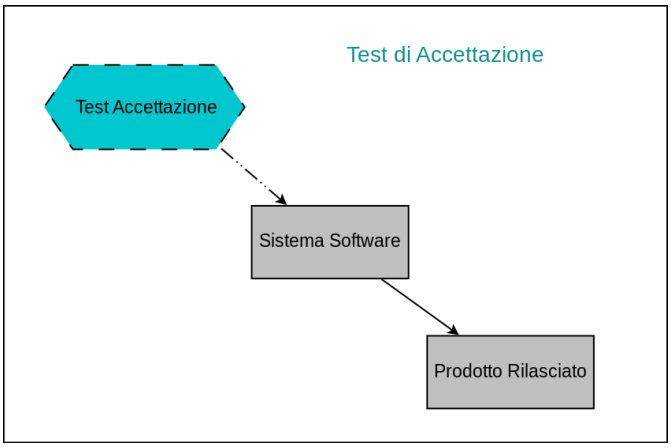

Immagine 12: Diagramma test di accettazione

Il test di accettazione rappresenta il collaudo del prodotto rispetto ai requisiti preposti, ed in presenza del Proponente. Al superamento di tale collaudo segue il rilascio ufficiale del prodotto sviluppato.

Codici identificativi dei test
La descrizione di ogni test è strutturata nel seguente modo:

- **Codice identificativo**: per ciascun test è assegnato un codice identificativo univoco che deve avere la seguente sintassi:

  T[Tipo][Codice identificativo]

  Dove:

  - **Tipo**: identifica uno dei seguenti tipi di test:
    - **U**: test di unità;
    - **I**: test di integrazione;
    - **S**: test di sistema;
    - **V**: test di validazione.
  - **Codice identificativo**: può assumere, in base al tipo di test, uno dei seguenti valori:
    - **Codice numerico**: associato ai test di unità e di integrazione, è un codice numerico intero progressivo a partire da 1;
    - **Codice requisito**: associato ai test di sistema e validazione, identifica il codice univoco associato ad ogni requisito, descritto in § 3.1.7.3.

- **Descrizione**: descrizione del test;

- **Stato**: può assumere i seguenti valori:

  - Implementato;
  - Non implementato;
  - Non eseguito;
  - Superato;
  - Non superato.

4 Processi Organizzativi

4.1 Gestione di progetto

4.1.1 Scopo

Lo scopo di questo processo è definire le modalità alle quali i membri del gruppo dovranno attenersi nello svolgimento dei vari processi inerenti all'attività di progetto.
Questo processo sfocerà nella redazione del *Piano di Progetto*, necessario ad un'ottimale organizzazione e gestione dei ruoli di ogni componente del team.

4.1.2 Aspettative

Le aspettative di tale processo sono:

- La redazione del *Piano di Progetto*
- La definizione dei ruoli che ciascun membro del team dovrà assumere
- La sintetizzazione di un piano di lavoro che formalizzi l'assegnazione di attività a ciascun membro del gruppo e che contenga la scansione temporale delle stesse

4.1.3 Descrizione

Vengono gestiti i seguenti argomenti:

- Ruoli di progetto;
- Comunicazioni (interne ed esterne);
- Riunioni (interne ed esterne);
- Strumenti di coordinamento;
- Strumenti di versionamento.
- Rischi

4.1.4 Ruoli di progetto

Come da istruzioni fornite, ciascun membro del gruppo ricoprirà a rotazione ogni ruolo. Nel *Piano di progetto* vengono definite le attività assegnate ad ogni ruolo.

4.1.4.1 Amministratore di Progetto

L'amministratore è colui che si occupa dell'efficienza del gruppo e dell'operatività delle risorse.
E' responsabile di:

- Scegliere e amministrare gli strumenti di versionamento;
- Ricercare strumenti che agevolino e automatizzino il lavoro quanto più possibile;
- Risolvere eventuali problemi di gestione di processi;
- Attua piani e procedure di gestione della *qualità*$_G$.

4.1.4.2 Responsabile di progetto

Il responsabile è, agli occhi del committente e del fornitore, il rappresentante del progetto. Inoltre, è colui che ha il compito di prendere decisioni ed incaricarsene la responsabilità.
E' responsabile di:

- Approvare i documenti;
- Redigere l'organigramma e il Piano di progetto;
- Gestire le risorse umane;
- Monitorare i progressi nell'avanzamento del progetto.

4.1.4.3 Analista
L'analista è colui che ha una vasta conoscenza del dominio del problema. Si occupa di:

- Capire al meglio il problema ed esporlo in modo chiaro;
- Redarre lo studio di fattibilità e l'analisi dei requisiti.

4.1.4.4 Progettista
Il Progettista si occupa di gestire gli aspetti tecnologici e tecnici del progetto. Nello specifico si occupa di:

- Effettuare scelte inerenti ad aspetti tecnici del progetto, applicando quanto più possibile *best practice*$_{\text{G}}$ per risolvere in modo più ottimizzato possibile problemi di natura ricorrente;
- Compiere decisioni che agevolino la manutenibilità del progetto e il suo riuso.

4.1.4.5 Verificatore
Il verificatore è colui che, grazie ad una solida conoscenza delle *Norme di progetto*, ne verifica l'avvenuta applicazione. Nello specifico:

- Controlla che le *Norme di progetto* vengano rispettate;
- Segnala al *Responsabile* eventuali discordanze tra quanto preventivato nel *Piano di progetto* e quanto realizzato.

4.1.4.6 Programmatore
Il programmatore è colui che si occupa della codifica del progetto e della sua manutenzione. I suoi compiti sono:

- Trasformare le indicazioni del progettista in codice documentato e manutenibile, scritto in modo coerente con quanto stabilito nelle norme di codifica;
- Creazione di test automatici che dimostrino la correttezza del codice scritto.
- Redazione del manuale utente;

4.1.5 Procedure

4.1.5.1 Gestione delle comunicazione

Comunicazioni Interne Le comunicazioni interne vengono gestite attraverso *Telegram*$_{\text{G}}$, un'app di messaggistica istantanea multipiattaforma. Inoltre, per l'assegnazione dei vari task, viene fatto uso di *Trello*$_{\text{G}}$, una board online che permette in modo molto semplice di avere una panoramica sui lavori svolti, in corso di svolgimento e da svolgere. Viene, inoltre, utilizzata la piattaforma *Slack*$_{\text{G}}$ per la comunicazione tra i vari componenti a cui è stato assegnato lo stesso ruolo.

Comunicazioni Esterne Le comunicazioni esterne avvengono prevalentemente usando la casella di posta elettronica *commandlineteam@gmail.com* , gestita dal *Responsabile* di progetto ma configurata per fare un inoltro automatico delle email a ciascun componente del gruppo. Viene, inoltre, utilizzato un gruppo *Slack* con La proponente per lo scambio di informazioni.

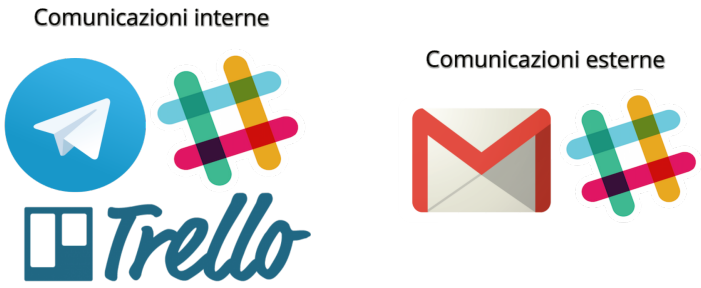

Immagine 13: Strumenti usati per le comunicazioni del gruppo

4.1.5.2 Gestione Riunioni

Riunioni Esterne E' compito del *Responsabile* di progetto organizzare eventuali riunioni esterne. Ciascun membro del gruppo, indipendentemente dal ruolo che sta assumendo, ha diritto di effettuare una richiesta di riunione esterna (coadiuvata da fondate motivazioni) al *Responsabile* di progetto, il quale è tenuto ad esaminare le motivazioni di tale richiesta e, se ritenute valide, a fissare la riunione. Fatto ciò, il *Responsabile* di progetto è tenuto a comunicare luogo e data della riunione ai componenti del gruppo, attraverso i mezzi di comunicazione preposti. Infine, lo stesso *Responsabile* di progetto è tenuto ad incaricare un componente del gruppo a redigere il verbale.

Riunioni Interne E' compito del *Responsabile* di progetto organizzare riunioni interne. Esso, su eventuale proposta dei membri del gruppo, è incaricato di scegliere luogo, data e orario della riunione e comunicarlo attraverso i canali di comunicazione interni agli altri membri del gruppo. Sarà compito del *Responsabile* di progetto assicurarsi della presenza di tutti i componenti del gruppo. Inoltre, analogamente a quanto avviene per le riunioni esterne, è compito del *Responsabile* di progetto incaricare un componente del gruppo a stilare il verbale.

4.1.5.3 Gestione dei rischi

La gestione dei rischi prevede l'identificazione, la valutazione della probabilità di accadimento e la stima dell'impatto che i rischi potrebbero avere sul progetto. Tali informazioni vanno inserite nel *Piano di progetto* (sezione analisi dei rischi), che il *Responsabile* di progetto è tenuto a controllare ed eventualmente integrare.
La gestione dei rischi prevede la seguente procedura:

- Controllo dei rischi previsti e verifica dell'efficacia delle strategie di risposta pianificate;
- Nel caso si verifichi un rischio previsto, evidenziarlo nel *Piano di progetto*;
- Nel caso di rischi non previsti, pianificare una strategia di risposta e inserirla nel *Piano di progetto*;
- Se risultano inefficaci, ridefinire le strategie di progetto.

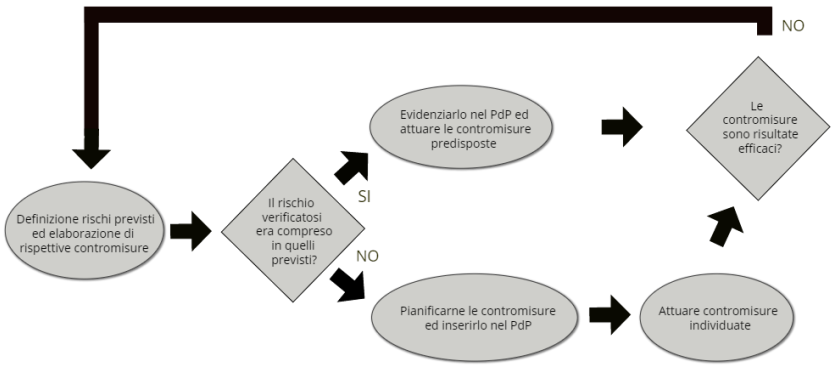

Immagine 14: Diagramma riassuntivo della procedura di gestione dei rischi

Codice identificativo dei rischi I rischi verranno identificati con i seguenti codici:

- **RT** per i rischi tecnologici;
- **RC** per i rischi dei e tra componenti;
- **RO** per i rischi di tipo organizzativo;
- **RS** per i rischi strumentali;
- **RR** per i rischi dei requisiti.

4.1.6 Strumenti

4.1.6.1 Sistema Operativo

I sistemi operativi utilizzati dai componenti del gruppo sono i seguenti:

- Windows 10 Pro x64;
- Windows 7 32 bit;
- Ubuntu 16.04 LTS;
- Mac OS High Sierra x64;
- Gentoo Linux - profilo v17.0;
- Elementary OS Loki 0.4.1 x64.

4.1.6.2 Git

Git è un sistema di versionamento gratuito e open source creato da Linus Torvalds nel 2005. La sua interfaccia è a riga di comando, ma esistono numerosi tool che forniscono una GUI. La versione utilizzata è almeno la 2.7.4.

4.1.6.3 GitLab

GitLab è un *repository manager*$_G$, che usa Git come sistema di controllo di versione, uniti a servizi di *issue tracking*$_G$ e di *wiki*$_G$.
GitLab prevede due varianti, una gratuita e open source (GitLab CE) e una a pagamento (GitLab EE), che differiscono per i servizi offerti. Sia la versione gratuita che quella a pagamento, però, permettono la creazione di repository privati.

4.1.6.4 Telegram

Telegram è un'applicazione di instant messaging cross platfrom$_G$ che può essere utilizzata su diversi dispositivi anche contemporaneamente. Telegram, inoltre, permette la creazione di gruppi, lo scambio di file di qualsiasi tipo e anche chiamate vocali tra due persone.

4.1.6.5 Slack

Slack è una piattaforma studiata apposta per la comunicazione interna tra membri di un team. L'applicazione è organizzata in workspace (nel nostro caso ne abbiamo due, uno per le comunicazioni interne e l'altro per le comunicazioni con il proponente). I workspace a loro volta sono suddivisi in canali, che permettono di "catalogare" le conversazioni in base ad un determinato argomento, in modo da tenere più ordinate e monotematiche possibili le chat.
Slack prevede tre abbonamenti, uno gratuito (che è quello che abbiamo scelto) e due a pagamento (che offrono più funzionalità). Inoltre Slack è cross platform, in quanto è dotato sia di applicazione per iOS/Android, che di applicazione per Windows/Mac OS/Linux che di web-application.

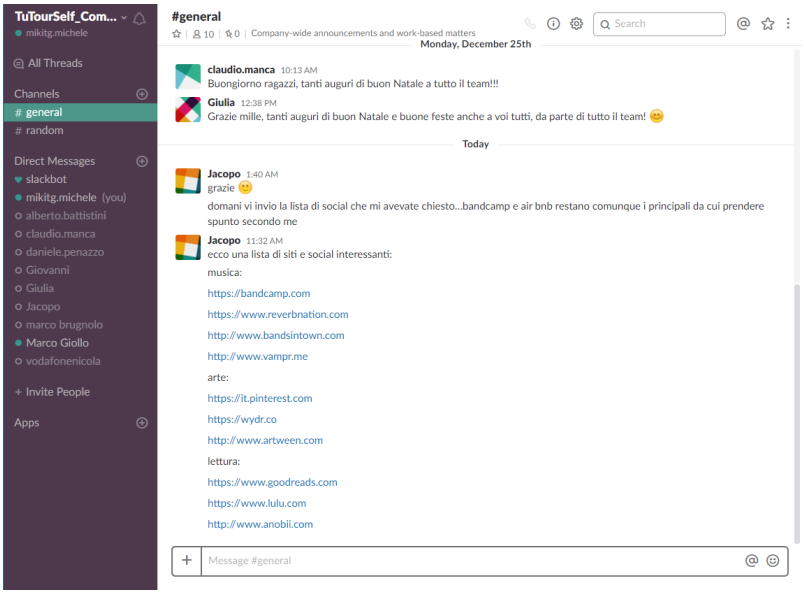

Immagine 15: Slack versione Web su Windows con Chrome

4.1.6.6 Trello

Trello è uno software di amministrazione di progetto web-based, con tanto di relative applicazioni per Android e iOS. Trello permette la creazione di una bacheca condivisa, organizzata in *cards*, all'interno delle quali è possibile inserire dei compiti da svolgere e assegnare a questi compiti una data di scadenza. Su ogni compito è possibile pubblicare dei commenti per, ad esempio, porre domande o fare determinate osservazioni.
Trello, inoltre, possiede numerose estensioni (alcune gratuite, altre a pagamento), tra le quali spicca *TrelloGantt*, che permette di visualizzare i contenuti inseriti in Trello sotto forma di diagramma di Gantt.

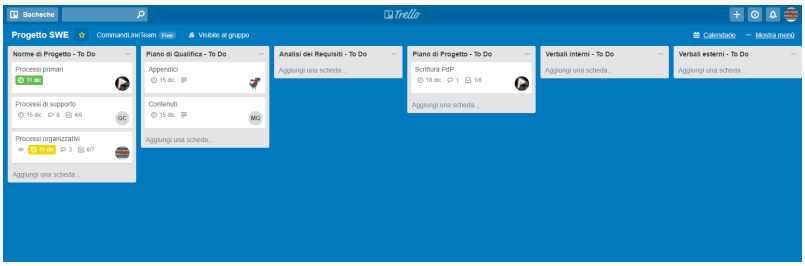

Immagine 16: Trello versione Web su Windows con Chrome

4.2 Adattamento e Manutenzione dei processi

Al fine di operare secondo il principio di miglioramento continuo, ogni processo dovrà essere sottoposto a monitoraggio costante.

4.2.1 Pianificazione

Per una gestione efficace dei processi è necessaria una dettagliata pianificazione dei tempi e delle modalità con cui tali processi si andranno a svolgere.

4.2.2 Esecuzione

Allo scopo di assicurare qualità ed economicità dei processi, è necessario eseguire e monitorare come i processi stanno performando rispetto alla pianificazione.

4.2.3 Valutazione

Al termine di ogni periodo specificato nel *Piano di Progetto v3.0.0*, si andranno ad effettuare valutazioni su ciascun processo secondo la metodologia PDCA (ciclo di Deming).
All'interno del *Piano di Qualifica v3.0.0*, per ogni processo dovrà essere riportata la seguente tabella:

Nome Processo	Il nome del processo sotto valutazione
Durata	Durata del processo
Tecnologie e Strumenti	Tecnologie e strumenti utilizzati nello svolgersi del processo
Metriche	Eventuali metriche di valutazione utilizzate
Problemi	Eventuali problematiche e ritardi incontrati
Piano di contingenza	Soluzioni e pianificazione futura atta al contenimento dei problemi rilevati
Valutazione	Una valutazione su una scala da 1 (pessimo) a 10 (eccellente)

Tabella 5: Tabella di valutazione dei processi

Allo scopo di garantire l'economicità dei processi, sarà utile effettuare revisioni dei processi ad intervalli appropriati.

4.2.4 Miglioramento

Alla fine del periodo, sulla base di quanto rilevato nella valutazione, si definiranno i miglioramenti da applicare e gli aggiornamenti sulla documentazione appropriata.
Dati tecnici, storici e di valutazione vanno raccolti ed analizzati all'interno del *Piano di Qualifica* allo scopo di rilevare criticità ed opportunità date dai processi impiegati.
L'analisi così effettuata sarà usata come feedback per il miglioramento dei processi, oltre ad indicare le modifiche da applicare a strumenti, procedure e tecnologie.

4.3 Gestione della formazione individuale

Ciascun componente del gruppo è tenuto a documentarsi autonomamente riguardo le tecnologie e i mezzi adottati tramite le risorse fornite dagli *Amministratori* (vedi 1.4.2).