Norme di Progetto

Gruppo MILCTdev – ***Progetto OpenAPM***
milctdev.team@gmail.com

Versione	4.0.0
Redazione	Carlo Munarini
Verifica	Dragos Cristian Lizan
Approvazione	Isacco Maculan
Uso	Interno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo MILCTdev

Descrizione
Questo documento descrive le regole, gli strumenti e le convenzioni che il *team*$_{\text{G}}$ MILCTdev dovrà rispettare per tutta la realizzazione del *progetto*$_{\text{G}}$ OpenAPM

Registro delle modifiche

Versione	Ruolo	Nominativo	Descrizione	Data
4.0.0	Responsabile	Isacco Maculan	Approvazione del documento per il rilascio	2018-06-03
3.1.0	Verificatore	Dragos Cristian Lizan	Verifica degli incrementi	2018-06-02
3.0.1	Amministratore	Carlo Munarini	Aggiornamento §2.1 e §2.2.4	2018-05-23
3.0.0	Responsabile	Isacco Maculan	Approvazione del documento per il rilascio	2018-05-04
2.3.0	Verificatore	Dragos Cristian Lizan	Verifica degli incrementi	2018-05-02
2.2.6	Verificatore	Leonardo Nodari	Stesura §A.3	2018-04-26
2.2.5	Verificatore	Tommaso Carraro	Stesura §A.2	2018-04-26
2.2.4	Amministratore	Carlo Munarini	Incremento §2.1 e §3.5	2018-04-23
2.2.3	Amministratore	Isacco Maculan	Incremento §2.2.4.5 e §2.2.4.6	2018-04-17
2.2.2	Amministratore	Isacco Maculan	Aggiornamento §2.2.4.4 e §2.2.4.2	2018-04-15
2.2.1	Amministratore	Carlo Munarini	Aggiornamento §2.2.3.1 e §2.2.5	2018-04-14
2.2.0	Verificatore	Leonardo Nodari	Verifica degli incrementi	2018-04-12
2.1.4	Amministratore	Isacco Maculan	Incremento §2.2.3	2018-04-10
2.1.3	Amministratore	Carlo Munarini	Migrazione ”Comandi Git maggiormente utilizzati” in appendice §D	2018-04-10
2.1.2	Amministratore	Carlo Munarini	Migrazione ”Liste di controllo” in appendice §A	2018-04-07
2.1.1	Amministratore	Isacco Maculan	Ristrutturazione generale del documento per renderla uniforme	2018-04-06
2.1.0	Verificatore	Leonardo Nodari	Verifica degli incrementi	2018-04-04
2.0.5	Amministratore	Isacco Maculan	Incremento §3.1	2018-03-27
2.0.4	Amministratore	Carlo Munarini	Incremento §3.3.3	2018-03-23
2.0.3	Amministratore	Carlo Munarini	Migrazione §3.3.2 e delle appendici §B e §C dal Piano di Qualifica	2018-03-23
2.0.2	Amministratore	Isacco Maculan	Correzione dei riferimenti	2018-03-22
2.0.1	Amministratore	Carlo Munarini	Modifica layout al registro delle modifiche	2018-03-20
2.0.0	Responsabile	Luca Dal Medico	Approvazione del documento per il rilascio	2018-03-06

Versione	Ruolo	Nominativo	Descrizione	Data
1.1.0	Verificatore	Isacco Maculan	Verifica dell'intero documento	2018-03-05
1.0.4	Amministratore	Mattia Bano	Modifica §A	2018-02-21
1.0.3	Amministratore	Mattia Bano	Modifica delle date sul documento	2018-02-21
1.0.2	Amministratore	Mattia Bano	Stesura §3.5	2018-02-20
1.0.1	Amministratore	Mattia Bano	Riscrittura parte del documento	2018-02-20
1.0.0	Responsabile	Dragos Cristian Lizan	Approvazione del documento per il rilascio	2017-11-23
0.4.0	Verificatore	Mattia Bano	Verifica del documento	2017-11-22
0.3.0	Verificatore	Carlo Munarini	Verifica del documento	2017-11-18
0.2.3	Amministratore	Luca Dal Medico	Stesura §4	2017-11-17
0.2.2	Amministratore	Tommaso Carraro	Stesura §3.4	2017-11-17
0.2.1	Amministratore	Luca Dal Medico	Aggiunte correzioni dopo verifica	2017-11-17
0.2.0	Verificatore	Mattia Bano	Verifica documento	2017-11-16
0.1.2	Amministratore	Tommaso Carraro	Aggiunta riferimenti per la parte Supporto : Documentazione	2017-11-15
0.1.1	Amministratore	Tommaso Carraro	Aggiunte correzioni dopo verifica	2017-11-15
0.1.0	Verificatore	Carlo Munarini	Verifica documento	2017-11-14
0.0.6	Amministratore	Luca Dal Medico	Stesura §3.3	2017-11-14
0.0.5	Amministratore	Tommaso Carraro	Stesura §3.2	2017-11-14
0.0.4	Amministratore	Tommaso Carraro	Stesura §3.1	2017-11-14
0.0.3	Amministratore	Tommaso Carraro	Stesura §2.2	2017-11-14
0.0.2	Amministratore	Luca Dal Medico	Stesura §2.1	2017-11-13
0.0.1	Amministratore	Luca Dal Medico	Stesura §1	2017-11-12
0.0.0	Amministratore	Tommaso Carraro	Inserimento template latex	2017-11-10

Indice

Tabelle

Immagini

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di definire le regole, gli strumenti e le convenzioni adottate dal gruppo MILCTdev durante l'intero svolgimento del progetto.

1.2 Scopo del prodotto

Lo scopo del *prodotto*$_{\text{G}}$ è realizzare un set di funzioni basate su *Elasticsearch*$_{\text{G}}$ e *Kibana*$_{\text{G}}$ per interpretare i dati raccolti da un *Agent*$_{\text{G}}$. I dati interpretati forniranno a *DevOps*$_{\text{G}}$ statistiche e informazioni utili per comprendere il funzionamento della propria applicazione. In particolare si richiede lo sviluppo di un motore di generazione di *metriche*$_{\text{G}}$ da *trace*$_{\text{G}}$, un motore di generazione di *baseline*$_{\text{G}}$ basato sulle metriche del punto precedente, e un motore di gestione di *critical event*$_{\text{G}}$.

1.3 Glossario

All'interno del documento sono presenti termini che possono assumere significati diversi a seconda del contesto. Per evitare ambiguità, i significati dei termini complessi adottati nella stesura della documentazione sono contenuti nel documento *Glossario v3.0.0*. Per segnalare un termine del testo presente all'interno del Glossario verrà aggiunta una $_{\text{G}}$ a pedice e il testo sarà in corsivo.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Standard ISO/IEC 12207:1995:**
  http://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
  (ultima consultazione effettuata in data 2018-03-05).
- **Verbale Interno - 2017-11-28**
- **Verbale Interno - 2018-04-05**
- **Verbale Interno - 2018-04-07**

1.4.2 Riferimenti informativi

- ***Analisi dei Requisiti v4.0.0* §4.2 - Requisiti funzionali**
- ***Piano di Qualifica v4.0.0* §2 - Specifica dei test**

- ***Piano di Progetto v4.0.0* §5 - Pianificazione**

- **Documentazione - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L12.pdf
  (ultima consultazione effettuata in data 2018-03-05);
- **Documentazione di TexMaker:**
  http://www.xm1math.net/texmaker/doc.html
  (ultima consultazione effettuata in data 2018-03-05);
- **Processi Software - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L03.pdf
  (ultima consultazione effettuata in data 2018-03-05);
- **Progettazione - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L10.pdf
  (ultima consultazione effettuata in data 2018-04-10).
- **Diagrammi UML - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E01.pdf
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E02.pdf
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E03.pdf
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E04.pdf
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E05.pdf
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E06.pdf
  (ultima consultazione effettuata in data 2018-04-10).
- **Documentazione di IntelliJ:**
  https://www.jetbrains.com/idea/documentation
  (ultima consultazione effettuata in data 2018-05-02);
- **Gestione di Progetto - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L06.pdf
  (ultima consultazione effettuata in data 2018-03-05).
- **Qualità di prodotto - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L13.pdf
  (ultima consultazione effettuata in data 2018-03-07);
- **Qualità di processo - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L15.pdf
  (ultima consultazione effettuata in data 2018-03-07);
- **Sommerville Ian, Software Engineering, 10th ed., Pearson (2015)**
  - §24 Quality management
- **Sommerville Ian, Software Engineering, 9th ed., Pearson (2010)**
  - §26 Process improvement
- **Verifica e validazione - Slide del corso "Ingegneria del Software":**
  http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/L17.pdf
  (ultima consultazione effettuata in data 2018-05-07);

- **Informazioni su Julia**
  http://www.math.unipd.it/~ranzato/vds_17-18/SLIDES/Ferrara_lecture.pdf
  (ultima consultazione effettuata in data 2018-05-02);
- **Standard ISO/IEC 15504**:
  https://en.wikipedia.org/wiki/ISO/IEC_15504
  (ultima consultazione effettuata in data 2018-03-07);
- **PDCA**:
  https://it.wikipedia.org/wiki/Ciclo_di_Deming
  (ultima consultazione effettuata in data 2018-03-07);
- **Standard ISO/IEC 9126**:
  https://it.wikipedia.org/wiki/ISO/IEC_9126
  (ultima consultazione effettuata in data 2018-03-07);
- **Indice Gulpease**:
  http://it.wikipedia.org/wiki/Indice_Gulpease
  (ultima consultazione effettuata in data 2018-03-07);
- **Logical SLOC**:
  https://en.wikiversity.org/wiki/Software_metrics_and_measurement
  (ultima consultazione effettuata in data 2018-03-07).
- **Documentazione su Git:**
  http://www.piratpartiet.it/mediawiki/index.php?title=Mini_Guida_a_GIT#Comandi_di_base_a_riga_di_comando
  (ultima consultazione effettuata in data 2018-03-05);

2 Processi primari

2.1 Fornitura

2.1.1 Scopo del processo

Il processo di fornitura contiene tutte le attività e i compiti che i membri del gruppo MILCTdev sono tenuti a rispettare al fine di proporsi e diventare Fornitori nei confronti della *Proponente*$_{\text{G}}$ Kirey Group e dei *Committenti*$_{\text{G}}$ Prof. Tullio Vardanega e Prof. Riccardo Cardin.

A contratto stipulato vengono determinate le procedure e le risorse necessarie alla gestione dello sviluppo del prodotto OpenAPM, tra cui la stesura del *Piano di Progetto v4.0.0* e la consegna del prodotto realizzato.

2.1.2 Studio di Fattibilità

Il documento *Studio di Fattibilità v1.0.0* deve contenere la valutazione tecnica sui capitolati proposti e le motivazioni che hanno portato alla scelta del progetto da realizzare.

Il Responsabile ha il compito di organizzare riunioni tra i componenti del gruppo per discutere dei capitolati e discutere le opinioni di ogni componente. Successivamente l'Analista deve redigere il documento indicando per ogni capitolato:

- **Descrizione generale**: breve presentazione del progetto;
- **Obiettivo finale**: descrizione in linea di massima delle caratteristiche principali richieste nel prodotto completato ed il suo ambito di utilizzo;
- **Tecnologie richieste**: elenco delle tecnologie da impiegare nella realizzazione del prodotto;
- **Valutazione finale**: riassunto degli aspetti principali che hanno portato il gruppo ad accettare o scartare il capitolato in questione.

2.1.3 Rapporti con la Proponente

Durante l'intero svolgimento del progetto il gruppo intende instaurare con la Proponente Kirey Group, in particolar modo nelle figure dei referenti Stefano Bertolin e Stefano Lazzaro, un rapporto di collaborazione costante e costruttiva al fine di:

- determinarne i bisogni;
- individuare norme consone all'esecuzione dei processi;
- ricevere feedback riguardo all'andamento del progetto;
- stimare costi e tempi;
- accordarsi circa la qualifica di prodotto ultimato.

Le modalità di comunicazione sono descritte in §4.1.2.2.

2.1.4 Documentazione fornita

Al fine di assicurare la massima trasparenza circa le attività progettuali, verranno distribuiti alla Proponente Kirey Group ed ai Committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin i documenti descritti in §3.1.5.

In particolare, in vista della Revisione di Accettazione del 15 giugno 2018 verranno consegnati, con annessa Lettera di Presentazione, i seguenti documenti:

- *Norme di Progetto v4.0.0*;
- *Analisi dei Requisiti v4.0.0*;
- *Piano di Progetto v4.0.0*;
- *Piano di Qualifica v4.0.0*;
- *Product Baseline v1.0.0*;
- *Manuale Utente v2.0.0*;
- *Manuale Sviluppatore v2.0.0*;
- i *Verbali* stilati nello trascorrere dalla scorsa revisione.

2.1.5 Consegna e collaudo del prodotto

Al fine di validare il prodotto ultimato rispetto agli obblighi contrattuali, il gruppo deve effettuare un collaudo sotto la supervisione della Proponente e dei Committenti.

La data prevista è il 15 giugno 2018 in concomitanza con la Revisione di Accettazione e per la quale è stato richiesto al Fornitore di stilare i test di accettazione da eseguirvisi.

Inoltre, il gruppo si impegnerà a consegnare entro il giorno lavorativo precedente il prodotto finito, su supporto DVD, comprensivo di: utilità di installazione, istruzioni per l'uso, sorgenti completi, utilità di compilazione, documentazione ed eventuali utilità di collaudo.

In preparazione di ciò il gruppo deve assicurare la correttezza, completezza ed affidabilità di ogni parte del prodotto realizzato affinché si possa dimostrare che:

- tutti i requisiti obbligatori descritti in *Analisi dei Requisiti v4.0.0* siano completamente soddisfatti;
- l'esecuzione di tutti i test descritti nel *Piano di Qualifica v4.0.0* abbia esito positivo.

2.1.6 Completamento del progetto

A seguito della consegna del prodotto e del superamento del collaudo, il progetto si intende concluso.

Ciò significa che, salvo diversi accordi con la Proponente Kirey Group, il gruppo MILCTdev non seguirà l'attività di manutenzione del prodotto *OpenAPM*.

2.2 Sviluppo

2.2.1 Scopo del processo

Il seguente processo contiene le attività di analisi dei requisiti, progettazione e codifica. Queste, partendo da quanto descritto nel Capitolato e tramite la conduzione del processo di Gestione, risulteranno nello sviluppo del prodotto software.

2.2.2 Analisi dei requisiti

In seguito al completamento dello Studio di Fattibilità, è compito degli Analisti redigere il documento di Analisi dei Requisiti. Esso ha l'intento di formalizzare e rendere tracciabili i requisiti e i casi d'uso individuati, presentandoli seguendo le specifiche sottostanti.

2.2.2.1 Requisiti

Ogni requisito è classificato tramite il seguente formalismo:

R{X}{Y}{codice_identificativo}

dove:

- **X** specifica la tipologia di requisito:
  - *F*: requisito funzionale;
  - *P*: requisito prestazionale;
  - *Q*: requisito qualitativo;
  - *V*: vincolo progettuale.
- **Y** indica uno dei seguenti gradi di necessità:
  - *O*: requisito obbligatorio;
  - *D*: requisito desiderabile;
  - *F*: requisito facoltativo.
- **codice_identificativo**: è un codice composto da una serie di numeri separati tramite punto che identificano il requisito in maniera univoca e lo esprimono gerarchicamente.

È imperativo che un requisito non possa cambiare denominazione col passare del tempo. La lista dei requisiti è espressa in forma tabellare e ciascun requisito è esplicato nel seguente modo:

- **ID Requisito**: rappresenta un identificativo del requisito, costruito come sopra;
- **Descrizione**: breve ma chiara e precisa; non deve lasciare spazio ad ambiguità;
- **Fonte**: indica la fonte da cui è stato estrapolato il requisito e può essere:
  - **capitolato**: derivante direttamente dal capitolato;
  - **verbale**: derivante da un incontro verbalizzato;
  - **caso d'uso**: derivante da uno o più casi d'uso.

Esempi di requisiti:

ID Requisito	Descrizione	Fonti
RVO1.2	Esempio 1	Capitolato; UC1
RPF3.1	Esempio 2	Capitolato; UC2

2.2.2.2 Casi d'uso

Ogni caso d'uso è classificato tramite il seguente formalismo:

UC{codice_identificativo}

dove:

- **codice_identificativo** è un codice composto da una serie di numeri separati tramite punto che identificano il caso d'uso in maniera univoca e lo esprimono gerarchicamente.

È imperativo che un caso d'uso non possa cambiare denominazione col passare del tempo. Per i casi d'uso che si sviluppano in sotto-casi viene incluso un diagrammma UML denominato *Use Case Diagram*$_{\text{G}}$. Ciascun caso d'uso è esplicato dai seguenti punti:

- **ID Caso d'uso**: identificativo del caso d'uso, costruito come scritto sopra;
- **Titolo**: titolo del caso d'uso;
- **Descrizione**: breve descrizione del caso d'uso;
- **Attori**: lista di *attori*$_{\text{G}}$ principali e secondari coinvolti nel caso d'uso;
- **Precondizione**: condizione che deve essere verificata prima dell'esecuzione del caso d'uso;
- **Postcondizione**: condizione che deve essere verificata dopo dell'esecuzione del caso d'uso;
- **Scenario principale**: descrizione composta dal flusso dei casi d'uso figli;
- **Scenari alternativi**: descrizione composta dai casi d'uso che non appartengono al flusso principale di esecuzione, se presenti;
- **Estensioni**: indica quali sono tutte le estensioni, se presenti;
- **Inclusioni**: indica quali sono tutte le inclusioni, se presenti;
- **Generalizzazioni**: indica quali sono tutte le generalizzazioni, se presenti.

Esempio di caso d'uso:

UC2.2 - Filtraggio delle traces per parametri configurati

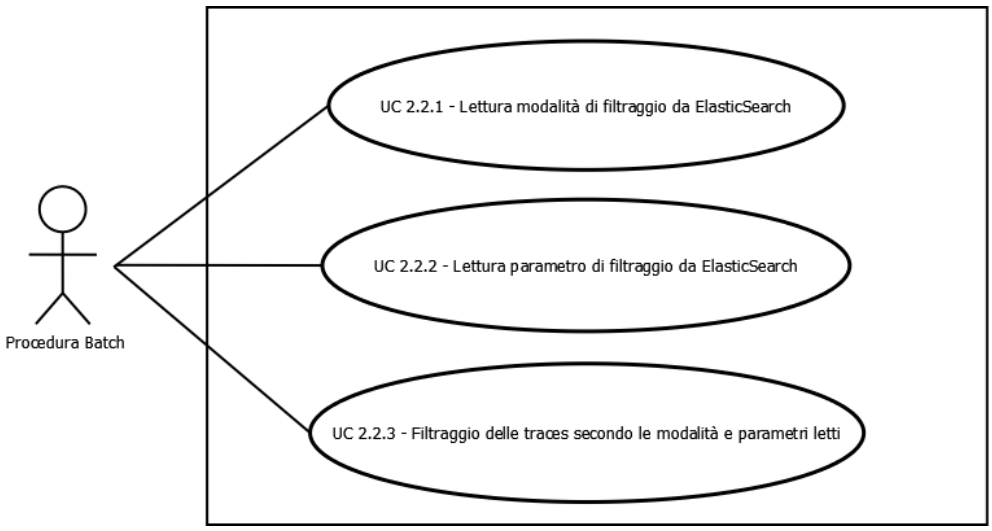

Figura 1: Esempio di caso d'uso

- **Attori** - Procedura $Batch_{\mathrm{G}}$;
- **Descrizione** - L'attore filtra le traces in base a parametri configurati e salvati in ElasticSearch;
- **Precondizione** - I parametri per il filtraggio sono stati precedentemente configurati;
- **Postcondizione** - Le traces sono state filtrate secondo i parametri configurati;
- **Scenario principale**
  1. L'attore legge la modalità di filtraggio da ElasticSearch (UC2.2.1);
  2. L'attore legge il parametro di filtraggio da ElasticSearch (UC2.2.2);
  3. L'attore filtra le traces secondo le modalità e i parametri letti (UC2.2.3).

2.2.3 Progettazione

Questa attività, compito dei Progettisti, consiste nel design dell'architettura logica e dell'architettura in dettaglio del prodotto software da realizzare, perseguendo le seguenti caratteristiche qualitative:

- **sufficienza**: capacità di soddisfare tutti i requisiti documentati nell'*Analisi dei Requisiti v4.0.0*;
- **comprensibilità**: capacità di essere capibile dai suoi utenti;
- **modularità**: suddivisione in parti chiare e distinte;
- **robustezza**: sopportazione di input dalla diversa natura, qualità e quantità;
- **semplicità**: avversione al superfluo;
- **incapsulazione**: identificazione di interfacce per la comunicazione che nascondano il funzionamento interno delle parti;
- **coesione**: raggruppamento delle parti rispetto a criteri che le avvicinino per scopo;
- **basso accoppiamento**: minimizzazione delle dipendenze ed assenza di dipendenze indesiderate.

L'attività di progettazione precede quella di codifica, permettendo di perseguire la correttezza per costruzione, con conseguenti vantaggi di efficienza ed efficacia.

2.2.3.1 Tecnica di progettazione

La modellazione viene fatta utilizzando il linguaggio *UML 2.0*$_{\text{G}}$ in quanto standard nell'ambito dell'Ingegneria del Software. Questa scelta facilita la comunicazione con Committente, Programmatori, Responsabile e Verificatori.

Viene fatto uso delle seguenti tipologie di diagrammi:

- **Diagrammi di package**: descrivono l'organizzazione dei package e degli elementi UML in essi contenuti, specificandone la visibilità; questi diagrammi permettono inoltre di esprimere dipendenze fra classi;
- **Diagrammi delle classi**: descrivono tipi di entità, mostrandone visibilità, attributi, operazioni e relazioni statiche;
- **Diagrammi di attività**: descrivono la logica procedurale delle attività, individuando come il flusso di controllo percorre certe azioni; questi diagrammi permettono inoltre di esprimere aspetti dinamici dei casi d'uso;
- **Diagrammi di sequenza**: descrivono una determinata sequenza di azioni, enfatizzando la collaborazione di un gruppo di oggetti al fine di implementare un certo comportamento.

Per la stesura dei diagrammi ci si avvale del programma Visual Paradigm 15.0 Community Edition poiché intuitivo da utilizzare ed impone il rispetto della grammatica UML 2.0.

2.2.3.2 Strategia di progettazione

La progettazione deve dominare la complessità del prodotto, portando ad una sua completa comprensione. Ciò è attuabile mediante una strategia *divide et impera*, andando a scomporre il sistema in sottosistemi più semplici, che verranno a loro volta raffinati fino ad essere ridotti ad elementi atomici.

Il requisito RVD4 su l'utilizzo di *Spring Batch*$_G$, tuttavia, vincola la specifica architetturale ad adattarsi alle soluzioni offerte dal framework, specializzandole a proprio uso. Per tanto, la progettazione avviene mediante un approccio *Meet-in-the-middle*, che combina la decomposizione di problemi con la composizione di soluzioni. Un aspetto positivo è che ciò facilita l'individuazione e la specifica dei moduli perché basati su entità concrete fornite dai framework.

2.2.3.3 Stile di progettazione

Nel progettare si dovrà:

- perseguire l'*Acyclic Dependency Principle*, eliminando le eventuali dipendenze circolari riscontrate nei digrammi di package mediante fattorizzazioni o riduzioni;
- sfruttare i *design pattern*$_G$, risolvendo problemi già conosciuti con soluzioni progettuali già consolidate;
- tracciare ogni requisito funzionale al componente, quindi alla corrispettiva unità che lo realizza, così da garantire congruenza e completezza rispetto all'analisi dei requisiti effettuata;
- definire test di integrazione e di unità atti a verificare la conformità con quanto progettato;
- documentare mediante commenti UML, le scelte progettuali effettuate; in particolare vanno descritti: l'impiego di design pattern, i requisiti implementati, le dipendenze riscontrate e le interfacce definite;
- perseguire il principio dell'information hiding, al fine di raggiungere un'organizzazione propensa alla codifica in parallelo da parte di più Programmatori;
- nei diagrammi delle classi descrivere l'utilizzo di design pattern tramite commenti di colore rosso;
- segnalare le librerie e le API esterne; nei diagrammi delle classi farlo colorandole di arancione.

2.2.4 Codifica

Questa attività, competenza dei Programmatori, consiste nella scrittura del codice sorgente di quanto progettato nell'attività di progettazione. Il codice deve compilare senza errori né warning. Deve inoltre rispettare le successive specifiche ed essere accompagnato da relativa documentazione per poter assicurare manutenibilità e leggibilità.

2.2.4.1 Nomenclatura

- I nomi dei file devono essere tutti in minuscolo;
- i nomi di variabili e metodi devono avere la prima lettera minuscola;
- i nomi delle classi devono avere la prima lettera maiuscola;
- i nomi di costanti devono essere tutti in maiuscolo;
- i nomi di variabili, metodi e classi devono essere in *camel case*$_{\text{G}}$.

2.2.4.2 Commenti

Classi e metodi devono essere preceduti da commenti secondo le convenzioni Javadoc:

http://www.oracle.com/technetwork/articles/java/index-137868.html

Il codice va commentato per semplificarne la comprensione e la manutenzione; questo significa anche evitare di farlo quando ciò risulti ovvio.

Inoltre:

- i commenti riguardo uno statement è preferibile che siano posizionati sopra lo statement stesso;
- una modifica del codice deve essere seguita da una modifica ai commenti che vi ci si riferiscono.

2.2.4.3 Ricorsione

La ricorsione va evitata il più possibile. Ogni metodo ricorsivo dovrà essere accompagnato da una prova di terminazione e dall'analisi dell'occupazione della memoria. Risultati non soddisfacenti ne comporteranno il rifiuto.

2.2.4.4 Formattazione

Il codice va formattato come da default in IntelliJ IDEA, consistente nello stile di indentazione KR con indentazioni composte da 4 spazi, come mostrato dall'esempio qui sottostante:

```
public class Foo {
    public void foo(boolean a, int x, int y, int z) {
        do {
            try {
                if (x > 0) {
                    int someVariable = a ? x : y;
                    int anotherVariable = a ? x : y;
                } else if (x < 0) {
                    int someVariable = (y + z);
                    someVariable = x = x + y;
                } else {
                    for (int i = 0; i < 5; i++) doSmtng(i);
                }
                switch (a) {
                    case 0:
                        doCase0();
                        break;
                    default:
                        doDefault();
                }
            } catch (Exception e) {
                processException(e.getMessage(), x, z, a);
            } finally {
                processFinally();
            }
        }

        for (int i = 0; i < 5; i++) System.out.println(i);
    }

    private class InnerClass implements I1, I2 {
        public void bar() throws E1, E2 {
        }
    }
}
```

2.2.4.5 Codifica componente

Il seguente diagramma illustra la procedura da seguire per la codifica di un componente architetturale.

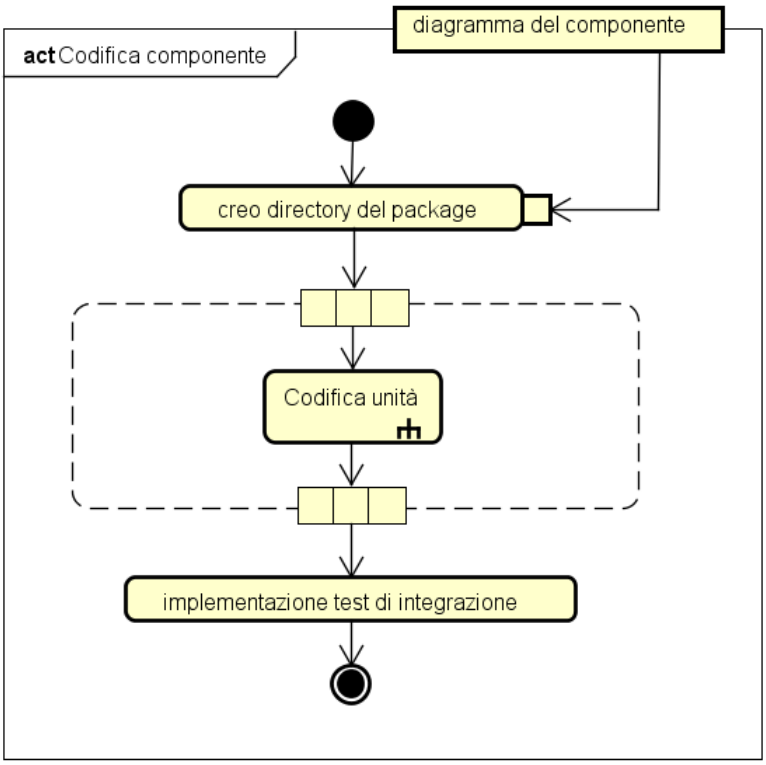

Figura 2: Procedura di codifica di un componente

2.2.4.6 Codifica unità

Il seguente diagramma illustra la procedura da seguire per la codifica di un'unità architetturale.

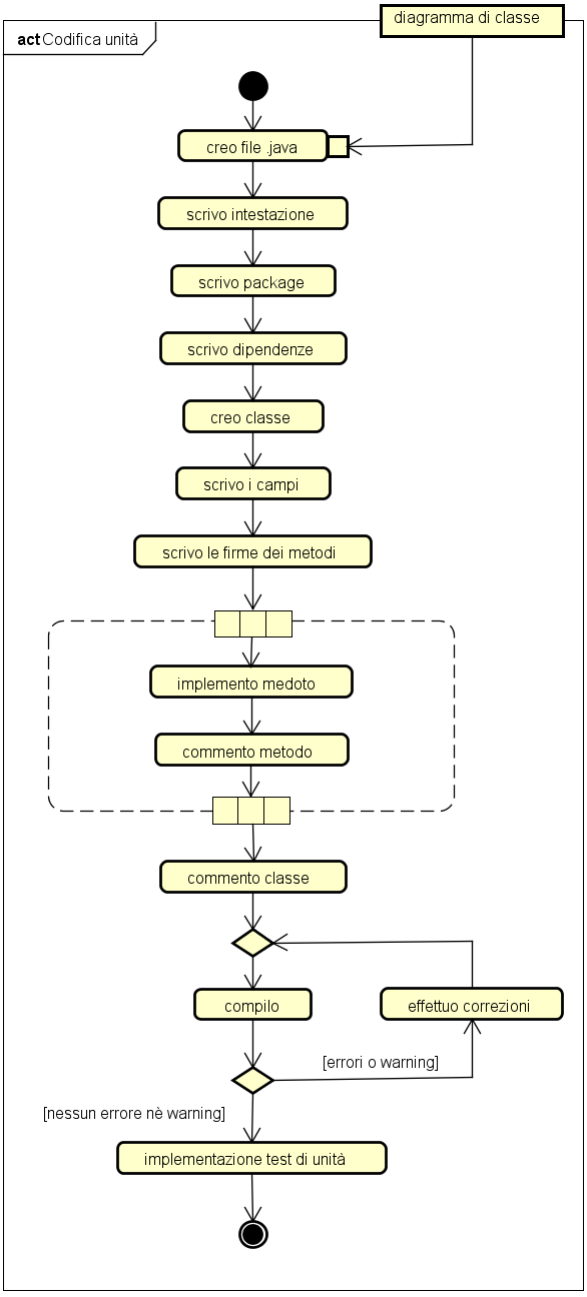

Figura 3: Procedura di codifica di un'unità

2.2.5 Strumentazione

2.2.5.1 SWEgo

Figura 4: SWEgo: applicazione web utilizzata dal gruppo per il tracciamento dei requisiti

SWEgo è un potente strumento di tracciamento dei requisiti disponibile online. Esso offre una moltitudine di funzioni che semplificano e velocizzano il modo di produrre software.

2.2.5.2 Papyrus

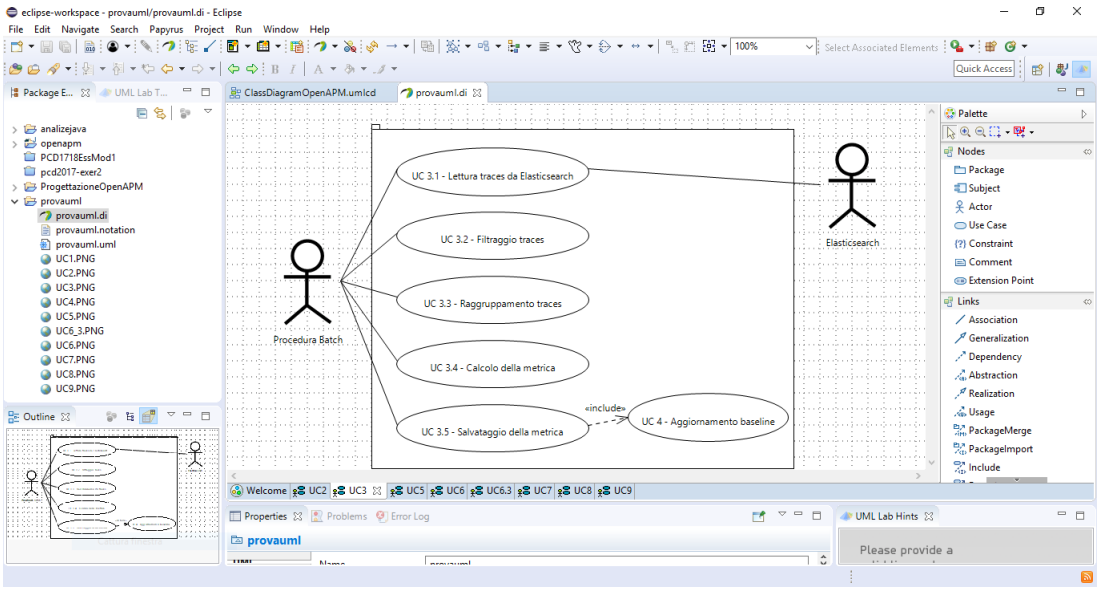

Figura 5: Papyrus: ambiente di modellazione UML utilizzata dal gruppo per disegnare i casi d'uso

Papyrus è un sofisticato e potente ambiente di modellazione UML integrato ad Eclipse; inizialmente utilizzato dal gruppo per disegnare casi d'uso, il suo utilizzo è stato accantonato perché risultava confusionario ai progettisti.

2.2.5.3 Visual Paradigm 15.0 Community Edition

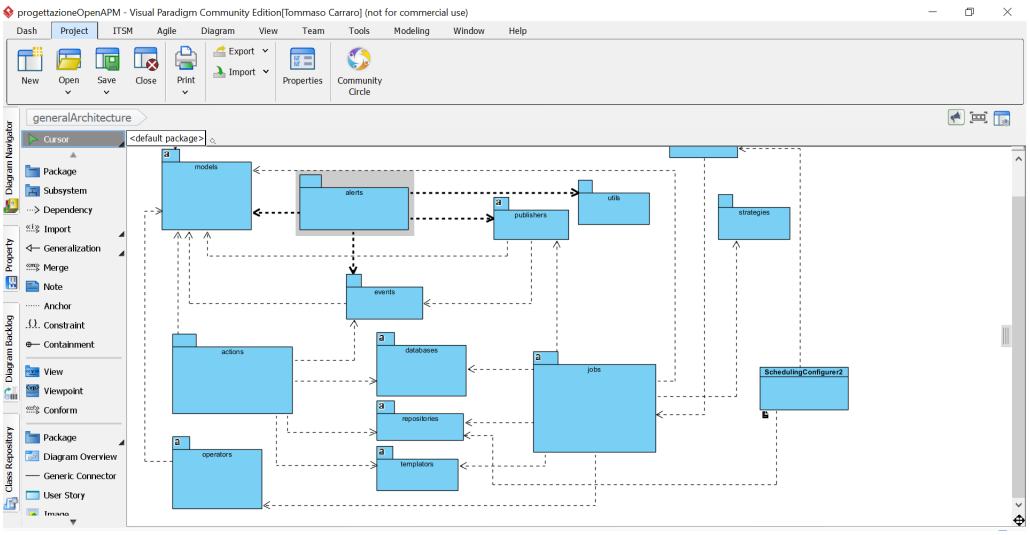

Figura 6: Visual Paradigm 15.0 Community Edition: applicazione utilizzata dal gruppo per la modellazione UML

Visual Paradigm 15.0 Community Edition è stato scelto, a scapito di Papyrus a seguito di *VI_20180407.3*, come applicazione per modellare diagrammi UML di classe e di sequenza per via della sua intuitività di utilizzo.

2.2.5.4 IntelliJ IDEA

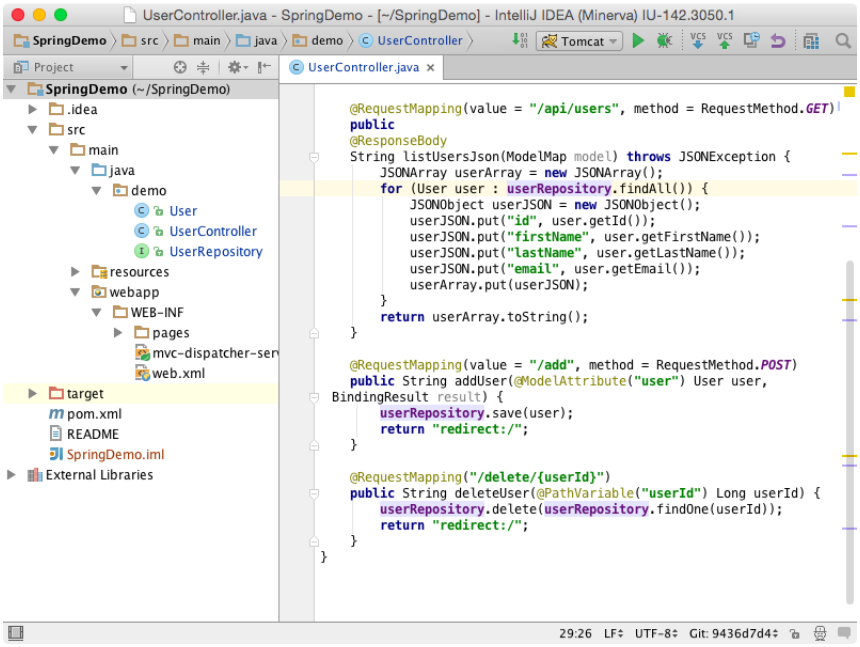

Figura 7: IntelliJ IDEA: IDE utilizzato dal gruppo per la codifica del software

È un potente IDE che presenta integrazione con *GitHub*$_G$ e strumenti di analisi statica utili ai processi di supporto allo sviluppo. L'utilizzo dello stesso IDE, con la stessa configurazione, da parte di tutti i Programmatori preverrà o quantomeno faciliterà la soluzione di problemi riguardanti l'ambiente di sviluppo.

3 Processi di supporto

3.1 Documentazione

In questa sezione vengono illustrate le norme e le decisioni prese dal gruppo di progetto in merito alla stesura, la verifica e l'approvazione della documentazione. Queste norme devono essere rispettate per la produzione di documenti validi, comprensibili e coerenti.

3.1.1 Scopo del processo

Il processo di documentazione si prefigge lo scopo di registrare le informazioni prodotte dai processi del ciclo di vita del software. Questo processo contiene l'insieme di attività atte alla pianificazione, progettazione, sviluppo, produzione, modifica, distribuzione e mantenimento dei documenti necessari al team di progetto e agli utenti del sistema o del prodotto software.

3.1.2 L'importanza della documentazione

Il processo di documentazione è uno dei processi di supporto più importanti, esso aiuta a:

- assicurare che i processi produttivi si svolgano con la qualità attesa;
- garantire una completa, accurata, tempestiva e non-invasiva circolazione di informazione;
- facilitare il controllo di avanzamento secondo le regole del modello di sviluppo adottato;
- segnare il confine tra creatività e disciplina.

3.1.3 Categorie di documenti

Esistono tre categorie di documenti formali:

- **Documentazione tecnica:** fanno parte di questa categoria la *Technology Baseline*$_G$ e la *Product Baseline*$_G$. Questi documenti vengono presentati e discussi in stile *Agile*$_G$ in finestre di opportunità che precedono le corrispondenti revisioni;
- **Documentazione gestionale:** fanno parte di questa categoria il Piano di Progetto, il Piano di Qualifica e le Norme di Progetto. Questi documenti vengono redatti e sottomessi per notificare candidatura alla revisione;
- **Documentazione di relazione con l'utente:** fanno parte di questa categoria i manuali e l'Analisi dei Requisiti. Questi documenti vengono redatti e sottomessi per notificare candidatura alla revisione.

Inoltre ogni documento può essere:

- **Interni**: sono i documenti utilizzati dal team di progetto;
- **Esterni**: sono i documenti condivisi con la Proponente in sede di revisione dei requisiti, e con il *Committente*$_G$ nelle revisioni successive.

3.1.4 Struttura repository documentale

Nella repository, la cartella relativa alla documentazione (Documenti) deve contenere quattro cartelle:

- RR (Revisione dei Requisiti);
- RP (Revisione di Progettazione);
- RQ (Revisione di Qualifica);
- RA (Revisione di Accettazione).

Ognuna di queste deve contenere due cartelle:

- **Interni** - contiene i documenti interni;
- **Esterni** - contiene i documenti esterni.

I verbali devono essere posti in un'apposita cartella, denominata "Verbali", posta all'interno della documentazione interna (se si tratta di un verbale interno) o esterna (se si tratta di un verbale esterno).

3.1.5 Documenti da produrre

Questa sezione illustra i documenti che dovranno essere presentati alle varie revisioni di progetto. I documenti sono elencati in ordine alfabetico e tra parentesi viene indicato il loro utilizzo (esterno o interno):

- **Analisi dei Requisiti** (uso esterno): lo scopo del documento è di esporre i requisiti del progetto a *stakeholder*$_{\text{G}}$, Proponente e Committenti. Una descrizione più accurata sarà presente nella sezione "Scopo" del documento stesso;
- **Glossario** (uso esterno): il Glossario è un documento unico per tutti i documenti formali. Lo scopo di tale documento è raggruppare tutti i termini ambigui al fine di aiutare i membri del team a comprendere tali termini ed evitare usi impropri degli stessi;
- **Manuale Utente** (uso esterno): lo scopo del documento è di fornire una guida, a supporto dell'utente finale, per l'installazione e l'esecuzione del prodotto collaudato;
- **Norme di Progetto** (uso interno): lo scopo del documento è di illustrare le regole con le quali il progetto viene sviluppato dai membri del team. Sono presenti regole per i processi primari, di supporto e organizzativi;
- **Piano di Progetto** (uso esterno): lo scopo del documento è di mostrare come il team gestisce le proprie risorse, come attribuisce responsabilità ed autorità e come il progetto viene distribuito nel tempo di calendario;
- **Piano di Qualifica** (uso esterno): lo scopo del documento è di illustrare le metodologie adottate per effettuare verifica e validazione in maniera tale da ottenere un prodotto di qualità, conforme ad aspettative e requisiti;
- **Studio di Fattibilità** (uso interno): lo scopo del documento è di mostrare i motivi per i quali il gruppo ha scelto il capitolato d'appalto. Il documento contiene inoltre le motivazioni per le quali il gruppo ha deciso di escludere gli altri capitolati proposti;
- **Technology Baseline**: lo scopo di questo semi-elaborato è presentare le tecnologie, i framework e le librerie selezionate per lo sviluppo del prodotto. L'adeguatezza e il grado di integrazione

di queste componenti sono dimostrati tramite un *Proof-of-Concept*$_G$ correlato agli obiettivi del progetto. La Technology Baseline forma parte integrante della revisione di progettazione e deve essere presentata insieme al PoC, in forma di discussione Agile, al docente Cardin;

- **Product Baseline**: lo scopo di questo semi-elaborato è presentare la baseline architetturale del prodotto, mostrandone la coerenza con quanto dimostrato in Technology Baseline. La baseline viene presentata tramite diagrammi delle classi e di sequenza, comprensivi della contestualizzazione dei design pattern adottati, all'interno dell'architettura del prodotto. Questo elaborato forma parte integrante della revisione di qualifica e deve essere presentato insieme al prodotto finale, in forma di discussione Agile, al docente Cardin;
- **Verbali** (uso interno/esterno): sono documenti redatti da un segretario in occasione di incontri interni al gruppo o con altre entità esterne. Possono contenere un elenco di decisioni prese, in caso di incontro interno, oppure risposte a domande poste ai Committenti, in caso di incontro esterno.

3.1.6 Nomenclatura e versionamento dei documenti

Ciascun documento deve avere un nome significativo ed un numero di versione (esempio: NomeDocumento_vX.Y.Z), ad eccezione dei verbali che non necessitano di controllo di versione. Le regole per il nome e la versione sono le seguenti:

- **Nome del documento**: il nome non deve contenere spazi e bisogna usare una lettere maiuscola per iniziare ogni parola;
- **Numero di versione del documento**: il numero di versione viene separato dal nome tramite un underscore (_) ed il significato della sua composizione è spiegato nella sezione §3.2.2.2.

3.1.7 Ciclo di vita dei documenti

Ogni documento può trovarsi in una delle seguenti tre fasi di ciclo di vita:

1. **Sviluppo**: lo sviluppo di un documento inizia da quando ha inizio la sua stesura e termina quando esso viene approvato in versione 4.0.0 (ultima versione pianificata). Il documento può passare alla fase di verifica da parte di un Verificatore se il Responsabile di Progetto ritiene che esso sia pronto per la verifica;
2. **Verifica**: la verifica di un documento deve essere eseguita da un Verificatore secondo le modalità espresse dal Piano di Qualifica per la documentazione. Al termine della verifica, il Responsabile, dopo averne esaminato accuratamente l'output, può decidere di approvare il documento per il rilascio oppure segnalare i problemi riscontrati agli sviluppatori dello stesso. In tal caso il documento rientra in fase di sviluppo per la correzione degli errori;
3. **Approvazione**: il documento viene approvato per il rilascio dal Responsabile di Progetto in seguito ad un esito di verifica positivo da parte dei Verificatori. Per segnalare l'importanza dell'approvazione di un documento, il suo numero di versione viene aggiornato dal Responsabile di progetto nelle modalità descritte dalla sezione §3.2.2.2.

3.1.8 Struttura dei documenti

3.1.8.1 Template

Per la formattazione dei documenti deve essere utilizzato il *template*$_G$, creato appositamente, così da unificare la struttura dei documenti, in modo da renderli il più uniformi possibili. Il template è stato creato mediante il linguaggio LaTeX.

3.1.8.2 Frontespizio

Il frontespizio è la prima pagina di ogni documento e deve essere composto dalle seguenti parti in ordine di apparizione:

- **Logo**: è il logo del gruppo MILCTdev;
- **Nome del documento**: indica il nome del documento che si sta visionando (es. Norme di Progetto o Analisi dei Requisiti);
- **Informazioni su gruppo e progetto**: sono indicati il nome del gruppo ed affianco il nome del capitolato scelto. Sotto queste informazioni è indicata la mail del gruppo;
- **Informazioni sul documento**: sotto forma di tabella sono indicate le informazioni principali del documento:
  1. **Versione**: indica la versione corrente del documento;
  2. **Redazione**: indica il nome dei membri del gruppo che si sono impegnati nella redazione del documento;
  3. **Verifica**: indica il nome dei membri del gruppo che si sono impegnati nella verifica del documento (Verificatori);
  4. **Approvazione**: indica il nome del componente del gruppo che si è impegnato nell'approvazione del documento (Responsabile di Progetto);
  5. **Uso**: specifica l'uso del documento, in particolare se si tratta di un documento interno oppure esterno;
  6. **Distribuzione**: indica il nome delle persone o dei gruppi alle quali il documento sarà distribuito.
- **Descrizione**: contiene una breve descrizione del documento che si sta visionando.

3.1.8.3 Registro delle modifiche

Il registro delle modifiche deve contenere, sotto forma di tabella, tutte le informazioni riguardanti le modifiche che sono state effettuate sul documento durante la sua stesura.
Ogni componente del gruppo deve aggiornare il registro delle modifiche ogni qual volta lavora in modo significativo in un documento. Il registro deve essere presente nella seconda pagina di ogni documento. Ogni riga della tabella è composta da 5 colonne:

- **Versione**: indica la versione del documento; le modalità tramite le quali questo numero deve essere incrementato sono spiegate nella sezione §3.2.2.2;
- **Ruolo**: indica il ruolo della persona che ha effettuato la modifica al documento;

- **Nominativo**: indica il nome della persona che ha effettuato la modifica al documento;
- **Descrizione**: riporta una piccola descrizione della modifica apportata al documento;
- **Data**: indica la data della modifica.

3.1.8.4 Indice

Ogni documento, ad eccezione dei verbali, deve avere un indice che ne permetta una lettura *ipertestuale*$_G$ e non per forza sequenziale. Nell'indice sono riportati capitoli, sottosezioni e sottosezioni di sottosezioni. La numerazione delle sezioni deve rispettare le seguenti regole:

1. **Capitolo**: la numerazione deve partire da uno ed essere del tipo "a." dove "a" è l'indice del capitolo;
2. **Sottosezione**: la numerazione deve partire da uno ed essere del tipo "a.b" dove "a" è l'indice del capitolo e "b" è l'indice della sottosezione del capitolo "a";
3. **Sottosezione di sottosezione**: la numerazione deve partire da uno ed essere del tipo "a.b.c" dove "a" è l'indice del capitolo, "b" è l'indice della sottosezione del capitolo "a" e "c" è l'indice della sottosezione della sottosezione "b" del capitolo "a".

Se nel documento sono presenti delle immagini o delle tabelle, deve essere presente un indice contenente il nome delle immagini e delle tabelle.

3.1.8.5 Sezione introduttiva

Il corpo del documento inizia sempre sotto l'indice, con una sezione denominata "Introduzione", la quale struttura è standard per ogni documento. Questa sezione contiene le seguenti quattro sottosezioni:

- **Scopo del documento:** varia tra i vari documenti e contiene una descrizione dettagliata dello scopo del documento;
- **Scopo del prodotto:** è replicata per ogni documento e contiene una descrizione dettagliata dello scopo del prodotto;
- **Glossario:** è replicata per ogni documento e illustra come utilizzare il Glossario in presenza di termini marcati nel documento;
- **Riferimenti:** varia tra i vari documenti e contiene i riferimenti **normativi** e **informativi**. La regole per la scrittura dei riferimenti sono indicate in sezione §3.1.9.6.

3.1.8.6 Contenuto principale

Tutte le pagine di ogni documento, ad eccezione del frontespizio, devono contenere un'intestazione ed un piè di pagina, costruiti tramite il template. L'intestazione e il piè di pagina sono separati dal contenuto principale da una linea orizzontale.
L'intestazione contiene:

- **Indirizzo e-mail**: il nome e l'indirizzo e-mail del gruppo sono situati a sinistra;
- **Nome documento**: il nome del documento, con la rispettiva versione, è posto a destra.

Il piè di pagina contiene:

- **Logo**: il logo del gruppo è situato a sinistra;
- **Indicazione pagina**: l'indicazione del numero di pagina è posta a destra.

3.1.8.7 Immagini

Se sono presenti immagini nel documento queste devono essere riportare in un indice separato. Le immagini devono essere inoltre centrate rispetto al documento e devono comprendere necessariamente una didascalia che ne descriva il contenuto.

3.1.8.8 Verbali

I verbali hanno una loro struttura, diversa dalle altre tipologie di documenti. Come gli altri documenti contengono il frontespizio, ma non contengono l'indice essendo documenti brevi. La struttura di ogni verbale interno o esterno deve essere la seguente:

- **Informazioni incontro**: questa sezione contiene informazioni riguardanti l'incontro tenuto, in particolare:
  1. **Luogo**: indica il luogo dove si è tenuto l'incontro. Potrebbe essere un'aula, in caso di incontro interno, oppure la sala riunioni di un'azienda, in caso di incontro esterno;
  2. **Data**: indica la data dell'incontro;
  3. **Orario**: indica l'ora di inizio incontro e l'ora di fine incontro, separate da un trattino;
  4. **Componenti interni**: indica i nomi dei componenti del gruppo che hanno preso parte all'incontro;
  5. **Componenti esterni**: indica i nomi delle figure esterne al gruppo che hanno preso parte all'incontro, questa voce è presente solo se si tratta di un incontro esterno.
- **Argomenti**: questa sezione contiene una breve descrizione di quali sono stati gli argomenti discussi durante l'incontro;
- **Riassunto incontro**: questa sezione descrive concretamente le decisioni prese, in caso di incontro interno, oppure le risposte a delle domande fatte, in caso di incontro esterno;
- **Riepilogo decisioni**: poiché ogni decisione, così come ogni requisito, deve essere tracciabile (una decisione può portare alla definizione di un nuovo requisito, per cui è importante sapere dove e quando esso ha avuto origine) è importante tenere traccia delle decisioni prese tramite opportuni codici. La struttura del codice deve essere la seguente:
  1. **ID verbale**: è un identificativo del verbale. Può essere VI, se si tratta di un verbale interno, oppure VE, se si tratta di un verbale esterno;
  2. **Data**: la data, nel codice, è separata dal ID tramite un underscore. La data è scritta senza spazi tra anno, mese e giorno e deve coincidere con la data dell'incontro (es. 20171128 indica il 28 Novembre 2017);
  3. **Numero decisione**: il numero della decisione deve partire sempre da uno ed essere incrementato per ogni decisione diversa presa. È separato dalla data tramite un punto.

  Un esempio di codice per una decisione di verbale è il seguente: VI_20171128.1, che sta per decisione uno del verbale interno tenutosi in data 28 Novembre 2017.

3.1.9 Stile di stesura dei documenti

3.1.9.1 Date

Le date devono essere scritte nel formato **AAAA-MM-GG**, dove GG rappresenta il giorno, MM il mese e AAAA l'anno.

3.1.9.2 Orari

Gli orari devono essere scritti nel formato **HH:MM**, dove HH rappresenta l'ora e MM rappresenta i minuti. I secondi non sono ritenuti importanti per lo scopo del progetto.

3.1.9.3 Elenchi puntati

Negli elenchi puntati ciascuna voce interna deve finire con un punto e virgola, mentre l'ultima voce dell'elenco deve terminare con un punto. Se l'elenco puntato rappresenta un elenco di definizioni, o punti salienti, dopo il nome del termine si inserisce ":" oppure "-" e poi si scrive la definizione del termine o la spiegazione del punto saliente (es. Analisi dei requisiti: "..." oppure Descrizione - "..."). Il nome del termine deve essere in grassetto.

3.1.9.4 Menzione ruoli

Ciascun ruolo deve essere indicato con la lettera iniziale maiuscola (es. Verificatore, Responsabile).

3.1.9.5 Riferimenti a termini nel glossario

I termini all'interno del documento che rientrano nel Glossario, vanno indicati con una maiuscola posta a pedice ed il testo deve essere in corsivo. Deve essere marcata solo la prima occorrenza del temine in ogni documento.

3.1.9.6 Riferimenti a risorse esterne

Ciascun link o collegamento ad una risorsa esterna al documento deve essere indicato con caratteri di colore blu.

3.1.9.7 Riferimenti a documenti di progetto

I nomi dei documenti legati al progetto vengono indicati con le lettere maiuscole (es. Analisi dei Requisiti).

3.1.10 Strumentazione

Per la stesura di tutta la documentazione il gruppo deve utilizzare il linguaggio LaTeX. Gli strumenti utilizzati a supporto di tale stesura sono:

- **TexMaker**: IDE gratuito per LaTeX, moderno e multipiattaforma.

  Esso include: supporto Unicode, controllo ortografico, auto-completamento ed un visualizzatore pdf integrato con supporto a SyncTeX per la sincronizzazione continua;

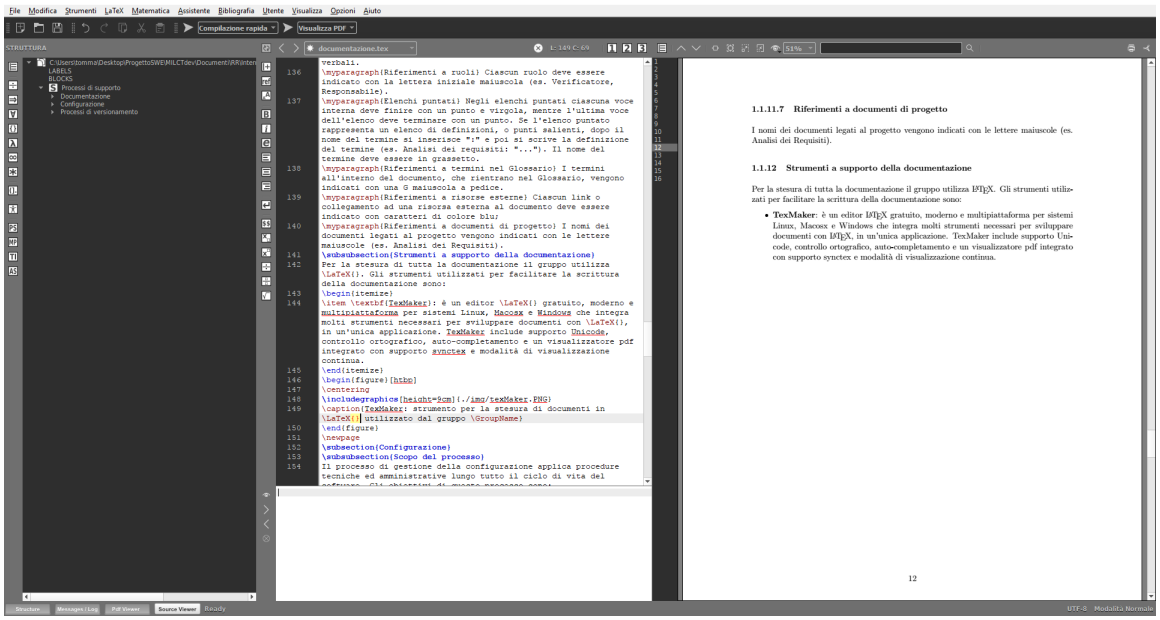

Figura 8: TexMaker: editor usato dal gruppo per la stesura di documenti in LaTeX

- ***Gantt Project***$_G$: software gratuito per la creazione di diagrammi di Gantt, rappresentanti task e *milestone*$_G$;
- **Microsoft Office Excel 2016**: software che permette di creare istogrammi e diagrammi a torta dinamici, in maniera veloce ed intuitiva;
- **Astah**: applicazione per la creazione di diagrammi UML di attività;
- **Script per la generazione di tabelle fonti-requisiti**: script sviluppato internamente al team in linguaggio PHP per generare il codice LaTeX delle tabelle "fonti-requisiti" e "requisiti-fonti" presenti in *Analisi dei Requisiti v4.0.0*;
- **Script per l'individuazione di termini di Glossario**: script sviluppato dal team in linguaggio PHP che individua, dato un file .tex, tutte le prime occorrenze di termini di Glossario in tale file.

3.2 Configurazione

3.2.1 Scopo del processo

Il processo di gestione della configurazione applica procedure tecniche ed amministrative lungo tutto il ciclo di vita del software. Gli obiettivi di questo processo sono:

- identificare e definire gli elementi software in un sistema;
- controllare modifiche e rilasci degli elementi software;
- registrare e riportare lo stato degli elementi software e delle richieste di modifica;
- garantire completezza, correttezza e coerenza degli elementi software;
- controllare conservazione, gestione e consegna degli elementi software.

3.2.2 Processi di versionamento

Per il versionamento del software e della documentazione, si utilizza il software di controllo di versione Git_{G}. In particolare il servizio GitHub.

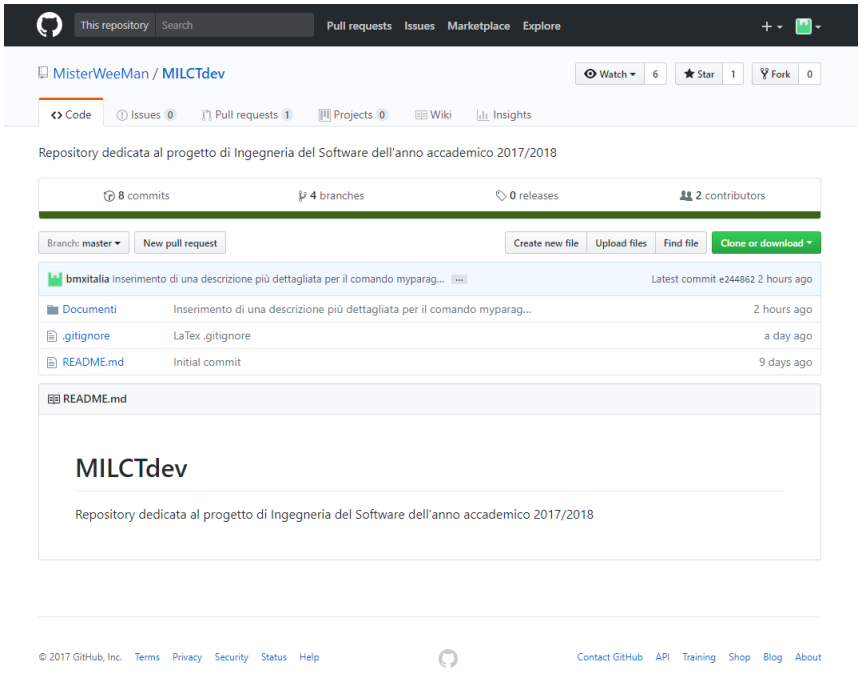

Figura 9: GitHub: strumento utilizzato dal gruppo per il controllo di versione

3.2.2.1 Struttura della copia locale del repository

La copia locale della repository è composta da tre "alberi" mantenuti da Git. Il primo è la directory di lavoro che contiene i files attuali su cui si sta lavorando. Il secondo è l'index ($stage_{G}$) che fa da spazio di transito per i files. Per finire c'è l'$head_{G}$, che punta all'ultimo $commit_{G}$ fatto.

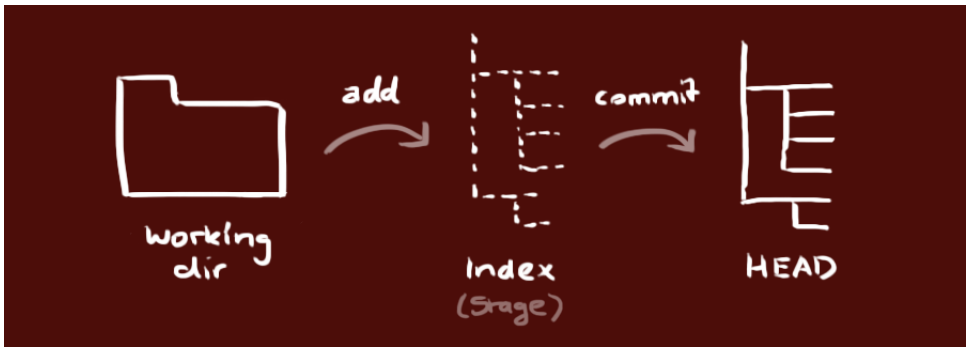

Figura 10: Struttura della repository locale, divisa nei tre alberi: working dir, stage e head

3.2.2.2 Codici di versione della documentazione

Come illustrato nella §3.1.6. il nome di un documento presenza il codice _vX.Y.Z che rappresenta la versione del documento. Il significato delle lettere è il seguente:

- **Z**: è un numero che viene incrementato di uno ogni volta che viene apportata una modifica al documento. Questo numero viene incrementato da colui che ha apportato la modifica;
- **Y**: è un numero che viene incrementato di uno ogni volta che il documento è stato verificato da un Verificatore. È quindi compito del Verificatore incrementare questo numero, nessun altro membro del team può farlo;
- **X**: è un numero che viene incrementato di uno ogni volta che il documento viene approvato dal Responsabile di Progetto per il rilascio. È compito del Responsabile di Progetto incrementare questo numero. Ogni volta che un documento raggiunge l'approvazione può essere generato il file pdf del documento.

Ogni volta che un numero viene incrementato, tutti i numeri alla sua destra devono essere settati a zero.

3.3 Garanzia di qualità

In questa sezione vengono definite le norme e la struttura delle metriche e degli obiettivi di qualità, descritti nel *Piano di Qualifica v4.0.0*. Metriche ed obiettivi devono essere decise dagli Amministratori in collaborazione con i Verificatori.

3.3.1 Notazione per la classificazione

Sia gli obiettivi di qualità che le metriche devono essere classificati secondo la seguente notazione:

{**Classe**}{**Tipo**}{**Oggetto**}*{**codice_identificativo**} : {**Nome**}

- **Classe:** indica se si tratta di un obiettivo di qualità o una metrica. Può essere:
  - **O:** per un obiettivo;
  - **M:** per una metrica.
- **Tipo:** stabilisce se riguarda un prodotto o un processo e può assumere i valori:
  - **PD:** per gli obiettivi di prodotto;
  - **PC:** per gli obiettivi di processo.
- **Oggetto:** nel caso di obiettivi o metriche di prodotto, indica se si riferisce alla parte software oppure ad un documento. Può essere:
  - **D:** per i documenti;
  - **S:** per il software.
- **codice_identificativo:** codice numerico incrementale necessario per l'identificazione;
- **Nome:** titolo che descrive l'obiettivo o la metrica.

3.3.2 Obiettivi di qualità

In questa sezione vengono illustrati gli obiettivi che MILCTdev intende raggiungere per assicurare la qualità di processo e di prodotto per quanto riguarda la realizzazione di OpenAPM. Inoltre, per ognuno di questi obiettivi, vengono fissate metriche per rendere quantificabile il raggiungimento della qualità di processo e di prodotto; queste sono descritte nella §3.3.3.

3.3.2.1 Qualità di processo

Per realizzare un prodotto valido, MILCTdev ha deciso di adottare lo standard ISO/IEC 15504 per valutare la qualità di ogni processo necessario allo sviluppo di OpenAPM. Viene inoltre utilizzato il ciclo di Deming per assicurare un miglioramento continuo dei processi, senza eventuali regressioni. Nell'appendice C vengono approfonditi questo metodo e lo standard utilizzato.

Gli obiettivi fissati per i processi sono:

- rispettare tempi e costi descritti nel *Piano di Progetto v4.0.0*;

- avere prestazioni sempre misurabili;
- perseguire un miglioramento continuo delle stesse.

3.3.2.2 Qualità di prodotto

Basandosi sullo standard ISO/IEC 9126, descritto nell'appendice C, sono stati fissati obiettivi che mirano a garantire la qualità del prodotto finale. Questi sono:

- i **documenti** devono:
  - essere leggibili e comprensibili a chiunque;
  - essere corretti dal punto di vista ortografico, sintattico, semantico e logico.
- il **software** deve:
  - soddisfare tutti i requisiti obbligatori descritti in *Analisi dei Requisiti v4.0.0*;
  - garantire usabilità e manutenibilità;
  - essere affidabile.

3.3.2.3 Tabella degli obiettivi

Viene qui riassunto ogni obiettivo, classificandolo con il suo codice identificativo e indicando le metriche che ne quantificano il raggiungimento. Per una descrizione delle metriche vedere nella sezione §3.3.3.

ID	Nome	Metrica
OPC1	Coerenza con Piano di Progetto	MPC1:Schedule Variance MPC2:Cost Variance
OPC2	Miglioramento continuo	MPC3:SPICE
OPDD1	Leggibilità documenti	MPDD1:Indice Gulpease
OPDS1	Implementazione requisiti obbligatori	MPDS6:Requisiti obbligatori soddisfatti
OPDS2	Manutenibilità e usabilità	MPDS1:Grado di accoppiamento MPDS2:Code coverage MPDS3:Rapporto linee di commento per linee di codice MPDS4:Complessità ciclomatica
OPDS3	Affidabilità	MPDS5:Percentuale superamento test

Tabella 3: Tabella degli obiettivi

Ogni obiettivo si riterrà raggiungo solamente al raggiungimento del valore minimo di ogni metrica che concorre alla quantificazione del suo grado di raggiungimento.

3.3.3 Metriche e misure

Ogni processo ed ogni prodotto dovrebbero sempre presentare un set di KPI_G che permettano il tracciamento, la comunicazione ed il miglioramento della loro qualità.
In questa sezione pertanto, si provvederà alla presentazione delle metriche che permettano di quantificare e valutare la qualità dei processi e dei prodotti di MILCTdev. Le spiegazioni e le modalità di calcolo di ogni metrica sono definite nell'appendice B.

Per valutare gli esiti ottenuti dall'utilizzo delle metriche, sono stati individuati tre diversi range di risultati possibili che indicano ciascuno un diverso grado di raggiungimento dell'obiettivo di qualità. Questi sono:

- **Valore negativo:** il valore della misurazione in questo caso viene anche definito inaccettabile, in quanto non soddisfa la qualità minima desiderata;
- **Valore minimo:** valori al di sopra di questa soglia possono essere accettati, ma saranno oggetto di ulteriore analisi in vista di un miglioramento desiderato;
- **Valore ottimale:** rappresenta il valore indice di qualità, da mantenere nel tempo.

Le soglie di accettazione per ogni metrica sono descritti nel *Piano di Qualifica v4.0.0*.

3.3.3.1 Tabella delle metriche

Nella seguente tabella vengono indicati, oltre a Identificativo e Nome, anche l'obiettivo a cui si riferisce.

ID	Nome	Obiettivo
MPC1	Schedule Variance	OPC1:Coerenza con Piano di Progetto
MPC2	Cost Variance	OPC1:Coerenza con Piano di Progetto
MPC3	SPICE	OPC2:Miglioramento continuo
MPDD1	Indice Gulpease	OPDD1:Leggibilità documenti
MPDS1	Grado di accoppiamento	OPDS2:Manutenibilità e usabilità
MPDS2	Code coverage	OPDS2:Manutenibilità e usabilità
MPDS3	Rapporto linee di commento per linee di codice	OPDS2:Manutenibilità e usabilità
MPDS4	Complessità ciclomatica	OPDS2:Manutenibilità e usabilità
MPDS5	Percentuale superamento test	OPDS3:Affidabilità

ID	Nome	Obiettivo
MPDS6	Requisiti obbligatori soddisfatti	OPDS1:Implementazione requisiti obbligatori

Tabella 5: Tabella delle metriche

3.4 Verifica

3.4.1 Scopo del processo

Il processo di verifica è un processo che determina se i prodotti software di un'attività soddisfano i requisiti o le condizioni imposte ad essi nelle attività precedentemente svolte. Per costi ed efficacia delle prestazioni, la verifica dovrebbe essere integrata, quanto prima possibile, nel processo che la impiega.

3.4.2 Analisi

3.4.2.1 Analisi statica

É una tecnica che studia il codice e la documentazione e ne verifica la conformità alle regole, l'assenza di difetti e la presenza di proprietà positive. Questa tecnica non richiede esecuzione del prodotto software in alcuna sua parte per cui è essenziale finché il sistema non è completamente disponibile. É attuabile tramite due tecniche:

- **Walkthrough**: attività che richiede la collaborazione di più persone per effettuare una lettura a largo spettro di tutto il documento e il codice in esame, con lo scopo di rivelare la presenza di difetti. Ogni difetto riscontrato verrà discusso tra Verificatore e autore; è importante che ci sia una terza figura che mantenga il controllo della discussione, detta arbitro. Per ogni documento deve essere redatta una lista di controllo con i difetti rilevati e le decisioni prese. Le fasi del walkthrough sono le seguenti:
  - pianificazione;
  - lettura;
  - discussione;
  - correzione dei difetti.

  Ognuna delle precedenti fasi elencate deve essere documentata;
- **Inspection**: attività normalmente svolta da una sola persona che effettua una lettura mirata e strutturata del documento, volta a localizzare gli errori segnalati nella lista di controllo; tramite controlli ripetuti verrà progressivamente ampliata per rendere più efficace l'attività di inspection. Le fasi di inspection sono:
  - pianificazione;
  - definizione della lista di controllo;
  - lettura;
  - correzione dei difetti.

  Ognuna delle precedenti fasi elencate deve essere documentata.

Essendo walkthrough una tecnica poco efficiente verrà impiegata principalmente nella prima parte del progetto. Nelle fasi successive si utilizzerà la lista di controllo prodotta da walkthrough e dalle segnalazioni dei committenti per effettuare inspection.

3.4.2.2 Analisi dinamica

É una tecnica di analisi del prodotto software che richiede l'esecuzione del programma. Vengono effettuati dei test su parti del sistema per verificare che ognuna di esse produca il risultato desiderato.
Prima di eseguire un qualsiasi test è necessario conoscere la precondizione e postcondizione del codice analizzato per poterne decretare l'esito finale. Ci sono diversi tipi di test:

- **Test di Unità**: va ad isolare la parte più piccola di software testabile nell'applicazione, chiamata unità, per stabilire se essa funziona esattamente come previsto. Per effettuare test di unità è necessaria la scrittura di codice fittizio. Può essere di due tipologie:
  - **Driver**: è un software che guida il test d'unità sostituendo l'unità chiamante per testare l'unità chiamata;
  - **Stub**: è un software che simula il comportamento delle unità chiamate per testare un'unità chiamante.

  Ogni unità deve essere sottoposta a test prima di poter essere integrata con quelle già testate. Inoltre, ogni test di unità dovrà avere un codice identificativo che segue la seguente nomenclatura:

  $$TU = \text{[Numero progressivo]}$$

- **Test di Integrazione**: prevede la combinazione di unità già testate in un unico componente per verificare che il loro funzionamento integrato abbia esito atteso. Richiede una strategia di integrazione incrementale che può essere di due tipologie:
  - **Bottom-up**: si sviluppano e si integrano prima le unità con minore dipendenza funzionale e maggiore utilità e poi si risale l'albero delle dipendenze;
  - **Top-down**: si sviluppano prima le unità più esterne poste sulle foglie dell'albero delle dipendenze e poi si scende l'albero.

  Ogni test di integrazione prevede un codice che lo identifica così formato:

  $$TI = \text{[Numero progressivo]}$$

- **Test di Regressione**: questo test deve essere eseguito ad ogni modifica ad un implementazione del sistema. Vengono ripetuti sul codice modificato i test precedentemente utilizzati, per verificare che le modifiche effettuate non abbiano alterato elementi precedentemente funzionanti. I contenuti di tale test devono essere decisi nel momento in cui si approvano modifiche al software. Ogni test di regressione prevede un codice che lo identifica così formato:

  $$TR = \text{[Numero progressivo]}$$

- **Test di Sistema**: ha inizio con il completamento del test d'integrazione e verifica il comportamento dinamico del sistema completo rispetto ai requisiti software. Ogni test di sistema dovrà avere un requisito correlato da testare e dovrà rispettare la seguente nomenclatura:

  $$TS = \text{[Tipologia][Importanza][Codice]}$$

  Dove Tipologia e Importanza sono dedotti dal requisito correlato che il test andrà a verificare, mentre Codice è un codice composto da una serie di numeri separati tramite punto che identificano il test in maniera univoca e lo esprimono gerarchicamente;

- **Test di Accettazione**: prevede il collaudo del prodotto in presenza della Proponente. Il superamento del test permette il rilascio ufficiale del prodotto sviluppato.

3.4.3 Strumentazione

- **Hunspell**: Hunspell è un correttore ortografico integrabile con TexMaker e che supporta dizionari italiani;
- **Julia**: Julia è un analizzatore statico per Java basato sulla tecnica scientifica dell'interpretazione astratta che garantisce la precisione e l'affidabilità dei suoi risultati.

3.5 Validazione

3.5.1 Scopo del processo

Il seguente processo ha lo scopo di determinare in maniera oggettiva la conformità dei requisiti e del prodotto software realizzato, rispetto all'uso prestabilito all'assegnazione dell'appalto come dimostrazione di adempimento contrattuale alla Committente.

3.5.2 Attività di validazione

La validazione può effettuarsi solo sul prodotto software ultimato; precondizione è, quindi, il superamento di tutti i test di unità e di integrazione, indicativi della corretta implementazione dell'architettura progettata.

Sul software vengono, dunque, effettuati in maniera interna al Fornitore e da parte di Verificatori estranei alla loro specifica a garanzia di imparzialità, i test di sistema definiti in *Piano di Qualifica v4.0.0* §2. L'esito positivo coincide, come mostrato dal *modello a* V_G, con il soddisfacimento dei requisiti individuati dall'attività di analisi e permette di affrontare il collaudo.

Il collaudo è un'attività formale supervisionata dalla Committente che consiste nella dimostrazione pratica del software prodotto di poter superare tutti i test di accettazione concordati contrattualmente.

Il superamento del collaudo coincide con il superamento della procedura di validazione, poiché conferma che i requisiti analizzati e realizzati nel software adempiono alle richieste espresse nel capitolato, e con il rilascio del prodotto secondo le modalità di fornitura stabilite.

È compito del Responsabile istanziare e pianificare la validazione affinché la si superi nei tempi dettati dal *Piano di Progetto v4.0.0* §5.6.

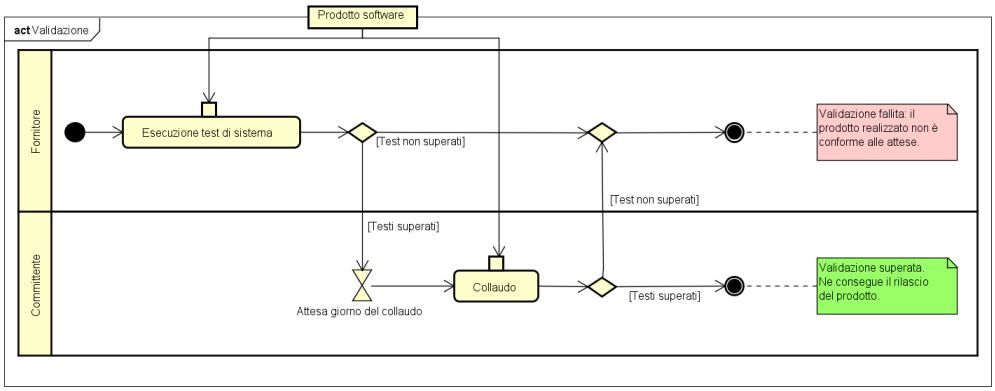

Figura 11: Procedura di validazione

4 Processi organizzativi

4.1 Gestione di processo

4.1.1 Scopo del processo

Il processo di gestione contiene le attività e i compiti generici che possono essere utilizzati da qualunque parte che deve gestire i rispettivi processi. Il manager è responsabile della gestione di prodotto, di progetto e dei compiti dei processi applicabili, come i processi di acquisizione, fornitura, sviluppo, funzionamento, manutenzione e di supporto.

4.1.2 Comunicazione

Questa sezione è dedicata all'illustrazione delle modalità di comunicazione che verranno adottate dal gruppo MILCTdev durante l'intera durata del progetto. Le comunicazioni potranno essere interne al gruppo, o potranno coinvolgere soggetti esterni ed esso quali Proponente, Committenti o altri stakeholder.

4.1.2.1 Comunicazioni interne

Le comunicazioni interne avranno luogo tramite lo strumento di collaborazione aziendale Slack.

Slack è stato preferito ad altri software di messaggistica per la possibilità di creare canali tematici e per l'alta integrabilità con altri servizi utilizzati dal team nel corso di questo progetto. All'interno del workspace Slack, i membri del gruppo dovranno comunicare nel rispetto dei temi dei canali creati, avvalendosi anche di features quali @everyone, @channel, @NomeUtente per l'invio di notifiche rispettivamente a tutti i soggetti interni al gruppo, a quelli che partecipano alla conversazione o a un utente specifico.

Il workspace Slack è suddiviso in canali specifici per permettere un corretto scambio di idee e informazioni, inerenti solamente a determinate aree del progetto. I canali presenti nel workspace sono:

- **canali dedicati a servizi utilizzati**: questi sono spazi dedicati ad integrazioni con altri servizi quali ad esempio Asana e GitHub. Viene creato un canale dedicato ad ogni integrazione;
- **canali dedicati alla documentazione**: questi sono spazi dedicati alla documentazione scritta dal team. Anche qui viene creato un canale per ogni documento da redigere, ad esempio sono presenti canali quali "norme di progetto" e "piano di qualifica". All'interno di ogni spazio verranno discusse sia le attività di sviluppo, che di verifica ed approvazione;
- **canali dedicati ad argomenti specifici**: dal canale riguardante incontri e comunicazioni con soggetti esterni, al canale dedicato al materiale informativo, a canali per la discussione dei capitolati d'appalto o della scelta di nome e logo del gruppo;
- **general e random**: canali di default, dedicati a comunicazioni non riferibili ad un singolo processo o ad una singola attività.

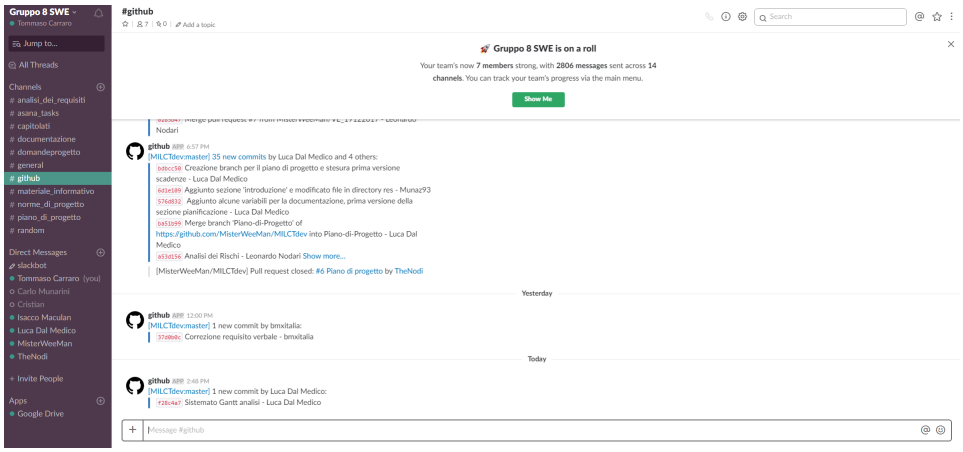

Figura 12: Slack: strumento per la comunicazione del gruppo

4.1.2.2 Comunicazioni esterne

Le comunicazioni intraprese con soggetti esterni al gruppo sono di competenza del Responsabile di Progetto che utilizzerà la casella di posta elettronica creata in fase di istituzione del team MILCTdev: milctdev.team@gmail.com. Il Responsabile di Progetto è quindi il soggetto incaricato dal gruppo per comunicare con i soggetti esterni, rappresentati da:

- **Kirey Group**, nella figura di Proponente, raggiungibile all'indirizzo di posta elettronica: stefano.bertolin@iks.it;
- **Prof. Tullio Vardanega e Prof. Riccardo Cardin**, nella figura di Committenti, raggiungibili agli indirizzi di posta elettronica istituzionali.

Tutti i membri del gruppo dovranno essere sempre aggiornati sulle conversazioni avvenute. Il compito di notificarli spetta al Responsabile di Progetto utilizzando l'apposito canale tematico all'interno del workspace Slack.

4.1.3 Incontri del team

Le riunioni del team, interne o esterne, verranno sempre organizzate in accordo con tutto il gruppo. L'organizzazione di esse spetta al Responsabile di Progetto.

4.1.3.1 Frequenza degli incontri

La frequenza è fissata ad almeno un incontro al mese, nel quale è richiesta la presenza di tutti i membri.

4.1.3.2 Verbali di riunione

Per ogni riunione verrà nominato un Segretario, a turno dal Responsabile di Progetto, il cui compito sarà redigere un verbale seguendo le norme della sezione 3.1.10.7.

4.1.3.3 Decisioni e vincoli

Il numero minimo di membri richiesto per rendere valido un incontro è cinque. Tale numero impone una collaborazione tra gli elementi del team ed è sufficientemente grande a garantire equità nelle decisioni prese. Nel caso di riunioni straordinarie, il "quorum" si abbassa a tre ma le decisioni prese dovranno essere riviste nell'incontro successivo. Ogni decisione verrà presa a maggioranza e, nel caso di parità di voti, la parola spetterà al Responsabile di Progetto che, dapprima si presterà nel ruolo di mediatore e, nel caso di fallimento delle trattative, dovrà prendere lui stesso una decisione.

4.1.4 Ruoli di progetto

Vista la natura didattica del progetto, nell'arco dell'intera durata dello stesso, ogni soggetto coprirà almeno una volta ognuno dei ruoli che verranno di seguito illustrati. Ogni membro dovrà svolgere le attività assegnate al ruolo ricoperto, come organizzato e stabilito dal documento *Piano di Progetto v4.0.0*. Al fine di non minare la qualità di prodotto all'interno del *Piano di Progetto v4.0.0* verranno pianificati dei turni tali da non creare conflitti d'interesse. Un esempio di conflitto d'interesse potrebbe essere dover verificare qualcosa che si era precedentemente scritto. I ruoli che verranno ricoperti dai membri del team sono illustrati nelle sezioni sottostanti.

4.1.4.1 Responsabile di progetto

Il Responsabile di progetto, anche chiamato Project manager, è una figura di estrema importanza all'interno del team in quanto su di lui ricadono responsabilità di pianificazione, gestione, controllo e coordinamento. Il Project manager rappresenta inoltre MILCTdev all'esterno del gruppo in quanto intrattiene rapporti con Committente e Proponente. In sunto, egli:

- gestisce, controlla e coordina risorse, umane e non;
- gestisce, controlla e pianifica le attività di progetto;
- analizza e gestisce i rischi;
- approva documenti.

Questa figura è presente durante tutta la durata del progetto.

4.1.4.2 Amministratore

L'Amministratore è una figura di supporto che mette a disposizione strumenti e tecnologie per perseguire qualità ed efficienza all'interno dell'ambiente lavorativo. Egli quindi non opera scelte gestionali ma:

- monitora e lavora per il perseguimento della qualità di prodotto;
- ricerca strumenti per l'automazione di attività, processi e compiti;
- supervisiona e si adopera per la gestione del versionamento e l'archiviazione della documentazione;
- controlla versioni e configurazioni del prodotto software;
- norma l'utilizzo degli strumenti utilizzati durante il progetto.

4.1.4.3 Analista

L'Analista è una figura che non sarà sempre presente durante il progetto, ma di estrema importanza per i compiti svolti. Egli ha l'onere di comprendere appieno il dominio del problema e, tramite le sue azioni, vincolerà anche l'operato di alcuni ruoli, quali, ad esempio, i Progettisti. I suoi compiti sono:

- studiare il dominio del problema ed il problema stesso, definendone complessità e requisiti;
- redigere documenti quali Analisi dei Requisiti e Studio di Fattibilità.

4.1.4.4 Progettista

Il Progettista è colui che, partendo dallo studio e dai vincoli posti dall'Analista, definisce una soluzione che soddisfi i requisiti. Egli è responsabile degli aspetti tecnologici e tecnici del progetto e dovrà:

- produrre una soluzione attuabile al problema;
- sviluppare un'architettura che sfrutti soluzioni note ed ottimizzate che portino ad un prodotto stabile e manutenibile.

4.1.4.5 Programmatore

Il Programmatore è la figura che provvederà alla codifica della soluzione, studiata e spiegata dal Progettista. Egli in particolar modo esegue queste operazioni:

- codifica la soluzione descritta dal Progettista nel rispetto di documenti, tra i quali le Norme di Progetto;
- crea o gestisce componenti di supporto per la verifica e la validazione del codice.

Questa figura ha inoltre il compito di manutenere il codice del prodotto e redigere un eventuale Manuale Utente.

4.1.4.6 Verificatore

Il Verificatore è una figura che è presente sin dalle prime fasi del progetto e lo sarà per tutto il ciclo di vita del software. Egli si focalizza sui processi e, grazie alla sua conoscenza delle Norme di Progetto, garantirà che essi siano conformi alle norme e alle attese. Il Verificatore controlla quindi che:

- ogni stadio del ciclo di vita del prodotto sia conforme al *Piano di Qualifica v4.0.0*;
- ogni processo sia eseguito nel rispetto delle *Norme di Progetto v4.0.0*.

4.1.5 Strumentazione

4.1.5.1 Strumenti di comunicazione

Per la comunicazione i componenti del gruppo dovranno utilizzare Slack (di cui si è ampiamente parlato in precedenza) e Gmail.

4.1.5.2 Strumenti di condivisione

Come strumento di condivisione il team dovrà utilizzare Google Drive, servizio cloud based, per il quale è presente un'integrazione Slack, che permette al gruppo un rapido scambio di documenti e file.

4.1.5.3 Strumenti di coordinamento

Il team utilizzerà Asana come strumento di coordinamento orientato al ticketing. Alcune delle funzionalità che questo strumento offre sono:

- creare un task, eventualmente inseribile in una "sezione" o all'interno di un altro task;
- indicare una persona a cui viene assegnato il task/sottotask;
- indicare una data di scadenza del task, andando così a popolare un calendario utile al coordinamento;
- inserire una descrizione del task e creare una conversazione relativa al task stesso.

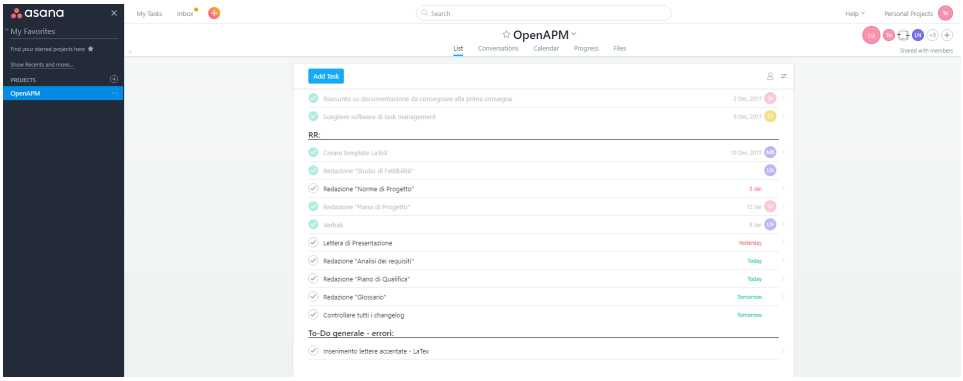

Figura 13: Asana: strumento utilizzato dal gruppo per il coordinamento del progetto didattico

4.1.5.4 Strumenti di versionamento

Per il salvataggio e versionamento dei file il team utilizzerà una repository su GitHub, il quale si basa sul sistema di versionamento Git. La repository contenente la documentazione, ha la seguente struttura (cartelle), alla quale i membri dovranno attenersi:

- **LatexLayout**: contenente file per l'agevolazione della scrittura e della creazione dei documenti in LaTeX;
- **NOMEREVISIONE**: all'interno di queste cartelle saranno presenti documenti, suddivisi nelle directory INTERNI/ESTERNI, che ospitano i file LaTeX e .pdf di ogni singolo tipo di documento.

Anche questo strumento è stato integrato nel workspace Slack, per favorire la notifica di nuovi eventi al gruppo.

4.1.5.5 Sistemi operativi

Data la mancata presenza di requisiti che impongano restrizioni sul sistema operativo da utilizzare, i membri del team potranno indistintamente usare sistemi Windows, MacOSX o Linux.

4.2 Formazione del personale

4.2.1 Scopo del processo

Il processo di formazione è un processo per fornire e mantenere personale qualificato. L'acquisizione, la fornitura, lo sviluppo, il funzionamento e la manutenzione di prodotti software dipende anche dalla preparazione e della competenza del personale. Per cui, è imperativo che il processo di formazione del personale deve essere pianificato e implementato in anticipo in modo che il personale sia sempre preparato e disponibile a lavorare in maniera efficiente ed efficace sul prodotto software.

4.2.2 Attività

Ogni membro studierà individualmente per essere preparato alla realizzazione delle varie parti del progetto. Saranno attribuiti dei task relativi allo studio di materiale complesso necessario allo svolgimento del progetto come ad esempio "imparare ad utilizzare il linguaggio LaTeX".

A Liste di controllo

Vengono qui presentate le liste di controllo da utilizzare nell'effettuare inspection.

A.1 Documenti

Documento	Lista di controllo
Tutti i documenti	• conformità delle date al formato standard; • conformità degli elenchi puntati; • conformità nell'uso delle maiuscole nei titoli; • correttezza ortografica; • correttezza della rappresentazione degli accenti in LaTeX; • correttezza della spaziatura prima della punteggiatura; • correttezza delle versioni dei documenti riferiti; • presenza della data di ultima consultazione delle fonti web.
Analisi dei Requisiti	• conformità alla struttura dei casi d'uso; • correttezza diagrammi casi d'uso; • correttezza enumerazione casi d'uso; • verificabilità dei requisiti.
Piano di Qualifica	• assenza di contenuti sovrapposti alle Norme di Progetto.
Norme di Progetto	• assenza di descrizioni di procedure in stile narrativo.
Glossario	• assenza di voci spurie.

Tabella 6: Lista di controllo per i documenti

A.2 Diagrammi UML

Diagramma	Lista di controllo
Diagrammi delle classi	• presenza della marcatura <<interface>> se la classe è un'interfaccia; • presenza della marcatura <<enumeration>> se la classe è un'enumerazione; • presenza della marcatura <<create>> in caso di Factory pattern; • separazione tra nome della classe, attributi e metodi; • ogni attributo deve avere un nome e un tipo, scritti nella corretta sintassi; • ogni metodo deve avere un tipo di ritorno, ad eccezione del costruttore; • se un metodo ha una lista di parametri, allora ogni parametro deve avere nome e tipo, scritti nella corretta sintassi; • utilizzo corretto dei modificatori d'accesso (+, -, #); • descrizione di design pattern tramite commenti con colorazione rossa; • colorazione arancione per classi e package esterni (framework o API); • l'impiego di classi esterne al package implica la modellazione tramite diagrammi di package; • sottolineatura per metodi e attributi statici; • dicitura in corsivo per le classi astratte; • dicitura maiuscola delle costanti di classe; • utilizzo della corretta sintassi per le classi parametriche; • esplicitazione del tipo con cui viene istanziata una classe parametrica nella dipendenza; • utilizzo del corretto tipo di relazione tra le classi; • verificare che non siano stati introdotti concetti legati al software.
Diagrammi di attività	• confluire di flussi mediante nodo di *merge*; • presenza freccia entrante nei nodi di *timeout*.
Diagrammi di sequenza	• presenza di un nome per ogni istanza di classe; • presenza di una barra di attivazione se un oggetto è attivo; • utilizzo della freccia piena se un messaggio è una chiamata a un metodo di un oggetto; • utilizzo del corretto operatore nei frame di iterazione; • presenza della marcatura <<create>> in caso di messaggio di creazione di un oggetto; • presenza della marcatura <<destroy>> in caso di messaggio di distruzione di un oggetto; • utilizzo della corretta sintassi per i vari tipi di messaggi.

Tabella 7: Lista di controllo per i diagrammi UML

A.3 Codice

Parte del software	Lista di controllo
Tutto il codice	• utilizzo dello stile di default di IntellJ IDEA; • ogni metodo pubblico deve avere un JavaDoc con una breve descrizione; • ogni classe deve avere un JavaDoc con una breve descrizione; • i metodi auto-generati devono essere in coda alle classi; • i metodi auto-generati devono essere opportunamente segnalati; • non devono essere specificate versioni ridondanti delle dipendenze esterne.
Spring	• la configurazione XML di Spring non deve essere usata.
Spring Batch	• i componenti di un Job non devono essere dichiarati come Bean.
Spring Data	• index e type nelle annotazioni dei Model devono provenire da Bean.
JUnit	• le linee di codice devono avere una test coverage superiore all'80%.

Tabella 8: Lista di controllo per il software

B Calcolo ed uso delle metriche

In questa sezione si provvederà alla descrizione delle metriche, presenti nella §3.3.3, che permettono di quantificare e valutare la qualità dei processi e dei prodotti di MILCTdev. All'interno della spiegazione di ogni metrica verrà illustrato quando, come e su cosa viene fatta la misurazione durante il processo di verifica.

B.1 Misure e metriche per i processi

B.1.1 Schedule Variance

La Schedule Variance è un indice di efficienza che ha come oggetto la durata temporale di un processo o di un'attività. Questa metrica aiuta il Responsabile di Progetto nella creazione dei prospetti orari inseriti nei consuntivi di periodo, e di conseguenza aiuta il team nell'analisi dell'utilizzo di risorse temporali.

Il calcolo della Schedule Variance avviene in questo modo:

$$SV = data\ conclusione\ reale - data\ conslusione\ preventivata$$

Entrambe le date si riferiscono alla conclusione dell'attività o del processo.

B.1.2 Cost Variance

La Cost Variance, o Variazione di Costo, è una metrica che analizza il costo, nonché le risorse legate ad un processo o ad un attività. Essa può essere influenzata anche dalla metrica sopracitata.

La Variazione di Costo viene così calcolata:

$$CV = costo\ delle\ risorse\ effettivo - costo\ delle\ risorse\ preventivato$$

B.1.3 SPICE

Al termine di ogni periodo, il team MILCTdev provvederà alla valutazione della qualità dei processi tramite lo standard ISO/IEC 15504 conosciuto come SPICE. Lo standard SPICE, ed i livelli di maturità, vengono illustrati in maniera completa ed approfondita nell'Appendice C.

B.2 Misure e metriche per i prodotti

B.2.1 Misure e metriche per i documenti

B.2.1.1 Indice Gulpease Per analizzare la leggibilità della documentazione prodotta, il team MILCTdev ha deciso di avvalersi dell'*indice Gulpease*$_{\text{G}}$. Questo è stato creato per venire incontro alla complessità della lingua italiana, non contemplata in altri indici, come l'*indice di Flesch*$_{\text{G}}$.

L'indice Gulpease viene calcolato tramite questa formula:

$$IG = 89 + \frac{(300 \times \textit{numero delle frasi}) - (10 \times \textit{numero delle lettere})}{\textit{numero delle parole}}$$

Il valore ottenuto indicherà la leggibilità del testo e può variare da 0, indice di bassissima leggibilità, a 100, indice di ottima leggibilità.

B.2.2 Misure e metriche per il software

Al fine di poter correttamente quantificare e valutare la qualità del prodotto software, il team MILCTdev ha deciso utilizzare diverse metriche. Gran parte delle metriche indicate in questa sezione verranno riviste ed aggiornate nel corso dei successivi periodi.

B.2.2.1 Grado di accoppiamento Questa metrica indica una misura la dipendenza di un componente del prodotto con le altri parti di OpenAPM.
Per questo calcolo si utilizza:

Structural Fan-Out (SFOUT): il grado di accoppiamento efferente permette di avere una visione del numero di moduli che usufruiscono della componente oggetto di analisi. Il valore di questo indice è semplicemente dato dal conteggio delle componenti indicate poco sopra. Un valore molto basso può indicare una scarsa utilità del modulo analizzato, all'opposto un grado troppo alto potrebbe indicare un pericoloso livello di dipendenza.

Structural Fan-In (SFIN): il grado di accoppiamento afferente, così come l'accoppiamento efferente, ha come oggetto di analisi il numero di moduli che sono legati alla componente in analisi. Questa volta si prende in considerazione il numero di moduli esterni che vengono utilizzati. Un indice ottimale dovrebbe avere un valore di 0 o 1, questo perché minore è il suo valore, più il modulo è indipendente ai cambiamenti del resto del sistema. Un valore eccessivamente alto è indice di troppa dipendenza rispetto al resto del sistema.

B.2.2.2 Code Coverage Il code coverage è una metrica che, sfruttando una misura detta Logical Source Lines of Code (Logical SLOC), indica la percentuale di statements coperti dai test.

La Logical SLOC è una metrica che dà un idea della grandezza del prodotto software contando il numero di linee di codice. Il team ha scelto di utilizzare la variante definita Logical SLOC, andando quindi a contare solamente il numero di *statements*$_{\text{G}}$ all'interno del codice.

Il valore della code coverage è così calcolato:

$$CC = \frac{Numero\ di\ statement\ coperti\ da\ test}{Logical\ SLOC} \times 100$$

B.2.2.3 Rapporto linee di commento per linee di codice

Un indice di buona manutenibilità del codice potrebbe essere il rapporto tra *Physical SLOC*$_{\text{G}}$ e numero di linee di commento all'interno dello stesso.

Il valore viene espresso in percentuale e viene così calcolato:

$$RLCLC = \frac{Numero\ di\ linee\ di\ commento}{Numero\ di\ linee\ di\ codice\ totali} \times 100$$

B.2.2.4 Complessità ciclomatica

L'indice di complessità di un programma aiuta ad identificare il numero di test necessari al raggiungimento di un coverage completo. Questa metrica software può essere applicata anche a packages, moduli, metodi o classi.

Il calcolo avviene sfruttando il *grafo di controllo di flusso*$_{\text{G}}$ e l'indice non è altro che il numero di cammini indipendenti attraverso il codice sorgente. La formula è quindi la seguente:

$$v(G) = e - n + p$$

Dove:

- **n:** è il numero di nodi del grafo, nonché il numero di tutti i gruppi indivisibili di istruzioni;
- **e:** rappresenta il numero di archi del grafo, cioè il numero di collegamenti tra due nodi tali che, il nodo seguente possa essere eseguito immediatamente dopo il nodo preso di riferimento;
- **p:** è il numero di componenti connesse.

B.2.2.5 Percentuale superamento test

Questa metrica indica quanti dei test implementati hanno esito positivo e può essere ottenuta così:

$$PST = \frac{Numero\ test\ superati}{Numero\ test\ implementati} \times 100$$

B.2.2.6 Requisiti obbligatori soddisfatti

Questa metrica aiuta il team a capire in che quantità sono stati soddisfatti i requisiti obbligatori indicati in'*Analisi dei Requisiti v4.0.0*.

Il valore viene espresso in percentuale e viene calcolato come segue:

$$ROS = \frac{Num.\ Requisiti\ obbligatori\ soddisfatti}{Num.\ Requisiti\ obbligatori\ individuati} \times 100$$

C Standard di qualità

C.1 ISO/IEC 15504

Lo standard ISO/IEC 15504, altresì conosciuto come SPICE (Software Process Improvement and Capability Determination), stabilisce un modello di riferimento per la valutazione della maturità (*capability dimension*) dei processi software (*process dimension*).

In particolare, la *process dimension* è definita in riferimento allo standard ISO/IEC 12207 per la gestione del ciclo di vita.

La *capability dimension*, invece, definisce una scala di sei livelli di maturità di processo:

- **0 - Incomplete**: il processo è fallito oppure non è stato implementato;
- **1 - Performed**: il processo è stato implementato ed ha ademptito al proprio obiettivo;
- **2 - Managed**: il processo, oltre ad essere semplicemente *performed*, è gestito in maniera organizzata, con responsabilità ben definite, pianificandone e tracciandone l'esecuzione e garantendone la qualità;
- **3 - Established**: il processo, oltre ad essere *managed*, è implementato aderendo ai principi dell'ingegneria del software e agli standard esistenti;
- **4 - Predictable**: il processo, oltre ad essere *established*, è attuato entro limiti prestazionali definiti per il raggiungimento degli obiettivi previsti;
- **5 - Optimizing**: il processo, oltre ad essere *predictable*, è oggetto di miglioramento continuo per il soddisfacimento di obiettivi di business, attuali, previsti e futuri.

Ogni processo è classificabile in base al livello di soddisfacimento dei seguenti nove attributi:

- **1.1 Process performance**: capacità del processo di raggiungere gli obiettivi prefissati;
- **2.1 Performance management**: misura del grado di gestione dell'attuazione del processo in esame;
- **2.2 Work product management**: misura del grado di gestione dei prodotti del processo in esame;
- **3.1 Process definition**: misura dell'adeguatezza del processo rispetto agli standard di riferimento;
- **3.2 Process deployment**: capacità del processo di sfruttare le risorse allocate;
- **4.1 Process measurement**: capacità del processo di produrre misurazioni utili a fini di controllo;
- **4.2 Process control**: capacità del processo di essere corretto o migliorato grazie all'analisi delle misurazioni rilevate;
- **5.1 Process innovation**: misura del grado in cui i cambiamenti strutturali e di esecuzione del processo sono controllati a fini di innovare e migliorare gli standard presenti;
- **5.2 Process optimization**: capacità del processo di implementare le modifiche effettuate in modo da ottenere un miglioramento continuo nella realizzazione degli obiettivi prefissati.

La scala di valutazione degli attributi di processo è la seguente:

- **N**: non posseduto (0 – 15%);
- **P**: parzialmente posseduto (>15% – 50%);
- **L**: largamente posseduto (>50% – 85%);
- **F**: pienamente posseduto (>85% – 100%).

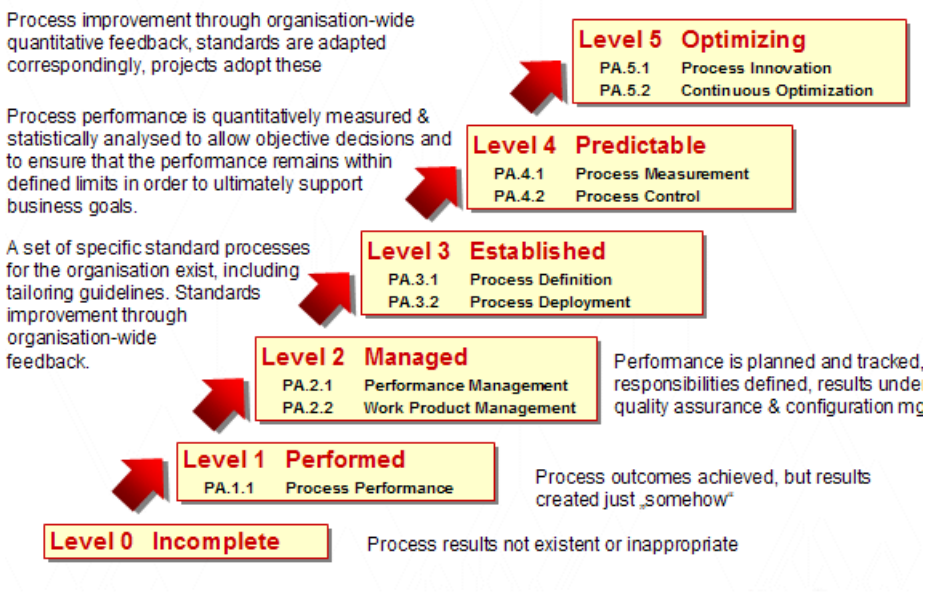

Figura 14: Schema della *capability dimension* di SPICE (tratta da SPiCE 1-2-1)

C.2 PDCA

Il PDCA, conosciuto anche come *Ciclo di Deming*, è un metodo di gestione dei processi durante il loro ciclo di vita con il fine di controllare il miglioramento continuo della loro qualità e, quindi, anche quella dei loro prodotti. L'approccio che propone è suddiviso in quattro fasi da ripetere iterativamente fino al raggiungimento dell'obiettivo finale:

- **Plan**: fase di pianificazione in cui vengono stabiliti gli obiettivi ed i processi necessari per il raggiungimento dei risultati attesi;
- **Do**: fase di esecuzione di quanto pianificato al punto precedente con rilevamento di dati significativi da poter analizzare nelle fasi successive;
- **Check**: fase di controllo dei dati rilevati nella fase *Do* per confrontare i risultati ottenuti con quelli attesi dalla fase *Plan*. Le differenze riscontrate e le deviazioni nell'attuazione del piano osservate serviranno alla fase successiva;
- **Act**: fase di attuazione del miglioramento della qualità, tramite l'adozione di strategie emerse dallo studio dei risultati della fase di *Check*, eventualmente anche al di fuori del processo in questione.

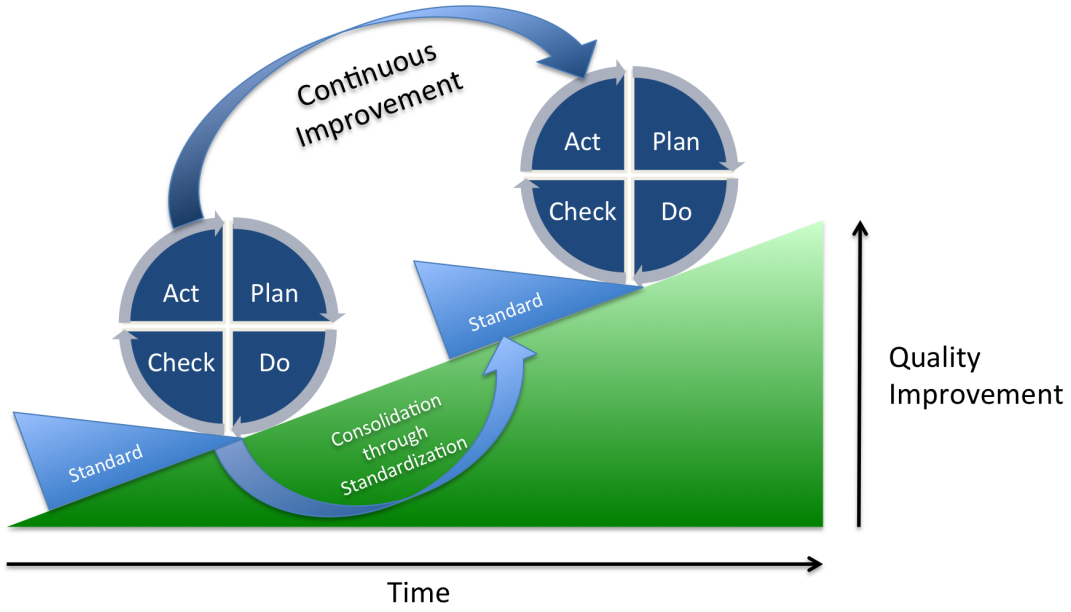

Figura 15: Schema del miglioramento continuo tramite PDCA (creato da Johannes Vietze)

C.3 ISO/IEC 9126

Lo standard ISO/IEC 9126 fornisce un modello per la definizione della qualità di un software.

In particolare, esso distingue tre punti di vista sul software rispetto ai quali valutarne la qualità:

- **Qualità interna**: relativa al software sorgente non in esecuzione ed alla documentazione correlata. Viene rilevata tramite analisi statica ed è influenzata dalla qualità dei processi del ciclo di vita del prodotto;
- **Qualità esterna**: relativa al software in esecuzione. Viene rilevata tramite test, in funzione degli obiettivi stabiliti, ed è influenzata dalla qualità interna;
- **Qualità in uso**: relativa alla percezione dell'utente del prodotto finito in contesti reali d'uso. È influenzata dalla qualità esterna.

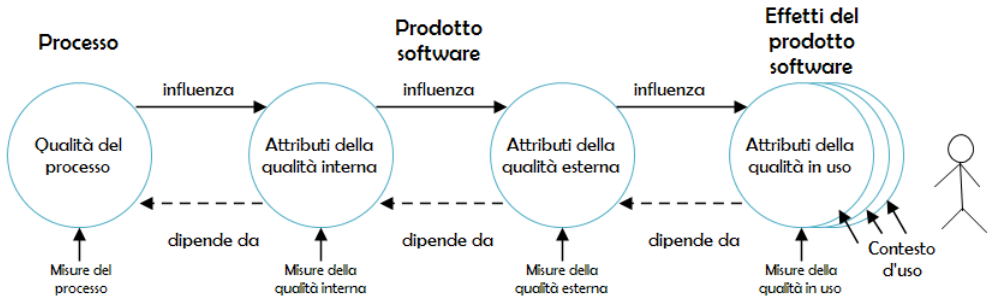

Figura 16: Schema del ciclo di qualità del software (creato da Giuseppe Manuele)

Per ciascuno dei punti di vista vengono inoltre delineate delle caratteristiche e sotto-caratteristiche qualitative, eventualmente misurabili quantitativamente, mediante apposite metriche.

Per la qualità interna ed esterna esse sono:

- **Funzionabilità**: capacità di fornire funzioni che soddisfino le esigenze stabilite, nei relativi contesti di presentazione.
  - **appropriatezza**;
  - **accuratezza**;
  - **interoperabilità**;
  - **conformità**;
  - **sicurezza**.
- **Affidabilità**: capacità di mantenere un determinato livello di prestazioni in date condizioni per un dato periodo.
  - **maturità**;
  - **tolleranza agli errori**;
  - **recuperabilità**;
  - **aderenza**.
- **Efficienza**: capacità di fornire appropriate prestazioni relativamente alle risorse utilizzate.

  - **comportamento rispetto al tempo**;
  - **utilizzo di risorse**;
  - **conformità**.
- **Usabilità**: capacità del prodotto software di essere capito, appreso e usato dall'utente, al verificarsi di determinate condizioni.
  - **comprensibilità**;
  - **apprendibilità**;
  - **operabilità**;
  - **attrattiva**;
  - **conformità**.
- **Manutenibilità**: capacità del prodotto software di essere modificato, corretto o migliorato facilmente nel tempo.
  - **analizzabilità**;
  - **modificabilità**;
  - **stabilità**;
  - **testabilità**;
  - **collaudabilità**.
- **Portabilità**: capacità del prodotto software di essere trasportato da un ambiente di lavoro all'altro.
  - **adattabilità**;
  - **installabilità**;
  - **conformità**;
  - **sostituibilità**.

Le caratteristiche per la qualità in uso sono:

- **Efficacia**: capacità di permettere all'utente di raggiungere gli obiettivi specificati con accuratezza e completezza;
- **Produttività**: capacità di permettere all'utente di spendere una quantità di risorse appropriata all'efficacia ottenuta dall'uso del prodotto;
- **Soddisfacibilità**: capacità di soddisfare l'utente;
- **Sicurezza**: capacità di raggiungere accettabili livelli di rischio nei confronti di persone e dell'ambiente di lavoro.

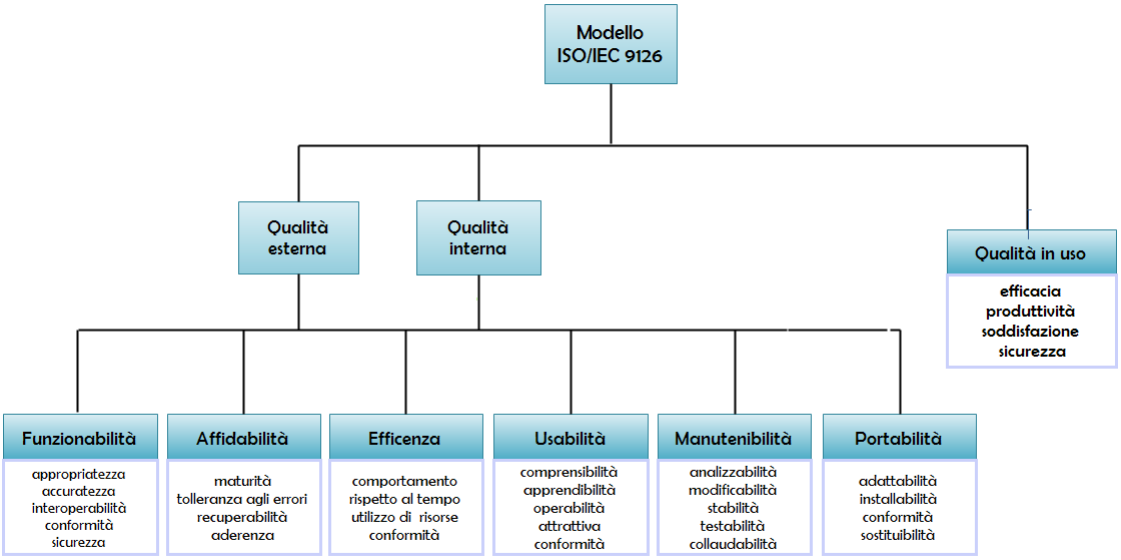

Figura 17: Schema delle caratteristiche definite in ISO/IEC 9126 (creato da Giuseppe Manuele)

D Guida all'uso di Git

Questa appendice contiene una serie di comandi Git che permetterà ai membri del team di progetto di interagire con GitHub:

- ***git clone***: comando che permette di clonare il repository remoto su GitHub in locale. La sintassi è *git clone indirizzoRepository* dove indirizzoRepository è l'indirizzo raggiungibile tramite il menù a tendina "Clone or Download" su GitHub;
- ***git add nomeFile***: comando che aggiunge il file alla lista dei file tracciati da Git, contenuti nell'index. Il comando deve essere utilizzato per aggiungere nuovi file o per proporre la modifica di file già esistenti in una versione successiva. Se si modifica un file e non si esegue questo comando, le modifiche non verranno apportate nella versione successiva. Il comando può essere usato per aggiungere una cartella oppure si può eseguire *git add .* per aggiungere tutti i file, comprese le sotto cartelle;
- ***git pull***: comando che permette di aggiornare il contenuto del repository locale con il contenuto di quello remoto. In questo modo eventuali modifiche attuate da altre persone verranno riportate sul repository locale e si può lavorare sui file più recenti;
- ***git status***: comando che visualizza lo stato dei file, dividendoli in tre gruppi:
  1. **File non tracciati**: mostra i file che non sono mai stati inseriti nel controllo versione di Git;
  2. **File modificati ma non aggiornati**: mostra i file modificati, rispetto al commit precedente, che non sono stati aggiunti tramite *git add* e che quindi non verranno inseriti nel prossimo commit;

3. **File modificati pronti per il commit**: mostra i file modificati, rispetto al commit precedente, che sono stati aggiunti tramite *git add* e che quindi sono pronti per il commit.

- ***git diff***: mentre *git status* mostra i file che sono stati modificati, per entrare nei dettagli delle righe modificate è necessario utilizzare *git diff*. Il comando può essere utilizzato in due modalità:
  1. ***git diff***: mostra le righe che sono state cambiate nei file non ancora preparati per la commit, confrontati con la copia presente nell'ultima commit;
  2. ***git diff –staged***: mostra le righe che sono state cambiate nei file preparati per la commit, confrontati con la copia presente nell'ultima commit.
- ***git commit -m "Messaggio commit"***: comando che permette di rendere definitivi i file aggiunti o modificati tramite il comando *git add*. Se non si lancia il comando *git commit* e si esegue *git push*, non verranno caricate eventuali modifiche o file aggiunti al repository locale sul repository remoto. *-m* è l'opzione per inserire un messaggio significativo (obbligatorio) per far comprendere ai membri del team la tipologia di modifiche che sono state effettuate;
- ***git log***: comando che permette di visualizzare lo storico dei commit;
- ***git checkout***: comando che permette di fare cose differenti a seconda di come viene utilizzato:
  1. ***git checkout -b nomeBranch***: crea un nuovo branch chiamato "nomeBranch". Una volta creato il branch, tutti i comandi che verranno lanciati saranno eseguiti su quel branch;
  2. ***git checkout nomeBranch***: permette di spostarsi sul branch "nomeBranch".
- ***git merge nomeBranch***: comando che permette di riportare le modifiche apportate a "nomeBranch" nel branch attualmente selezionato. Il comando deve essere lanciato dal branch master;
- ***git push***: comando che aggiorna il repository remoto sovrascrivendolo con il repository locale;
- **configurazioni utente**: se si tratta della prima volta che si utilizza Git allora al primo tentativo di commit verrà lanciato un errore relativo al fatto che non si sono settati il nome utente e la e-mail. Per far questo si utilizzano i comandi:
  1. ***git config –global user.name "nomeUtente"***: comando che permette di configurare Git con il nostro nome. In questo modo ogni qualvolta che si effettuerà una commit, il nostro nome verrà associato alla commit;
  2. ***git config –global user.email "e-mail"***: comando identico al precedente ma che permette di configurare l'indirizzo e-mail.