Norme di Progetto

Informazioni Documento

Versione	3.0.0
Data approvazione	13 Febbraio 2018
Responsabile	Kevin Silvestri
Redattori	Kevin Silvestri, Marco Focchiatti, Manfredi Smaniotto
Verificatori	Cristiano Tessarolo, Giulio Rossetti
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo Graphite
Uso	Interno
Recapito	graphite.swe@gmail.com

Registro delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
3.0.0	2018-02-13	-	Responsabile	Approvazione
2.3.0	2018-02-12	-	Verificatore	Verifica §3
2.2.2	2018-02-12	-	Amministratore	Redatto §3.3.10
2.2.1	2018-02-12	-	Amministratore	Inseriti §3.3.2-§3.3.8-§3.3.9 (spostamento da PQ)
2.2.0	2018-02-12	-	Verificatore	Verifica §3-§4
2.1.2	2018-02-12	-	Amministratore	Rivisti schemi procedurali in §3.4-§3.5-§4
2.1.2	2018-02-12	-	Amministratore	Rivista procedura in §3.5.4
2.1.1	2018-02-12	-	Amministratore	Modificato formato data in §3.1.4.3
2.1.0	2018-02-12	-	Verificatore	Verifica §1-§2
2.0.3	2018-02-12	-	Amministratore	Dettagliata maggiormente §2.2.5
2.0.2	2018-02-12	-	Amministratore	Corretto errore link in §1.4
2.0.1	2018-02-12	-	Amministratore	Dettagliata maggiormente §1.4
2.0.0	2018-02-13	Kevin Silvestri	Responsabile	Approvazione
1.3.0	2018-02-12	Cristiano Tessarolo	Verificatore	Verifica §3
1.2.1	2018-02-11	Kevin Silvestri	Amministratore	Revisione e modifica §3
1.2.0	2018-02-08	Giulio Rossetti	Verificatore	Verifica §4
1.1.0	2018-02-02	Cristiano Tessarolo	Verificatore	Verifica §2-§3
1.0.3	2018-01-30	Marco Focchiatti	Amministratore	Revisione e modifica §4
1.0.2	2018-01-28	Manfredi Smaniotto	Amministratore	Revisione e modifica §3
1.0.1	2018-01-27	Kevin Silvestri	Amministratore	Revisione e modifica §2
1.0.0	2017-12-08	Matteo Rizzo	Responsabile	Approvazione
0.2.0	2017-12-08	Samuele Modena	Verificatore	Verifica §3-§4
0.1.0	2017-12-08	Kevin Silvestri	Verificatore	Verifica §1-§2

0.0.6	2017-12-07	Giulio Rossetti	Amministratore	Stesura §4
0.0.5	2017-12-06	Cristiano Tessarolo	Amministratore	Fine stesura §2
0.0.4	2017-12-05	Cristiano Tessarolo	Amministratore	Stesura §3
0.0.3	2017-12-04	Giulio Rossetti	Amministratore	Inizio stesura §2
0.0.2	2017-12-03	Samuele Modena	Amministratore	Stesura §1
0.0.1	2017-12-03	Samuele Modena	Amministratore	Creazione del template

Indice

1. Introduzione

1.1 Scopo del documento

Questo documento definisce le norme che i membri del gruppo Graphite seguiranno durante lo svolgimento del progetto per assicurare un modo di lavorare comune che garantisca una collaborazione efficiente tra tutti i diversi membri. Nello specifico, esso definisce:

- Modalità di lavoro durante le fasi di progetto;
- Convenzioni per la stesura di documenti;
- Strumenti utilizzati dai membri del gruppo.

Si fa notare la natura incrementale del documento e come quindi esso non definisca una versione finale.

1.2 Scopo del prodotto

Lo scopo del *prodotto*$_{\text{G}}$ è quello di fornire un *frontend grafico*$_{\text{G}}$ utilizzabile come strumento di supporto allo sviluppo di *plugin*$_{\text{G}}$ sulla piattaforma *Speect*$_{\text{G}}$. Lo strumento darà la possibilità all'utente di salvare i grafi generati a schermo dall'applicazione.
Il funzionamento dell'applicazione sarà garantito su un terminale *Linux Ubuntu*$_{\text{G}}$ versione 16.04.

1.3 Glossario

Al fine di evitare ogni ambiguità linguistica e di massimizzare la comprensibilità dei documenti, i termini tecnici e di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportati nel documento *Glossario v3.0.0*. Ogni termine presente nel glossario è marcato da una "G" maiuscola in pedice.

1.4 Riferimenti

Normativi

- *Standard ISO/IEC 12207:1995:* http://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf.

  Definisce la struttura del presente documento.

Informativi

- **Piano di Progetto v3.0.0**: documento *Piano di Progetto v3.0.0*;

  Definisce nel dettaglio ruoli e periodi.

- **Piano di Qualifica v3.0.0**: documento *Piano di Qualifica v3.0.0*;

  Definisce nel dettaglio obiettivi e strategie di qualità.

- **Amministrazione di progetto - Slide del corso "Ingegneria del Software":** http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L07.pdf;

  Introduzione all'amministrazione di progetto.

- **Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004:** http://www.math.unipd.it/~tullio/IS-1/2007/Approfondimenti/SWEBOK.pdf;

  - §7 "Software configuration management";
  - §8 "Software Engineering management";
  - §9 "Software Engineering process";
  - §10 "Software Engineering Tools and methods".

- **Software Engineering - Ian Sommerville - 9 th Edition (2010):**

  (Disponibile nel solo formato cartaceo);

  §25 "Configuration management".

- **GNU GCC Coding Conventions:** https://gcc.gnu.org/codingconventions.html.

  Definizione delle norme di codifica.

2. Processi primari

2.1 Fornitura

2.1.1 Scopo

Il *processo*$_{\text{G}}$ di fornitura ha lo scopo di trattare le norme e i termini che i membri del gruppo Graphite sono tenuti a rispettare per diventare fornitori della *proponente*$_{\text{G}}$ MIVOQ S.R.L. e dei *committenti*$_{\text{G}}$ Prof. Tullio Vardanega e Prof. Riccardo Cardin per quanto concerne il prodotto "DeSpeect: interfaccia grafica per Speect".

2.1.2 Rapporti di fornitura

Durante l'intero progetto si intende instaurare con la proponente MIVOQ S.R.L., nella persona del referente Giulio Paci, un profondo e quanto più possibile costante rapporto di collaborazione orientato a:

- Determinare aspetti chiave per soddisfare i bisogni della Proponente;
- Stabilire scelte volte alla definizione e realizzazione del prodotto (vincoli sui requisiti);
- Stabilire scelte volte alla definizione ed esecuzione di processi (vincoli di progetto);
- Stimare i costi;
- Concordare la qualifica del prodotto.

2.1.3 Attività della fornitura

Il processo di fornitura consiste delle seguenti attività:

- Analisi dei capitolati;

- Definizione modello di sviluppo;
- Pianificazione della Qualità.

Le suddette attività vengono di seguito esposte secondo scopo e descrizione.

2.1.4 Analisi dei capitolati

2.1.4.1 Scopo

L'attività consiste di un'analisi dettagliata di ogni capitolato proposto, con il fine di evidenziare le ragioni che hanno portato il gruppo Graphite a optare per quello scelto.

2.1.4.2 Descrizione

In seguito alla presentazione ufficiale dei Capitolati d'appalto, è compito del *Responsabile di Progetto*$_{\text{G}}$ convocare una riunione interna al gruppo per valutare le proposte di progetto pervenute. Gli *Analisti*$_{\text{G}}$ conducono quindi un'approfondita attività di analisi dei rischi e delle opportunità su ciascun capitolato. Lo studio effettuato deve evidenziare le motivazioni che hanno portato il gruppo Graphite a proporsi come fornitore per il prodotto indicato, nonché analizzare ogni capitolato proposto valutandone:

- **Dominio applicativo:** analisi del *Dominio Applicativo*$_{\text{G}}$, cioè l'ambito di utilizzo del prodotto da sviluppare;
- **Dominio tecnologico:** analisi del *Dominio Tecnologico*$_{\text{G}}$ richiesto dal capitolato e raggruppamento delle tecnologie da impiegare nello sviluppo del progetto. Tale analisi include valutazioni sulle conoscenze attuali e sulle possibilità di apprendimento in relazione alle tecnologie richieste per realizzare il prodotto proposto nel capitolato;
- **Aspetti positivi:** analisi sul costo in rapporto ai risultati previsti e all'interesse del gruppo rispetto alle tematiche del capitolato;
- **Fattori di rischio:** analisi delle criticità di realizzazione, quali ad esempio mancanza di conoscenze adeguate o difficoltà nell'individuazione di requisiti dettagliati.

2.1.5 Definizione del modello di sviluppo

2.1.5.1 Scopo

Lo scopo della attività è la pianificazione del progetto e del ciclo di vita del software a cui attenersi nel corso della realizzazione dello stesso, effettuata dal *Responsabile di Progetto* coadiuvato dagli *Amministratori*$_{\text{G}}$.

2.1.5.2 Descrizione

Lo studio deve analizzare i seguenti punti:

- **Pianificazione:** pianificazione delle attività da svolgere nel corso del progetto che include delle scadenze temporali precise sulle stesse;
- **Rischi:** analisi dettagliata dei rischi che potrebbero insorgere nel corso del progetto e proposta di metodi per affrontarli. Tale analisi include la comprensione della probabilità che i rischi evidenziati si concretizzino e del livello di gravità ad essi associato;
- **Preventivo e Consuntivo:** stima della quantità di lavoro necessaria per ogni fase, sulla base della pianificazione effettuata. A tale stima consegue un preventivo per il costo totale del progetto da aggiornarsi ad ogni fase.

2.1.5.3 Classificazione dei rischi

Ogni rischio verrà catalogato secondo il seguente codice identificativo:

R[Tipologia].[Identificativo]

Dove:

- **Tipologia**: indica il tipo di rischio in esame, e può assumere uno dei seguenti valori:
  - **T**: rischi inerenti le tecnologie;
  - **S**: rischi inerenti gli strumenti;
  - **G**: rischi inerenti il gruppo di progetto;
  - **R**: rischi inerenti i requisiti;
  - **O**: rischi inerenti l'organizzazione.
- **Identificativo**: indica un codice incrementale per il rischio.

2.1.6 Pianificazione della qualità

2.1.6.1 Scopo

Lo scopo della attività è lo studio di una strategia atta a garantire la qualità del materiale prodotto dal gruppo.

2.1.6.2 Descrizione

Per assicurare il conseguimento di una qualità coerente con gli obiettivi fissati dal gruppo, l'attività dovrà definire:

- **Obiettivi di Qualità:** obiettivi di processo e di prodotto a cui attenersi;
- **Politica di qualità:** la strategia generale atta al conseguimento dei succitati obiettivi;
- **Gestione della revisione:** definizione delle modalità di comunicazione delle anomalie e delle procedure di controllo per la qualità di processo.

2.2 Sviluppo

2.2.1 Scopo

Il processo di sviluppo ha il fine di pianificare dettagliatamente e poi realizzare le attività che il gruppo di lavoro deve svolgere per fornire il prodotto richiesto dal capitolato.

2.2.2 Descrizione

Per una corretta implementazione di tale processo le aspettative sono le seguenti:

- Realizzare un prodotto finale *conforme*$_{\mathrm{G}}$ alle richieste della Proponente;
- Realizzare un prodotto finale soddisfacente i test di verifica;
- Realizzare un prodotto finale soddisfacente i test di validazione;
- Fissare gli obiettivi di *sviluppo*$_{\mathrm{G}}$;
- Fissare i vincoli tecnologici;

- Fissare i vincoli di design.

Il processo di sviluppo si svolge in accordo con lo standard ISO/IEC 12207, pertanto si compone delle seguenti attività:

- Analisi dei requisiti;
- Consolidamento dei requisiti e delle tecnologie;
- Progettazione;
- Codifica.

2.2.3 Analisi dei requisiti

2.2.3.1 Scopo

Gli *Analisti*$_{\text{G}}$ devono individuare ed elencare i *requisiti*$_{\text{G}}$ del progetto da realizzare.

2.2.3.2 Descrizione

L'attività di Analisi deve:

- Descrivere il fine del progetto;
- Fissare le funzionalità e i requisiti richiesti dal committente.

Durante l'Analisi dei Requisiti gli *Analisti* analizzano individualmente le varie fonti, quindi, a seguito di una riunione, si confrontano e stilano le varie liste di requisiti suddivise per importanza. Le fonti sono:

- **Capitolato d'appalto:** requisiti emersi dall'analisi del documento fornito dalla Proponente;
- **Verbali esterni:** requisiti emersi a seguito di colloqui con i responsabili della Proponente;
- **Casi d'uso:** requisiti emersi a seguito di uno o più casi d'uso analizzati;
- **Tracciabilità Interna (TI):** requisiti emersi tramite discussioni ed incontri tra gli *Analisti* del gruppo Graphite.

2.2.3.3 Classificazione requisiti

I requisiti devono essere suddivisi per importanza e classificati come segue:

R[Importanza][Tipologia][Codice]

- **Importanza:** Ogni requisito può appartenere solo ad una delle classi di Importanza elencate di seguito:
  - **O (Requisito Obbligatorio):** requisito fondamentale per la corretta realizzazione del progetto;
  - **D (Requisito Desiderabile):** requisito non fondamentale al progetto ma il cui soddisfacimento comporterebbe una maggiore completezza del prodotto;
  - **F (Requisito Facoltativo):** requisito non richiesto per il corretto funzionamento del prodotto ma che se incluso arricchirebbe il progetto. Prima di soddisfare il requisito è necessaria un'analisi di tempi e costi per evitare ritardi nella consegna e/o costi superiori a quelli preventivati.
- **Tipologia:** Di seguito sono riportate le tipologie di requisito:
  - **V:** Identifica un *requisito di vincolo*$_{\mathrm{G}}$;
  - **F:** Identifica un *requisito funzionale*$_{\mathrm{G}}$;
  - **P:** Identifica un *requisito prestazionale*$_{\mathrm{G}}$;
  - **Q:** Identifica un *requisito di qualità*$_{\mathrm{G}}$.
- **Codice:** Ogni requisito è formato da un codice numerico che lo indentifica in modo univoco.

Il codice stabilito secondo la convenzione precedente, una volta associato ad un requisito, non potrà cambiare nel tempo.
Ciascun requisito dovrà inoltre essere accompagnato dalle sue fonti, dalle sue relazioni di dipendenza con altri requisiti e da una descrizione che ne specifichi lo scopo. Sarà inoltre indicata l'importanza.

Esempio:

Codice	Importanza	Descrizione	Fonti
ROF0	Obbligatorio	L'utente può visualizzare la pagina di DeSpeect	UC0 $_{4.2}$ Capitolato

Figura 2.1: Esempio di tabella dei requisiti

2.2.3.4 Classificazione casi d'uso

I *casi d'uso*$_{\text{G}}$ verranno identificati nel seguente modo:

UC[Codice padre]*.[Codice identificativo]

- **Codice padre:** identifica il codice del caso d'uso da cui è stato generato il caso d'uso identificato, se non esiste il campo va tralasciato;
- **Codice identificativo:** identifica il caso d'uso univocamente.

Ogni caso d'uso è inoltre definito secondo la seguente struttura:

- **Attore Principale:** indica gli attori principali (ad esempio l'utente generico) del caso d'uso;
- **Attore Secondario:** indica gli attori secondari (ad esempio entità di autenticazione esterne) del caso d'uso;
- **Descrizione:** riporta una breve descrizione del caso d'uso;
- **Precondizione:** specifica le condizioni che sono identificate come vere prima del verificarsi degli eventi del caso d'uso;
- **Postcondizione:** specifica le condizioni che sono identificate come vere dopo il verificarsi degli eventi del caso d'uso;
- **Scenario Principale:** descrive il flusso degli eventi, a volte attraverso l'uso di una lista numerata o non, specificando i casi d'uso generati;
- **Scenari Alternativi:** specifica casi di errore o eventi non previsti nello scenario principale.

2.2.4 Consolidamento dei requisiti e delle tecnologie

2.2.4.1 Scopo

In questa attività gli *Analisti* devono affinare, ed eventualmente ampliare, i requisiti analizzati nel periodo di Analisi e, prima di iniziare l'attività di progettazione, i *Progettisti* devono condurre un analisi delle tecnologie mirata alla scelta delle più appropiate ai fini del progetto, motivando e dimostrando la scelta mediante *Proof of Concept*$_{\mathrm{G}}$.

2.2.5 Progettazione

2.2.5.1 Scopo

L'attività di Progettazione deve obbligatoriamente precedere la produzione del software e consiste nel descrivere una soluzione del problema che sia soddisfacente per tutti gli *stakeholders*$_{\mathrm{G}}$. Essa serve a garantire che il prodotto sviluppato soddisfi le proprietà e i bisogni specificati nell'attività di analisi.

2.2.5.2 Descrizione

La progettazione permette di:

- Costruire un'architettura logica del prodotto;
- Ottimizzare l'uso delle risorse;
- Garantire una determinata qualità del prodotto, perseguendo la *correttezza per costruzione*: è questo il principio secondo il quale il software deve funzionare correttamente e soddisfare tutti i requisiti e i vincoli perché è stato progettato per farlo e per essere conforme ed *efficace*$_{\mathrm{G}}$. Si trova in netta contrapposizione alla *correttezza per correzione*;
- Organizzare e dividere le parti del progetto in modo da poter ottenere componenti singole e facili da implementare attraverso la codifica.

La progettazione si colloca nel periodo "Progettazione e codifica" dettagliato nel PP (§4.4), qui vengono individuati i design pattern e costruita l'architettura del prodotto realizzando diagrammi di classi e di sequenza: il lavoro svolto rappresenta la *Product Baseline*.
E' compito dei *Progettisti* svolgere l'attività di progettazione, definendo l'architettura logica del prodotto identificando componenti chiare, riusabili e coese, rimanendo nei costi fissati. L'architettura definita deve attenersi alle seguenti qualità:

- **Sufficienza:** soddisfare i requisiti definiti nel documento *Analisi dei Requisiti*;
- **Comprensibilità:** essere capita dagli stakeholders e quindi descritta in modo comprensibile, oltre ad essere tracciabile sui requisiti;
- **Modularità:** essere divisa in parti chiare e distinte, ognuna con la sua specifica funzione;
- **Robustezza:** essere in grado di gestire malfunzionamenti in modo da rimanere operativa di fronte a situazioni erronee improvvise;
- **Flessibilità:** permettere modifiche a costo contenuto nel caso in cui i requisiti dovessero evolversi o se ne dovessero aggiungere di nuovi;
- **Riusabilità:** essere costruita in modo da poter permettere il riutilizzo di alcune sue parti;
- **Efficienza:** soddisfare tutti i requisiti in modo tale da ridurre gli sprechi di tempo e spazio;
- **Affidabilità:** garantire che i servizi esposti siano sempre disponibili, ovvero svolgere i suoi compiti quando viene usata;
- **Disponibilità:** necessitare di tempo ridotto per la manutenzione in modo da garantire un servizio il più continuo possibile;
- **Sicurezza:** essere sicura rispetto ad intrusioni e malfunzionamenti;
- **Semplicità:** prediligere la semplicità, contenendo solo il necessario, rispetto ad una inutile complessità;
- **Incapsulazione:** fare in modo che l'interno delle componenti non sia visibile dall'esterno, seguendo il principio del *data hiding*$_{\text{G}}$;
- **Coesione:** raggruppare le parti secondo l'obiettivo a cui concorrono, in modo da avere maggiore manutenibilità e riusabilità;
- **Basso accoppiamento:** non avere dipendenze indesiderabili.

Tali qualità vengono perseguite attraverso:

- Uso di adeguati design pattern;
- Una progettazione guidata da diagrammi UML delle classi, di sequenza e attività dettagliati e precisi;

- Confronto diretto con la Proponente;
- Appropriata conoscenza e padronanza delle tecnologie.

2.2.5.3 Progettazione architetturale

Conclusa l'attività di Analisi, è compito dei *Progettisti* adottare opportune soluzioni progettuali a problemi ricorrenti. Tali soluzioni devono essere flessibili e consentire una certa libertà d'uso ai *Programmatori*. Al fine di conseguire queste qualità è necessario utilizzare design pattern coerenti con gli obiettivi di progetto, compatibili con le tecnologie in uso e ben documentati. Altresì importante è l'approfondita conoscenza delle tecnologie (maturata tra l'altro durante la realizzazione del *Proof of Concept* nel contesto della *Technology Baseline*) che permette di integrarle efficacemente e limita possibili problemi che potrebbero sorgere durante la codifica.

2.2.5.4 Progettazione in dettaglio

La progettazione in dettaglio deve definire per ogni componente software il progetto della stessa e i relativi test. Quest'ultimi devono essere pensati per garantire il funzionamento del software ed essere contemporaneamente coerenti con le scelte tecnologiche effettuate nell'ambito dell'integrazione continua e delle misurazioni metriche relative alla qualità di prodotto.

2.2.5.5 Diagrammi

La progettazione deve utilizzare le seguenti tipologie di diagrammi UML:

- **Diagrammi delle classi**: illustrano una collezione di elementi che rappresenta un modello come classi e tipi, con i loro contenuti e relazioni;
- **Diagrammi dei package**: raggruppamenti di classi in una unità ad un livello più alto;
- **Diagrammi delle attività**: rappresentano il flusso di operazioni relativo ad un'attività e vengono utilizzati in particolare per descrivere la logica di un algoritmo. Tali diagrammi vengono utilizzati solo se ne sopraggiunge la necessità;
- **Diagrammi di sequenza**: descrivono una determinata sequenza di azioni. Nel diagramma non compaiono scelte, nè flussi alternativi.

L'utilizzo dei succitati diagrammi rispetta la notazione del linguaggio UML v2.0.

2.2.5.6 Linee guida per i progettisti

Al fine di implementare le qualità dell'architettura descritte in §2.2.5.2, ai *Progettisti* viene richiesto di seguire le seguenti linee guida:

- evitare package vuoti, classi e parametri non utilizzati e parametri senza tipo;
- evitare la presenza di dipendenze circolari;
- evitare modifiche ai servizi offerti dalle librerie esterne utilizzate;
- utilizzare ove possibile classi astratte;
- utilizzare nomi significativi per le classi e i metodi in esse contenuti, in modo da facilitare la comprensione immediata di ogni componente;
- assegnare ad ogni metodo e attributo minor visibilità possibile.

2.2.6 Codifica

2.2.6.1 Scopo

Nella seguente sezione sono riportate le norme da seguire durante la programmazione da parte dei *Programmatori*. Lo scopo di queste norme consiste nel dare delle linee guida in modo tale che il codice risulti leggibile e facilmente analizzabile durante la fase di mantenimento, verifica e validazione, migliorando così la qualità del prodotto.

2.2.6.2 Descrizione

L'attività di codifica deve sviluppare il software e i test per le componenti descritti dalla progettazione in dettaglio assicurandosi che sia conforme a quanto definito.

2.2.6.3 Nomenclatura dei file

I file contenenti il codice sorgente devono essere suddivisi in cartelle, in modo tale che la loro struttura rispecchi l'architettura del sistema. I nomi dei file devono essere univoci e in minuscolo.

2.2.6.4 Intestazione dei file

Ogni file contenente codice deve avere la seguente intestazione:

```
/*
*  File : nome file
*  Version : versione file
*  Type : tipo file
*  Date : data di creazione
*  Author : nome cognome autore/i
*  E- mail : email gruppo
*  License : tipo licenza
*/
```

2.2.6.5 Commenti

Ogni metodo del codice del prodotto, al netto delle librerie esterne deve avere un commento la cui struttura segue la seguente sintassi:

```
/**
*  Descrizione del metodo
*  @param { TipoParametro } NomeParametro - Descrizione parametro
*/
```

Nella stesura del commento il *Programmatore* è tenuto a rispettare le seguenti convenzioni:

- Documentare con un commento in linea parti di codice non immediatamente comprensibili;
- Usare frasi di senso compiuto in modo da essere più chiari possibili;
- Porre il commento sopra una riga di codice anziché alla fine di essa (in particolare all'inizio della porzione di codice interessata), per una miglior facilità di lettura.

2.2.6.6 Stile di codifica

Il gruppo aderisce alle *GNU GCC Coding Conventions*, reperibili al seguente link:

https://gcc.gnu.org/codingconventions.html

Segue quindi nelle successive sezioni una descrizione delle principali tra tali convenzioni di codifica.

Linee di codice

Le linee di codice devono essere lunghe al massimo 80 colonne.

Nomi

- **Univocità dei nomi:** classi, metodi e variabili devono avere un nome univoco e quanto più possibile esplicativo della loro funzione al fine di evitare ambiguità;
- **Macro:** i nomi delle macro devono essere tutti in maiuscolo;
- **Classi:** le classi iniziano sempre con una lettera maiuscola;
- **Metodi:** i nomi dei metodi iniziano con una lettera minuscola e se sono composti da più parole le successive devono iniziare con una lettera maiuscola. Le sigle e gli acronimi, all'interno dei nomi dei metodi, vengono trattate come parole;
- **Strutture:** quando le strutture e/o le classi hanno funzioni membro, i dati membro vanno nominati con il prefisso m_, mentre i dati statici con il prefisso s_;
- **Template:** i nomi dei parametri dei template devono usare *CamelCase*$_{\mathrm{G}}$, seguendo lo standard C++;
- **Altri nomi:** devono essere minuscoli e separati dal trattino basso.

Espressioni

Espressione	Convenzione adottata	Convenzione scorretta
not logico	!x	! x
complemento bit a bit	$\sim x$	$\sim x$
meno unario	-x	- x
conversione esplicita	(foo) x	(foo)x
deferenziazione puntatore	*x	* x

Ereditarietà

Per quanto riguarda l'ereditarietà è necessario attenersi ai seguenti accorgimenti:

- È consentita l'ereditarietà singola;

- Deve essere utilizzata l'ereditarietà pubblica per l'interfaccia, cioè le relazioni *"is-a"*;
- Deve essere utilizzata l'ereditarietà privata e protetta per l'implementazione;
- Le gerarchie complesse devono essere evitate;
- L'ereditarietà multipla deve essere trattata con molta attenzione e nel caso venga usata deve essere scritta una documentazione particolarmente chiara dell'intera gerarchia. In generale, è preferibile non usarla.

Definizioni

- **Variabili:** le variabili devono essere definite nel momento del loro primo utilizzo. Possono essere definite e testate simultaneamente nelle espressioni di controllo;
- **Strutture:** la parola chiave *struct* deve essere utilizzata solo per i POD_{G};
- **Classi:** Per la definizione di classe devono essere usate le seguenti regole:
  - **Parola chiave:** la parola chiave *class* deve essere utilizzata solo per i tipi non POD.
  - **Indentazione:** metodi e campi dati membri della classe devono essere indentati di due spazi;
  - **Definizione in singola riga:** se possibile, la classe deve essere definita su una singola riga, come nel seguente esempio:

    ```
    class gnuclass: base {
    ```

    Se ciò non fosse possibile, si deve iniziare la lista delle classi base con i due punti dell'elenco all'inizio della riga successiva, come di seguito mostrato.

    ```
    class a_rather_long_class_name
    : with_a_very_long_base_name, and_another_just_to_make_life_hard {
    ...
    };
    ```

Se l'elenco dovesse superare la lunghezza di una riga, si devono spostare le classi base in eccesso alla riga successiva con il rientro di due spazi, come illustrato nel seguente esempio:

```
class gnuclass
: base1 <template_argument1>, base2 <template_argument1>,
base3 <template_argument1>, base4 <template_argument1> {
....
};
```

- **Ordine degli elementi:** quando si definisce una classe, l'ordine in cui definire i suoi elementi (metodi o campi dati) deve essere il seguente:
  1. Tipi pubblici;
  2. Tipi non pubblici;
  3. Costruttori pubblici;
  4. Distruttore pubblico;
  5. Metodi pubblici;
  6. Campi dati pubblici;
  7. Costruttori non pubblici;
  8. Distruttore non pubblico;
  9. Metodi non pubblici;
  10. Campi dati non pubblici.

  Se vincoli semantici richiedessero un diverso ordine di dichiarazione, se deve cercare di minimizzare la potenziale confusione (per esempio attraverso appositi commenti al codice).
- **Apertura e chiusura** una definizione di classe deve essere aperta con una parentesi graffa sinistra in linea e chiusa con una parentesi graffa destra, punto e virgola, commento di chiusura facoltativo e una nuova riga, come illustrato nel seguente esempio:

```
class gnuclass: base {
...
}; // class gnuclass
```

- **Membri di una classe:** tutti i membri di una classe vanno definiti al di fuori della definizione della stessa, cioè non ci sono corpi di funzioni o inizializzatori di campi dati all'interno della definizione della classe. È inoltre necessario attenersi alle seguenti indicazioni:

– È preferibile scrivere l'intera definizione di un metodo su di una singola riga, come nel seguente esempio:

```
gnuclass::gnuclass () : base_class () {
...
};
```

Se ciò non fosse possibile, si deve iniziare la lista di inizializzazione con i due punti dell'elenco all'inizio della riga successiva, come di seguito mostrato.

```
gnuclass::gnuclass ()
: base1 (), base2 (), member1 (), member2 (), member3 (), member4 () {
...
};
```

Se l'elenco dovesse superare la lunghezza di una riga, si devono spostare gli inizializzatori in eccesso alla riga successiva con il rientro di due spazi, come illustrato nel seguente esempio:

```
gnuclass::gnuclass ()
: base1 (some_expression), base2 (another_expression),
member1 (my_expressions_everywhere) {
...
};
```

– Se il nome di una funzione è sufficientemente lungo da permettere che il primo parametro di funzione con il suo tipo superi 80 caratteri, deve apparire nella riga successiva preceduto da quattro spazi di indentazione, come illustrato nell'esempio seguente:

```
void
very_long_class_name::very_long_function_name (
very_long_type_name arg)
{
```

– Se la somma delle lunghezze del qualificatore della classe e del nome della funzione supera le 80 colonne, la riga va interrotta per andare a capo con l'operatore *"::"*, come illustrato nell'esempio seguente:.

```
void
very_long_template_class_name <with, a, great, many, arguments>
```

```
::very_long_function_name (
very_long_type_name arg)
{
```

Costruttori

Ogni costruttore deve inizializzare i membri dei dati nell'elenco di inizializzazione dei membri, piuttosto che nel corpo del costruttore.

Distruttori

Una classe con funzioni virtuali deve necessariamente avere un distruttore virtuale.

Conversioni

I costruttori a singolo argomento devono essere sempre dichiarati esplicitamente. Gli operatori di conversione devono essere il quanto più possibile evitati.

Overloading

- **Overloading delle funzioni:** è consentito fare *overloading*$_{\text{G}}$ di funzioni. Non tuttavia è consigliato farlo per le funzioni virtuali;
- **Overloading degli operatori:** è consentito fare overloading degli operatori. L'overloading degli operatori, ad eccezione dell'operatore di chiamata, non deve essere utilizzato per implementazioni costose.

Argomenti di default

Gli argomenti di default sono un altro tipo di overloading di funzioni, per cui vi si applicano le medesime regole. Gli argomenti di default devono sempre essere valori POD, cioè non possono eseguire costruttori. Le funzioni virtuali non devono avere argomenti di default.

Funzioni inline

Sono preferite funzioni non in linea, a meno che non si abbia prova che la versione in linea della stessa funzione sia significativamente più piccola o meno impattante sulle prestazioni.

Template

Una dichiarazione che segue un elenco di parametri del template non dovrebbe avere indentazione aggiuntiva, ed è preferito il typename sulla classe nella lista di parametri del template. Per non gravare eccessivamente sulle prestazioni, si raccomanda di fare uso parsimonioso dei template.

Namespace

I namespace sono incoraggiati, tutte le librerie separabili devono avere un unico namespace globale. I nomi delle directory annidate devono essere associati a namespace annidati quando possibile. I file header non devono avere direttive *using*.
Un namespace va aperto con il nome seguito da una parentesi graffa sinistra e una nuova riga e chiuso con una parentesi graffa destra, un commento di chiusura facoltativo e una nuova riga, come evidenziato nel seguente esempio:

```
namespace gnutool {
...
} // namespace gnutool
```

Le definizioni all'interno del corpo di un namespace non sono indentate.

Extern

è preferito un blocco extern al posto di un qualificatore di dichiarazione.
Un blocco extern va aperto e chiuso nel modo seguente:

```
extern "C" {
...
} // extern "C"
```

Le definizioni all'interno del corpo di un blocco extern non sono indentate.

3. Processi di supporto

3.1 Documentazione

3.1.1 Scopo

Lo scopo del processo di documentazione è descrivere l'insieme di regole adottate dal gruppo per la redazione della documentazione di progetto. Il gruppo si prefigge di definire in questo capitolo norme atte alla stesura di una documentazione il più possibile formale, corretta, coerente e coesa.

3.1.2 Produzione Documentale

Durante il corso del progetto è richiesta la stesura dei seguenti documenti:

Tabella 3.1: Tabella produzione documentale

Processo	**Attività**	**Documento**
Fornitura	Analisi dei capitolati	Studio di fattibilità
Fornitura	Definizione del ciclo di sviluppo	Piano di Progetto
Fornitura	Pianificazione della qualità	Piano di Qualifica
Sviluppo	Analisi dei requisiti	Analisi dei requisiti

Tabella 3.1: Tabella produzione documentale

Processo	**Attività**	**Documento**
Sviluppo	Progettazione	Product Baseline Prima bozza manuali
Sviluppo	Codifica	Manuali
Gestione di progetto	Gestione degli incontri	Verbali

Durante l'intero svolgimento del progetto verrà redatto e incrementato un Glossario.

3.1.3 Struttura dei documenti

3.1.3.1 Template

Il gruppo utilizza un template LaTeX appositamente codificato per uniformare l'aspetto dei documenti e velocizzare il processo di documentazione. Il template include frontespizio, registro delle modifiche, indice, piè di pagina e intestazione, elementi del documento approfonditi di seguito.

3.1.3.2 Frontespizio

Il frontespizio di ogni documento è strutturato nel seguente modo:

- **Logo del gruppo:** il logo identificativo del gruppo;
- **Nome del documento:** indica il nome del documento (ex: Norme di progetto);
- **Informazioni sul documento:** tabella riassuntiva indicante le seguenti informazioni di spicco sul documento:
  - **Versione:** indica un numero che identifica la versione corrente del documento. La struttura del numero di versionamento è approfondita di seguito in questa sezione;

- **Data di approvazione:** indica l'ultima data in cui il documento è stato approvato;
- **Responsabile:** indica il Responsabile che ha approvato il documento;
- **Redattori:** indica il sottoinsieme dei membri del gruppo che ha partecipato alla redazione del documento;
- **Verificatori:** indica il sottoinsieme dei membri del gruppo che ha partecipato alla verifica del documento;
- **Distribuzione:** indica a chi è destinato il documento;
- **Uso:** indica se l'uso del documento è interno o esterno al gruppo;
- **Recapito:** la mail di recapito del gruppo

3.1.3.3 Nomenclatura di versionamento dei documenti

Ogni documento è accompagnato da un numero di versionamento, indicato nella tabella *Informazioni sul documento*, dove ogni versione corrisponde ad una riga nel registro delle modifiche, che è espresso nel modo seguente:

$$\mathbf{v}\ \left\{\mathbf{A}\right\}.\left\{\mathbf{B}\right\}.\left\{\mathbf{C}\right\}$$

Dove:

- **A:** è l'indice principale. Viene incrementato dal *Responsabile di Progetto* all'approvazione del documento e corrisponde al numero di revisione.
- **B:** è l'indice di verifica. Viene incrementato dal *Verificatore*$_{\mathrm{G}}$ ad ogni verifica. Quando viene incrementato A, riparte da 0.
- **C:** è l'indice di modifica. Viene incrementato dal redattore del documento ad ogni modifica. Quando viene incrementato B, riparte da 0.

3.1.3.4 Registro delle modifiche

Dopo il frontespizio segue il registro delle modifiche, che cataloga le modifiche apportate al documento durante il suo sviluppo indicando per ognuna:

- Versione del documento dopo la modifica;
- Data della modifica;
- Nome e cognome dell'autore della modifica;
- Breve descrizione della modifica.

3.1.3.5 Indice

Ogni documento possiede un indice che ne agevola la consultazione e permette una visione generale degli argomenti trattati nello stesso. L'indice è strutturato gerarchicamente ed è collocato dopo il registro delle modifiche.

3.1.3.6 Intestazione

Fatta eccezione per il frontespizio, tutte le pagine del documento contengono un'intestazione. Questa è costituita da:

- **Logo del gruppo:** il logo identificativo del gruppo Graphite, collocato a sinistra;
- **Titolo del capitolo:** indicazione del titolo del capitolo corrente, collocata a destra.

3.1.3.7 Piè di pagina

Fatta eccezione per il frontespizio, tutte le pagine del documento contengono un piè di pagina. Questo è costituito da:

- **Nome documento e versione:** indica il nome del documento e la sua versione attuale, collocate a sinistra;
- **Numero di pagina:** numerazione progressiva delle pagine, collocato a destra.

3.1.3.8 Note a piè di pagina

In caso di presenza in una pagina interna di note da esplicare, esse vanno indicate nella pagina corrente, in basso a sinistra. Ogni nota deve riportare un numero, corrispondente alla relativa parola o frase che la riferisce, e una descrizione.

3.1.3.9 Contenuto principale

Il contenuto principale del documento è organizzato secondo la seguente struttura gerarchica:

1. Capitolo;
2. Sezione;
3. Sottosezione;
4. Sottosottosezione.

3.1.4 Norme tipografiche

3.1.4.1 Stile del testo

- **Glossario:** ogni parola contenuta nel glossario deve essere marcata, solo alla sua prima occorrenza in ogni documento, in corsivo e con una G maiuscola a pedice. Tale formattazione viene assicurata dalla macro *glossario*{*termine*}{*riferimento al glossario*} che stampa il termine con la giusta formattazione ($termine_{\mathrm{G}}$) e controlla che il suo riferimento sia presente nel glossario, restituendo un errore di compilazione in caso contrario, il termine di riferimento è la parola in minuscola senza accenti;
- **Grassetto:** il grassetto viene applicato ai titoli e agli elementi di un elenco puntato seguiti da una descrizione. Esso può inoltre essere usato per mettere in particolare risalto termini significativi;
- **Corsivo:** il corsivo viene utilizzato per:
  - citazioni;
  - termini di glossario;
  - attività del progetto;
  - ruoli del progetto;
  - riferimenti ad altri documenti.

  Esso può inoltre essere usato per mettere in particolare risalto termini particolari solitamente poco usati o conosciuti;
- **Maiuscolo:** il maiuscolo viene usato solo per gli acronimi.

3.1.4.2 Elenchi puntati

Gli elenchi puntati servono ad esprimere in modo sintetico un concetto, evitando frasi lunghe e dispersive. Ogni voce di un elenco puntato termina con un punto e virgola, ad eccezione dell'ultima, che termina invece con un punto.

3.1.4.3 Formati

- **Date:**

  AAAA-MM-GG

  - **AAAA:** rappresenta l'anno in cifre per intero;

  - **MM:** rappresenta il mese in cifre;
  - **GG:** rappresenta il giorno del mese in cifre.

- **Orari:**

  HH:MM

  - **HH:** rappresenta l'ora;
  - **MM:** rappresenta i minuti.

- **Nomi ricorrenti:**

  - **Ruoli di progetto:** ogni nome di ruolo di progetto viene scritto in corsivo e con l'iniziale maiuscola;
  - **Nomi dei documenti:** ogni nome di documento viene scritto in corsivo e con l'iniziale di ogni parola che non sia un articolo maiuscola;
  - **Nomi propri:** ogni nome proprio di persona deve essere scritto nella forma *Nome Cognome*.

- **Link:** i link esterni devono essere scritti attraverso il comando LaTeX *url* e sono distinti dal colore blu. I link interni, per esempio quelli dell'indice, non vengono evidenziati.

3.1.4.4 Sigle

È previsto l'utilizzo delle seguenti sigle:

- **AR:** *Analisi dei Requisiti*;
- **PP:** *Piano di Progetto*;
- **NP:** *Norme di Progetto*;
- **SF:** *Studio di Fattibilità*;
- **PQ:** *Piano di Qualifica*;
- **TB:** *Technology Baseline*$_{\mathrm{G}}$;
- **MU:** *Manuale utente*$_{\mathrm{G}}$;
- **MS:** *Manuale sviluppatore*$_{\mathrm{G}}$;
- **PB:** *Product Baseline*$_{\mathrm{G}}$;

- **PoC:** *Proof of Concept*;
- **RR:** Revisione dei requisiti;
- **RP:** Revisione di progettazione;
- **RQ:** Revisione di qualifica;
- **RA:** Revisione di accettazione;
- **Re:** *Responsabile di Progetto*;
- **Am:** *Amministratore di Progetto*;
- **An:** *Analista*;
- **Pt:** *Progettista*;
- **Pr:** *Programmatore*$_{\mathrm{G}}$;
- **Ve:** *Verificatore*.

3.1.5 Elementi grafici

3.1.5.1 Tabelle

Ogni tabella è corredata da una didascalia in cui compare il suo numero identificativo (per agevolarne il tracciamento) ed una breve descrizione del suo contenuto.

3.1.5.2 Immagini

Ogni immagine deve essere centrata e separata dai paragrafi a lei precedenti e successivi. Le immagini sono corredate da una didascalia analoga a quella delle tabelle. Tutti i diagrammi vengono inseriti come immagini.

3.1.6 Classificazione dei documenti

3.1.6.1 Documenti informali

Tutte le versioni dei documenti che non siano state approvate dal *Responsabile di Progetto* sono ritenute informali e, in quanto tali, sono considerate esclusivamente ad uso interno.

3.1.6.2 Documenti formali

Una versione di un documento viene considerata formale quando è stata approvata dal *Responsabile di Progetto*. Solo i documenti formali possono essere distribuiti all'esterno del gruppo.

3.1.6.3 Documenti interni

Un documento viene considerato interno quando il suo utilizzo è destinato al solo gruppo Graphite.

3.1.6.4 Documenti esterni

Un documento viene considerato esterno quando il documento viene condiviso con i Committenti e con la Proponente.

3.1.6.5 Verbali

Per ogni incontro interno o esterno è prevista la redazione di un verbale, contenente le seguenti informazioni:

- **Informazioni sull'incontro:**
  - **Luogo**: indica il luogo in cui si svolge la riunione. In caso di videochiamate, questo campo può essere riempito con la voce *"Videochiamata"* e/o con il punto di ritrovo in cui si è svolta (ex: Aula AC150, Torre Archimede (Videochiamata));
  - **Data:** indica la data in cui si è svolto l'incontro;
  - **Orari:** indica orari di inizio e di fine dell'incontro;
  - **Partecipanti del gruppo:** indica i membri del gruppo partecipanti all'incontro;
  - **Partecipanti esterni:** indica, se dovessero essercene, eventuali partecipanti esterni presenzianti all'incontro.
- **Ragioni dell'incontro:** indica le ragioni che hanno spinto ad indire l'incontro in esame. In questa sezione viene inserito l'eventuale ordine del giorno;
- **Resoconto:** contiene annotazioni riguardo gli argomenti discussi e capisaldi dell'incontro;

- **Tracciamento delle decisioni:** indica eventuali decisioni prese durante l'incontro, tracciate mediante il seguente codice:

  $$V[Tipologia] - [Dataincontro].[ID]$$

  Dove:

  - **Tipologia:** indica se il verbale è interno oppure esterno;
  - **Data incontro:** indica la data in cui si è svolto l'incontro in formato *AAAA-MM-GG*;
  - **ID:** è un codice identificativo numerico incrementale.

I verbali dovranno essere approvati dal *Responsabile di Progetto* al termine di ogni incontro.

3.1.7 Ciclo di vita del documento

Ogni documento non formale completato deve essere sottoposto al *Responsabile di Progetto*, che si occupa di incaricare i *Verificatori* di controllarne la correttezza del contenuto e della forma. Se vengono individuati degli errori, i *Verificatori* li riportano al *Responsabile di Progetto*, che a sua volta incarica il redattore del documento di correggerli. Questo ciclo va ripetuto fino a che il documento non è considerato corretto. Successivamente esso viene sottoposto al Responsabile di Progetto, che può o meno approvarlo. Quando approvato, il documento è da considerarsi formale. In caso contrario il *Responsabile di Progetto* deve comunicare le motivazioni per cui il documento non è stato approvato, specificando le modifiche da apportare.

3.1.8 Nomenclatura dei documenti

I documenti formali, fatta eccezione per i Verbali di riunione, adottano il seguente sistema di nomenclatura:

$$NomeDelDocumentovX.Y.Z$$

Dove:

- **NomedelDocumento:** indica il nome del documento, senza spazi con lettera maiuscola per ogni parola che non sia un articolo o una preposizione;
- **vX.Y.Z:** indica l'ultima versione del documento approvata dal *Responsabile di Progetto*.

3.1.9 Formato dei file

I file legati alla documentazione vengono salvati in formato .tex durante il loro ciclo di vita. Quando un documento raggiunge lo stato di "Approvato" viene creato un file in formato PDF contenente la versione del documento approvata dal *Responsabile*.

3.2 Gestione della configurazione

3.2.1 Versionamento

Ogni componente versionabile del progetto, nonché la documentazione formale, è versionata mediante la tecnologia *Git*$_{\text{G}}$, nello specifico utilizzando il servizio gratuito GitHub. Per la condivisione di materiale informale e/o non versionabile si predilige invece una cartella condivisa sullo spazio cloud offerto da *Google Drive*$_{\text{G}}$.

3.2.2 Controllo della configurazione

Per identificare e tracciare le richieste di cambiamenti, analizzare e valutare le modifiche effettuate e approvare o meno quelle proposte, viene utilizzato il sistema di *issue*$_{\text{G}}$ di Github. Le issue riportano nella loro descrizione la ragione della modifica effettuata, nonché la o le *commit*$_{\text{G}}$ ad esse associate ed eventuali ulteriori issue correlate.

3.2.3 Stato della configurazione

Successivamente ad ogni revisione, l'ultima commit effettuata viene marcata con un'etichetta di versione (essa costituirà la nuova *baseline*$_{\text{G}}$ di riferimento). L'esito delle revisioni vengono riportate in appendice al PQ con le relative correzioni da apportare ai prodotti oggetto della revisione.

3.2.4 Rilasci e consegne

I rilasci e le consegne dei prodotti software e della documentazione correlata sono sottoposti a verifica e validazione. Prima di ogni revisione viene consegnato al *Committente*, secondo tempistiche stabilite dal PP e mezzi concordati, un archivio contenente tutta la documentazione richiesta in formato *PDF*$_{\text{G}}$. La consegna è accompagnata da una Lettera di Presentazione, anch'essa inclusa nell'archivio.

3.3 Gestione della qualità

3.3.1 Scopo

Il processo di gestione della qualità è finalizzato al conseguimento degli obiettivi di qualità definiti dal gruppo per garantire uno sviluppo efficace ed efficiente del prodotto richiesto.

3.3.2 Definizione degli standard qualitativi di riferimento

Vengono di seguito sinteticamente esposti gli standard di qualità a cui il gruppo intende aderire e le motivazioni di tale scelta.

3.3.2.1 Qualità di processo - ISO/IEC 15504

ISO/IEC 15504$_{\mathrm{G}}$, anche noto come *SPICE*$_{\mathrm{G}}$, è lo standard scelto per la definizione degli obiettivi di processo. Si rimanda all'appendice §A.1 del PQ per un'approfondita descrizione di tale standard. La scelta di SPICE è motivata dal fatto che esso fornisce gli strumenti utili a valutare la qualità di processo, parametro da tenere in grande considerazione per il conseguimento di un prodotto qualitativamente valido entro tempi prestabiliti. Per poter applicare correttamente SPICE, viene utilizzato il *ciclo di Deming* o *ciclo di PDCA*$_{\mathrm{G}}$. Si rimanda all'appendice §A.3 del PQ per un'approfondita descrizione del ciclo di Deming. Tale ciclo definisce un metodo di controllo orientato al miglioramento continuo del livello qualitativo dei processi, evitando nel contempo regressioni. Il ciclo di Deming si applica solo conoscendo lo stato di maturità attuale dei processi di interesse, definendo specifici obiettivi di miglioramento, e studiando i risultati delle azioni migliorative sperimentate. Esistono dunque stringenti precondizioni alla sua applicabilità, ovvero l'attuazione di processi ripetibili e misurabili, qualità di processo che il gruppo ha intenzione di perseguire. L'affiancamento dello standard ISO al ciclo PDCA permette di:

- Misurare costantemente le performance di processo;
- Perseguire un miglioramento continuo di tali performance;
- Rispettare tempi e costi indicati nel PP.

3.3.2.2 Qualità di prodotto - ISO/IEC 9126

ISO/IEC 9126$_{\text{G}}$ è lo standard scelto per la definizione degli obiettivi di prodotto. Si rimanda all'appendice §A.2 del PQ per un'approfondita descrizione di tale standard. La scelta di ISO/IEC 9126 è motivata dal fatto che esso definisce criteri di applicazione nell'ambito di metriche per la qualità interna esterna e in uso del software, qui approfondite in §3,3.4 *Misure e metriche* e successive, utili a valutare il grado di raggiungimento degli obiettivi prefissati. I prodotti realizzati durante lo svolgimento del progetto sono di due tipologie:

- **Documentazione:** deve essere leggibile, comprensibile e corretta dal punto di vista ortografico, sintattico e semantico.
- **Software:** deve avere le seguenti caratteristiche:
  - Possedere funzionalità che soddisfino i requisiti fissati;
  - *Manutenibilità*$_{\text{G}}$;
  - Essere ampiamente testato;
  - *Robustezza*$_{\text{G}}$;

3.3.3 Classificazione degli obiettivi di qualità

Gli obiettivi di qualità stabiliti nel PQ rispettano la seguente notazione:

OQ[Tipo][Oggetto]*[ID]: [Nome]

Dove:

- **Tipo:** indica se l'obiettivo si riferisce a prodotti o a processi. Può assumere i valori:
  - **P:** per indicare i processi;
  - **PP:** per indicare i prodotti;
- **Oggetto:** indica, per gli obiettivi di prodotto, se si riferisce a documentazione o a software. Può assumere i valori:
  - **D:** per indicare i documenti;
  - **S:** per indicare il software;

- **ID:** identifica univocamente l'obiettivo tramite un codice numerico incrementale;
- **Nome:** titolo dell'obiettivo;
- **Descrizione:** breve descrizione dell'obiettivo.

3.3.4 Misure e metriche

Classificazione delle metriche

Le metriche stabilite in questo documento e riferite nel PQ rispettano la seguente notazione:

M[Tipo][Oggetto]*[ID]: [Titolo]

Dove:

- **Tipo:** indica se la metrica si riferisce a prodotti o a processi. Può assumere i valori:
  - **P:** per indicare i processi;
  - **PP:** per indicare i prodotti;
- **Oggetto:** indica, per le metriche di prodotto, se si riferisce a documentazione o a software. Può assumere i valori:
  - **D:** per indicare i documenti;
  - **S:** per indicare il software;
- **ID:** identifica univocamente la metrica tramite un codice numerico incrementale;
- **Nome:** titolo della metrica.

Dettaglio delle metriche

Segue dunque l'esposizione nel dettaglio delle metriche applicate e delle rispettive scale di riferimento e metodologie di calcolo. Le metriche sono univocamente identificate da un codice che ne agevola il tracciamento. La presentazione di ogni metrica si articolerà nelle seguenti sottosezioni:

- **Nome:** nome descrittivo della metrica;

- **Codice:** codice univocamente identificativo della metrica;
- **Descrizione:** illustrazione della metrica e del suo significato;
- **Modalità di calcolo:** dove definito, viene riportato il modo in cui la metrica viene calcolata.

3.3.5 Metriche per i processi

3.3.5.1 ISO/IEC 15504 (SPICE)

- **Codice:** MP001;
- **Descrizione:** La metrica definita dallo standard ISO/IEC 15504 viene utilizzata alla fine di ogni periodo per monitorare e valutare la qualità dei processi impiegati. Tale standard è illustrato dettagliatamente in Appendice §A.1 del PQ;

3.3.6 Metriche per i documenti

3.3.6.1 Errori ortografici corretti

- **Codice:** MPPD001;
- **Descrizione:** La documentazione prodotta deve essere corretta dal punto di vista ortografico e grammaticale;
- **Modalità di calcolo:** Controllo ortografico di TexStudio e del *Verificatore*;

3.3.6.2 Gulpease

- **Codice:** MPPD002;
- **Descrizione:** L'*indice Gulpease*$_{\mathrm{G}}$ 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. Tale indice considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere;
- **Modalità di calcolo:** L'indice Gulpease si calcola con la seguente formula:

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

Il risultato è un valore compreso nell'intervallo tra 0 e 100, dove il valore 100 indica la più alta leggibilità, mentre 0 la più bassa. In generale risulta che testi con indice:

- 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;
- inferiore a 40 sono difficili da leggere per chi ha un diploma superiore.

Nonostante l'importanza riconosciuta all'indice Gulpease, il gruppo tiene conto delle seguenti considerazioni:

- l'indice non rileva la comprensibilità del testo, il cui contenuto potrebbe essere totalmente incomprensibile ma ottenere ugualmente un buon valore;
- ai fini dei documenti vengono necessariamente usati termini tecnici possibilmente ostici ma insostituibili;
- il gruppo decide di prediligere la chiarezza e la precisione del contenuto dei documenti anche qualora questa dovesse minarne (in modo superficiale) la leggibilità.

3.3.7 Metriche per il software

3.3.7.1 Copertura requisiti obbligatori

- **Codice:** MPPS001;
- **Descrizione:** La copertura dei requisiti obbligatori permette di monitorare la percentuale di requisiti obbligatori soddisfatti;
- **Modalità di calcolo:** La copertura dei requisiti obbligatori si calcola con la seguente formula:

$$CRO = \frac{\textit{numero requisiti obbligatori soddisfatti}}{\textit{totale dei requisiti obbligatori}}$$

3.3.7.2 Copertura del codice

- **Codice:** MPPS002;
- **Descrizione:** La copertura del codice (o code coverage) indica la percentuale di istruzioni, rispetto al totale, che vengono eseguite durante i test. Maggiore è il numero delle istruzioni testate e maggiore è la possibilità di individuare e risolvere errori di codifica. Un valore troppo basso indica che molte istruzioni non vengono testate, divenendo possibile causa di anomalie.
- **Modalità di calcolo:** La copertura del codice si calcola con la seguente formula:

$$CC = \frac{numero\ di\ righe\ di\ codice\ testate}{totale\ delle\ righe\ di\ codice}$$

3.3.7.3 Percentuale superamento test

- **Codice:** MPPS003;
- **Descrizione:** La metrica indica la percentuale di test implementati superati dal oggetto del test;
- **Modalità di calcolo:** Percentuale superamento test si calcola con la seguente formula:

$$PST = \frac{numero\ test\ superati}{totale\ dei\ test\ implementati}$$

3.3.7.4 Numero di parametri per metodo

- **Codice:** MPPS004;
- **Descrizione:** Il numero di parametri di un metodo incide sulla sua complessità. Un valore elevato può indicare che esso si fa carico di una quantità eccessiva di responsabilità e che potrebbe essere decomposto in metodi più semplici;
- **Modalità di calcolo:** controllo del *Verificatore*.

3.3.7.5 Numero di attributi per classe

- **Codice:** MPPS005;
- **Descrizione:** Il numero di attributi di una classe incide sulla sua complessità. Un valore elevato può indicare che essa si fa carico di una quantità eccessiva di responsabilità e che potrebbe essere decomposta in classi più semplici;
- **Modalità di calcolo:** controllo del *Verificatore*.

3.3.7.6 Numero di metodi per classe

- **Codice:** MPPS006;
- **Descrizione:** Il numero di metodi di una classe incide sulla sua complessità. Un valore elevato può indicare che essa si fa carico di una quantità eccessiva di responsabilità e che potrebbe essere decomposta in classi più semplici;
- **Modalità di calcolo:** controllo del *Verificatore*.

3.3.7.7 Complessità ciclomatica

- **Codice:** MPPS007;
- **Descrizione:** La *Complessità ciclomatica*$_{\text{G}}$ (o complessità condizionale) è una metrica software utilizzata per misurare la complessità di un programma. Nello specifico, essa stima la complessità di funzioni, moduli, metodi o classi di un programma. Il valore della Complessità Ciclomatica rappresenta quanto un metodo è complesso, tramite la misura del numero di cammini linearmente indipendenti che attraversano il *grafo*$_{\text{G}}$ di flusso di controllo. 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. Durante l'identificazione dei test, la Complessità Ciclomatica è inoltre utile per determinare il numero di casi di test necessari: l'indice di Complessità Ciclomatica è infatti un limite superiore al numero di test necessari per raggiungere la completa *copertura del codice*$_{\text{G}}$ (o *code coverage*) del modulo testato. Un valore troppo elevato indica un'eccessiva complessità del codice, cui consegue una complessa manutenzione, al contrario un valore ridotto potrebbe indicare una scarsa efficienza dei metodi;

- **Modalità di calcolo:** Rappresentando un programma con il grafo di controllo del flusso, un modo di calcolare il numero ciclomatico *V(G)* è il seguente:

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

Dove:

  - **N:** indica il numero di nodi del grafo;
  - **E:** indica il numero di archi del grafo;
  - **P:** indica il numero di componenti connesse del grafo;

3.3.7.8 Grado di instabilità

- **Codice:** MPPS008;
- **Descrizione:** Il livello di stabilità di un software indica il rischio con una modifica apportata ad un $package_{\mathrm{G}}$ influenzi il funzionamento dell'intero sistema;
- **Modalità di calcolo:** Il valore di instabilità è calcolato usando il grado di $accoppiamento_{\mathrm{G}}$ tramite la seguente formula:

$$Instabilità = \frac{Accoppiamento\ efferente}{Accoppiamento\ efferente + Accoppiamento\ afferente}$$

Dove:

  - **Accoppiamento efferente:** indica 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;
  - **Accoppiamento afferente:** indica il grado di utilità delle classi di un package verso classi esterne ad esso. Un valore troppo basso potrebbe essere indice di scarsa utilità della classe, dato che poche delle sue funzionalità sono usate al suo esterno. Al contrario, un valore troppo elevato può indicare un livello di dipendenza pericoloso del package in esame, che può portare ad effetti indesiderati nelle classi esterne in caso di modifiche. Un valore elevato, d'altro canto, non è necessariamente indice di un errore di progettazione bensì potrebbe indicare semplicemente la criticità del package in esame.

3.3.7.9 Altezza albero della gerarchia

- **Codice:** MPPS09;
- **Descrizione:** L'altezza degli alberi di gerarchia delle classi va limitata così da limitare l'accoppiamento. Preferibilmente, le classi dovranno dipendere solo da classi astratte e potranno implementare una o più interfacce. Viene invece proibito l'uso di ereditarietà multipla.
- **Modalità di calcolo:** controllo del *Verificatore*.

3.3.7.10 Rapporto tra linee di codice e linee di commento

- **Codice:** MPPS010;
- **Descrizione:** Il rapporto tra linee di codice (escluse le righe vuote) e linee di commento rappresenta un indice utile a stimare la manutenibilità del codice sorgente. Un rapporto troppo basso indica una scarsa documentazione del codice scritto, cui consegue una possibile elevata complessità nel manutenerlo.
- **Modalità di calcolo:**

$$\frac{\textit{numero linee di commento}}{\textit{numero linee di codice (non vuote)}}$$

3.3.8 Politica della qualità

Il gruppo intende perseguire gli obiettivi di qualità definiti nel PQ (§2.1) attraverso:

- Applicazione del principio del miglioramento continuo, da effettuarsi per tutta la durata del progetto a livello personale e di team;
- Stretta collaborazione con la proponente che si sviluppi in rapporti caratterizzati da una cooperazione attenta e propositiva orientata alla fornitura di un servizio di qualità;
- Utilizzo di appositi strumenti automatici;
- Costante verifica, validazione e conseguente miglioramento del sistema qualità attraverso l'istanziazione di processi specializzati supportati da strumenti appropriati;

- Lauto investimento temporale in termini di formazione dei membri del gruppo attraverso studio personale e condivisione delle conoscenze specifiche dei membri.

Segue il dettaglio delle strategie di conseguimento della qualità per ogni obiettivo definito dal gruppo.

3.3.8.1 Strategie di conseguimento degli obiettivi di qualità

Vengono di seguito listate le strategie di perseguimento della qualità per gli obiettivi definiti nel PQ (§2.1).

3.3.8.2 OQP001: Miglioramento continuo

Tale obiettivo viene perseguito mediante la costante applicazione del ciclo di Deming, descritto in dettaglio nell'appendice §A.3 del PQ.

3.3.8.3 OQPPD001: Leggibilità dei documenti

Tale obiettivo viene perseguito mediante il rilevamento degli errori ortografici in real time di *TexStudio*$_{\text{G}}$ e istruendo il personale ad uno stile di scrittura sintetica e corretta mediante il processo di formazione del personale (approfondito in §4.6 di questo documento).

3.3.8.4 OQPPS001: Implementazione requisiti obbligatori

Per ogni incremento, l'insieme dei task da assegnare ai membri del gruppo durante lo sviluppo viene pianificato in modo tale che il suo completamento porti al soddisfacimento di almeno un requisito obbligatorio.

3.3.8.5 OQPPS002: Copertura del codice

Tale obiettivo viene perseguito mediante l'applicazione dei test definiti in fase di progettazione attuata tramite lo strumento automatico *Travis*$_{\text{G}}$, che assicura che il codice caricato sulla repository superi tali test, e monitorato attraverso i report automatici generati da *coveralls.io* (tali tecnologie sono approfondite in §4.7.4.2 di questo documento).

3.3.8.6 OQPPS003: Manutenzione e comprensione del codice

Tale obiettivo viene perseguito mediante la messa in pratica delle norme relative alla codifica definite nelle NP, garantita dagli strumenti di integrazione continua approfonditi in §3.3.10 di questo documento.

3.3.9 Definizione delle anomalie

L'identificazione delle anomalie ha come scopo la loro risoluzione e rappresenta un dato rilevante per il monitoraggio dello stato del prodotto. Distinguere e catalogare le anomalie permette di organizzare (in particolar modo di prioritizzare) e affinare le correzioni da attuare per eliminarle. Di seguito vengono quindi elencate le definizioni di anomalie (secondo glossario IEEE 610.12-90) adottate dal gruppo:

- **Error:** differenza riscontrata tra risultato di una computazione e valore teorico atteso;
- **Fault:** un passo, un processo o un dato definito in modo erroneo che corrisponde a quanto viene definito come bug;
- **Failure:** il risultato di un fault;
- **Mistake:** azione umana che produce un risultato errato.

Nello specifico, rappresentano un'anomalia:

- La violazione delle norme tipografiche definite nelle NP;
- La presenza di contenuti non pertinenti l'argomento trattato o il documento in cui risiedono;
- Errori di codifica;
- Il mancato rispetto dei valori di accettazione fissati in questo documento;
- Incongruenze tra il prodotto e le sue funzionalità determinate nell'*Analisi dei Requisiti*.

3.3.10 Integrazione Continua

L'integrazione continua è un metodo di sviluppo software in cui gli sviluppatori aggiungono regolarmente modifiche al codice in un repository centralizzato, quindi la creazione di build e i test vengono eseguiti per mezzo di strumenti automatici e contestualmente si eseguono misurazioni atte a certificare la qualità del software. Gli obiettivi principali dell'integrazione continua sono:

- Individuazione e risoluzione di bug con maggiore tempestività;
- Incremento della qualità del software;
- Riduzione del tempo richiesto per convalidare e pubblicare nuovi aggiornamenti.

Il gruppo intende fare uso dell'integrazione continua al fine di perseguire i succitati obiettivi. Segue la presentazione del flusso di integrazione continua (si noti che gli strumenti citati sono approfonditi in §4.7)

3.3.10.1 Flusso di integrazione continua

Il flusso dell'integrazione continua instaurata è il seguente:

1. Scrittura del codice all'interno dell'ambiente di sviluppo;
2. Push dei dati modificati verso il repository *GitHub*;
3. Esecuzione automatica della build e della suite di test mediante *Travis CI*, con eventuale notifica su *Slack* relativa a problemi di building o di mancato superamento dei test;
4. Rilascio dei dati sul code coverage a *coveralls.io*;
5. Calcolo dei valori per le ulteriori metriche relative alla qualità del software da parte di *Codacy* e *Better Code Hub*.

Il seguente diagramma riassume il flusso di integrazione continua:

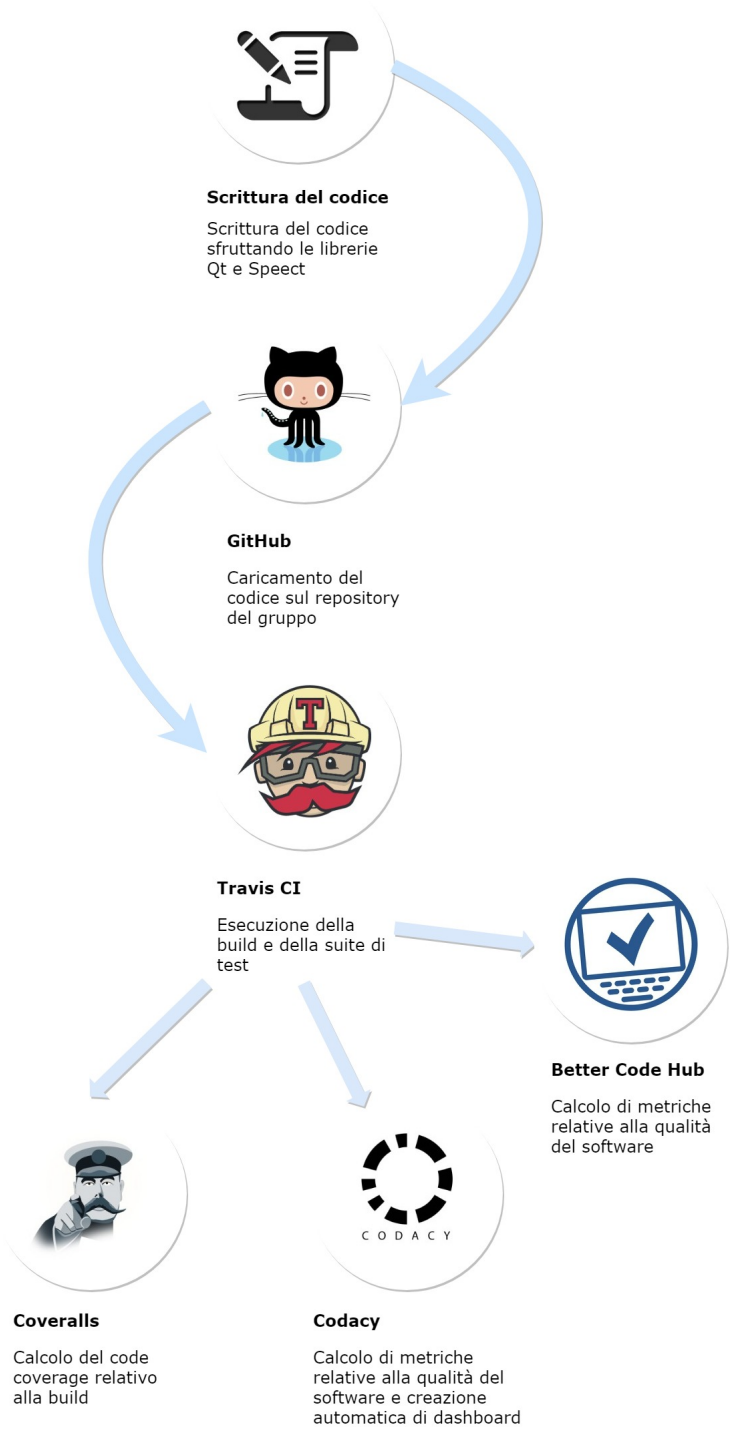

Figura 3.1: Diagramma del flusso di integrazione continua

3.4 Verifica

3.4.1 Scopo

Il processo di verifica ha lo scopo di accertare che non vengano introdotti errori nel prodotto a seguito dello svolgimento delle attività dei processi.

3.4.2 Descrizione

Il processo di verifica viene istanziato per ogni processo in esecuzione, qualora esso raggiunga un livello di maturità significativo o vi fossero modiche sostanziali al suo stato. Per ogni processo viene verificata la qualità dello stesso e dei suoi prodotti. Ognuno dei periodi descritti nel PP produce degli esiti diversi, dunque le procedure di verifica saranno specializzate per ognuno di essi. Gli esiti delle attività di verifica sono riportati in un'appendice dedicata nel PQ. In relazione alle attività di verifica, il gruppo si riferisce alle metriche descritte in §3.3 di questo documento. Il processo di verifica agisce su:

- **Processi**: L'esecuzione dei processi viene monitorata e documentato a fine periodo nell'appendice §C "Resoconto delle attività di verifica di periodo" nel PQ, secondo le metodologie indicate dallo standard *ISO/IEC 15504*$_{\text{G}}$ illustrato in appendice §A.1 dello stesso;
- **Prodotti**: La verifica dei prodotti assume connotazioni differenti a seconda della tipologia di prodotto in esame: per i documenti si prediligono tecniche di analisi statica, per il software tecniche di analisi dinamica, di seguito approfondite.

3.4.3 Analisi

3.4.3.1 Analisi statica

L'analisi statica è una tecnica che permette di individuare anomalie all'interno di documenti e codice sorgente durante tutto il loro ciclo di vita. Si può realizzare tramite due tecniche diverse:

- **Walkthrough:** viene svolta effettuando una lettura a largo spettro. Si tratta di un'attività collaborativa onerosa e poco efficiente. Essa viene utilizzata principalmente durante la fase iniziale del progetto, in cui non tutti i membri del gruppo hanno piena padronanza e conoscenza delle NP e del PQ. Tramite analisi *Walkthrough* è possibile stilare una lista di controllo contenente gli errori più comuni;

- **Inspection:** viene svolta una lettura mirata e strutturata, volta a localizzare gli errori segnalati nella lista di controllo con il minor costo possibile. Con l'incremento dell'esperienza, la lista di controllo viene progressivamente estesa rendendo l'*inspection* via via più efficacie.

Il gruppo utilizza entrambe le tecniche di analisi statica. Seguono le liste di controllo per gli errori comuni relativi a documentazione e codice.

Lista di controllo per la documentazione

Tabella 3.2: Tabella errori comuni - documentazione

Anomalia	**Descrizione**
Comandi LaTeXdeprecati	Uso di comandi LaTeXsuccessivamente ridefiniti
Date	Formato delle date non conforme alle NP
Elenchi puntati	Elenchi puntati non conformi alle NP
Errori grammaticali	Errori grammaticali non rilevati dal controllo automatico dell'editor

Lista di controllo per il codice

Tabella 3.3: Tabella errori comuni - codice

Anomalia	**Descrizione**
Classe non delimitata	Omissione di parentesi graffa di chiusura o di punto e virgola al termine di una classe
Funzione non delimitata	Omissione di parentesi graffa di chiusura o di punto e virgola al termine di una funzione
Statement non delimitato	Omissione di punto e virgola al termine di uno statement
Richiamo errato di variabili	Utilizzo di variabili già definite ma richiamate con errori di battitura

3.4.3.2 Analisi dinamica

L'analisi dinamica è una tecnica di analisi del prodotto software che richiede la sua esecuzione. Viene effettuata mediante dei test volti a verificare il funzionamento del prodotto e nel caso in cui vengano riscontrate anomalie ne permette l'identificazione. I test devono essere ripetibili, cioè deve essere possibile, dato lo stesso input nello stesso ambiente d'esecuzione, otte-

nere lo stesso output. Per ogni test devono dunque essere definiti i seguenti parametri:

- **Ambiente d'esecuzione:** il sistema hardware e software sul quale verrà eseguito il test;
- **Stato iniziale:** lo stato iniziale dal quale il test viene eseguito;
- **Input:** l'input inserito;
- **Output:** l'output atteso;
- **Istruzioni aggiuntive:** ulteriori istruzioni su come eseguire il test e su come interpretare i risultati ottenuti.

I test verranno eseguite automaticamente ad ogni commit effettuato tramite l'integrazione continua attuata grazie a *Travis CI*, mentre la copertura del codice verrà assicurata dal servizio web *coveralls.io*, anch'esso integrabile con Travis CI e Github, che permette di ottenere viste immediate circa la copertura del codice dopo ogni build. Tali test si differenziano tra loro secondo quanto illustrato nei paragrafi che seguono; una trattazione più approfondita, assieme al tracciamento, è reperibile nel PQ (Appendice §B).

3.4.4 Procedure

3.4.4.1 Verifica dei documenti

1. Assegnazione della issue di verifica ad un *Verificatore* da parte del *Responsabile di progetto*;
2. Controllo degli errori comuni secondo checklist;
3. Lettura dei contenuti e correzione degli errori logici e sintattici;
4. Tracciamento degli errori rilevati;
5. Notifica dell'issue come *verified* (precedentemente *verify*).

Segue il diagramma di flusso che riassume la procedura di verifica dei documenti:

Figura 3.2: Diagramma della procedura di verifica dei documenti

3.4.4.2 Verifica del software

La verifica del software avviene mediante tecniche di analisi dinamica, con la creazione di test automatici e tramite misurazioni apposite delle metriche illustrate in §3.3. Segue la procedura di verifica del software:

1. Implementazione del codice;
2. Commit del codice sul repository;
3. Esecuzione della build;
4. Esecuzione della suite di test;
5. Esecuzione della build testata;
6. Calcolo della copertura del codice e delle altre metriche pertinenti con relativa documentazione dei risultati.

Segue il diagramma di flusso che riassume la procedura di verifica del software:

Figura 3.3: Diagramma della procedura di verifica del software

3.4.4.3 Accertamento di qualità di prodotto

Il controllo della qualità del prodotto viene garantito dai processi di verifica e di validazione, nonché dal rispetto delle norme di progetto.

- **Verifica**: questo processo si occupa di accertare che l'esecuzione delle attività di processo siano corrette. Si applica ad ogni attività o risultato che fa progredire il progetto da una baseline alla successiva;
- **Validazione**: questo processo viene svolto come azione conclusiva per accertare che il prodotto rispecchi le aspettative utilizzando un metodo sistematico, disciplinato e quantificabile;
- **Rispetto delle norme** il prodotto deve rispettare le norme e gli strumenti descritti dal presente documento.

3.4.4.4 Accertamento di qualità di processo

La qualità di un processo viene raffinata attraverso al miglioramento continuo indotto dal modello PDCAG (Plan-Do-Check-Act) che apporta in modo incrementale qualità sempre maggiore. Ogni attività del modello deve avere una scrupolosa pianificazione ed un'ottima strategia di verifica che, all'interno di una determinata attività, permetta una serie di iterazioni fino al raggiungimento di un grado di qualità accettabile per poter procedere ad un ulteriore incremento. Per poter parlare di un possibile miglioramento della qualità di un processo, bisogna quantificare la qualità studiandone caratteristiche di interesse. Ciò viene attuato sulla base dello standard SPICE.

3.4.5 Test

La tecnologia utilizzata per la realizzazione dei test è *Google Test*, approfondita in §4.7.4 di questo documento. Di seguito vengono riportate le tipologie di test eseguite sul software prodotto, le cui specifiche sono dettagliate nell'appendice §B del PQ:

- **Test di unità:** i test di unità hanno l'obiettivo primario di isolare la parte più piccola di software testabile, indicata col nome di unità, per stabilire se essa funziona esattamente come previsto. Ogni test di unità deve avere almeno un metodo correlato da testare e deve rispettare la seguente nomenclatura:

  TU[Progressivo]

- **Test di integrazione:** i test di integrazione valutano due o più unità già sottoposte a test come fossero un solo componente, testando le interfacce presenti tra loro al fine di rilevare eventuali inconsistenze. Risulta conveniente testare le unità a coppie aggiungendone gradualmente altre, così da poter rendere immediatamente tracciabile l'origine delle anomalie. Ogni test d'integrazione deve avere almeno un paio di componenti correlate da testare e rispettare la seguente nomenclatura:

  TI[Progressivo]

- **Test di regressione:** i test di regressione devono essere eseguiti ad ogni modifica di un'*implementazione*$_{\text{G}}$ all'interno del sistema. A tal fine è necessario eseguire sul codice modificato i test esistenti, in modo da stabilire se le modifiche apportate hanno alterato elementi precedentemente funzionanti.

- **Test di sistema:** i test di sistema determinano la validazione del prodotto software finale e verificano dunque che esso soddisfi in modo completo i requisiti fissati. Ogni test di sistema deve avere un requisito correlato da testare e deve rispettare la seguente nomenclatura:

  TS[Tipologia][Importanza][Codice]

  Dove Tipologia, Importanza e Codice sono dedotti dal requisito correlato che il test va a verificare.

- **Test di validazione:** il test di accettazione prevede il *collaudo*$_{\text{G}}$ del prodotto in presenza del *proponente*$_{\text{G}}$ e, in caso di superamento di tale collaudo, consegue il rilascio ufficiale del prodotto sviluppato. Ogni test di validazione deve avere un requisito correlato da testare e deve rispettare la seguente nomenclatura:

  TV[Tipologia][Importanza][Codice]

  Dove Tipologia, Importanza e Codice sono dedotti dal requisito correlato che il test va a verificare.

3.5 Validazione

3.5.1 Scopo

Il processo di validazione ha lo scopo di accertare se il prodotto software e relativa documentazione verificati sono conformi a quanto preventivato.

3.5.2 Descrizione

Concluso il processo di verifica, viene istanziato quello di validazione, in cui nuovi *Verificatori* designati dal *Responsabile* accertano che i risultati prodotti siano conformi agli obiettivi di qualità. Se l'esito del processo di validazione è positivo, il *Responsabile di Progetto* provvede all'approvazione dei documenti e/o del software a lui sottoposti.
Questo processo consta di due principali attività:

- Test di sistema e di validazione;
- Collaudo.

Le attività svolte nel processo di validazione sono dunque le seguenti:

- Pianificazione dei test da eseguire e relativo tracciamento;
- Conduzione di test che stressino il software nei suoi punti critici, ovvero laddove è più probabile il verificarsi di errori;
- Verifica del soddisfacimento dei requisiti secondo le informazioni di tracciamento.

È compito dei *Progettisti* definire la pianificazione e la progettazione dei test. Compito dei *Verificatori* è invece eseguirli e tracciarne i risultati tramite gli strumenti a ciò preposti. Per garantire la dovuta imparzialità nell'esecuzione, chi esegue un determinato test è necessariamente un membro del gruppo che non lo ha progettato ed implementato. Ad ulteriore garanzia di indipendenza, è intenzione del gruppo concordare un incontro con il committente in cui effettuare i test di sistema e validazione.

3.5.3 Procedure

3.5.3.1 Validazione dei documenti

1. Assegnazione della issue relativa alla validazione ad un *Verificatore* da parte del *Responsabile*;
2. Calcolo dell'indice Gulpease e confronto del valore ottenuto con i parametri di riferimento;
3. Se i risultati rilevati vengono ritenuti soddisfacenti, il documento viene approvato.

Segue il diagramma di flusso che riassume la procedura di validazione dei documenti:

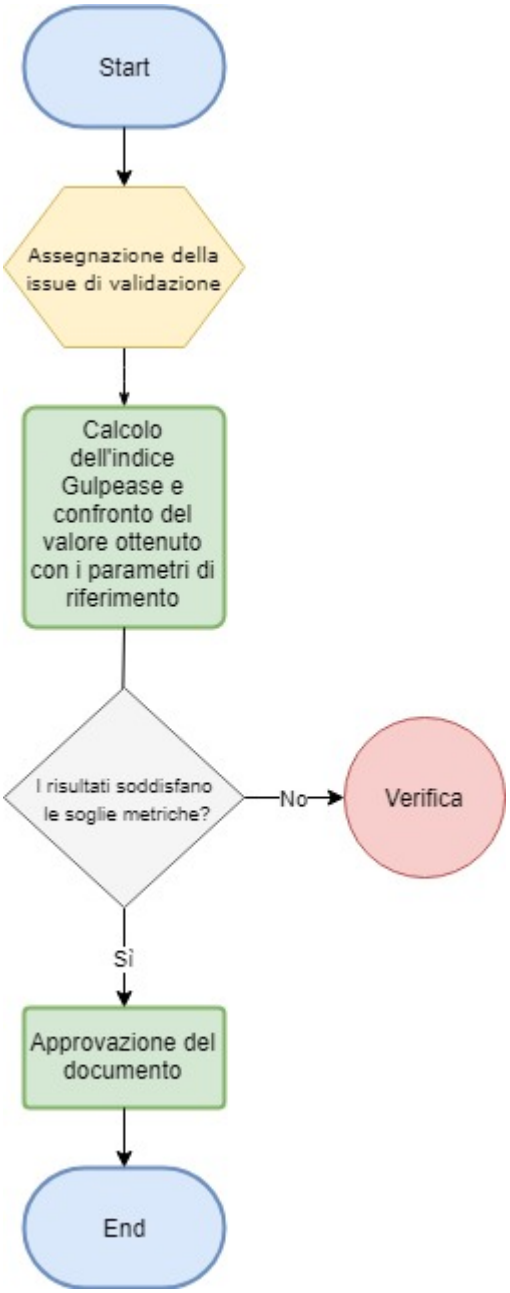

Figura 3.4: Diagramma della procedura di validazione dei documenti

3.5.3.2 Validazione del software

1. Esecuzione dei test di sistema;
2. Esecuzione dei test di validazione.

Segue il diagramma di flusso che riassume la procedura di validazione del software:

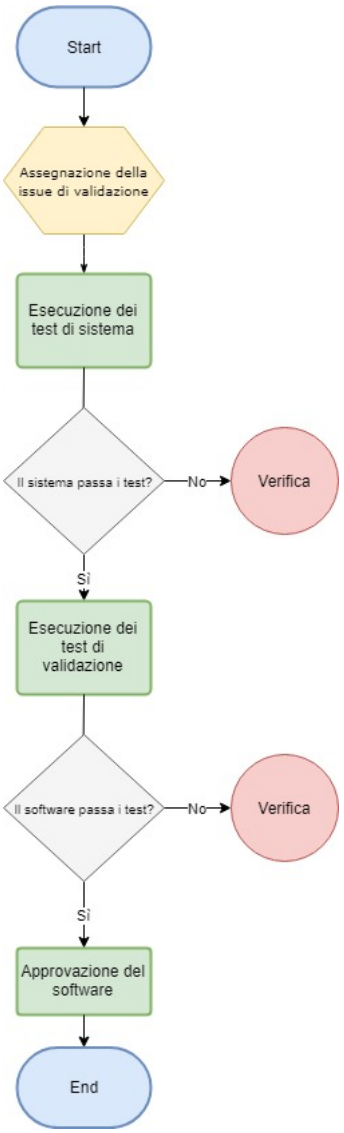

Figura 3.5: Diagramma della procedura di validazione del software

3.5.4 Collaudo

A seguito di una soddisfacente verifica e validazione del software, è intenzione del gruppo concordare un incontro con il Committente per eseguire un collaudo.

4. Processi organizzativi

4.1 Scopo

Lo scopo di questo processo è fornire norme e strumenti per organizzare il lavoro del gruppo per il corretto svolgimento del progetto.

4.2 Descrizione

Durante questo processo sono trattati:

- Ruoli di progetto;
- Comunicazioni;
- Incontri;
- Strumenti di coordinamento;
- Strumenti di versionamento.

4.3 Ruoli di progetto

Ogni ruolo viene ricoperto da ciascun componente del gruppo a turno, dando la possibilità ad ogni membro di fare esperienza in ognuno di essi. L'organizzazione e la pianificazione delle attività da svolgere in ogni ruolo è regolata dal PP. La descrizione dei ruoli viene approfondita nelle sezioni seguenti.

4.3.1 Amministratore di progetto

L'*Amministratore di Progetto* deve controllare e amministrare tutto l'ambiente di lavoro con piena responsabilità sulla capacità operativa e sull'efficienza. Le sue mansioni sono:

- ricerca di strumenti che migliorino l'ambiente di lavoro e che lo automatizzino ove possibile;
- gestione del versionamento;
- controllo di versioni e configurazioni del prodotto software;
- risoluzione dei problemi di gestione dei processi;
- controllo della *qualità*$_{\text{G}}$ sul prodotto.

4.3.2 Responsabile di progetto

Il *Responsabile di Progetto* è il punto di riferimento sia per il *committente*$_{\text{G}}$ che per il *fornitore*. Esso deve anche approvare le scelte prese dal gruppo e se ne assume la responsabilità.
Le sue mansioni sono:

- coordinare e pianificare le attività di progetto;
- approvare la documentazione;
- effettuare uno studio e gestire in modo corretto i rischi;
- approvare l'offerta economica;
- gestire le risorse umane distribuendo in modo corretto i carichi di lavoro.

4.3.3 Analista

L'*Analista* si occupa dell'analisi dei problemi e del dominio applicativo. Normalmente questo ruolo non rimane attivo per tutta la durata del progetto, bensì concentra tutta la propria attività nelle fasi iniziali.
Le sue principali mansioni sono:

- comprensione del problema e della sua complessità;
- produzione dello *Studio di Fattibilità* e dell'*Analisi dei Requisiti*.

4.3.4 Progettista

Il *Progettista* gestisce gli aspetti tecnologici e tecnici del progetto.
Le sue mansioni sono:

- rendere facilmente mantenibile il progetto;
- effettuare scelte efficienti ed ottimizzate su aspetti tecnici del progetto.

4.3.5 Verificatore

Il *Verificatore* deve garantire una verifica completa ed esaustiva del progetto basandosi sulle solide conoscenze delle sue normative.
Le sue mansioni sono:

- controllare le attività del progetto secondo le normative prestabilite.

4.3.6 Programmatore

Il *Programmatore* è il responsabile della codifica del progetto e delle componenti di supporto, che serviranno per effettuare le prove di verifica e validazione sul prodotto.
Le sue mansioni sono:

- versionamento del codice prodotto;
- implementare le decisioni del *Progettista*;
- realizzazione degli strumenti per la verifica e la validazione del software;
- scrittura di un codice pulito e facile da mantenere, che rispetti le *Norme di Progetto*.

4.4 Gestione di progetto

La gestione di progetto viene effettuata tramite meccanismo di ticketing, di seguito illustrato.

4.4.1 Pianificazione tramite ticketing

All'inizio di ogni incremento il *Responsabile di Progetto* è tenuto a pianificare un insieme di task da assegnare ai componenti del gruppo sulla base di:

- Una stima delle risorse disponibili per suddetto incremento;
- Una stima dei possibili rischi;
- Rispetto degli obiettivi di qualità definiti.

Per l'assegnazione del carico di lavoro in Task equamente distribuiti tra i componenti viene utilizzata la piattaforma *Wrike*$_{\mathrm{G}}$, che mostra in modo efficace nella propria interfaccia il quadro completo di tutti i Task inseriti con relativo status (completato, in corso o libero), persona assegnata e scadenza. Quando viene inserito, viene assegnato o cambia stato un Task viene inviata una mail ad ogni componente del gruppo e, per tutti i componenti che fanno uso dell'applicazione mobile, viene inviata una notifica sugli smartphone collegati a *Wrike*.
L'assegnazione dei Task avviene secondo il seguente schema:

- Inserire un titolo al Task;
- Dividere in più Subtask il Task e titolarli;
- Indicare la persona a cui è stato assegnato ogni Subtask;
- Inserire la data entro cui consegnare i documenti/file nella repository prefissata;
- Inserire una descrizione che contiene un breve riassunto del compito assegnato e il ruolo assunto in quella fase del progetto.

Segue il diagramma di flusso che riassume la procedura di individuazione e assegnazione dei task:

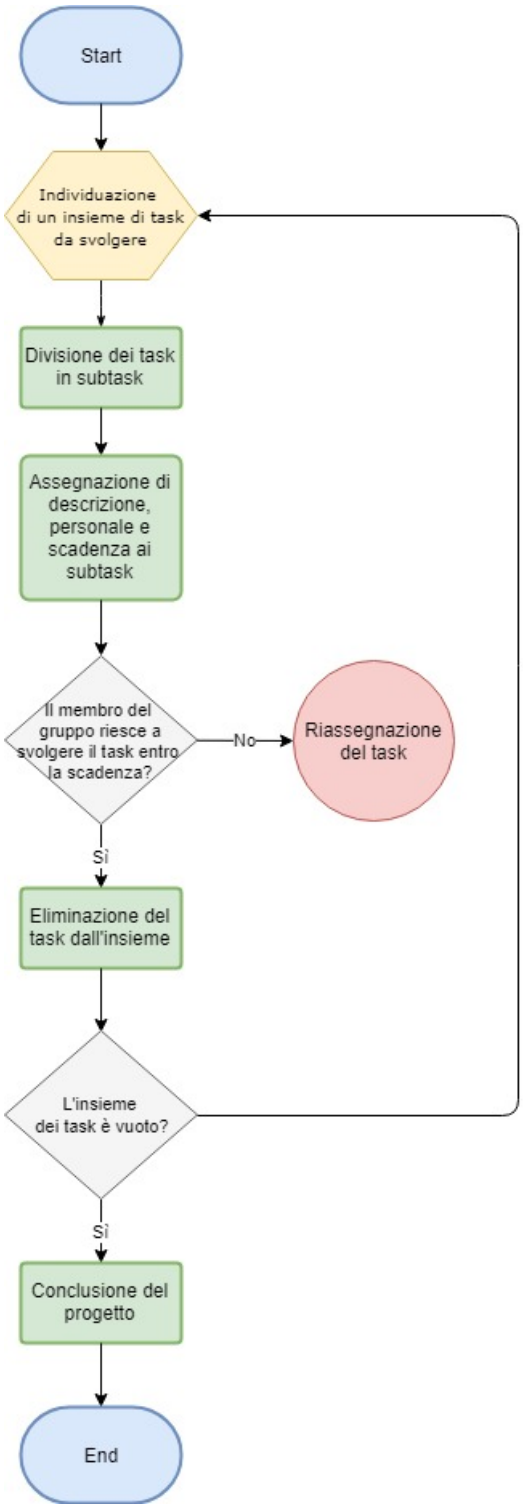

Figura 4.1: Diagramma della procedura di individuazione e assegnazione dei task

4.4.2 Gestione delle comunicazioni

4.4.2.1 Comunicazioni interne

Le comunicazioni interne (ed alcune comunicazioni esterne con la Proponente compatibilmente con le sue necessità) avvengono tramite il tool di messaggistica multi-piattaforma con funzionalità specifiche per gruppi di lavoro *Slack*$_{\text{G}}$. Tale strumento si integra con altre tecnologie selezionate dal gruppo offrendo la possibilità di ricevere notifiche riguardanti le modifiche della *repository*$_{\text{G}}$ *GitHub*$_{\text{G}}$ o il progresso dei task assegnati con Wrike. Permette inoltre di creare canali tematici per rendere più efficiente lo scambio e il reperimento delle informazioni, nonché condividere file ed immagini utili allo sviluppo del progetto.
Per mantenere sotto continuo controllo l'avanzamento dei lavori è prevista, almeno una volta a settimana, una conversazione su *Slack* in cui ogni membro del gruppo indica lo stato di avanzamento del *Task*$_{\text{G}}$ assegnatogli.
Per comunicazioni informali e per ricevere notifiche sommarie riguardanti le modifiche della *repository*$_{\text{G}}$ *GitHub*$_{\text{G}}$ verrà usato un ulteriore tool di messaggistica, di nome *Telegram*$_{\text{G}}$.

4.4.2.2 Comunicazioni esterne

Il *Responsabile del Progetto* è tenuto a mantenere le comunicazioni esterne utilizzando una cartella di posta elettronica appositamente creata:

graphite.swe@gmail.com.

Il *Responsabile del Progetto* deve mantenere informati i restanti componenti del gruppo riguardo alle discussioni con terzi utilizzando i canali di comunicazione interna, ma per maggiore sicurezza ed evitare disguidi interni è previsto che alla ricezione di una mail sull'indirizzo di posta elettronica il messaggio venga inoltrato a tutti i componenti del gruppo. Compatibilmente con impegni e richieste della Proponente, alcune rapide comunicazioni esterne avverranno tramite canale dedicato Slack.

4.4.3 Gestione degli incontri

4.4.3.1 Incontri interni

Il *Responsabile di progetto* è incaricato di organizzare gli incontri interni dando comunicazione di data e ora tramite i canali di comunicazione. Inoltre deve essere comunicato prima di ogni riunione l'ordine del giorno sempre dal *Responsabile di progetto.*

Per mantenere sotto continuo controllo l'avanzamento dei lavori è previsto almeno un incontro settimanale con l'intero gruppo.
Ogni componente del gruppo ha diritto di presentare una richiesta di organizzazione di un incontro al *Responsabile di progetto* il quale può accogliere o meno la proposta.

4.4.3.2 Incontri esterni

Il *Responsabile di Progetto* deve organizzare gli incontri esterni con il committente e comunicare al proprio gruppo data e ora in cui essi avvengono. Come per gli incontri interni, ogni componente del gruppo ha diritto ad una richiesta di organizzazione di un incontro esterno.
Per ogni incontro esterno deve essere preventivamente stilata una serie di domande tale da giustificare la richiesta di un appuntamento con il committente.

4.4.4 Gestione degli strumenti di versionamento

4.4.4.1 Repository

Per il versionamento e il salvataggio dei file è previsto l'utilizzo di repository su GitHub. L'*Amministratore di Progetto* si deve occupare della creazione delle repository e dell'aggiunta di tutti i componenti del gruppo, in possesso di un account personale, come collaboratori.
È previsto l'utilizzo di tre repository facenti capo all'account GitHub del gruppo Graphite:

1. Repository dedicata alla Documentazione;
2. Repository dedicata al *Proof of Concept* (e relativa pubblicazione);
3. Repository dedicata al codice del prodotto.

4.4.4.2 Tipi di file e .gitignore

Nelle cartelle contenenti tutti i documenti saranno presenti solamente i file .tex, .pdf, .jpg, .png. Le estensioni dei file generati automaticamente dalla compilazione sosno stati aggiungi a .gitignore, e quindi vengono ignorati e resi invisibili a Git_{G}.

4.4.4.3 Norme sui commit

Ogni volta che vengono effettuate delle modifiche ai file del repository, le quali poi vengono caricate su di esso, bisogna specificarne le motivazioni. Questo avviene utilizzando il comando *commit* accompagnato da un messaggio riassuntivo e una descrizione in cui va specificato:

- la lista dei file coinvolti;
- la lista delle modifiche effettuate, ordinate per ogni singolo file.

4.4.5 Gestione dei rischi

Il *Responsabile di Progetto* ha il compito di rilevare i rischi indicati nel PP. Nel caso ne vengano individuati di nuovi dovrà aggiungerli nell'appendice dedicata dello stesso documento (§A "Esito della rilevazione dei rischi").
La procedura da seguire per la gestione dei rischi è la seguente:

- Registrazione di ogni riscontro dei rischi nel PP;
- Aggiunta dei nuovi rischi rilevati nell'appendice dedicata del PP;
- Individuazione di problemi non calcolati e monitoraggio di rischi già previsti;
- Ridefinizione, se necessaria, delle strategie di progetto.

4.5 Miglioramento continuo

Il gruppo intende misurare, controllare, valutare e migliorare i processi a beneficio del ciclo di sviluppo del software, operando conformemente al principio del miglioramento continuo.

4.5.1 Costituzione dei processi

Per ogni processo stabilito, esso deve essere controllato, sviluppato e monitorato costantemente allo scopo di garantirne il raggiungimento degli obiettivi fissati nella pianificazione.

4.5.2 Valutazione dei processi

Allo scadere di ogni periodo pianificato, viene attivata una procedura di valutazione dei processi conformemente al ciclo *Ciclo di PDCA*$_{\text{G}}$. Per ogni processo viene valutato:

- Esito;
- Tempistiche;
- Inconvenienti;
- Metriche SPICE.

4.5.3 Miglioramento dei processi

Giunti al periodo successivo, è necessario definire e applicare i miglioramenti necessari ed aggiornare la relativa documentazione. Dati storici, tecnici e di valutazione devono essere raccolti ed analizzati all'interno del PQ al fine di evidenziate eventuali punti deboli e le potenzialità dei processi impiegati. L'analisi è utilizzata come feedback per migliorare i processi e indicare modifiche da applicare in questo senso alle procedure, agli strumenti e alle tecnologie.

4.6 Formazione del personale

I membri del gruppo devono procedere autonomamente allo studio individuale del *project management* e delle tecnologie che verranno utilizzate nel corso del progetto, riferendosi a:

- Materiale elencato nella sezione "Riferimenti informativi" di ogni documento;
- Siti web e relativa documentazione delle tecnologie riportati in forma di link in §4.6 di questo documento;
- Membri del gruppo più esperti;
- Seminario su CMAKE offerto dalla Proponente e reperibile al seguente link:

https://github.com/giuliopaci/cmake-tutorial

Occorre stabilire e fornire in tempo l'adeguata preparazione in termini di sviluppo di competenze e conoscenze richieste. Il personale dove essere formato preventivamente alla pianificazione fissata, in modo da poter svolgere adeguatamente i compiti assegnati. Qualora alcuni membri non riescano a colmare determinate lacune, essi devono informare al più presto il *Responsabile di Progetto* che provvede a redistribuire i compiti assegnati o a indicare materiale utile allo sviluppo delle competenze/conoscenze necessarie allo svolgimento del task. Il versionamento dei prodotti viene usato anche come strumento per apprendere dall'operato altrui, così da integrare le conoscenze personali migliorando la qualità e l'efficienza delle attività di formazione.

4.7 Strumenti

4.7.1 Organizzazione

4.7.1.1 Sistema operativo

Il gruppo di progetto lavora sui seguenti sistemi operativi:

- Ubuntu 17.10 x64;
- Ubuntu 16.04 LTS_{G} x64;
- Windows 10 Home x64;
- Windows 10 Pro x64;
- Windows 7 Home Premium.

4.7.1.2 Google Drive

Google Drive è un servizio di memorizzazione e sincronizzazione online introdotto da Google. Il servizio comprende il file hosting, il file sharing e la modifica collaborativa di documenti fino a 15 GB gratuiti ed è perfettamente integrato con altri servizi Google quali Gmail, Docs, Sheets, Slides e Forms. Il gruppo intende utilizzare Drive per il rilascio della documentazione prevista per le revisioni di progetto e per il salvataggio di materiale informale che possa necessitare di elaborazione collaborativa. Il servizio prevede un piano gratuito automaticamente associato alla creazione di un account Google, ed è reperibile al seguente link:

https://www.google.com/intl/it_ALL/drive/using-drive/

4.7.1.3 Hangouts

Hangouts è un software di messaggistica istantanea e di VoIP sviluppato da Google. È disponibile per le maggiori piattaforme mobili e come estensione per il browser web Google Chrome, e si integra perfettamente con l'ecosistema di prodottoti Google. Il gruppo intende utilizzare tale tecnologia per effettuare videochiamate tra i membri e/o con la Proponente (caso in cui il contatto in remoto si è rivelato indispensabile a causa degli obblighi logistici della stessa) nel contesto di riunioni e confronti. La tecnologia è gratuita sia in versione mobile che desktop, maggiori informazioni sono reperibili al seguente link:

https://gsuite.google.it/learning-center/products/hangouts/get-started/

4.7.1.4 Slack

Slack è uno strumento di collaborazione aziendale utilizzato particolarmente per lo scambio di messaggi tra componenti di un team. L'idea di partenza di questo software è di essere un totale rimpiazzo degli scambi di E-mail e sms ma a questo si aggiunge la possibilità di aggiungere plugin per la notifica di attività annesse al proprio ambiente di lavoro.
Verranno in particolare sfruttati i plugin per la notifica di attività sulla repository e per discussioni riguardo ai task presenti sulla piattaforma Wrike. La tecnologia offre un piano gratuito che non limita il gruppo entro i confini degli obiettivi di progetto, ed è accessibile al seguente link:

https://slack.com/features

4.7.1.5 Telegram

Telegram è una applicazione di messaggistica nata come applicazione mobile e successivamente portata anche su Windows, Mac e varie distribuzioni Linux. Rispetto agli altri sistemi di messaggistica *Telegram* consente un facile passaggio di immagini e documenti in più formati mantenendo inoltre nel proprio cloud storage tali file per un agevole recupero su qualsiasi dispositivo. È possibile creare gruppi di utenti la cui chat ha soprattutto il valore aggiunto di poter contenere sistemi automatici per l'organizzazione di sondaggi e la comunicazione di messaggi importanti da tenere in sovrimpressione. Telegram è accessibile gratuitamente al seguente link:

https://telegram.org/

4.7.1.6 Wrike

Wrike è un' applicazione web disponibile anche per dispositivi mobile creata per aiutare i team a tracciare il proprio lavoro. Le sue principali funzionalità si dividono in:

- possibilità di creare dei task, assegnarvi delle persone, darne una priorità e controllarne lo stato di completamento;
- capacità di creare in modo automatico diagrammi di Gantt basati sui task inseriti dall'utente;
- possibilità di condivisione file con la possibilità di farci delle modifiche online;
- possibilità di creazione di resoconti e discussioni riguardo a specifici argomenti;
- servizio di notifica e modifica rapida tramite l'applicazione per dispositivi mobile.

Il gruppo intende fare di Wrike il principale strumento a supporto del *project management* e quindi utilizzarlo per la suddivisione e assegnazione del lavoro in task da parte del *Responsabile*, sfruttando nel contempo le altre sue funzionalità per documentare e monitorare la gestione di progetto. Pur essendo uno strumento prettamente commerciale, Wrike è disponibile gratuitamente per gli studenti al seguente link:

https://www.wrike.com/it/tour/

4.7.1.7 Git

Git è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando. Pensato per mantenere un grande progetto di sviluppo distribuito, Git supporta fortemente lo sviluppo non lineare del software.
Tramite strumenti appositi è possibile creare più diramazioni di sviluppo del software con la garanzia di poter mantenere in locale la cronologia di sviluppo completa.
In sostituzione ai comandi testuali molti membri del gruppo utilizzeranno *Gitkraken*$_{\text{G}}$ o *GitHub Desktop* come interfaccia grafica per la gestione della repository. La tecnologia è open source e reperibile gratuitamente al seguente link:

https://git-scm.com/about

4.7.1.8 GitHub

GitHub è un servizio di *hosting*$_G$ per progetti software. Il sito è principalmente utilizzato dagli sviluppatori, che caricano il codice sorgente dei loro programmi e lo rendono scaricabile dagli utenti. Può essere utilizzato anche per la condivisione e la modifica di file di testo e documenti revisionabili.
Un utente può interagire con lo sviluppatore tramite un sistema di issue tracking, pull request e commenti che permette di migliorare il codice della repository, risolvendo bug o aggiungendo funzionalità. Lo strumento è reperibile gratuitamente al seguente link:

https://github.com/features

4.7.2 Documentazione

Per redarre la documentazione, il gruppo utilizza i seguenti strumenti:

4.7.2.1 LaTex

LaTeXè un linguaggio di markup, usato per la preparazione di testi, basato sul programma di composizione tipografica TEX. Il gruppo intende utilizzare tale tecnologia per la produzione collaborativa della documentazione formale, in particolare attraverso lo stumento *TeXstudio*. La tecnologia è reperibile gratuitamente al seguente link:

https://www.latex-project.org/

4.7.2.2 TexStudio

TexStudio è un IDE per LaTeXche il gruppo utilizza per la stesura della documentazione in virtù delle innumerevoli funzionalità che offre gratuitamente e del fatto che si tratta di un'applicazione cross-platform supportata dai maggiori sistemi operativi. Lo strumento è reperibile gratuitamente al seguente link:

https://www.texstudio.org/

4.7.2.3 SWEgo

Il tracciamento dei requisiti e dei casi d'uso viene effettuato tramite il software *SWEgo*$_G$ e successivamente controllato manualmente per assicurarne la correttezza. Lo strumento è accessibile gratuitamente al seguente link:

http://www.swego.it/

4.7.2.4 LucidChart

Per la produzione di diagrammi UML viene utilizzato il software gratuito *Lucidchart*$_{\text{G}}$. Lo strumento è accessibile gratuitamente al seguente link:

https://www.lucidchart.com/pages/tour

4.7.3 Sviluppo

Per lo sviluppo software relativo al progetto, il gruppo utilizza i seguenti strumenti:

4.7.3.1 Sistema operativo di riferimento

Ubuntu è un sistema operativo focalizzato sulla facilità di utilizzo. È prevalentemente composto da software libero proveniente dal ramo instabile di Debian GNU/Linux, ma contiene anche software proprietario, ed è distribuito liberamente con licenza GNU GPL. È orientato all'utilizzo sui computer desktop, ma presenta delle varianti per altri dispositivi, ponendo grande attenzione al supporto hardware. Il gruppo intende utilizzare Ubuntu v16.04 LTS come sistema operativo di riferimento per lo sviluppo del prodotto, offrendone garanzia di corretto funzionamento sullo stesso. Ubuntu è reperibile al seguente link:

https://www.ubuntu-it.org/progetto

4.7.3.2 Libreria di riferimento

Speect è un sistema di *Text To Speech* (TTS) multilingua. Esso offre un sistema TTS completo (analisi e decodifica del testo e sintesi vocale) con annesse varie API, nonché un ambiente per la ricerca e lo sviluppo di sistemi e voci TTS. Speect è scritto in linguaggio C, con una stretta conformità allo standard ISO / IEC 9899: 1990, consentendo così la massima portabilità su diverse piattaforme di calcolo. Le chiamate di sistema specifiche della piattaforma sono astratte per consentire le porte a nuove piattaforme. Speect v1.1.0-69-g65f4 rappresenta il cuore del prodotto che il gruppo intende sviluppare ed in particolare è l'applicazione per la quale si vuole realizzare un'interfaccia grafica atta a semplificarne l'utilizzo e il debug. La tecnologia è open source e la sua documentazione è accessibile al seguente link:

http://speect.sourceforge.net/contents.html

4.7.3.3 IDE

QT è una libreria multipiattaforma per lo sviluppo di programmi con interfaccia grafica tramite l'uso di widget (congegni o elementi grafici). La libreria è scritta in C++ e gode di ampia diffusione e supporto. Il gruppo intende utilizzare questa tecnologia nella versione v5.9 LTS per lo sviluppo dell'interfaccia grafica del prodotto. Qt è reperibile al seguente link:

https://www.qt.io/what-is-qt/

4.7.3.4 Compilazione

Per la compilazione vengono utilizzati i seguenti strumenti:

- **GCC**: il compilatore che verrà usato per la compilazione del software è il *GCC*$_{\text{G}}$ (GNU Compiler Collection). Il compilatore è reperibile al seguente link:

  https://gcc.gnu.org/

- **Cmake**: CMake è una famiglia di strumenti open source e multipiattaforma progettati per creare, testare e pacchettizzare software. CMake viene utilizzato per controllare il processo di compilazione del software utilizzando semplici file di configurazione indipendenti dalla piattaforma e dal compilatore e generare makefile e aree di lavoro nativi che possono essere utilizzati nell'ambiente del compilatore di propria scelta. Il gruppo intende utilizzare questa tecnologia nella versione v3.10.2 per l'automazione della compilazione del prodotto. Lo strumento è reperibile al seguente link:

  https://cmake.org/overview/

4.7.3.5 GUI

Per progettare l'interfaccia grafica di DeSpeect è stato utilizzato *Qt Creator*$_{\text{G}}$. Questo strumento permette di realizzare interfacce grafiche mediante le librerie grafiche Qt, diventate in questo ambito quasi uno standard per piattaforme Linux Based. Maggiori informazioni su Qt Creator sono disponibili al seguente link:

https://www.qt.io/qt-features-libraries-apis-tools-and-ide/

4.7.4 Verifica

4.7.4.1 Verifica della documentazione

- **Verifica ortografica:** Per eseguire controlli ortografici sulla documentazione viene utilizzata la verifica dell'ortografia in tempo reale, strumento integrato in TexStudio che sottolinea in rosso le parole errate secondo la lingua italiana;
- **Verifica della leggibilità:** Per calcolare l'*indice Gulpease*$_{\text{G}}$ a verifica della leggibilità dei documenti viene utilizzato uno script apposito reperibile al link seguente:

https://github.com/LeafSWE/Leaf/tree/master/Documents/Gulpease

4.7.4.2 Verifica del software

- **Test:** Google Test è un framework per la realizzazione di test per il linguaggio C++. Il gruppo intende utilizzare questa tecnologia per la realizzazione dei test del software, ulteriori informazioni sono reperibili al seguente link:

https://github.com/google/googletest/blob/master/googletest/docs/Primer.md

- **Verifica della build e testing automatico:** Travis CI è un servizio di integrazione continua distribuito utilizzato per costruire e testare progetti software ospitati su GitHub. I progetti open source possono essere testati gratuitamente attraverso il sito web travis-ci.org. Il gruppo intende utilizzare questa tecnologia per l'esecuzione di test automatici a seguito del caricamento di codice sulla repository relativa al progetto, così da garantirne la correttezza. Travis CI è accessibile al seguente link:

https://docs.travis-ci.com/

- **Analisi statica:** Per l'analisi statica del codice viene usato il software *Valgrind*$_{\text{G}}$. La *suite*$_{\text{G}}$ di strumenti Valgrind fornisce numerosi strumenti di *debugging*$_{\text{G}}$ e di *profiling*$_{\text{G}}$ che aiutano a rendere i programmi più performanti e più corretti. Il più popolare di questi strumenti è chiamato Memcheck, ed è in grado di rilevare molti errori relativi alla

memoria comuni nei programmi C e C++ e che possono causare arresti anomali e comportamenti imprevedibili. Valgrind è accessibile al seguente link:

http://valgrind.org/

- **Analisi dinamica:** Per l'esecuzione dei test di analisi dinamica viene usato il software *SonarQube*$_G$, una piattaforma open source per la gestione della qualità del codice. SonarQube è un'applicazione web che produce report sul codice duplicato, sugli standard di programmazione, i test di unità, il *code coverage*$_G$, la complessità, i bug potenziali, i commenti, la progettazione e l'architettura. SonarQube è accessibile al seguente link:

  https://www.sonarqube.org/

- **Metriche legate al codice:** Per il controllo delle varie metriche vengono utilizzati i software:

  - **Better Code Hub**: *Better Code Hub*$_G$ è un servizio di analisi del codice sorgente web-based che controlla il codice per la conformità rispetto a 10 linee guida per l'ingegneria del software e fornisce un *feedback*$_G$ immediato per capire dove concentrarsi per miglioramenti di qualità. Better Code Hub è accessibile al seguente link:

    https://bettercodehub.com/

  - **Codacy**: Codacy è uno strumento di analisi / qualità del codice automatizzato, con cui si ottengono analisi statiche, complessità ciclomatica, indicazioni sulla duplicazione e le variazioni della copertura dei test dell'unità di codice in ogni richiesta di commit e pull. È inoltre possibile utilizzare Codacy per applicare uno standard di qualità del codice e applicare le best practice sulla sicurezza, il tutto integrato con GitHub. Ulteriori informazioni su Codacy sono disponibili al seguente link:

    https://www.codacy.com/features

4.7.5 Schema riepilogativo dell'interazione delle tecnologie e degli strumenti

Segue uno schema che riassume l'interazione e l'uso delle tecnologie e degli strumenti selezionati.

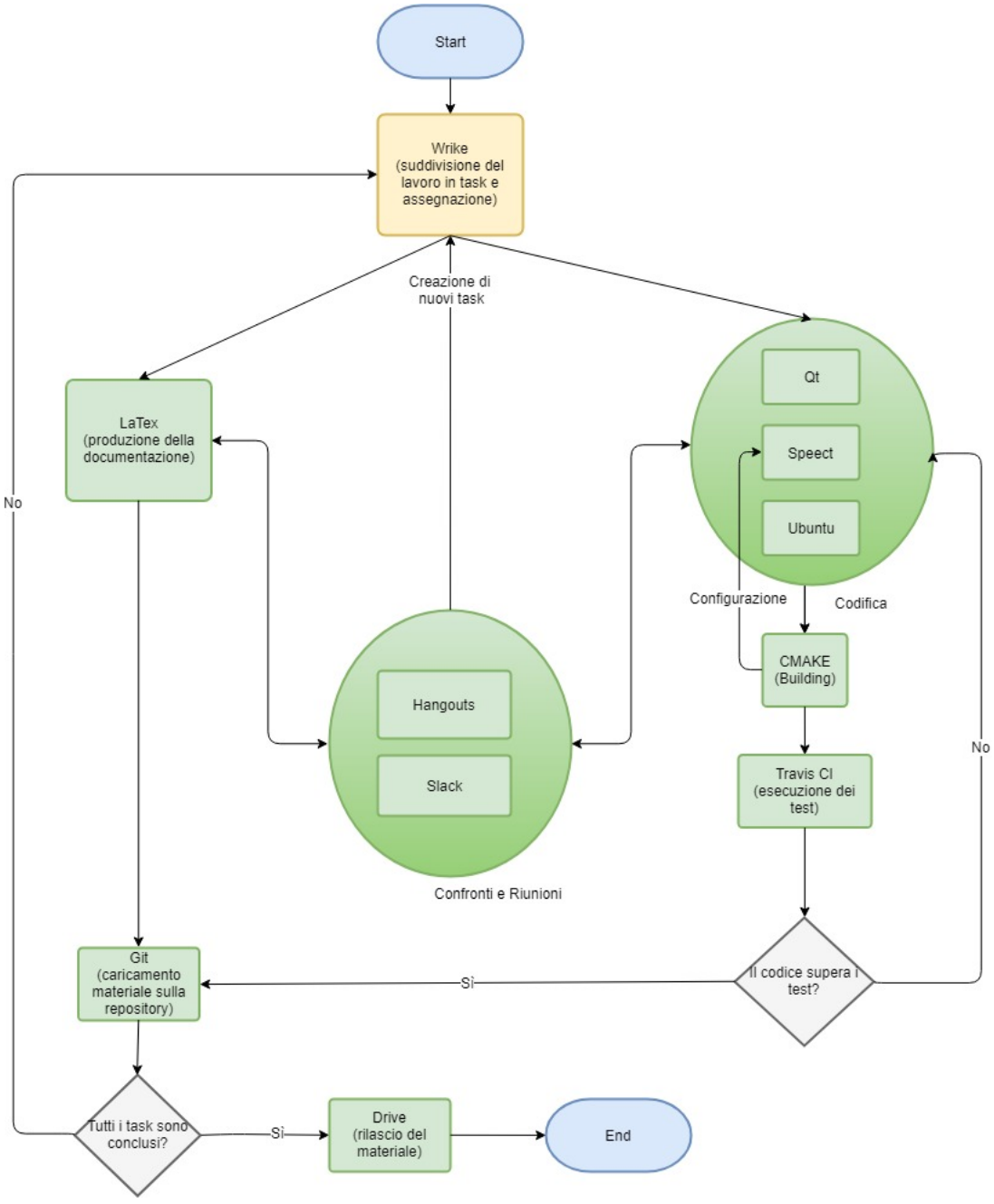

Figura 4.2: Schema riepilogativo dell'interazione delle tecnologie