Norme di Progetto

Informazioni Documento

Versione	1.0.0
Data approvazione	8 Dicembre 2017
Responsabile	Marco Focchiatti
Redattori	Samuele Modena, Giulio Rossetti, Cristiano Tessarolo
Verificatori	Kevin Silvestri, Matteo Rizzo
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo Graphite
Uso	Interno
Recapito	graphite.swe@gmail.com

Registro delle modifiche

Versione	**Data**	**Autore**	**Ruolo**	**Descrizione**
1.0.0	08-12-2017	Marco Focchiatti	Responsabile	Approvazione
0.2.0	08-12-2017	Matteo Rizzo	Verificatore	Verifica sezione processi di supporto e processi organizzativi
0.1.0	08-12-2017	Kevin Silvestri	Verificatore	Verifica sezione introduzione e processi primari
0.0.6	07-12-2017	Giulio Rossetti	Amministratore	Stesura sezione processi organizzativi
0.0.5	07-12-2017	Samuele Modena	Amministratore	Fine sezione processi primari
0.0.4	05-12-2017	Cristiano Tessarolo	Amministratore	Stesura sezione processi di supporto
0.0.3	04-12-2017	Giulio Rossetti	Amministratore	Inizio stesura sezione processi primari
0.0.2	03-12-2017	Cristiano Tessarolo	Amministratore	Stesura sezione introduzione
0.0.1	03-12-2017	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.

1.2 Scopo del Prodotto

Lo scopo del *prodotto* $_{G}$ è quello di fornire un *frontend grafico* $_{G}$ utilizzabile come strumento di supporto allo sviluppo di *plugin* $_{G}$ sulla piattaforma *Speect* $_{G}$.
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* $_{G}$ versione 17.04 o superiore.

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 v1.0.0*. Ogni termine presente nel glossario è marcato da una "G" maiuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

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

1.4.2 Informativi

- **Piano di Progetto v 1.0.0**;
- **Piano di Qualifica v 1.0.0**;
- **Amministrazione di progetto - Slide del corso "Ingegneria del Software":** http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L07.pdf;
- **Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004:** http://www.math.unipd.it/~tullio/IS-1/2007/Approfondimenti/SWEBOK.pdf;
- **Software Engineering - Ian Sommerville - 9 th Edition (2010):** (Disponibile nel solo formato cartaceo);
- **GNU GCC Coding Conventions:** https://gcc.gnu.org/codingconventions.html.

2. Processi primari

2.1 Fornitura

2.1.1 Scopo

Il *processo* $_{G}$ 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* $_{G}$ MIVOQ S.R.L. e dei *committenti* $_{G}$ 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 del 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.

A seguito della consegna del prodotto, il gruppo Graphite non seguirà l'attività di manutenzione dello stesso.

2.1.3 Studio di fattibilità

2.1.3.1 Scopo

Il succitato documento 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.

2.1.3.2 Descrizione

In seguito alla presentazione ufficiale dei Capitolati d'appalto, è compito del *Responsabile di Progetto* $_{G}$ convocare una riunione interna al gruppo per valutare le proposte di progetto pervenute. Gli *Analisti* $_{G}$ conducono quindi un'approfondita attività di analisi dei rischi e delle opportunità su ciascun capitolato, che culmina nella stesura del documento *Studio di Fattibilità v1.0.0*. Tale documento include le motivazioni che hanno portato il gruppo Graphite a proporsi come fornitore per il prodotto indicato, nonché l'analisi di ogni capitolato proposto articolata in:

- **Descrizione generale:** sintesi del prodotto da sviluppare secondo quanto stabilito dal capitolato d'appalto;
- **Dominio applicativo:** analisi del *Dominio Applicativo* $_{G}$, cioè l'ambito di utilizzo del prodotto da sviluppare;
- **Dominio tecnologico:** analisi del *Dominio Tecnologico* $_{G}$ 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;
- **Valutazione Finale:** sintesi delle motivazioni, rischi e criticità evidenziate per cui il capitolato in questione è stato respinto o accettato.

2.1.4 Piano di progetto

2.1.4.1 Scopo

Lo scopo del documento è l'esposizione dettagliata della pianificazione cui attenersi nel corso della realizzazione del progetto, redatta dal *Responsabile di Progetto* coadiuvato dagli *Amministratori* $_{G}$.

2.1.4.2 Descrizione

Il documento contiene:

- **Analisi dei 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;
- **Pianificazione:** relazione sulla pianificazione delle attività da svolgere nel corso del progetto che include delle scadenze temporali precise sulle stesse;
- **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. Alla fine di ogni attività si redige inoltre un consuntivo di periodo per tracciare l'andamento rispetto a quanto preventivato.

2.1.5 Piano di qualifica

2.1.5.1 Scopo

Lo scopo del documento è l'esposizione della strategia individuata dai *Verificatori* $_{G}$ per la *Verifica* $_{G}$ e la *Validazione* $_{G}$ del materiale prodotto dal gruppo.

2.1.5.2 Descrizione

Il documento contiene:

- **Quadro generale della strategia di verifica:** definizione delle procedure di controllo sulla qualità di processo e di prodotto stabilite tenendo in considerazione le risorse a disposizione;

- **Misure e metriche:** definizione delle metriche stabilite per documenti, processi e software prodotto;
- **Gestione della revisione:** definizione delle modalità di comunicazione delle anomalie e delle procedure di controllo per la qualità di processo;
- **Resoconto delle attività di verifica:** sintesi del tracciamento delle attività di verifica e rapporto sulle metriche calcolate durante le stesse.

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* $_{G}$ alle richieste del proponente;
- realizzare un prodotto finale soddisfacente i test di verifica;
- realizzare un prodotto finale soddisfacente i test di validazione;
- fissare gli obiettivi di *sviluppo* $_{G}$;
- 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
- Progettazione
- Codifica

2.2.3 Analisi dei Requisiti

2.2.3.1 Scopo

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

2.2.3.2 Descrizione

I requisiti devono essere estratti dal capitolato d'appalto, dai verbali di riunione e dallo studio dei casi d'uso. Il documento *Analisi dei Requisiti* 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 per gli *Analisti* sono:

- **Capitolato d'appalto:** requisiti emersi dall'analisi del documento fornito dal committente;
- **Verbali esterni:** requisiti emersi a seguito di colloqui con i responsabili dell'azienda committente;
- **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* $_{G}$;
  - **F:** Identifica un *requisito funzionale* $_{G}$;
  - **P:** Identifica un *requisito prestazionale* $_{G}$;
  - **Q:** Identifica un *requisito di qualità* $_{G}$.
- **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.

2.2.3.4 Classificazione casi d'uso

I *casi d'uso* $_{G}$ 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;
- **Scenari Alternativi:** specifica casi di errore o eventi non previsti nel flusso di esecuzione;
- **Flusso di Esecuzione:** rappresenta il flusso degli eventi come lista numerata, specificando per ciascun evento: titolo, descrizione, attori coinvolti e casi d'uso generati;

Alcuni casi d'uso possono essere associati ad un Diagramma *UML* G 2.x dei casi d'uso riportante lo stesso titolo e codice.

2.2.4 Progettazione

2.2.4.1 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* G .

2.2.4.2 Descrizione

Essa serve a garantire che il prodotto sviluppato soddisfi le proprietà e i bisogni specificati nell'attività di analisi. 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* 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.

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* 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.

2.2.4.3 Design Patterns

Finita 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*. Per ogni *Design Pattern* G utilizzato, esso va riportato all'interno del documento di *Specifica Tecnica* riportando un diagramma che lo illustri e una descrizione testuale inerente la sua applicazione e utilità all'interno dell'architettura.

2.2.4.4 UML 2.0

Al fine di rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, è necessario fare largo uso di vari tipi di diagrammi UML 2.0, tra cui:

- **Diagrammi delle classi:** definiscono relazioni, metodi e attributi di classi e tipi, astraendosi da un qualsiasi linguaggio di programmazione;
- **Diagrammi dei *package* G :** illustrano raggruppamenti di classi che condividono la stessa causa di cambiamento e necessitano di essere riusate assieme;
- **Diagrammi di sequenza:** servono a descrivere le interazioni che avvengono fra gli oggetti che devono implementare collettivamente un comportamento, rappresentando sequenze di azioni tramite scelte definite (ad esempio, una sequenza di invocazioni tra metodi);
- **Diagrammi delle attività:** descrivono il flusso di operazioni di una attività (ad esempio un algoritmo), descrivendone la logica procedurale.

E' possibile inoltre fare uso di altre tipologie di diagrammi, come diagrammi delle componenti, di macchine a stati o di *deployment* G , se necessario.

2.2.5 Codifica

2.2.5.1 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 aiuti durante la fase di mantenimento, verifica e validazione, migliorando così la qualità del prodotto.

2.2.5.2 Descrizione

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.

2.2.5.3 Linee di codice

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

2.2.5.4 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* G , seguendo lo standard C++;
- **Altri nomi:** devono essere minuscoli e separati dal trattino basso.

2.2.5.5 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

2.2.5.6 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.

2.2.5.7 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)
{
```

2.2.5.8 Costruttori

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

2.2.5.9 Distruttori

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

2.2.5.10 Conversioni

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

2.2.5.11 Overloading

- **Overloading delle funzioni:** è consentito fare *overloading* $_{G}$ 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.

2.2.5.12 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.

2.2.5.13 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.

2.2.5.14 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.

2.2.5.15 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.

2.2.5.16 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 Gestione della configurazione

3.1.1 Versionamento

Ogni componente versionabile del progetto, nonché la documentazione formale, è versionata mediante la tecnologia *Git* $_{G}$, 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* $_{G}$.

3.1.2 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* $_{G}$ di Github. Le issue riportano nella loro descrizione la ragione della modifica effettuata, nonché la o le *commit* $_{G}$ ad esse associate ed eventuali ulteriori issue correlate.

3.1.3 Stato della configurazione

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

3.1.4 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 *Piano di Progetto v 1.0.0* e mezzi concordati, un archivio contenente tutta la documentazione

richiesta in formato *PDF* G . La consegna è accompagnata da una Lettera di Presentazione, anch'essa inclusa nell'archivio.

3.2 Gestione della qualità

3.2.1 Classificazione degli obiettivi

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.

3.2.2 Classificazione delle metriche

Le metriche stabilite 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.

3.3 Documentazione

3.3.1 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.

3.3.2 Struttura dei documenti

3.3.2.1 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.

3.3.2.2 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

3.3.2.3 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* $_{\text{G}}$ 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.

3.3.2.4 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.

3.3.2.5 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.

3.3.2.6 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.

3.3.2.7 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.

3.3.2.8 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.

3.3.2.9 Contenuto principale

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

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

3.3.3 Norme tipografiche

3.3.3.1 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 $\backslash glossario\{termine\}\{riferimento\ al\ glossario\}$ 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.

3.3.3.2 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.

3.3.3.3 Formati

- **Date:**

  GG-MM-AAAA

  - **GG:** rappresenta il giorno del mese in cifre;
  - **MM:** rappresenta il mese in cifre;
  - **AAAA:** rappresenta l'anno in cifre per intero.

- **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.

3.3.3.4 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*;
- **ST:** *Specifica Tecnica*;
- **MU:** *Manuale utente* G ;
- **DP:** *Definizione di Prodotto*;
- **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* G ;
- **Ve:** *Verificatore.*

3.3.4 Elementi grafici

3.3.4.1 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.

3.3.4.2 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.

3.3.5 Classificazione dei documenti

3.3.5.1 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.

3.3.5.2 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.

3.3.5.3 Documenti interni

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

3.3.5.4 Documenti esterni

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

3.3.5.5 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 *GG-MM-AAAA*;
  - **ID:** è un codice identificativo numerico incrementale.

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

3.3.6 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.

3.3.7 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*.

3.3.8 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*.

3.4 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 instanziato 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 5 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. Concluso il processo di verifica, viene instanziato 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. In relazione alle attività di verifica, il gruppo si riferisce alle metriche descritte nel capitolo §3, "Misure e metriche", del PQ.

3.4.3 Verifica dei processi

L'esecuzione dei processi viene monitorata e documentato a fine periodo nell'appendice §B "Resoconto delle attività di verifica" nel PQ, secondo le metodologie indicate dallo standard *ISO/IEC 15504* G illustrato in appendice §A.1 dello stesso.

3.4.4 Verifica dei 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.

3.4.5 Verifica dei documenti

3.4.5.1 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 e fa inoltre uso di uno script apposito per il calcolo dell' *indice Gulpease* G , la cui descrizione è illustrata di seguito in questo documento.

3.4.5.2 Procedura di 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*).

3.4.6 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 "Misure e metriche" del PQ.

3.4.6.1 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, ottenere 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.

3.4.6.2 Test

Di seguito vengono riportate le tipologie di test eseguite sul software prodotto:

- **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* $_{G}$ 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. Ogni test di regressione deve rispettare la seguente nomenclatura:

  TR[Progressivo]

- **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 accettazione:** il test di accettazione prevede il *collaudo* $_{G}$ del prodotto in presenza del *proponente* $_{G}$ 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.

3.4.6.3 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.

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

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 Procedura di 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.

3.5.4 Procedura di validazione del software

1. Esecuzione dei test di unità;
2. Esecuzione dei test di integrazione;
3. Esecuzione dei test di sistema;
4. Esecuzione dei test di validazione.

3.5.5 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;
- Rischi.

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 documento *Piano di Progetto v1.0.0*. 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* $_{G}$ sul prodotto.

4.3.2 Responsabile di Progetto

Il *Responsabile di Progetto* è il punto di riferimento sia per il *committente* $_{G}$ 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 Procedure

4.4.1 Gestione delle comunicazioni

4.4.1.1 Comunicazioni interne

Le comunicazioni interne avvengono tramite un tool di messaggistica multipiattaforma di nome *Slack* G con funzionalità specifiche per gruppi di lavoro. Tale strumento offre la possibilità di ricevere notifiche riguardanti le modifiche della *repository* G *GitHub* G , creare canali tematici per rendere più

efficiente lo scambio e il reperimento delle informazioni, condividere file ed immagini utili allo sviluppo del progetto.
Per comunicazioni informali e per ricevere notifiche riguardanti le modifiche della *repository* G *GitHub* G verrà usato un ulteriore tool di messaggistica di nome *Telegram* G .
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* G assegnatogli.

4.4.1.2 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.
In ogni caso per 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.

4.4.2 Gestione degli incontri

4.4.2.1 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.

4.4.2.2 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.

4.4.3 Gestione degli strumenti di coordinamento

4.4.3.1 Ticketing

Per la suddivisione del carico di lavoro in Task equamente distribuiti tra tutti i componenti del gruppo viene utilizzata la piattaforma *Wrike* $_{G}$. Tale compito è dato al *Responsabile di Progetto*. *Wrike* 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.

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 dei repository. In seguito l'*Amministratore* inserirà tutti i componenti del gruppo, i quali dovranno essere in possesso di un account personale, come collaboratori.
È previsto l'utilizzo di un repository di nome *progettoSWE* suddiviso nel seguente modo:

- **Documenti:** contiene la documentazione dell'attività di progetto.
- **Prodotto:** contiene i file del prodotto da realizzare.

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 *Piano di Progetto v1.0.0.* Nel caso ne vengano individuati di nuovi dovrà aggiungerli nell'analisi dei rischi.
La procedura da seguire per la gestione dei rischi è la seguente:

- registrare ogni riscontro dei rischi nel *Piano di Progetto v1.0.0*;
- aggiungere i nuovi rischi individuati nel *Piano di Progetto v1.0.0*;
- individuare problemi non calcolati e monitorare i rischi già previsti;
- ridefinire, se necessario, le strategie di progetto.

4.5 Strumenti

4.5.1 Organizzazione interna

4.5.1.1 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.

4.5.1.2 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.

4.5.1.3 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 sovraimpressione.

4.5.1.4 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.

4.5.1.5 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.

Questi servizi sono normalmente disponibili soltanto per la versione a pagamento del software, dando però la possibilità agli studenti di usufruirne con una limitazione di 15 persone per gruppo di lavoro.

4.5.1.6 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* $_{\text{G}}$ come interfaccia grafica per la gestione della repository.

4.5.2 Documentazione

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

- **LaTex**: per la stesura della documentazione viene utilizzato il linguaggio LaTeX, data la sua flessibilità, potenza e facilità d'uso;

- **TexStudio**: per la stesura del codice LaTeX viene utilizzato l'editor *TexStudio* G , in virtù delle innumerevoli feature che offre gratuitamente e del fatto che si tratta di un'applicazione cross-platform supportata dai maggiori sistemi operativi;

- **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. Il software è reperibile gratuitamente al link:

  http://www.swego.it/

- **Diagrammi UML**: per la produzione di diagrammi UML viene utilizzato *Astah* G . Lo strumento è accessibile al seguenti link:

  http://astah.net/

4.5.3 Sviluppo

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

4.5.3.1 IDE

Per la codifica relativa al software richiesto dal progetto il gruppo usa l' *IDE* G *Qt* G , in virtù della sua diffusione, potenza e semplicità d'uso. Qt è reperibile al seguente link:

https://www.qt.io/

4.5.3.2 Compilazione

Per la compilazione vengono utilizzati i seguenti strumenti:

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

  https://gcc.gnu.org/

- **Cmake**: per semplificare la compilazione verrà usato *Cmake* G . Lo strumento è reperibile al seguente link:

https://cmake.org/

4.5.3.3 GUI

Per progettare l'interfaccia grafica viene utilizzato *Qt Creator* G . Questo strumento permette di realizzare interfacce grafiche mediante le librerie grafiche Qt, diventate in questo ambito quasi uno standard per piattaforme Linux Based. Qt Creator è disponibile al seguente link:

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

4.5.4 Verifica

4.5.4.1 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* 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

4.5.4.2 Verifica del software

- **Analisi statica:** Per l'analisi statica del codice viene usato il software *Valgrind* G . La *suite* G di strumenti Valgrind fornisce numerosi strumenti di *debugging* G e di *profiling* 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 software:** Per il controllo delle varie metriche viene utilizzato il software *Better Code Hub* G . Better Code Hub 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/