Norme di Progetto

Gruppo SWEight - Progetto Colletta

SWEightGroup@gmail.com

Informazioni sul documento

Versione	4.0.0
Approvatore	Enrico Muraro
Redattori	Francesco Magarotto Damien Ciagola
Verificatori	Sebastiano Caccaro
Uso	Interno
Distribuzione	MIVOQ Prof. Vardanega Tullio Prof. Cardin Riccardo Gruppo SWEight

Descrizione

Questo documento si occupa di definire la way-of-working alla quale il gruppo *SWEight* dovrà attenersi per lo svolgimento del progetto.

Registro delle modifiche

Versione	Data	Descrizione	Nominativo	Ruolo
4.0.0	2019-05-15	Approvazione per il rilascio	Enrico Muraro	*Responsabile*
3.2.1	2019-05-04	Verifica documento	Sebastiano Caccaro	*Verificatore*
3.2.1	2019-05-03	Correzioni in §4.5, Aggiunte §4.5.1 e §4.5.2	Damien Ciagola	*Redattore*
3.1.1	2019-05-02	Correzioni in §3	Damien Ciagola	*Redattore*
3.1.0	2019-05-01	Verifica modifiche apportate	Sebastiano Caccaro	*Verificatore*
3.1.0	2019-04-30	Ampliamento e correzioni in §3, Aggiunte §3.1.4 e §3.1.5, spostata §3.2.6 in §4.3.1.5	Damien Ciagola	*Redattore*
3.0.0	2019-03-21	Approvazione per il rilascio	Damien Ciagola	*Responsabile*
2.2.2	2019-03-20	Verifica documento	Francesco Magarotto	*Verificatore*
2.2.2	2019-03-19	Normato codice identificativo test	Francesco Corti	*Redattore*
2.2.1	2019-03-17	Normati Spring e SpringData MongoDB	Francesco Corti	*Redattore*
2.2.0	2019-03-17	Diagrammi delle classi §3.2.2.2.4, spostata appendice qualità dal *PianoDiQualifica_v3.0.0*	Sebastiano Caccaro	*Redattore*
2.1.3	2019-03-15	Normati MongoDb e MongoDb Compass Aggiornato §3.2 e ampliato §3.2.6.4	Francesco Corti	*Redattore*
2.1.2	2019-03-14	Normati diagrammi di sequenza e di package Aggiunte sezioni §3.2.4.4 e §3.2.4.5	Francesco Corti	*Redattore*
2.1.1	2019-03-11	Stile tipografico corretto	Alberto Bacco	*Redattore*
2.1.0	2019-03-10	Normato Rosso SWEight §4.1.9	Sebastiano Caccaro	*Redattore*
2.0.0	2019-03-07	Approvazione per il rilascio	Francesco Magarotto	*Responsabile*
1.4.1	2019-03-07	Verifica documento	Francesco Corti	*Verificatore*
1.4.0	2019-03-07	Aggiunte nuove metriche MS010 e MS011	Enrico Muraro	*Redattore*
1.3.0	2019-03-07	Aggiunte §3.2.1.3.1, §3.1.2.1, §4.1.9.2, §4.3.2.2	Sebastiano Caccaro	*Redattore*

1.2.5	2019-03-06	Correzione errori ed aggiunta §3.2.5.9 Aggiunto §B	Francesco Corti	*Redattore*
1.2.4	2019-03-05	Verifica modificate	Caccaro Sebastiano	*Verificatore*
1.2.4	2019-02-28	Ampliamento §3.2.4 e §3.2.5, Aggiunta classificazione univoca delle metriche	Enrico Muraro	*Redattore*
1.2.3	2019-02-26	Ampliamento §4.1.2.3, §4.4.6.1 e §4.4.6.2	Francesco Corti	*Redattore*
1.2.2	2019-02-25	Ampliamento §5.4 e §3.2.5.2	Alberto Bacco	*Redattore*
1.2.1	2019-02-23	Aggiunta §3.2.5.8 Checkstyle	Sebastiano Caccaro	*Redattore*
1.2.0	2019-02-20	Aggiunta scelte tecnologiche §3.2.4.2, da §3.4.5.4 a §3.4.5.7, §4.3.1.4, §4.3.2.2, §4.4.6 e figlie	Sebastiano Caccaro	*Redattore*
1.1.5	2019-02-20	Modifica §2	Alberto Bacco	*Redattore*
1.1.4	2019-02-18	Correzione errori grammatica, spostate di asana da §4.3 a §5.2,	Alberto Bacco	*Redattore*
1.1.3	2019-02-14	Riorganizzazione e correzione errori §5	Enrico Muraro	*Redattore*
1.1.2	2019-02-03	Modifica §4.1.10, §4.3.1.4, §4.3.1.5, §4.3.1.6	Alberto Bacco	*Redattore*
1.1.1	2019-01-31	Modifica struttura e contenuti §3	Damien Ciagola	*Redattore*
1.1.0	2019-01-27	Qualità §4.2	Sebastiano Caccaro	*Redattore*
1.0.1	2019-01-25	Parziale ristrutturazione della struttura del documento	Sebastiano Caccaro	*Redattore*
1.0.0	2019-01-11	Approvazione per il rilascio	Sebastiano Caccaro	*Responsabile*
0.9.0	2019-01-9	Verifica finale	Francesco Corti	*Verificatore*
0.9.0	2019-01-8	Aggiunta lista di controllo	Gionata Legrottaglie	*Redattore*
0.8.0	2018-12-23	Correzioni errori ortografici	Gionata Legrottaglie	*Redattore*
0.7.0	2018-12-20	Verifica documento	Francesco Corti	*Verificatore*
0.6.0	2018-12-18	Aggiunta §5.2.2.2, §5.2.2.3, §5.2.2.4	Francesco Magarotto	*Redattore*
0.5.2	2018-12-16	Modifica §4.1.5.3	Alberto Bacco	*Redattore*
0.5.2	2018-12-16	Modifica §4.1.5.3	Alberto Bacco	*Redattore*

0.5.2	2018-12-16	Aggiunte	Alberto Bacco	*Redattore*
0.5.1	2018-12-15	Aggiunte §5.3, §5.4, §5.5, §5.6, §5.7, §5.8	Alberto Bacco	*Redattore*
0.5.0	2018-12-15	Aggiunta §5 e §5.1, §5.2	Gionata Legrottaglie	*Redattore*
0.4.1	2018-12-11	Aggiunta §4.1.7.3.1	Francesco Magarotto	*Redattore*
0.4.0	2018-12-10	Aggiunte §4.1.5, §4.1.6, §4.1.7, §4.1.8	Gionata Legrottaglie	*Redattore*
0.4.0	2018-12-09	Aggiunta §4 e §4.1.1, §4.1.2, §4.1.3, §4.1.4	Gionata Legrottaglie	*Redattore*
0.3.1	2018-12-07	Aggiunta §3.2	Gionata Legrottaglie	*Redattore*
0.3.0	2018-12-06	Aggiunta §3 e §3.1	Gionata Legrottaglie	*Redattore*
0.2.0	2018-12-05	Aggiunti i riferimenti	Gionata Legrottaglie	*Redattore*
0.1.0	2018-11-30	Aggiunta introduzione	Gionata Legrottaglie	*Redattore*
0.0.1	2018-11-28	Creazione scheletro del documento	Gionata Legrottaglie	*Redattore*

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il documento ha lo scopo di definire le norme che i membri del gruppo SWEight dovranno rispettare durante lo svolgimento del progetto "Colletta". I membri sono tenuti a leggere il documento per garantire uniformità al fine di ottenere una migliore efficacia ed efficienza nello svolgimento delle attività. I partecipanti potranno contattare l'*Amministratore di Progetto* per eventuali suggerimenti e opinioni riguardanti le norme di progetto. L'*Amministratore di Progetto* si dovrà consultare con i membri del team ed avrà la responsabilità di accettare o rifiutare eventuali suggerimenti proposti, argomentandone la decisione. Il documento pone l'accento sui seguenti punti:

- Interazioni tra il team di sviluppo ed esterni;
- Organizzazione dell'ambiente di lavoro;
- Modalità di lavoro durante lo sviluppo del progetto;
- Stesura documenti e convenzioni di scrittura utilizzate;
- Tecniche di analisi ed eventuali correzione degli errori.

1.2 Scopo del progetto

Lo scopo del progetto è la realizzazione di una piattaforma interattiva per la raccolta dati relativi ad esercizi di analisi grammaticale, che verranno impiegati come dati sorgente in algoritmi di apprendimento automatico.

1.3 Glossario

Per prevenire ed evitare qualsiasi dubbio, e per permettere una maggiore chiarezza e comprensione del testo inerentemente a termini ambigui, abbreviazioni e acronimi utilizzati nei vari documenti, è stato redatto un Glossario nel quale si possono trovare le definizioni di tali termini. Quest'ultimi sono contrassegnati all'interno dei documenti con il pedice $_{G}$.

2 Riferimenti

2.1 Normativi

- **Capitolato d'appalto C2:** Colletta: piattaforma raccolta dati di analisi di testo
  https://www.math.unipd.it/~tullio/IS-1/2018/Progetto/C2.pdf
- **Specifiche UML$_{G}$ 2.0**:
  http://www.omg.org/spec/UML/2.0/.

2.2 Informativi

- **Standard ISO$_{G}$-8601**:
  https://en.wikipedia.org/wiki/ISO_8601, sezione: "Calendar dates";
- **Standard ISO/IEC$_{G}$ 12207**
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf sezioni 5,6,7;

- **Dispense**
  https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L03.pdf
  http://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L05.pdf
  https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L06.pdf

- **Software Engineering (9th edition)**
  Ian Sommerville, Pearson Education, Part 1 e 4.

- **Linee guida da seguire per la codifica:**
  http://google.github.io/styleguide/

3 Processi primari

3.1 Fornitura

In questa sezione vengono descritte le norme che il team deve rispettare al fine di diventare fornitore nei confronti del proponente nell'ambito della Progettazione, Sviluppo e Validazione del prodotto.

3.1.1 Studio di fattibilità

In seguito alla pubblicazione dei capitolati è compito del *Responsabile di Progetto* convocare l'intero team al fine di discutere degli aspetti positivi e negativi dei vari capitolati.

Successivamente, gli *Analisti* devono effettuare lo Studio di Fattibilità per ogni capitolato, attenendosi anche a quello emerso dalla riunione precedente.
Per ogni capitolato deve essere scritto un documento contenente:

- **Informazioni sul capitolato**;
- **Descrizione**: una sintesi del prodotto da realizzare secondo le richieste del capitolato;
- **Dominio applicativo**: il contesto in cui l'applicazione opera;
- **Dominio tecnologico**: rappresenta il dominio tecnologico richiesto dal capitolato;
- **Considerazioni del gruppo**: racchiude tutte le valutazioni dei membri del gruppo che hanno portato ad una accettazione o ad un rifiuto;
- **Valutazione finale**: descrive le motivazioni decisive che hanno portato il gruppo a scegliere o a rigettare il capitolato.

Infine, questi documenti devono essere racchiusi in un unico documento: *StudioDiFattibilità_v2.0.0*, che verrà sottoposto a verifica dai componenti preposti a tale compito.

3.1.2 Piano di progetto

Redigere il *PianoDiProgetto_v3.0.0* è compito del *Responsabile di Progetto*, nel documento è delineata la pianificazione del gruppo *SWEight* per lo sviluppo di "Colletta: piattaforma raccolta dati di analisi di testo". Al fine di rendere chiara ed efficace l'esposizione, il piano di progetto contiene:

- **Analisi dei rischi**: vengono identificati nel dettaglio i rischi che potranno presentarsi, ognuno di essi viene categorizzato per probabilità e gravità, vengono fornite delle strategie atte a prevenire l'occorrenza di un dato rischio;
- **Descrizione del modello di sviluppo**: viene descritto il modello di sviluppo che è stato scelto per la pianificazione;

- **Pianificazione del tempo e delle risorse**: vengono pianificate le attività da eseguire nei diversi periodi del progetto e vengono stabilite le loro scadenze;
- **Preventivo sul costo dell'utilizzo delle risorse**: viene data una stima di lavoro necessaria per periodo, proponendo un preventivo per il costo totale del progetto.

3.1.2.1 Classificazione dei rischi

La classificazione dei rischi nel *PianoDiProgetto_v3.0.0* segue la seguente convenzione:

[ClasseDiRischio]-[Identificativo]

Dove:

- ClasseDiRischio indica la classe di rischio fra le seguenti alla quale il rischio appartiene:
  - **B**: Bassa;
  - **M**: Media;
  - **A**: Alta.
- Identificativo è un codice di tre cifre numeriche che identifica univocamente un rischio all'interno della sua classe.

Ad esempio, il tredicesimo rischio di livello medio è individuato dal seguente codice:

M-013

3.1.3 Piano di qualifica

Il *PianoDiQualifica_v3.0.0*, redatto dall'*Amministratore* è un documento contenente le strategie da adottare per garantire la qualità del materiale prodotto, esso è così suddiviso:

- **Qualità di processo**: viene illustrato lo standard adottato al fine di valutare la maturità e capacità dei processi;
- **Qualità di prodotto**: viene illustrato lo standard adottato al fine di valutare, monitorare e misurare la qualità del software;
- **Strategia**: viene illustrata la strategia adottata dal gruppo;
- **Metriche**: vengono illustrate le metriche atte a monitorare lo stato e la qualità dei processi dei documenti e del software prodotto.

3.1.4 Qualifica del prodotto

Una volta avviato il processo di sviluppo, i membri del gruppo devono affiancare ad ogni attività produttiva un'attività che verifichi ciò che è stato prodotto. La corrispondenza tra attività di produzione e attività di verifica è la seguente:

- **Analisi dei requisiti**: specifica dei test di sistema;
- **Progettazione logica**: specifica dei test di integrazione;
- **Progettazione di dettaglio**: specifica dei test di unità;
- **Codifica**: analisi statica del codice sorgente ed esecuzione dei test di unità e di integrazione;
- **Redazione della documentazione**: verifica della documentazione;

3.1.5 Consegna

Il gruppo deve consegnare il prodotto dopo averlo collaudato pubblicamente;
il collaudo deve includere almeno i seguenti punti:

- Illustrazione delle principali funzionalità del prodotto, che riassuma almeno i requisiti giudicati obbligatori nel documento *AnalisiDeiRequisiti_v3.0.0*;
- Esempio d'uso del prodotto, dal vivo.

3.2 Sviluppo

3.2.1 Analisi dei requisiti

Gli *Analisti* devono redigere l'*AnalisiDeiRequisiti_v3.0.0*al fine di:

- Fornire ai *Progettisti* riferimenti affidabili e precisi;
- Facilitare le revisioni del codice;
- Descrivere lo scopo del progetto;
- Fissare le funzionalità ed i requisiti$_G$ concordati con il proponente;
- Fornire ai *Verificatori* riferimenti per l'attività di test circa i casi d'uso principali e alternativi.

Deve essere redatto, sempre dagli stessi, un documento che elenchi i requisiti, classificandoli come di seguito riportato.

3.2.1.1 Classificazione dei requisiti

I vari requisiti potranno provenire dalle seguenti fonti:

- **Capitolato$_G$**: il requisito è stato scritto esplicitamente nel documento fornito dal proponente;
- **Verbali interni e studio di fattibilità**: il requisito è emerso durante la discussione del capitolato tra gli *Analisti*;
- **Casi d'uso$_G$**: il requisito è il risultato dell'analisi di uno o più casi d'uso.

I vari requisiti devono essere classificati secondo la seguente convenzione$_G$:

R-[Importanza][Tipo][Identificativo]

Dove:

- Importanza indica se il Requisito è:
  - **1**: Requisito obbligatorio;
  - **2**: Requisito desiderabile;
  - **3**: Requisito opzionale;
- Tipo indica se il requisito è:
  - **F**: Requisito funzionale;
  - **Q**: Requisito di qualità;
  - **P**: Requisito prestazionale;
  - **V**: Requisito di vincolo;
- Identificativo è un codice univoco che contraddistingue il requisito.

Ogni identificativo di requisito non deve essere più modificato nel tempo per evitare incongruenze tra i documenti. Deve essere realizzata una tabella che classifichi tutti i requisiti, incorporando una descrizione e la provenienza del requisito, indicata con il termine fonte.

Identificativo	Importanza	Tipo	Fonte	Descrizione
3V001	Requisito obbligatorio	Di vincolo	Capitolato	Gli sviluppatori devono poter accedere ai dati raccolti gratuitamente

Tabella 2: Esempio tabella classificazione requisiti

3.2.1.2 Casi d'uso Dopo la stesura dei requisiti, gli *Analisti* devono analizzare tutti i casi d'uso (d'ora in poi UC); ognuno di essi deve essere composto da un codice identificativo univoco, eventuali diagrammi UML e infine dalla sua descrizione testuale. In particolare, per quanto riguarda l'identificazione si procederà come descritto di seguito:

UC[`Codice identificativo`]

Dove `Codice identificativo` consiste in codice numerico gerarchico$_G$ e univoco. Nel caso si trattasse di un sottocaso d'uso$_G$, pertanto generato da un caso d'uso generico, si utilizzerà il seguente costrutto:

$$\underbrace{\text{UC}[\texttt{Codice identificativo UC generico}].[\texttt{Codice identificativo UC specifico}]}_{\text{Codice identificativo}}$$

L'utilizzo del "." (punto) viene utilizzato per determinare la profondità della gerarchia dei casi d'uso creatasi. Per quanto riguarda la descrizione testuale dei casi d'uso i membri del gruppo interessati seguiranno il seguente modello:

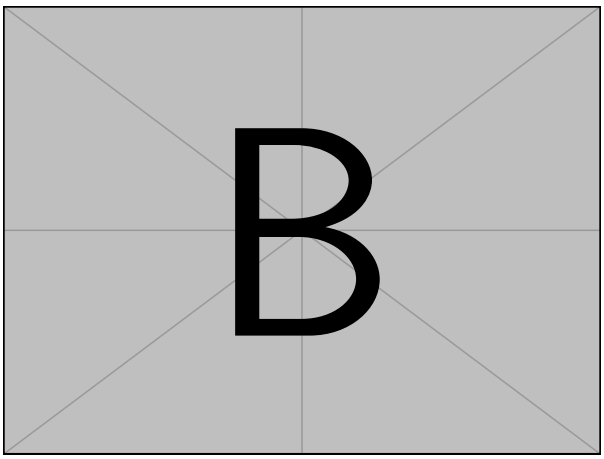

Figura 1: Caso d'uso Codice univoco o breve descrizione nel caso della panoramica attore

UC `Codice univoco` - Titolo del caso d'uso, la cui lunghezza deve essere ristretta

- **Attore**: gli attori coinvolti nell'interazione con il sistema;
- **Descrizione**: breve descrizione testuale del caso d'uso;
- **Precondizione**: definisce lo stato del sistema, pertanto anche le condizioni che devono essere vere, prima dell'esecuzione del caso d'uso;
- **Postcondizione**: definisce lo stato del sistema dopo il verificarsi del caso d'uso;
- **Flusso degli eventi**: il susseguirsi di eventi che conducono alla postcondizione. Deve essere realizzato con un elenco numerato, facendo riferimento eventualmente ad ulteriori casi d'uso, oppure con una breve descrizione testuale;

- **Estensioni**: eventuali estensioni coinvolte;
- **Inclusioni**: eventuali inclusioni coinvolte;
- **Generalizzazioni**: eventuali generalizzazioni coinvolte;
- **Flusso degli eventi alternativo**: (Non obbligatorio) il susseguirsi alternativo di eventi che alternativi al flusso principale.

3.2.1.3 Diagrammi UML

Con l'obiettivo di rendere chiare le soluzioni progettuali utilizzate, è necessario l'utilizzo di diagrammi UML$_G$. Quest'ultimi devono essere realizzati utilizzando lo standard$_G$ 2.0.

È richiesta la realizzazione di:

- **Diagrammi di attività**: descrivono un processo o un algoritmo;
- **Diagrammi dei package**: per raggruppare elementi e fornire un namespace$_G$ per gli elementi raggruppati;
- **Diagrammi di sequenza**: rappresentano una sequenza di processi o funzioni;
- **Diagrammi di classi**: rappresentano le classi utilizzate e le loro relazioni.

In caso vengano utilizzati dei Design Pattern$_G$ sarà necessario accompagnarli con una descrizione ed un diagramma UML.

3.2.2 Progettazione

Dopo aver terminato il periodo di Analisi si passerà a quella di Progettazione che vede come protagonisti i *Progettisti*, quest'ultimi hanno stabilito l'architettura$_G$ logica da definire e i vari diagrammi che la rappresentano. La Progettazione permette di:

- Ottimizzare l'uso delle risorse;
- Garantire la qualità del prodotto sviluppato;
- Suddividere il problema principale in tanti sotto problemi di complessità minore.

3.2.2.1 Architettura logica

Bisogna definire un'architettura logica del prodotto che dovrà:

- Soddisfare i requisiti definiti nel documento di *AnalisiDeiRequisiti_v3.0.0*;
- Essere sicura in caso di malfunzionamenti o intrusioni;
- Essere modulare$_G$ e formato da componenti riutilizzabili;
- Essere affidabile;
- Essere comprensibile per future manutenzioni.

3.2.2.2 Design Pattern

I *Progettisti* devono utilizzare il design pattern che ritengono più adatto al contesto per rendere l'applicazione più sicura ed efficiente possibile. Ogni utilizzo di design pattern deve essere brevemente descritto ed accompagnato da un diagramma UML posto nella directory padre dei file sorgenti.

3.2.2.2.1 Diagrammi di attività

I diagrammi di attività sono rappresentati usando i formalismi definiti in UML 2. I principali sono riportati in seguito. Per indicare il consumo e la generazione di token, viene usata la seguente notazione: (T:token generati/token consumati).

- **Nodo iniziale**: punto dal quale inizia l'esecuzione del processo. È rappresentato da un pallino nero (T:1/0);
- **Activity**: azione elementare svolta dal programma. Si rappresenta con un rettangolo che ne contiene una descrizione il più sintetica possibile (T:1/1);
- **Branch**: punto di decisione. Il flusso può intraprendere solo uno di n rami. Si rappresenta mediante un rombo con una freccia entrante e n uscenti. Ogni ramo è accompagnato da una *guardia*, un'etichetta fra parentesi quadre che ne descrive la condizione;

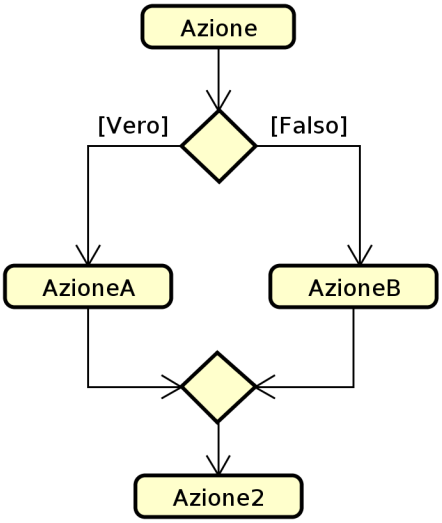

Figura 2: Merge e branch

- **Merge**: punto in cui rami derivanti da un branch si uniscono. Si rappresenta come un rombo (T:1/1);
- **Fork**: punto da cui partono 2 o più processi paralleli. Si rappresenta come una spessa linea orizzontale o verticale, con una freccia entrante e n uscenti (T:1/N);
- **Join**: punto in sincronizzazione di processi paralleli. Si rappresenta come un Fork, ma con n frecce entranti e una uscente. Può essere accompagnato da un'espressione booleana che ne specifica il funzionamento (T:N/1);
- **Pin**: passaggio di parametri fra activity. Si rappresenta tramite un quadratino dal quale escono o entrano delle frecce. É accompagnato dal tipo del parametro trasferito, posto a fianco;
- **Subactivity**: indica che l'azione che l'activity svolge è descritta più in dettaglio in un diagramma separato. Una sotto-attività ha sempre un input e un output. Si indica con box simile a quello dell'activity che presenta un piccolo tridente in basso a sinistra (T:1/1);
- **Partizioni**: forniscono una responsabilità all'esecuzione delle azioni. Si indicano tramite delle linee verticali che dividono il diagramma in più partizioni;
- **Segnali**: rappresentano l'invio/ricezione di eventi a/da processi esterni. Si rappresentano attraverso un rettangolo con una concavità triangolare per la ricezione, e tramite un rettangolo con una convessità triangolare per l'invio. Entrambi contengono una breve descrizione del segnale;
- **Timeout**: modellano dei timeout. Si presentano come una clessidra, possono avere frecce entranti e uscenti, e sono accompagnati dalla descrizione in linguaggio naturale dell'intervallo di tempo che deve trascorrere;

- **Eventi Ripetuti**: modellano degli eventi che si ripetono nel tempo, generando un token ad ogni ripetizione. Possono avere solo frecce uscenti, e si rappresentano con una descrizione dell'intervallo della ripetizione;
- **Nodi di fine flusso**: rappresenta la morte di un ramo di esecuzione. Si rappresenta tramite un cerchio con una X (T:1/0);
- **Nodo finale** : rappresenta la fine dell'esecuzione di un processo. Si rappresenta tramite un cerchio pieno racchiuso in una circonferenza di raggio maggiore (T:0/1).

Figura 3: Nodi, da sinistra a destra: iniziale, finale, di di fine flusso.

3.2.2.2.2 Diagrammi di sequenza

Ogni diagramma di sequenza avrà un senso di lettura verticale dall'alto al basso, il verso indica lo scorrere del tempo. Gli oggetti coinvolti verranno rappresentati tramite un rettangolo, al cui interno si troverà un nome per identificarli nel formato di istanza: **NomeClasse**.
Sotto ad ogni istanza si troverà una linea della vita tratteggiata, che sarà sormontata in alcuni tratti da una barra di attivazione che indica i momenti in cui l'oggetto è attivo. Da una barra di attivazione partiranno delle frecce, che rappresentano un messaggio o segnale, verso la linea della vita di oggetti già instanziati, in alternativa, verso una nuova istanza di classe per crearla. In particolare, verranno utilizzati i seguenti tipi di frecce:

- Freccia piena, per indicare un messaggio sincrono: corrisponde alla chiamata di un metodo. Sopra tale freccia si dovrà specificare il metodo invocato secondo il formato:

  `nomeMetodo(lista parametri formali)`

- Freccia, per indicare un messaggio asincrono, chi invoca il metodo non attendendo il return;
- Freccia tratteggiata, per indicare il ritorno di un metodo chiamato. Sopra tale freccia si dovrà indicare il tipo di ritorno secondo il formato: `TipoRitorno`;
- Freccia trattegiata sormontata da `«create»`, indica la creazione di un nuovo oggetto e termina sempre in un rettangolo che ne contiene il nome nel formato `istanza: NomeClasse`;
- Freccia piena sormontata da `«destroy»`, indica la distruzione di un oggetto e termina sempre in una X, nella quale muore anche la linea della vita dell'oggetto.

Sarà inoltre possibile determinare sui diagrammi di sequenza dei frame di interazione associati ad una guardia o condizione:

- **alt**: indica dei frammenti in alternativa fra loro, verrà eseguito quello per cui la guardia è verificata;
- **opt**: indica un frammento che viene eseguito solo se la guardia è specificata;
- **par**: indica frammenti eseguiti in parallelo;
- **loop**: indica un frammento che viene eseguito più volte, la condizione di arresto del ciclo è la guardia;
- **region**: indica un frammento critico che deve essere eseguito in mutua esclusione.

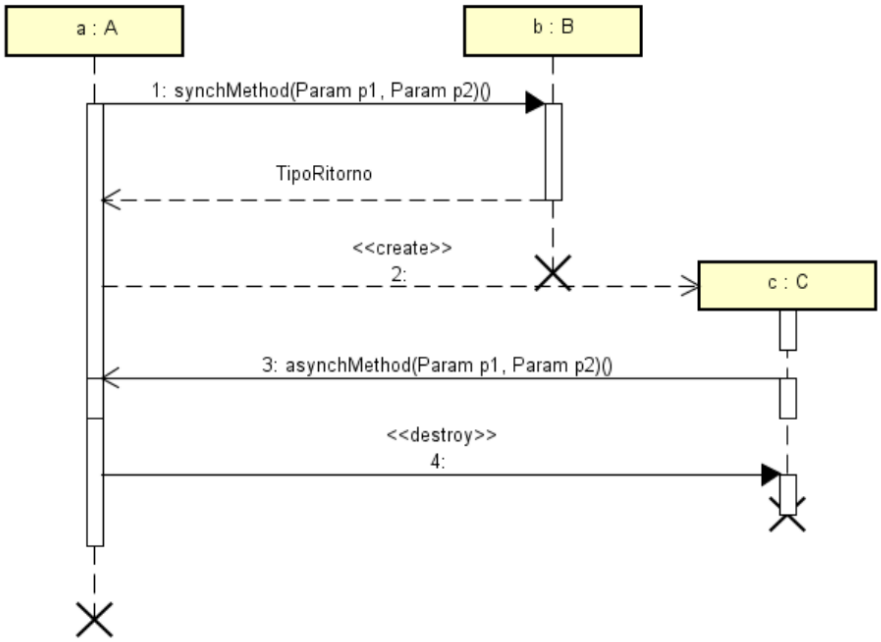

Figura 4: Diagramma di sequenza con tutti i tipi di segnale

3.2.2.2.3 Diagrammi dei package

Ogni package verrà rappresentato tramite un rettangolo con un'etichetta per il nome, che dovrà contenere i diagrammi delle classi appartenenti al package ed eventuali sotto-package. Le dipendenze fra i vari package dovranno essere segnalate con una freccia tratteggiata. Tale freccia, disegnata dal package A al package B, indica una dipendenza di A ne confronti di B. Si devono evitare dipendenze cicliche.

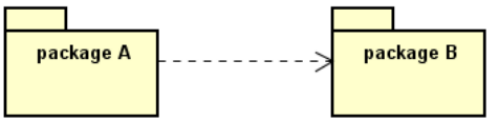

Figura 5: Esempio di diagramma dei package

3.2.2.2.4 Diagrammi delle classi

Tutte le classi devono seguire l'ultima specifica UML 2.X. Allo stesso modo ogni relazione o dipendenza fra classi deve essere indicata con l'opportuna notazione. Le classi rappresentate nei diagrammi devono avere lo stesso nome di quelle nel codice vero e proprio, così come attributi e nomi dei metodi. È possibile, a seconda del livello di dettaglio del diagramma, omettere metodi o attributi che non risultino fondamentali per la comprensione del diagramma stesso (come, ad esempio, `getters` e `setters`).

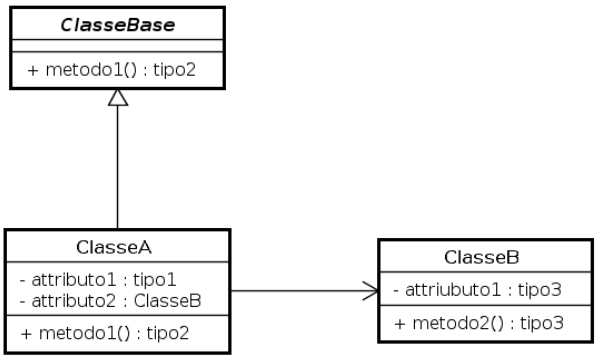

Figura 6: Esempio di diagramma delle classi

3.2.3 Qualità

3.2.3.1 Classificazione dei test

Ogni Test è strutturato come segue:

- Codice identificativo;
- Descrizione;
- Stato.

Per ciascun test è assegnato un codice identificativo la cui sintassi segue il seguente formalismo:

T{tipo}{codiceIdentificativo}

Dove:

- **lettera**: identifica uno dei seguenti tipi di test:
  - **U**: test di unità;
  - **I**: test di integrazione;
  - **S**: test di sistema.
- **codiceIdentificativo**: assume i seguenti valori in base al tipo di test:
  - **codiceNumerico**: associato ai test di unità e di integrazione, è un codice progressivo di tre cifre che parte da 001;
  - **codiceRequisito**: associato ai test di sistema. Identifica il codice univo- co associato ad ogni requisito descritto nel documento Analisi dei Requisiti v3.0.0.

Inoltre lo Stato del test può assumere i seguenti valori:

- Non implementato;
- Superato;
- Non superato.

3.2.3.2 Test di unità

Devono essere definiti dei testi di unità necessari a garantire che tutte le componenti del sistema funzionino correttamente.

3.2.3.3 Test di integrazione
Devono essere definite le classi di verifica necessarie a garantire che tutte le componenti del sistema funzionino correttamente.

3.2.3.4 Test di sistema
Richiede che i vari componenti del sistema vengano integrati al fine di garantire che tutte le componenti del sistema funzionino correttamente.

3.2.4 Codifica

In questa sezione vengono descritte le norme che i *Programmatori* devono seguire con l'obiettivo di scrivere codice leggibile, affidabile e manutenibile.

3.2.4.1 Linguaggi e Framework
Lo sviluppo del progetto didattico richiede l'uso di diversi framework e linguaggi di programmazione, ognuno mirato ad uno scopo preciso:

- **JavaScript:** viene adottato Javascript alla specifica ES6 per lo sviluppo di gran parte della backend e della frontend;
- **React:** viene adottato il framework React 16.8 per lo sviluppo della parte di frontend;
- **Java:** viene adottato Java con le specifiche del JDK 11 per lo sviluppo del modulo Server;
- **Spring:** viene adottato il framework Spring 2.0.5 per facilitare lo sviluppo del modulo sopracitato, avvalendosi solo del modulo RESTful. Vedi paragrafo §3.2.5.12.
- **Spring Data MongoDB**: viene adottato il framework Spring Data MongoDB per interrogare e aggiornare le informazioni presenti nel database, vedi paragrafo §3.2.5.13.

3.2.4.2 Stile di codifica
Al fine di produrre codice uniforme, leggibile e manutenibile è richiesto che vengano rispettate le seguenti convenzioni:

- I nomi utilizzati devono essere chiari, descrittivi rispetto alla loro funzione e in inglese;
- Evitare nomi troppo simili tra loro che possano creare difficoltà nella comprensione del codice;
- Deve essere presente almeno un breve commento descrittivo per ogni classe e metodo;
- I commenti devono essere scritti in lingua inglese senza utilizzare abbreviazioni o altre ambiguità;
- Le modifiche al codice devono sempre riflettersi sui relativi commenti;
- Evitare commenti superflui o inappropriati;
- Codifica dei file UTF-8.

3.2.4.3 Intestazione
Ogni file deve presentare un'intestazione con le seguenti informazioni:

- @path: percorso e nome del file;
- @author: nome e cognome dell'autore o autori separati da una virgola;
- @date: data ultima modifica;
- @description: breve descrizione del contenuto del file.

3.2.4.4 Java Style Code

Lo stile di codifica fa riferimento al Gooogle style code reperibile:

https://google.github.io/styleguide/javaguide.html

Sono di particolare rilevanza §4 e §5, ovvero:

- Empty blocks: may be concise

```
1  // This is acceptable
2  void doNothing() {}
3
4  // This is equally acceptable
5  void doNothingElse() {
6  }
```

- Where to break
- Variable declarations
- Annotations
- Comments

```
1 /*
2  * This is          // And so           /* Or you can
3  * okay.            // is this.          * even do this. */
4  */
```

Per quanto riguarda il paragrafo §5 è necessaria la lettura completa. Al fine di garantire l'adozione del stylecode di Google è possibile impostare Eclipse IDE con lo style appropriato. Lo style in formato XML_G per gli IDE è disponibile al seguente link:

Styleguide by Google

Per vedere come importare gli stili di codifica fare riferimento a §3.2.6.4.

3.2.4.5 JavaScript Style Code

Lo stile di codifica utilizzato per JSX di React è quello fornito da Airbnb e reperibile al link:

Styleguide for React by Airbnb

Seguono alcune convenzioni per una più facile fruibilità:

- Vanno utilizzati 2 spazi per ogni livello di indentazione, inoltre dopo la chiusura delle parentesi tonde dell'if è necessario uno spazio:

```
1 function() {
2   let name;
3 }
```

- È necessario uno spazio prima della parentesi apertura del blocco per le seguenti istruzioni: `if`, `while`, `switch` e `do`.
- Il nome delle classi deve utilizzare la PascalCase notation:

```
1 //YES
2 class Animal {
3 }
4
5 //NO
6 class animal {
7 }
```

- Utilizzare PascalCase per i componenti React e la notazione camelCase le loro istanze:

```
// bad
import reservationCard from './ReservationCard';

// good
import ReservationCard from './ReservationCard';

// bad
const ReservationItem = <ReservationCard />;

// good
const reservationItem = <ReservationCard />;
```

- Utilizzare i doppi apici (") per gli attributi JSX, un apice singolo (') per tutti gli altri file JS:

```
// bad
<Foo bar='bar' />

// good
<Foo bar="bar" />

// bad
<Foo style={{ left: "20px" }} />

// good
<Foo style={{ left: '20px' }} />
```

- Racchiudere i tag JSX all'interno di parentesi quando questi occupano più di una linea:

```
// bad
render() {
  return <MyComponent variant="long body" foo="bar">
           <MyChild />
         </MyComponent>;
}

// good
render() {
  return (
    <MyComponent variant="long body" foo="bar">
      <MyChild />
    </MyComponent>
  );
}
```

- Utilizzare le arrow-function per gestire le variabili locali:

```
function ItemList(props) {
  return (
    <ul>
      {props.items.map((item, index) => (
        <Item
          key={item.key}
          onClick={(event) => doSomethingWith(event, item.name, index)}
        />
      ))}
    </ul>
  );
}
```

- Sinstassi per creare componenti correttamente:

```
// bad
React.createClass({
  _onClickSubmit() {
    // do stuff
  },
```

```
6
7     // other stuff
8   });
9
10  // good
11  class extends React.Component {
12    onClickSubmit() {
13      // do stuff
14    }
15
16    // other stuff
17  }
```

3.2.4.6 Regole comuni

3.2.4.6.1 Ricorsione

La ricorsione va sempre evitata se possibile. Per ogni funzione ricorsiva è necessario fornire una prova di terminazione nei commenti.

3.2.4.6.2 Variabili globali

L'uso di variabili globali va sempre evitato se possibile.

3.2.4.6.3 Funzioni anonime

Per definire funzioni anonime bisogna usare la notazione a freccia, sono più concise e mantengono il contesto del this$_G$.

3.2.4.6.4 Lunghezza massima righe

La lunghezza massima per ciascuna riga di codice è di 100 caratteri, si suggerisce comunque ove possibile di rimanere entro gli 80 caratteri per favorire la leggibilità.

3.2.4.6.5 Regole di denominazione

- Utilizzare nomi descrittivi;
- Utilizzare la camelCase$_G$ per nominare oggetti, funzioni e istanze;
- Utilizzare la PascalCase$_G$ solamente per le classi e i costruttori;
- Non utilizzare trattini bassi prefissi o postfissi.

3.2.4.6.6 Codetags

I codetags sono delle parole chiavi informali all'interno dei commenti per segnalare dei problemi, note o delle parti di codice ancora da fare. La struttura 'e la seguente:
<tag>: <descrizione in una singola linea>
I tag che verranno utilizzati sono:

- **TODO**: qualcosa che va completato;
- **NOTE**: annotazioni particolari;
- **FIXME**: qualcosa che va risolto perchè non funzionante;
- **HACK**: qualcosa che funziona ma andrebbe migliorato.

Potranno essere usati all'interno di commenti blocco o inline. Sono supportati da molti IDE ed editor moderni.

3.2.5 Strumenti di supporto

3.2.5.1 RQConnect

Il tracciamento dei requisiti e dei casi d'uso avviene attraverso la piattaforma *RQConnect*, installata su un server Firebase$_G$ ad hoc per il gruppo *SWEight*. È un tool sviluppato da un componente del gruppo in occasione di questo progetto e liberamente disponibile su GitHub.
È possibile aggiungere requisiti e casi d'uso alla piattaforma e successivamente collegarli tra di loro, la schermata principale è divisa in due colonne, in quella a sinistra c'è l'elenco dei requisiti e in quella a destra i casi d'uso, cliccando su un elemento si apre una vista dettagliata nella quale è possibile leggere i dettagli e collegare l'elemento, cliccando sul pulsante *risolvi* si apre una finestra dalla quale è possibile selezionare gli elementi da collegare con l'aiuto di un menu a tendina.
Una volta completato l'inserimento ed il collegamento è possibile scaricare l'intera lista in tabella LaTeX.

3.2.5.2 Visual studio code

L'IDE$_G$ scelto per lo sviluppo è Visual studio code$_G$, versione: 1.31 disponibile al seguente link:

https://code.visualstudio.com/Download;

Esso è stato scelto principalmente perché gratuito, include supporto per il debugging e un controllo Git integrato oltre al Syntax highlighting$_G$, IntelliSense$_G$, Snippet$_G$ e code refactoring$_G$.
Si farà riferimento ai seguenti linguaggi, piattaforme, librerie e framework:

- JavaScript$_G$ 1.8.5;
- HTML5$_G$;
- Node.js$_G$ 10.15.1;
- React$_G$ 16.8.1;
- Bootstrap$_G$ 4.3.0.

I plugin utilizzati sono:

- **Prettier - Code formatter**: per la formattazione automatica del codice. Per settare la formattazione automatica al salvataggio del file si deve:
  - Premere CMD + Shift + P;
  - Andare su Open User Settings;
  - Andare in Text Editor > Formatting;
  - Spuntare la casella Format on save.
- **Simple React Snippet**: per facilitare la scrittura del codice con semplici comandi come:
  - **Imrc**: Per l'import;
  - **Cc**: Per creare le classi;

3.2.5.3 Docker

Lo strumento scelto per contenere la libreria di pos-tagging con il relativo server TCP/IP$_G$ è Docker. Esso permette di inserire l'applicativo scritto in C++$_G$ all'interno di contenitori software, fornendo un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo di Linux. Per installare Docker nella versione 18.09, è necessario installare Docker Desktop in Windows 10 Pro o Enterprise e in MacOS:

https://www.docker.com/products/docker-desktop

Mentre in Ubuntu Linux è consigliato seguire la seguente guida: "How To Install and Use Docker on Ubuntu 18.04". Il Dockerfile contiene la configurazione per la compilazione e l'installazione dell'applicativo.
Per generare l'immagine è necessario eseguire il seguente comando:

```
# docker build -tag=freeling:alpha .
```

Successivamente quest'ultima potrà essere eseguita con il comando:

```
# docker run -it -rm -p 50005:50005 freelingserver:alpha analyze -f es.cfg -server -p
50005
```

L'istruzione crea un'istanza dell'immagine che è in attesa di richieste sulla porta 50005 che analizza frasi scritte in lingua spagnola. Per permettere di analizzare frasi in più lingue creare istanze diverse con il comando precedente (`docker run`) sostituendo `es.cfg` con la lingua d'interesse, ad esempio: `it.cfg`.
Alcune delle lingue disponibili sono:

- **en.cfg**: predispone il sistema ad analizzare frasi in inglese;
- **it.cfg**: predispone il sistema ad analizzare frasi in italiano;
- **es.cfg**: predispone il sistema ad analizzare frasi in spagnolo;
- **fr.cfg**: predispone il sistema ad analizzare frasi in francese;
- **de.cfg**: predispone il sistema ad analizzare frasi in tedesco.

3.2.5.4 Eclipse

Per la parte di sviluppo in Java, il gruppo *SWEight* propone l'IDE Eclipse per le seguenti ragioni:

- È gratuito;
- Gran parte dei membri del gruppo ne hanno già conoscenza;
- Integrazione con Maven;
- È multi-piattaforma;
- Estensibilità tramite plugin per lo sviluppo con Spring.

Eclipse è reperibile al seguente link:

https://www.eclipse.org/downloads/

3.2.5.4.1 Integrazione con Spring

All'interno del marketplace di Eclipse è possibile installare il plugin per integrare il framework Spring, denominato Spring IDE. Segue la guida utilizzata per la configurazione dell'ambiente di sviluppo:

https://www.eclipse.org/community/eclipse_newsletter/2018/february/springboot.php

3.2.5.4.2 Configurazione dello stylecode

Per l'utilizzo del stylecode di Google seguire le i seguenti passi:

1. Scaricare il file xml dello stylecode per Eclipse
   http://code.google.com/p/google-styleguide/;
2. Selezionare in Eclipse `Window/Preferences` → `Java/Code Style/Formatter` → `Import` e selezionare il file xml precedentemente scaricato.

3.2.5.4.3 Plugin Fast-Code

Il plugin "Fast Code" aiuta nella stesura del codice Java con il framework Spring. Il plugin è reperibile all'indirizzo http://fast-code.sourceforge.net/.

3.2.5.4.4 Lombok
Nel progetto viene fatto uso di Lombok per la generazione automatica dei builder, setter e getter all'interno delle classi che rappresentano la persistenza. La dipendenza del plugin è già configurata all'interno del `pom.xml`, ma necessita di un'ulteriore configurazione in Eclipse, pertanto è fondamentale seguire la guida reperibile al link Install lombok in Eclipse o installare il plugin direttamente dal marketplace di Eclipse.

3.2.5.5 Node.js
Per lo sviluppo lato server in linguaggio Javascript ci si avvale dell'ultima versione Long Term Support (LTS) di Node.js, che, al momento della stesura di questo documento, è la 10.15.1 LTS. Node.js è reperibile al seguente link:

`https://nodejs.org/it/`

Il file package.json contiene tutte le configurazioni e dipendenze del progetto. Per installare tutti i moduli necessari è necessario eseguire il seguente comando nella cartella contenente il file package.json:

```
npm install
```

Per eseguire il progetto è necessario usare il comando:

```
npm start
```

3.2.5.6 Apache Tomcat
Per l'esecuzione e il deploy sul server del modulo client Freeling scritto in Java, viene usato Apache Tomcat, che permette l'esecuzione di codice Java lato server. La configurazione di Apache Tomcat viene gestita dal framework Spring. Maggiori informazioni sono reperibili al seguente link:

`http://tomcat.apache.org/`

3.2.5.7 Checkstyle
La conformità dello stile di codifica Java con le linee guida stabilite è garantita dal plugin Maven Checkstyle, che fa fallire la build in automatico in caso di errori nella forma del codice scritto. Il controllo sullo stile può essere eseguito anche manualmente con il seguente comando:

```
mvn verify
```

3.2.5.8 ESLint
Per rendere conforme lo stile di codifica per il linguaggio Javascript con la libreria ReactJs si utilizza il modulo ESLint di Node.js con la configurazione rilasciata da Airbnb. Ad ogni compilazione del progetto il compilatore stamperà un warning per ogni non conformità riscontrata. Durante la verifica, il progetto verrà ritenuto conforme e completo solo se il compilatore non segnalerà nessun errore o warning.
Per installare ESLint è necessario dare il comando: `npm install eslint -save-dev` che installerà il modulo localmente.
Per settarlo invece dare il comando: `./node_modules/.bin/eslint -init` oppure `npx eslint`.
Maggiori informazioni sono presenti al seguente indirizzo: ESLint. Una configurazione automatica è disponibile al seguente link: Eslint Prettier Airbnb

3.2.5.9 Google Java Format
Per rendere conforme lo stile di codifica per il linguaggio Java verrà utilizzato il programma `google-java-format`, il programma formatta automaticamente il codice di un file Java secondo le direttive di Google Java Style Guide. Per formattare un file bisogna recarsi nella cartella che lo contiene e dare il comando:

```
java -jar ../google.jar ––replace [NomeFile].java
```

va controllato che il programma `google-java-format` si trovi ad un livello di gerarchia delle cartelle "più alto" rispetto al file che si sta formattando, in caso contrario va modificato il campo `../google.jar` nel comando e bisogna indicare la posizione del file `google.jar`.
Se per esempio il programma si trova nella cartella Pluto presente al livello superiore delle cartelle va dato il comando `../Pluto/google.jar` al posto di `../google.jar`.
Se si vuole ad esempio formattare il file *HelloWorld.java* e il programma si trova al livello di cartella superiore va dato il comando:

```
java -jar ../google.jar --replace HelloWorld.java
```

se il file è stato formattato correttamente non vengono generati errori.
Maggiori informazioni sono presenti al link: google-java-format.

3.2.5.10 MongoDB

In seguito ad una riunione, `Verbale-5-E-2019-03-25`, il gruppo ha deciso, contattando precedentemente la proponente, di utilizzare come storage di raccolta dati il servizio MongoDB di MongoDB Inc.
MongoDB è un database non relazionale `NOSQL`, orientato ai documenti che si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile `JSON` con schema dinamico.
Per settare il database è necessario configurare il file `application.properties` dove verrà specificato l'indirizzo ip, il nome e la password per accedere al database.
Il file dovrà avere la seguente struttura:

```
server.port=8081
spring.data.mongodb.host=34.238.118.172
spring.data.mongodb.port=27017
spring.data.mongodb.username=sweightgroup
spring.data.mongodb.password=###################
spring.data.mongodb.database=colletta
spring.data.mongodb.authentication-database=admin
```

Maggiori informazioni sono presenti al seguente link: Mongo Docs, il quale rimanda alla documentazione ufficiale di MongoDB.

La configurazione del database compete all'*Amministratore* che fornirà la password per connettersi ai membri.

3.2.5.11 MongoDB Compass

Per rendere l'utilizzo e la visualizzazione dei dati presenti nel database più agevole ai membri del gruppo è stato scelto di utilizzare lo strumento `MongoDB Compass` che tramite una GUI_G di facile utilizzo permette di monitorare e consultare i dati inseriti dall'applicativo.
MongoDB compass è open source, pertanto per scaricarlo basta recarsi al seguente link: MongoDB Compass e procedere con l'installazione.
Per settarlo basterà recarsi nella sezione `New Connection` e impostare i parametri seguendo le direttive presenti nel file `application.properties` localizzato all'interno della cartella **Backend** del progetto.

3.2.5.12 Spring

Spring è un framework open source per lo sviluppo di applicazioni su piattaforma Java. A questo framework sono associati tanti altri progetti, che hanno nomi compositi come Spring Boot, Spring Data, Spring Batch.
Spring da la possibilità di creare applicazioni complesse senza la necessità di rispettare pienamente la specifica EJB_G, ma usando semplici oggetti Java (POJO_G).
Il cuore di Spring è un motore di `Inversion of Control`$_\text{G}$ che consente d'implementare applicazioni solide basate su componenti estremamente testabili e riutilizzabili.
Maggiori informazioni sono disponibili al seguente link: Spring: the source for modern java.

3.2.5.13 Spring Data MongoDB

Spring Data MongoDB è un subframework di Spring Data, lo scopo di Spring Data è quello di fornire un accesso semplice ai database, SQL e NOSQL, seguento il modello di programmazione di Spring. Nel nostro

caso il progetto Spring Data MongoDB prevede un integrazione con il database documentale MongoDB.
La chiave centrale sono i **modelli POJO**$_G$ per interagire con il database, questi permettono di interrogare e scrivere in modo semplice il database senza dover ricorrere alla sintassi di MongoDB ma effettuando delle chiamate da metodi scritti in Java.
Una collezione all'interno del database è rappresentata, in Java, mediante una classe che presenta la dicitura:

```
@Document(collection = "collectionName")
```

al di sopra della definizione di essa. Sfruttando la conversione effettuata automaticamente da Spring i documenti presenti nel database possono essere modificati attraverso funzioni e metodi di Java.
Più nello specifico la classe `MongoTemplate`, localizzata nel pacchetto `org.springframework.data.mongodb.core`, possiede dei metodi per interagire con il database: `create`, `update`, `delete`, `query` per MongoDb ed inoltre provvede a `mappare gli oggetti` presenti in Java in documenti di Mongo.
Gli oggetti vengono mappati da SpringData tramite l'implentazione dell'interfaccia `MongoConverter` nella classe `MappingMongoConverter`, questa può essere lasciata generare automaticamente o può essere ridefinita.
Per maggiori dettagli si veda: Overriding Default Mapping with Custom Converters.

4 Processi di supporto

4.1 Documentazione

4.1.1 Scopo

Lo scopo di questo processo è quello di redigere e mantenere la documentazione durante tutto il ciclo di vita del software. I documenti formali che verranno presentati saranno suddivisi in due categorie:

- **Interni**:
  - **Norme di progetto**: lo scopo del documento è di stabilire convenzioni e norme che il team SWEight deve seguire durante tutto il progetto;
  - **Studio di fattibilità**: il documento presenta il processo che ha portato il team alla scelta del capitolato, riportando le motivazioni di preferenza o rifiuto di ogni capitolato;
- **Esterni**:
  - **Piano di progetto**: documento per l'analisi e la pianificazione della gestione delle risorse di tempo e umane;
  - **Piano di qualifica**: documento che descrive standard e obiettivi che il gruppo deve raggiungere per garantire la qualità di processo e prodotto;
  - **Glossario**: documento che raccoglie i termini appartenenti ad un ambito specifico e circoscritto. Contiene la spiegazione dei termini desueti o specialistici utilizzati nei documenti;
  - **Analisi dei requisiti**: documento che contiene l'analisi dei casi d'uso del prodotto e i diagrammi di interazioni previste con l'utente;
  - **Manuale Utente**: documento che fornisce all'utente finale istruzioni su come usare il prodotto software;
  - **Manuale Sviluppatore**: documento che fornisce allo sviluppatore indicazioni utili per comprendere ed espandere il prodotto software.

4.1.2 Classificazione dei documenti

4.1.2.1 Documenti informali

Tutti i documenti non ancora approvati dal *Responsabile di progetto* sono da ritenersi informali quindi ad uso esclusivamente interno.

4.1.2.2 Documenti formali

Un documento diventa formale in seguito all'approvazione da parte del *Responsabile di progetto*. Solo i documenti formali possono essere divulgati all'esterno del gruppo. Prima di poter essere approvato un documento deve essere verificato.

4.1.2.3 Verbali

Per ogni incontro deve essere nominato un segretario che si occuperà della stesura di un verbale. Il verbale deve contenere i seguenti punti:

- **Estremi della riunione**:
  - Motivazione;
  - Luogo;
  - Data;

- Partecipanti del gruppo;
- Ora;
- Segretario.

- **Ordine del giorno**: lista degli argomenti che saranno oggetto di discussione;
- **Resoconto**: verbale effettivo degli argomenti discussi. Il tracciamento delle decisioni prese sarà fatto nel seguente modo: ogni decisione sarà identificata da un codice: *VER-NumeroVerbale-Data.x* dove *NumeroVerbale* è il numero assegnato al verbale, *Data* è la data in cui è stato svolto l'incontro e *x* è un numero sequenziale.

Una volta approvato dal *Responsabile di progetto*, il verbale deve essere distribuito a tutti i componenti del gruppo. In caso di verbale esterno il verbale dovrà essere inviato anche al proponente. I nomi dei file dei verbali devono rispettare il seguente formato:

Verbale-[numeroVerbale]-[T]-[Data].pdf

dove:

- **NumeroVerbale**: è l'identificativo numerico del verbale;
- **T** è la tipologia del verbale che si divide in:
  - **I**: interni;
  - **E**: esterni.
- **Data**: è la data in cui è stato svolto il verbale nel formato descritto nel paragrafo §4.1.6.1.

4.1.3 Nome dei documenti

I nomi dei documenti devono:

- Iniziare con la lettera maiuscola;
- Se composti da più parole anche esse devono avere la lettera maiuscola;
- Dopo il nome testuale, segue un carattere di underscore;
- Infine si indica la versione come descritto in §4.1.4.

Esempio:

`NomeDocumento_v1.0.0`

I documenti devono essere esportati in formato PDF.

4.1.4 Template

Per semplificare e standardizzare la stesura dei documenti previsti è stato creato un template LaTeX contenente tutte le impostazioni per la struttura grafica e lo stile di formattazione. Ogni membro del team deve obbligatoriamente utilizzare questo template$_G$. Quest'ultimo è disponibile all'interno dell'omonima cartella ed è denominato "SWEightStyle.sty", e dev'essere importato all'interno del documento LaTeX tramite la direttiva di inclusione di un pacchetto.

4.1.5 Versioni

Per ogni documento deve essere specificata obbligatoriamente la versione nella seguente forma:

X.Y.Z

Dove X, Y, e Z sono interi non negativi. X viene incrementata ad ogni revisione, Y viene incrementata ogni qualvolta venga inserito un nuovo capitolo, infine Z determina le modifiche al documento. Ogni elemento deve incrementare indipendentemente. Per esempio: 1.9.0 → 1.10.0 → 1.11.0 .

4.1.6 Struttura

Ogni documento deve essere realizzato a partire dal template descritto precedentemente, la cui struttura non deve essere modificata, ad eccezione dei verbali e della lettera di presentazione, dovrà essere composta da:

- **Frontespizio**: questa sezione si trova nella prima pagina di ogni documento e deve contenere:
  - **Logo**: il logo del gruppo;
  - **Titolo**: il titolo del documento;
  - **Nome del gruppo**;
  - **Nome del progetto**;
  - **E-mail**: indirizzo di posta elettronica in comune per le comunicazioni interne ed esterne;
  - **Versione**: versione del documento;
  - **Approvatore**: *Responsabile di progetto*;
  - **Redattori**: *Redattori* del documento;
  - **Verificatori**: *Verificatori* del documento;
  - **Uso**: interno o esterno;
  - **Distribuzione**: destinatari del documento;
  - **Descrizione**: breve descrizione del documento;
- **Registro delle modifiche**: il registro delle modifiche deve essere riportato subito dopo il frontespizio. Esso deve contenere, in forma tabellare: versione del documento, data di salvataggio, descrizione della nuova versione, nominativo e ruolo;
- **Indice**: contiene il nome di tutti i capitoli, sezioni e sottosezioni seguiti dal numero della pagina iniziale;
- **Indice delle immagini**: nel caso in cui il documento contenga immagini;
- **Indice delle tabelle**: nel caso in cui il documento contenga tabelle;
- **Introduzione**: contiene lo scopo del documento, una breve descrizione ed i vari riferimenti normativi e informativi utili al lettore;
- **Contenuto del documento**.

4.1.6.1 Formattazione delle pagine

Ogni pagina, escluso il frontespizio ed indice, devono contenere:

- **Intestazione**:
  - Logo del gruppo (a sinistra);
  - Numero del capitolo corrente seguito dal nome (a destra).
- **Piè di pagina**:
  - Nome e versione del documento (sinistra);
  - Numero pagina, in formato “N di M” dove N è la pagina corrente ed M è il numero di pagine totali.

4.1.7 Norme Tipografiche

- **Maiuscolo**: l'uso del maiuscolo è obbligatorio nei seguenti casi:
  - All'inizio di ogni elemento di un elenco puntato;
  - Nella scrittura degli acronimi;
  - Dopo il punto, punto di domanda, punto esclamativo;
  - Per i ruoli di progetto, i nomi dei documenti, le fasi di progetto, revisioni di progetto oltre a dove previsto dalla lingua italiana;
- **Grassetto**: il grassetto può essere utilizzato solo per evidenziare l'oggetto trattato di un elenco puntato e per le parole chiave;
- **Collegamenti**: tutti i collegamenti, URL, devono essere di colore rosso;
- **Riferimenti a sezioni**: i riferimenti interni al documento devono riportare il numero della sezione, preceduto dal simbolo di paragrafo (esempio §2.1.3);
- **Corsivo**: il corsivo deve essere utilizzato nei seguenti casi:
  - **Ruoli**: in ogni nome di ruolo;
  - **Documenti**: in ogni nome di documento;
  - **Citazioni**: in ogni citazione.
- **Codice**: i frammenti di codice devono essere racchiusi in un riquadro di colore nero;
- **Glossario**: per contrassegnare una parola che è presente nel glossario è necessario aggiungere una G maiuscola al pedice della parola (ad esempio: termine$_{G}$);
- **Elenchi puntati**:
  - Ogni elemento dell'elenco deve terminare con il punto e virgola, ad eccezione dell'ultimo elemento che deve terminare con il punto;
  - **Elenco puntato non ordinato**: gli elementi di primo livello devono avere come stile un pallino pieno nero, quelli di secondo livello un pallino nero vuoto. Elementi di livello successivo al secondo devono alternare lo stile del primo e del secondo livello;
  - **Elenco puntato ordinato**: gli elementi di primo livello devono avere come stile un numero progressivo, mentre quelli di secondo livello una lettera racchiusa all'interno di parentesi graffe.
- **Citazioni**: ogni citazione deve essere accompagnata dal riferimento bibliografico.

4.1.7.1 Formati

- **Date**: ogni data deve rispettare lo standard internazionale ISO 8601:2004, che prevede la seguente convenzione:

  AAAA-MM-GG

  dove:
  - **AAAA**: rappresenta il formato dell'anno scritto con quattro cifre;
  - **MM**: rappresenta il formato del mese scritto con due cifre;
  - **GG**: rappresenta il formato del giorno scritto con due cifre;
- **Orari**: se presenti, gli orari fanno riferimento anch'essi allo standard internazionale ISO 8601:2004:

  HH:MM

dove:

- **HH**: rappresenta l'ora espressa con due cifre il cui valore è compreso tra 00 e 23;
- **MM**: rappresenta i minuti espressi con due cifre il cui valore è compresi tra 00 e 59.

4.1.8 Componenti grafiche

4.1.8.1 Tabelle

Tutte le tabelle devono avere un indice univoco che la identifichi all'interno del documento ed una breve descrizione posizionata sotto di essa. Le intestazioni delle tabelle devono essere in grassetto. Le righe possono avere colori alternati per facilitarne la lettura.

4.1.8.2 Immagini

Tutte le immagini inserite nei vari documenti devono avere un eguale e sufficiente margine orizzontale e verticale in modo da non ridurre la leggibilità del testo. Ogni immagine deve anch'essa avere un indice univoco che la contraddistingua nell'intero documento. In particolare le immagini dovranno rispettare le seguenti proprietà, salvo indicato diversamente:

- **Nome**: deve rispettare la notazione a cammello$_G$;
- **Formato**: è preferibile utilizzare il formato PNG, ma è accettato anche il PDF in quanto scalabile;
- **Importazione in LaTeX**: quando importata nel documento, l'immagine dev'essere collocata all'interno del tag figure, accompagnata da una descrizione e con lunghezza massima uguale a 17cm;
- **Posizionamento**: preferibilmente deve essere centrata orizzontalmente.

4.1.8.2.1 Esportare immagini in Astah

Per esportare i diagrammi in Astah, dopo aver completato il diagramma, selezionare:

$$Tools \rightarrow Export\ image \rightarrow Current\ Diagram \rightarrow PNG.$$

L'immagine rappresentante l'UML deve avere le seguenti caratteristiche:

- **Nome file**: UC`Codice Identificativo`, privo di "."
  Ad esempio: UC4.2.png → UC42.png;
- **Formato file immagine**: `.png`;
- **Massima larghezza**: 17cm;
- **Didascalia:** figura con codice caso d'uso;
- **Salvataggio:** Il file deve essere salvato all'interno del folder img, destinato alle immagini.

Inoltre, anche il file con estensione asta deve essere salvato nella cartella predisposta al fine di permetterne modifiche successivamente. Per la realizzazione dell'Analisi dei Requisiti la cartella destinata al salvataggio dei diagrammi modificabili è: "Casi d'uso", reperibile all'interno del branch$_G$ Analisi dei Requisiti. Il nome del diagramma deve essere il medesimo dell'immagine esportata.

4.1.9 Colori

In presentazioni e documenti dove sia necessario inserire elementi che necessitino di una colorazione interna, è da preferirsi l'uso del *Rosso SWEight*, colore presente anche nel logo del gruppo.

#ff5252

4.1.10 Strumenti di supporto

4.1.10.1 LaTeX

L'intera documentazione deve essere prodotta utilizzando il linguaggio di markup$_G$ LaTeX, scelto dal team per i seguenti motivi:

- Gestisce elegantemente e con facilità gli indici, i pedici, i riferimenti ed il glossario;
- È un linguaggio che supporta il versionamento$_G$;
- Permette la suddivisione del documento in parti che rappresentano le varie sezioni facilitando la stesura.

Per utilizzare LaTeX è necessario installare il compilatore pdflatex, integrato in un software come TexLive reperibile all'indirizzo https://www.tug.org/texlive/. Quest'ultimo è cross platform$_G$, open source$_G$ e gratuito. Si consiglia l'installazione del pacchetto completo per evitare di installare pacchetti successivamente. Per la stesura del documento si consiglia Texmaker, il quale presenta le seguenti peculiarità:

- Compilatore e visualizzatore PDF$_G$ per il documento prodotto;
- Evidenziamento della sintassi;
- Completamento automatico.

Il software è reperibile all'indirizzo http://www.xm1math.net/texmaker/, è cross-platform ed è gratuito.

4.1.10.2 GNU Aspell

Al termine della stesura dei documenti, prima della fase di verifica del documento, un *Redattore* deve effettuare un controllo ortografico sui file tex tramite GNU Aspell, reperibile al seguente link:

http://aspell.net/

Il dizionario italiano per l'applicativo, insieme alle istruzioni per l'installazione, è disponibile al seguente link:

https://ftp.gnu.org/gnu/aspell/dict/0index.html

Il programma esegue da terminale con il seguente comando:

```
aspell -c -t nomeFile.tex -d it
```

4.1.10.3 MAGIC

Per il calcolo dell'indice di Gulpease, il gruppo *SWEight* si avvale dello script MAGIC (Magic Automatic Gulpease Index Calulator) sviluppato internamente. Il tool esegue automaticamente ogni giorno e svolge i seguenti step:

1. Compilazione di tutti i documenti;
2. Calcolo dell'indice di Gulpease per ogni documento;
3. Salvataggio dei valori ottenuti in foglio di calcolo Google.

I valori inseriti sono automaticamente plottati in un grafico, che permette ai verificatori di monitorare celermente il livello di leggibilità di ogni documento.

4.1.11 Ciclo di vita dei documenti

Per gestire il ciclo di vita dei documenti, deve essere utilizzato il servizio di task offerto da Asana, come descritto nella sezione §5.2 gestione dell'infrastruttura.

1. Il *Responsabile di progetto* crea il task, assegna i *Redattori* ed i *Verificatori*, e lo mette nella colonna "To Do" ;
2. Quando i *Redattori* assegnati iniziano la stesura del documento spostano il task nella colonna "In progress";

3. Finita l'attività di scrittura il task deve essere spostato in "Need Review";
4. I *Verificatori* assegnati controllano il documento decidendo se:
   (a) Accettarlo, e muoverlo in "Done";
   (b) Non accettarlo e rimetterlo in "In Progress", lasciando un commento al task;
5. Se non accettato, i *Redattori* devono apportare le modifiche descritte dai *Verificatori*;
6. Se accettato, il documento giunge al *Responsabile di progetto* che può:
   (a) Approvarlo, rendendo il documento formale e marcarlo come completato.
   (b) Non approvarlo, fornendo le opportune motivazioni come commento al task e spostandolo in "To Do";
7. Se approvato, sarà compito dell'*Amministratore di Progetto* effettuare il merge.

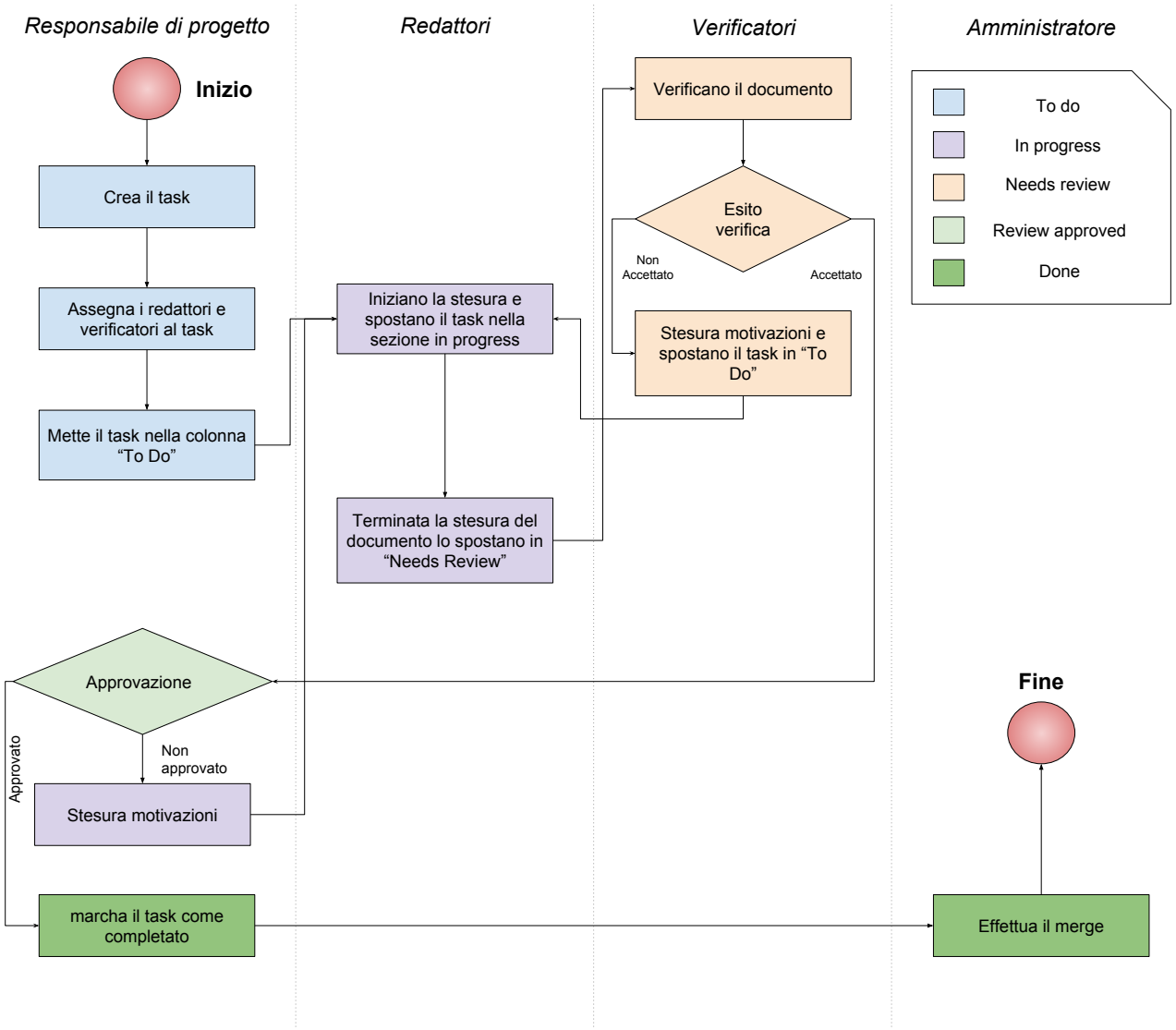

Figura 7: Ciclo di vita di un documento

4.1.11.1 Diagrammi UML

Il software che deve essere utilizzato per la creazione dei diagrammi UML è Astah versione 8.0.0, gli *Analisti*

devono utilizzare la documentazione del software per il suo studio e utilizzo. Ogni diagramma di caso d'uso è identificato in alto a sinistra dal suo codice identificativo e dal nome.

4.1.11.2 Draw.io

Un'alternativa ad Astah è Draw.io, un applicativo online gratuito per la realizzazione dei modelli UML, per l'esportazione dei diagrammi si deve utilizzare lo stesso metodo impiegato in §4.1.8.2.1. Si è predisposto l'utilizzo di Draw.io perché a differenza di Astah, perché permette di lavorare in contemporanea sullo stesso diagramma. Il formato modificabile dei diagrammi dev'essere XML.

4.2 Qualità

4.2.1 Scopo

Questa sezione descrive le classificazioni e le formule per il calcolo delle metriche necessarie per assicurare la qualità.

4.2.2 Classificazione delle metriche

Per garantire la qualità del lavoro gli Amministratori hanno definito delle metriche, che devono però rispettare la seguente notazione

M[categoria][numero]

dove:

- Categoria: indica la categoria della metrica, può assume i valori:
  - D per indicare le metriche di documenti;
  - S per indicare le metriche di software;
  - P per indicare le metriche di processi.
- Numero: identifica in maniera univoca la metrica in ogni categoria, assume un valore intero a 3 tre cifre incrementale a partire da 1.

Ad esempio la prima metrica sui documenti sarà classificata in questo modo:

MD001

4.2.2.1 Metriche Documenti

Le metriche presentate in questa sezione hanno come scopo fornire delle metriche per garantire un buon livello di leggibilità dei documenti.

4.2.2.1.1 MD001 - Indice Gulpease

L'Indice Gulpease è un indice di leggibilità di un testo tarato sulla lingua italiana. Rispetto ad altri ha il vantaggio di utilizzare la lunghezza delle parole in lettere anziché in sillabe, semplificandone il calcolo automatico. L'indice di Gulpease considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere. L'indice di Gulpease può assumere valori fra 0 e 100 e si calcola come segue:

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

4.2.2.2 Metriche Software

Le metriche presentate in questa sezione hanno come scopo fornire delle metriche per garantire un buon livello di qualità del software.

4.2.2.2.1 MS001 - Numero di Metodi
Numero medio di metodi contenuti nelle classi di un package. Un numero di metodi troppo alto può indicare la necessità di scomporre una classe. Un numero di metodi troppo basso, d'altro canto, deve far riflettere sull'effettiva utilità della classe presa in esame.

4.2.2.2.2 MS002 - Numero di Parametri
Numero di parametri passati a un metodo. Un eccessivo numero di parametri passati ad un metodo può indicare un'eccessiva complessità dello stesso, che va scomposto o quanto meno ripensato.

4.2.2.2.3 MS003 - Funzioni di interfaccia per package
Numero di funzioni che un package espone. Un valore troppo elevato potrebbe indicare un errore di progettazione.

4.2.2.2.4 MS004 - Complessità Ciclomatica
La Complessità Ciclomatica (CC) è una metrica software che misura la complessità di un programma contando il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. In questo grafo, i nodi corrispondono a gruppi indivisibili di istruzioni, mentre gli archi connettono due nodi se il secondo gruppo di istruzioni può essere eseguito immediatamente dopo il primo gruppo: sono quindi responsabili dell'aumento della CC i punti decisionali, *if* e *for*. Tenere bassa la CC può portare a vari vantaggi:

- Minore complessità durante lo sviluppo;
- Maggior facilità nell'aumentare la code coverage in fase di test;
- Maggior coesione del codice.

La Complessità Ciclomatica è calcolata come segue:

$$CC = v(G) = e - n + 2p$$

dove:

- e = numero di archi del grafo;
- n = numero di nodi del grafo;
- p = numero di componenti connesse.

4.2.2.2.5 MS005 - Campi dati per classe
Numero di campi dati contenuti da una classe. Una classe con troppi campi dati può essere sintomo di cattiva progettazione e va ripensata.

4.2.2.2.6 MS006 - Commenti per linee di codice
Rapporto fra numero di righe di commento e numero totale di righe (vuote escluse). Un codice ben commentato può essere compreso più facilmente e velocemente, facilitando le operazioni di manutenzione.

4.2.2.2.7 MS007 - Code Coverage
Percentuale delle linee di codice coperte dai test. Avere codice coperto da test riduce la possibilità di introdurre errori nel prodotto.

4.2.2.2.8 MS008 - Superamento test
Percentuale di test superati. Per avere un prodotto di qualità, è necessario che esso superi i test prestabiliti.

4.2.2.2.9 MS009 - Requisiti obbligatori soddisfatti

Percentuale di requisiti obbligatori stabiliti dalla proponente soddisfatti. È fondamentale, per la buona riuscita del progetto, soddisfare i requisiti obbligatori presenti nell'*AnalisiDeiRequisiti_v3.0.0*.

4.2.2.2.10 MS010 - Media di build Travis settimanali

Media del numero di build effettuate su Travis CI settimanalmente. Questo valore permette al gruppo di controllare l'andamento del lavoro sul prodotto.

4.2.2.2.11 MS011 - Percentuale build Travis superate

Percentuale delle build Travis superate con successo. Troppe build fallite indicano la presenza di errori che non sono stati risolti nei tempi prestabiliti.

4.2.2.2.12 MS012 - Accoppiamento tra classi

Calcola il numero di classi con cui ogni classe è accoppiata. Una classe si dice accoppiata con un'altra se è presente una dipendenza tra le due, indipendentemente dal verso della dipendenza. Un valore elevato di questa metrica non è desiderabile poiché evidenzia una bassa modularità del codice.

4.2.2.3 Metriche processi

Le metriche presentate in questa sezione hanno come scopo fornire delle metriche per garantire un buon livello di qualità sui processi.

4.2.2.3.1 MP001 - Schedule Variance

La Schedule Variance indica se una certa attività o processo è in anticipo, in pari, o in ritardo rispetto alla data di scadenza prevista. Se $SV < 0$ significa che l'attività o il processo è in pari o in anticipo, invece, se $SV \geq 0$ significa che l'attività è in ritardo. La Schedule Variance è calcolata come segue:

$$SV = data\ conclusione\ effettiva - data\ conclusione\ pianificata$$

4.2.2.3.2 MP002 - Budget Variance

La Budget Variance misura, ad una determinata data, lo scostamento fra quanto speso e quanto preventivato. Se $BV \geq 0\%$ significa che il progetto sta spendendo le risorse più velocemente di quanto pianificato. La Budget Variance è calcolata come segue:

$$BV = \frac{costo\ effettivo - costo\ preventivato}{costo\ preventivato}$$

4.2.2.3.3 standard ISO/IEC 9126

Lo standard ISO/IEC 9126 stabilisce una serie di linee guida mirate al miglioramento delle qualità del software sviluppato.

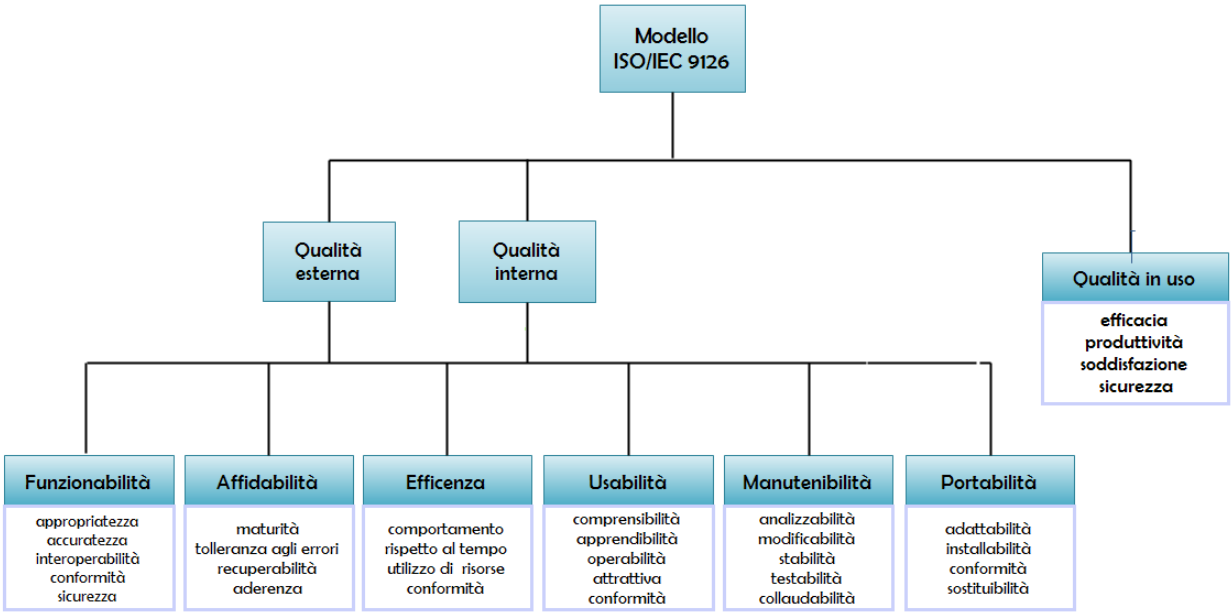

Figura 8: Rappresentazione grafica di ISO/IEC 9126 [Wikipedia]

Come presentato in Figura 12, ISO/IEC 9126 prescrive indicazioni su:

- **Qualità interna:** è misurata sul software non eseguibile, come, ad esempio il codice sorgente. Le misure effettuate permettono di avere una buona previsione della qualità esterna.
- **Qualità esterna:** è misurata tramite l'analisi dinamica su software eseguibile.
- **Qualità in uso:** definita in base all'esperienza utente. Sono da perseguire i seguenti obiettivi:
  - Efficacia;
  - Produttività;
  - Soddisfazione;
  - Sicurezza.

ISO/IEC 9126 definisce inoltre una serie di requisiti da soddisfare per produrre software di qualità:

- **Funzionalità:** capacità di un prodotto software di soddisfare le esigenze stabilite (vedi *AnalisiDeiRequisiti_v3.0.0*);
- **Affidabilità:** capacità di un prodotto di mantenere un determinato livello di prestazioni in date condizioni d'uso per un certo periodo;
- **Efficienza:** capacità di un prodotto software di eseguire il proprio compito minimizzando il numero di risorse usate;
- **Usabilità:** capacità del prodotto software di essere utilizzato, capito e studiato dall'utente a cui è rivolto;
- **Manutenibilità:** capacità del prodotto software di evolvere mediante modifiche, correzioni e miglioramenti;
- **Portabilità:** capacità del prodotto software di funzionare ed essere installato su più ambienti hardware e software.

4.3 Configurazione

4.3.1 Versionamento

4.3.1.1 Introduzione

Git, il più usato e conosciuto software di controllo di versione$_G$ open source, è stato scelto per il controllo di versione, GitHub invece è la piattaforma di web hosting.
Il repository dedicato ai documenti è formata dalle seguenti subdirectory:

- **RR**: contiene tutti i documenti da consegnare per la Revisione dei Requisiti;
- **RP**: contiene tutti i documenti da consegnare per la Revisione di Progettazione;
- **RQ**: contiene tutti i documenti da consegnare per la Revisione di Qualifica;
- **RA**: contiene tutti i documenti da consegnare per la Revisione di Accettazione.

4.3.1.2 Messaggi di commit

Ogni volta che si effettuano modifiche sui file del repository locale per poi esser caricate in quello remoto, bisogna specificarne le motivazioni. È importante che i messaggi siano esplicativi e che non creino ambiguità, pertanto si scrivano le varie modifiche facendo uso di un elenco puntato.

4.3.1.3 Merge

Nel caso in cui, nel tentativo di effettuare l'upload nella repository remota, si verifichi un conflitto$_G$ è compito del componente del gruppo informare l'*Amministratore di Progetto*. Nel caso in cui non si fosse in grado di effettuare il merge$_G$ sfruttando il tool automatico messo a disposizione da client di versionamento, chiedere all'*Amministratore di Progetto* di risolvere manualmente la cosa il prima possibile.

4.3.1.4 Branching

Per minimizzare e meglio gestire i conflitti, il gruppo *SWEight* lavora sulla propria repo mediante la tecnica del *feature branch*.È inoltre impostata una procedura standard di creazione e nomenclatura dei branch:

- È permesso lavorare sul branch principale solo in caso di fix veloci e autorizzati dal responsabile;
- In caso siano necessarie modifiche più profonde, ad un componente o ad un documento, è necessario branchare dal ramo principale. La nomenclatura deve seguire il seguente schema: "Modifico_XXXX_YY", dove:
  - XXXX è una sigla da 2 a 4 caratteri che individua la macro area che è soggetta a modifiche;
  - YY è la prossima revisione che il gruppo dovrà sostenere.

  Ad esempio, un membro modifica l'Analisi dei Requisiti, e il gruppo sta correntemente lavorando per soddisfare la Revisione di Progettazione, il nome del branch è: "Modifico_AR_RP";
- Se fosse necessario parallelizzare la modifica di un certo componente, allora si deve branchare dal ramo creato in precedenza secondo il seguente schema: "XXXX_ZZ", dove
  - XXXX è la sigla del ramo da cui ho branchato;
  - ZZ sono le iniziali del membro che ha creato il branch.

  Ad esempio, se Sebastiano Caccaro necessita di branchare da "Modifico_AR_RP", creerà il branch "AR_SC".

4.3.1.5 Feature branch Per l'attività di codifica si utilizza il modello feature branch, ovvero ogni funzionalità viene creata e testata in un branch a parte prima di essere mergiata, ovvero integrata, all'interno dell'applicativo. I punti forti di questo modello sono:

- Usare un branch per feature/caratteristica;
- L'incapsulamento consente di lavorare senza disturbare la base di codice principale;
- Consente una collaborazione più semplice;
- Unisce e mappa i conflitti concettuali: più facile da tracciare.

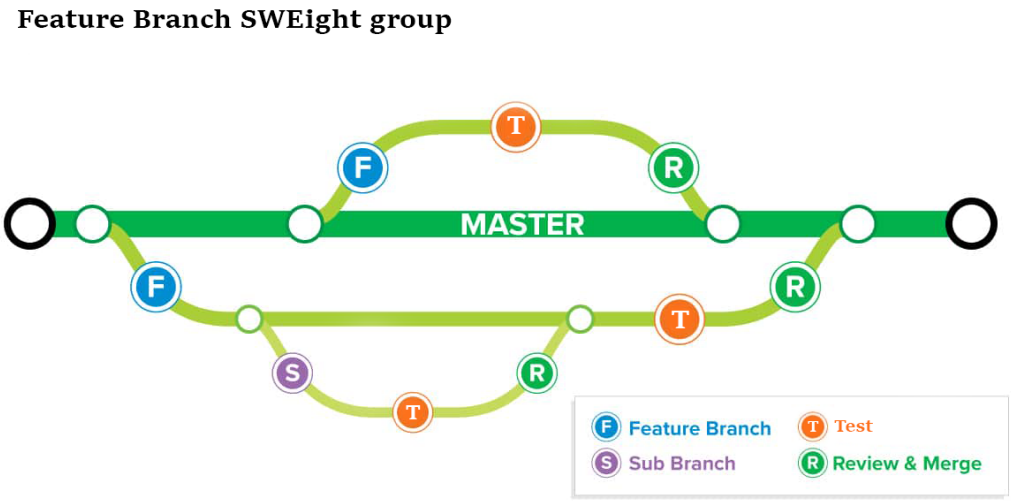

Figura 9: Panoramica feature branch
Immagine tratta da www.revelry.co

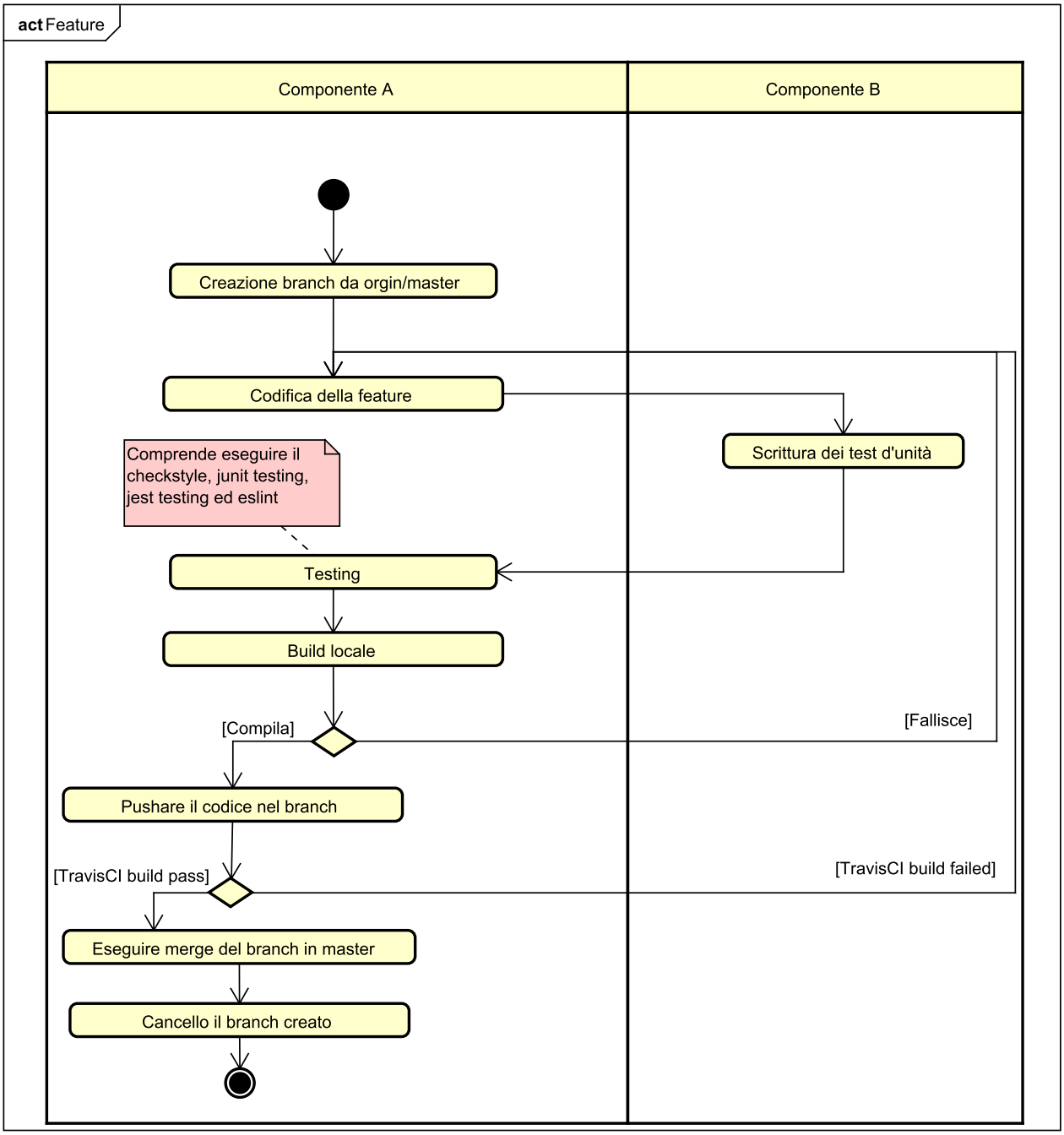

Figura 10: Diagramma di sequenza feature branch, con il termine componente si fa riferimento a due elementi del gruppo

4.3.2 Strumenti di supporto

4.3.2.1 GitHub Desktop

Il client$_G$ che viene utilizzato per il controllo di versione è GitHub Desktop, esso permette di gestire attraverso un'interfaccia grafica la repository$_G$ al fine di consentire anche a coloro che non hanno familiarità con tale tool di apprenderlo con una learning curve$_G$ meno ripida. Per utilizzare il programma è necessario recarsi al sito: https://desktop.github.com/ disponibile sia per Microsoft Windows$_G$ e Apple MacOS$_G$. Per quanto

riguarda le varie distribuzioni Linux$_G$ non è disponibile un client ufficiale, è disponibile però un porting$_G$ non ufficiale all'indirizzo: https://github.com/shiftkey/desktop

4.3.2.2 Maven

Il gruppo *SWEight*si avvale di Apache Maven per la gestione della configurazione del modulo client Freeling scritto in Java. Tale strumento è fondamentale per consentire la ripetibilità della build sul pc di ogni membro del gruppo. Inoltre, permette di eseguire in maniera automatica tutti i test del modulo JUnit. Maven è reperibile al seguente link: "Apache Maven Project".
Posizionandosi nella cartella gestita da Maven e dando il seguente comando da terminale:

```
# mvn clean install
```

si effettua la compilazione, si eseguono i test, si costruisce l'artefatto e lo si copia nel repository locale.
Per effettuare solo i test di unità basterà lanciare il comando:

```
# mvn test
```

Maven mette a disposizione molteplici configurazioni per il testing, vedi: "How to run unit test with Maven".

4.3.2.3 Travis CI

Il gruppo *SWEight* si avvale si Travis CI per eseguire automaticamente build e test ogni qual volta avvenga un commit sul branch master. Ciò consente di accorgersi immediatamente di eventuali bug introdotti durante la codifica.
Travis viene configurato attraverso il file `.travis.yml` posto nella root del progetto su GitHub. Per accedere alla dashboard di Travis CI, i membri del gruppo dovranno accedere a https://travis-ci.org/ col proprio account.

4.4 Verifica

4.4.1 Scopo

La verifica dei processi, documenti e prodotti è un'attività da eseguire continuamente durante lo sviluppo del progetto. Di conseguenza, servono modalità operative chiare e dettagliate per i *Verificatori*, in modo da uniformare le attività di verifica svolte ed ottenere il miglior risultato possibile.

La corretta implementazione del processo deve:

- Fornire le procedure di verifica necessarie;
- Individuare i criteri per la verifica;
- Individuare eventuali difetti perché possano essere corretti.

4.4.2 Analisi

4.4.2.1 Analisi statica

- **Walkthrough**: lettura completa del codice sorgente da analizzare. Va utilizzata unicamente durante le prime fasi del progetto in quanto risulta onerosa e non efficiente. Questa tecnica di analisi prevede una lettura critica del codice o del documento prodotto. Gli *Analisti* che la utilizzano devono stilare una lista di controllo con gli errori rilevati più frequentemente;
- **Inspection**: lettura mirata del codice sorgente da analizzare. Questa tecnica di analisi presuppone l'esperienza da parte del *Verificatore* nell'individuare gli errori e le anomalie più frequenti in modo tale da creare una lista di controllo per poter localizzare eventuali punti critici in cui cercare errori. Dopo ogni analisi la lista di controllo deve essere incrementata con eventuali nuovi errori rilevati.

4.4.2.2 Analisi dinamica

Si applica solo alle componenti software e consiste nella verifica e validazione attraverso i test. Per garantire la correttezza è necessario che i test siano ripetibili, dato lo stesso input il test deve produrre sempre lo stesso output.
Solo i test con queste caratteristiche riescono a verificare la correttezza del prodotto.
Per ogni test deve quindi essere definito:

- **Ambiente**: sistema hardware e software in cui viene eseguito il test;
- **Stato iniziale**: stato iniziale da cui si parte ad eseguire il test;
- **Input**: input inserito;
- **Output**: output atteso.

4.4.3 Test

4.4.3.1 Test di unità

Il test di unità verifica che ogni singola unità funzioni correttamente, le unità sono specificate nella progettazione di dettaglio.

In particolare, si verifica che i requisiti per quella determinata unità siano soddisfatti.

- **Test strutturale (white-box)**: verifica la logica interna del codice dell'unità cercando la massima copertura. Ogni singola prova deve attivare un singolo cammino di esecuzione all'interno dell'unità. L'insieme di dati di ingresso che ottiene quell'effetto costituisce un caso di prova;
- **Test funzionale (black-box)**: fa riferimento alla specifica dell'unità utilizzando dati di ingresso capaci di provocare l'esito atteso. Ciascun insieme di dati di ingresso che produca un dato comportamento funzionale costituisce un singolo caso di prova.

La forma più semplice, adottata all'interno del progetto, di questo tipo di test prevede la combinazione di due unità già sottoposte a test in un unico componente e il test dell'interfaccia presente tra le due.

4.4.3.2 Test di Integrazione

Si applica alle componenti specificate nella progettazione architetturale e rappresenta l'estensione logica del test di unità. Consiste nella combinazione di due unità già sottoposte a test in un solo componente e nel test dell'interfaccia presente tra le due. Il test di integrazione consente di individuare i problemi che si verificano quando due unità si combinano. Per effettuare tali test si farà uso di classi appositamente create per simulare e verificare l'interazione. Strategie di integrazione:

- **Assemblare parti in modo incrementale**:
  - Si sviluppano e si integrano prima le parti con minore dipendenza funzionale e maggiore utilità;
  - Poi si risale l'albero delle dipendenze;
  - Questa strategia riduce il numero di stub necessari al test, ma ritarda la disponibilità di funzionalità di alto livello;
- **Top-down**:
  - Si sviluppano prima le parti più esterne, quelle poste sulle foglie dell'albero delle dipendenze e poi si scende;
  - Questa strategia comporta l'uso di molti stub ma l'integrazione avviene a partire dalle funzionalità di più alto livello;
- **Assemblare produttori prima dei consumatori**;
- **Assemblare in modo che ogni passo di integrazione sia reversibile**.

4.4.3.3 Test di sistema
Il test di sistema consiste nella validazione del sistema ed è un test di tipo funzionale. Viene eseguito quando si ritiene che il prodotto sia giunto ad una versione definitiva, verificando la completa copertura dei requisiti software.

4.4.3.4 Test di regressione
Il test di regressione va eseguito ogni volta che viene modificata un'implementazione in un programma. È possibile eseguire nuovamente i test esistenti sul codice modificato, integrando solo le parti che abbiano precedentemente superato il test di unità, per stabilire se le modifiche apportate hanno alterato elementi precedentemente funzionanti. Se necessario è anche possibile scrivere nuovi test.
I contenuti del test di regressione vanno decisi nel momento in cui si approvano modifiche al software.

4.4.3.5 Test di accettazione (collaudo)
Il test di accettazione consiste nel collaudo del sistema eseguito dal Committente. Se l'esito è positivo si può procedere al rilascio ufficiale del prodotto.

4.4.4 Verifica dei documenti

Ogni volta che un documento viene modificato, per essere approvato dal *Responsabile di progetto*, deve essere verificato da un *Verificatore*.
Per essere certi che il documento sia conforme alle *NormeDiProgetto_v3.0.0* e agli altri documenti presentate, il *Verificatore* deve:

- **Verificare il contenuto**: controllare che tutto il contenuto del documento sia inserito nella giusta sezione ed adeguatamente impaginato;
- **Verificare il glossario**: controllare che tutte le parole da inserire nel glossario siano state inserite e che tutte le parole inserite nel glossario, siano state targate con il pedice "G";
- **Verifica tipografica**: controllare con il controllo ortografico dell'editor utilizzato eventuali errori e correggerli;
- **Segnalare gli errori**: una volta completata la verifica, deve segnalare tutti gli errori trovati e comunicarli al *Redattore*.

4.4.4.1 Regole a garanzia dell'assenza di conflitto di interessi
I *Redattori* dei documenti non possono anche esserne i *Verificatori*. Pertanto è necessario che nessuno dei componenti del gruppo sia messo in condizione di verificare la parte da lui redatta, e quindi affinché ciò non avvenga il gruppo si adopererà attuando una rotazione periodica dei ruoli, o in caso questa non sia possibile, semplicemente assicurarsi che nessun membro verifichi le sezioni dei documenti assegnatogli.

4.4.4.1.1 Verifica dell'Analisi dei Requisiti
L'attività di verifica per l'Analisi dei Requisiti sarà realizzata dai *Redattori* della stessa in modo tale che nessuno dei componenti verifichi la parte da lui curata. Ad esempio, se un componente ha realizzato la sezione "A" del documento, egli verificherà la sezione "C" stilata da un altro componente, che a sua volta verificherà la sezione di un altro componente. La scelta di questa soluzione è stata protratta poiché i componenti del gruppo, che hanno realizzato il documento hanno sviluppato una discreta conoscenza nella modellazione UML, e pertanto possono fornire una verifica più accurata.

4.4.5 Verifica dei diagrammi

I diagrammi devono essere verificati manualmente dal *Verificatore* che deve controllare che aderiscano correttamente allo standard 2.0 (vedi Riferimenti Normativi). In particolare deve controllare che i diagrammi

di flusso siano rappresentati in maniera corretta e che i diagrammi utilizzino correttamente le inclusioni e le estensioni, come ad esempio controllando che queste ultime siano accompagnate dalla condizione.

4.4.6 Strumenti di supporto

4.4.6.1 JUnit

Per la parte del codice scritta in Java, i test di unità saranno eseguiti tramite la libreria JUnit. La scelta è stata dettata dalla conoscenza dello strumento da gran parte dei membri del gruppo. I test sono localizzati all'interno della cartella *test*, ogni file di test ha lo stesso nome della classe che deve testare con l'aggiunta della keyword *Test* alla fine della parola.
Ad esempio se si vuole testare il file HelloWorld.java il file di test sarà chiamato HelloWorld**Test**.java.
I metodi di test devono avere l'annotazione *@Test* posta prima della definizione del metodo, senza spazi, che avrà segnatura *public void* e ogni metodo potrà avere un solo controllo al suo interno.
Il nome del metodo per il test dovrà essere significativo per la funzionalità che si sta andando a testare, vedi: "Naming standards for unit tests".
Esempio di `HelloWorldTest`:

```
1   public class HelloWorldTest{
2
3     Helloworld object = new Helloworld(); // classe da testare
4
5     @Test
6     public void checkText{
7       assertEquals("helloWorld", object.getHelloWorld());
8       // assertEquals() controlla se i parametri sono uguali
9     }
10  }
```

La classe *Assert* prevede molteplici metodi che verificano se il risultato atteso è uguale al risultato ritornato dal metodo che si sta testando, vedi "Class Assert".
Per maggiori informazioni riguardanti le proprietà delle annotazioni dei test vedi "Annotation Type Test".
In Eclipse IDE la creazione di un test d'unità può essere eseguita semplicemente evidenziando il metodo e premendo `Ctrl + Alt + Shift + U`, dopo aver installato il plugin "FAST CODE" citato in §3.2.6.4.3.

4.4.6.2 Jest ed Enzyme

Il framework Jest e la libreria Enzyme saranno utilizzati dal gruppo per la creazione di test sul codice React. Permettono, fra le altre cose, di creare mock e simulare input utente.
Per creare un test ci si dovrà posizionare all'interno della cartella *components* e creare la cartella *__tests__*, se non presente, e creare il file di test.
Per testare, ad esempio, il file HelloWorld.jsx il file di test sarà chiamato HelloWorld.**test**.jsx.

```
it('sums numbers', () => {
   expect(sum(1, 2)).toEqual(3);
   expect(sum(2, 2)).toEqual(4);
});
```

Il comando `expect` verifica che i parametri siano uguali, vedi: API Reference: "Expect" per altri controlli disponibili.
Tramite il comando `npm test` vengono eseguiti i test sui file modificati, è possibile applicare i test ogni volta che viene apportata una modifica localmente tramite il comando `npm run test - -watch`.
Maggiori informazioni e i link per i download sono disponibili di seguito:

- **Jest:** https://jestjs.io/;
- **Enzyme:** https://github.com/airbnb/enzyme;
- **Guida introduttiva:** https://tinyurl.com/sw8EnzymeJest.

4.5 Validazione

4.5.1 Scopo

Lo scopo del processo di validazione è di determinare in maniera oggettiva che il prodotto esaminato sia conforme ai requisiti richiesti e che soddisfi il compito per cui è stato creato. La corretta implementazione del processo deve:

- Utilizzare gli stessi strumenti e le stesse procedure del processo di verifica;
- Fornire tutti i dati necessari alla valutazione del prodotto;
- Verificare che tutte le metriche stabilite siano soddisfatte.

4.5.2 Procedure

L'attività di verifica deve essere svolta rispettando il seguente ordine:

1. i verificatori eseguono i test sul prodotto finale tracciando gli esiti ottenuti;
2. il Responsabile di progetto analizza i risultati ottenuti decidendo se accettarli o ripetere alcuni test;
3. una volta accettati il Responsabile di progetto consegna i risultati al proponente.

5 Processi organizzativi

5.1 Processo di gestione

5.1.1 Obiettivo

Gli obiettivi del processo di gestione sono:

- Raggiungere l'efficienza e l'efficacia seguendo un approccio sistematico dei processi software al fine di raggiungere gli obiettivi di progetto rispettando le scadenze;
- Introdurre nuovi processi o migliorare quelli già esistenti.

5.1.2 Incontri

5.1.2.1 Interni Una riunione dell'intero team è richiesta in caso di necessità del *Responsabile di progetto*. Nel momento in cui un membro del gruppo abbia la necessità di incontrarsi con il team, deve inviare una richiesta al *Responsabile di progetto* che la valuta e organizza un eventuale incontro. Questi sono fissati solo per discutere di argomenti attinenti al progetto. Le riunioni possono essere realizzate anche in forma telematica$_G$, adottando le stesse procedure e regole utilizzate per le riunioni fisiche. Dopo ogni riunione dovrà essere obbligatoriamente stilato un verbale che verrà strutturato secondo quanto scritto nel paragrafo §4.1.2.3.

5.1.2.2 Esterni Il *Responsabile di progetto* è l'unico a poter fissare gli incontri con il Proponente o il Committente e successivamente informare i componenti del team. Non è necessaria la presenza dell'intero gruppo di lavoro durante gli incontri con gli esterni. Il numero di partecipanti sarà quindi limitato a 2 rappresentanti del team di sviluppo.
Il *Responsabile di Progetto* deve essere sempre presente agli incontri con gli esterni mentre il secondo partecipante del team è scelto in base all'ordine del giorno.
Ogni riunione comprende la stesura di un verbale ufficiale contenente le seguenti informazioni:

- Data e ora;
- Luogo;
- Partecipanti esterni;
- Partecipanti interni;
- Ordine del giorno;
- Domande e risposte.

5.1.3 Comunicazione

5.1.3.1 Interna Per le comunicazioni interne al gruppo è stato adottato Slack$_G$, un'applicazione di messaggistica multi-piattaforma con funzionalità specifiche per gruppi di lavoro.
Su Slack si possono integrare Bot$_G$, creare canali tematici per rendere più efficiente lo scambio e il reperimento delle informazioni, fare ricerche di messaggi vecchi e condividere file di grosse dimensioni.
In Slack, sono stati creati dei canali per dividere le comunicazioni con temi comuni:

- **Analisti**: contenente le comunicazioni relative all'Analisi dei Requisiti;
- **Colletta**: relative alle comunicazioni generali;
- **Norme di progetto**: contenente le comunicazioni relative al documento "Norme di progetto";
- **Random**: contiene informazioni relative a domande dei componenti che possono riguardare informazioni di carattere personale;

- **Verifica**: per le comunicazioni relative alle attività di verifica;
- **Link**: contenente tutti i link utili.

Un altro strumento utilizzato per la comunicazione interna è Hangouts, il quale permette di realizzare videochiamate di gruppo utilizzando il browser web. Pertanto, come detto in §5.1.1.1, Hangouts è lo strumento per la realizzazione di incontri interni anche in remoto. È delegato al capo progetto il compito di creare la chiamata di gruppo e aggiungere i vari componenti.

5.1.3.2 Esterna Per le comunicazioni esterne è stata creata una casella di posta elettronica .
Tale indirizzo deve essere l'unico canale di comunicazione esistente tra il gruppo di lavoro e l'esterno.
Come descritto nel capitolato d'appalto:

- Il Proponente può essere contattato in qualsiasi momento, solo dal *Responsabile di Progetto*, all'indirizzo e-mail , al quale risponde il reparto tecnologico dell'azienda;
- Le e-mail devono contenere in oggetto la sigla "UNIPD-SWE" e dovranno essere indirizzate all'attenzione di Giulio Paci, che sarà il referente principale;
- Ogni e-mail ricevuta su questo indirizzo viene inoltrata automaticamente alla casella personale di ciascun membro, tramite l'utilizzo di filtri di Gmail;
- In alternativa, per le comunicazioni ritenute urgenti è possibile telefonare al numero 0490998335.

5.1.4 Gestione delle responsabilità e ruoli

5.1.4.1 Introduzione
I ruoli di progetto rappresentano le figure professionali che lavorano al progetto. Ogni membro del gruppo, in un dato momento, dovrà ricoprire un determinato ruolo.
La rotazione è obbligatoria e si terrà conto delle preferenze personali di ognuno. Ogni membro deve rispettare il proprio ruolo e svolgere le attività assegnategli, secondo quanto stabilito nel *PianoDiProgetto_v3.0.0*.
Si avrà l'accortezza di evitare situazioni di conflitto di interesse come, ad esempio, essere *Verificatori* di documenti prodotti da sé stessi, compromettendone così la qualità.

5.1.4.2 Responsabile di progetto
Il *Responsabile di Progetto*, o Project Manager, è una figura necessaria alla gestione dell'intero progetto. Egli raccoglie su di sé le responsabilità decisionali di scelta e approvazione e costituisce il centro di coordinamento per l'intero progetto; in particolare, rappresenta il gruppo di lavoro nei confronti del Committente e del Proponente.
In particolare, ha l'incarico di:

- Organizzare incontri interni ed esterni;
- Pianificare le attività svolte dal gruppo;
- Individuare per ciascun compito un membro del gruppo per svolgerlo;
- Analizzare, monitorare e gestire i rischi.

5.1.4.3 Amministratore di Progetto
L'*Amministratore di Progetto* è la figura professionale che deve gestire l'ambiente di lavoro, al fine di aumentare l'efficienza e portare qualità.
Ha l'incarico di:

- Ricercare nuovi strumenti che migliorino l'efficienza;
- Gestire la documentazione di progetto;
- Occuparsi del controllo di versione del prodotto;

- Occuparsi della configurazione del prodotto.

5.1.4.4 Analista

L'*Analista* è la figura che svolge le attività di analisi al fine di comprendere appieno il dominio del problema.
Ha l'incarico di:

- Analizzare i requisiti del prodotto;
- Analizzare i requisiti di dominio;
- Redigere il documento Studio di Fattibilità;
- Redigere il documento Analisi dei Requisiti.

5.1.4.5 Progettista

Il *Progettista* è il responsabile delle scelte architetturali del progetto e ne influenza gli aspetti tecnici e tecnologici.
Partendo dalle attività dell'*Analista*, il *Progettista* ha il compito di trovare una possibile soluzione per i problemi e i requisiti precedentemente individuati.
Ha l'incarico di:

- Comprendere a fondo i requisiti nel documento Analisi dei Requisiti;
- Comprendere a fondo i requisiti nel documento Analisi dei Requisiti;
- Redigere la documentazione tecnica per il prodotto software;
- Redigere il documento Manuale Sviluppatore.

5.1.4.6 Programmatore

Il *Programmatore* è la figura che provvederà alla codifica della soluzione, studiata e spiegata dal *Progettista*.
Ha l'incarico di:

- Scrivere il codice del prodotto software che rispetti le decisioni del *Progettista*;
- Redigere il documento Manuale Utente.

5.1.4.7 Verificatore Il *Verificatore* è una figura presente per l'intero ciclo di vita del software e controlla che le attività svolte siano conformi alle attese e alle norme prestabilite.
Ha l'incarico di:

- Verificare che ciascuna attività svolta sia conforme alle norme stabilite nel progetto;
- Controllare che, per ogni stadio del ciclo di vita del prodotto, questo sia conforme al Piano di Qualifica.

5.1.5 Pianificazione

Il *Responsabile di progetto* decide le scadenze tenendo conto degli impegni lavorativi e scolastici di ogni membro. Stima i costi e le risorse necessarie, pianifica le attività e le assegna alle persone.
Il *Responsabile di progetto* deve definire il piano di progetto, in cui descrive attività e compiti utili/necessari all'esecuzione di processo.
Il piano deve:

- Fissare un tempo di completamento dei compiti;
- Dare una stima del tempo richiesto dei compiti;
- Adeguare le risorse necessarie per eseguire i compiti;
- Allocare i compiti;

- Assegnare le responsabilità;
- Analizzare i rischi;
- Definire delle metriche per il controllo della qualità;
- Associare dei costi al processo di esecuzione;
- Fornire un'infrastruttura.

5.1.6 Monitoraggio del piano

Il *Responsabile di progetto* supervisiona l'esecuzione del progetto fornendo report interni sulla progressione del processo.
Egli deve investigare, analizzare e risolvere i problemi scoperti durante l'esecuzione ed eventualmente può rivedere la pianificazione temporale per far fronte a tali problemi e a cambi di strategia.

- I report redatti, in conclusione di ogni periodo, confluiranno nel Piano di Qualifica;
- Al termine del periodo di riferimento, il *Responsabile di progetto* deve confrontare i risultati ottenuti con gli obiettivi prefissati e valutare strumenti, attività e processi impiegati per il completamento dei processi;
- La valutazione finale e il tracciamento di strumenti e tecnologie dovranno confluire nel Piano di Qualifica, eventuali rischi e piani di contingenza utilizzati dovranno essere riportati nel Piano di Progetto.

5.2 Gestione dell'infrastruttura

5.2.1 Introduzione

Inizializzare, implementare e gestire processi software, richiede l'istanziazione di un'infrastruttura ad hoc per l'intero progetto. In questa sezione verrà normato e descritto l'ambiente di lavoro e gli strumenti o software utilizzati.

5.2.2 Strumenti di condivisione

Un altro servizio utilizzato dal team è Google Drive, servizio web in ambiente cloud$_G$ di memorizzazione dati e sincronizzazione online. Questo servizio ha anche un tool aggiuntivo per l'integrazione di Slack, che permette al gruppo un rapido scambio di documenti.

5.2.3 Strumenti di gestione del lavoro

Per gestire al meglio il tracciamento e la gestione del lavoro il SWEight ha deciso di usare la web application Asana.

5.2.3.1 Project Board

Per la gestione delle issue$_G$ si fa utilizzo di project board$_G$ di Asana, che grazie ad un intuitiva dashboard, permette la gestione dei compiti. In particolare, trascinando la card$_G$ di uno specifico task da una colonna ad un'altra si andrà ad indicare in che stato si trova quella determinata issue. Le colonne, indicanti lo stato in cui si trova la issue, utilizzate all'interno del ciclo di vita dei documenti sono le seguenti:

- **To do**: indica che l'attività deve essere ancora svolta;
- **In progress**: indica che l'attività è in svolgimento;
- **Needs review**: indica che l'attività necessita di verifica;

- **Done**: indica che l'attività è conclusa.

5.2.3.2 Gestione delle milestone

Per la gestione delle milestone$_G$ è utilizzato il sistema di gestione offerto da Asana, che permette di associare le milestone ai task. Per la creazione di una milestone all'interno del progetto Colletta, si deve:

1. Selezionare il progetto su cui si lavora;
2. Cliccare su "new" e selezionare task;
3. Scrivere il nome del task e assegnarlo;
4. Andare alla sezione timeline;
5. Trascinare il task da "unschedule task" e posizionarlo nel periodo desiderato;
6. Restringere o allargare il task per modificare la durata;
7. Determinare le dipendenze trascinando la freccia all'interno di un altro task.

5.3 Miglioramento continuo dei processi

Processi, attività o compiti istanziati possono non essere definitivi. Se al termine di uno dei periodi di progetto, un membro del gruppo dovesse trovare difficoltà, oppure se trovasse procedure più efficienti ed efficaci di quelle in uso, può decidere di proporre al *Responsabile di Progetto* e crearne di nuovi o modificare quelli già esistenti.
Il *Responsabile di progetto* organizzerà un incontro con tutti i membri di SWEight e procederà ad esporre i nuovi spunti suggeriti, ascoltando i pareri di tutti.
La decisione finale spetta comunque al *Responsabile di progetto*, che dopo aver valutato l'impatto di qualsiasi modifica al Piano di Progetto ed aver controllato che ciò non comporti l'impossibilità nel raggiungere gli obiettivi prefissati nei tempi stabiliti, potrà decidere di attuarla.

5.4 Formazione

È il processo che fornisce e mantiene la formazione del personale. Acquisizione, supporto, sviluppo, esecuzione e mantenimento del prodotto software sono largamente dipendenti dalle conoscenze e dalle capacità dei membri del gruppo.
Per questo ognuno deve procedere in modo autonomo con lo studio individuale delle tecnologie che verranno utilizzate nel corso del progetto, prendendo come riferimento oltre al materiale indicato nella sottosezione Riferimenti normativi, anche la seguente documentazione:

- **Latex**: https://www.latex-project.org;
- **GitHub**: https://guides.github.com;
- **Git**: https://git-scm.com/doc;
- **Hangouts**: https://support.google.com/hangouts/?hl=en;
- **Texmaker**: http://www.xm1math.net/texmaker/doc.html;
- **Slack**: https://get.slack.help/hc/en-us;
- **Astah**: http://astah.net/manual;l
- **React tutorial**: https://www.youtube.com/watch?v=Ke90Tje7VS0&t=3425s;
- **Documentazione react**: https://reactjs.org/docs/getting-started.html;
- **Documentazione bootstrap**: https://getbootstrap.com/docs/4.3/getting-started/introduction/;
- **Documentazione freeling**: http://nlp.lsi.upc.edu/freeling/node/9;

- **Guida spring**: https://spring.io/guides;
- **Docker**: https://www.docker.com/why-docker;
- **Tomcat**: http://tomcat.apache.org/tomcat-8.5-doc/index.html;
- **MongoDB**: https://docs.mongodb.com/guides/.

A Lista di controllo

Durante l'applicazione del walkthrough$_G$ ai documenti, sono qui riportati gli errori più frequenti. Per migliorare l'efficienza e l'efficacia da parte dei *Verificatori* è opportuno che i controlli si basino sui seguenti punti:

- **Lingua italiana**:
  - La prima parola di una voce dell'elenco puntato inizia con la lettera maiuscola;
  - La voce finale dell'elenco puntato non termina con il punto;
  - Una voce non finale dell'elenco puntato non termina con il punto e virgola.
- **LaTeX**:
  - Date e ore scritte non rispettano il formato stabilito;
  - Mancato utilizzo dei comandi personalizzati;
  - Utilizzo scorretto delle parentesi graffe dopo i comandi LaTeX;
  - Mancato aggiornamento dell'intestazione del documento dopo una modifica;
  - Link e riferimenti non funzionanti o assenti.
- **UML**:
  - Casi d'uso non proporzionati correttamente tra loro;
- **Glossario**:
  - Termini segnati con il comando `{Parola}\ped{G}` ed impropriamente non presenti nel Glossario;
  - Mancata segnatura di termini presenti nel Glossario.
- **Nomi dei documenti**:
  - Mancata indicazione della versione di riferimento di un documento.

B Ciclo di Deming

Ogni processo deve essere organizzato basandosi sul Ciclo di Deming.

- **Pianificazione**: stabilire gli obiettivi e i processi necessari per fornire risultati in accordo con i risultati attesi;
- **Eseguire**: vengono implementate le attività secondo le linee definite durante la fase pianificazione;
- **Valutare**: viene verificato l'esito delle azioni di miglioramento rispetto alle attese;
- **Act**: azione per rendere definitivo e/o migliorare il processo. Richiede azioni correttive sulle differenze significative tra i risultati effettivi e previsti.

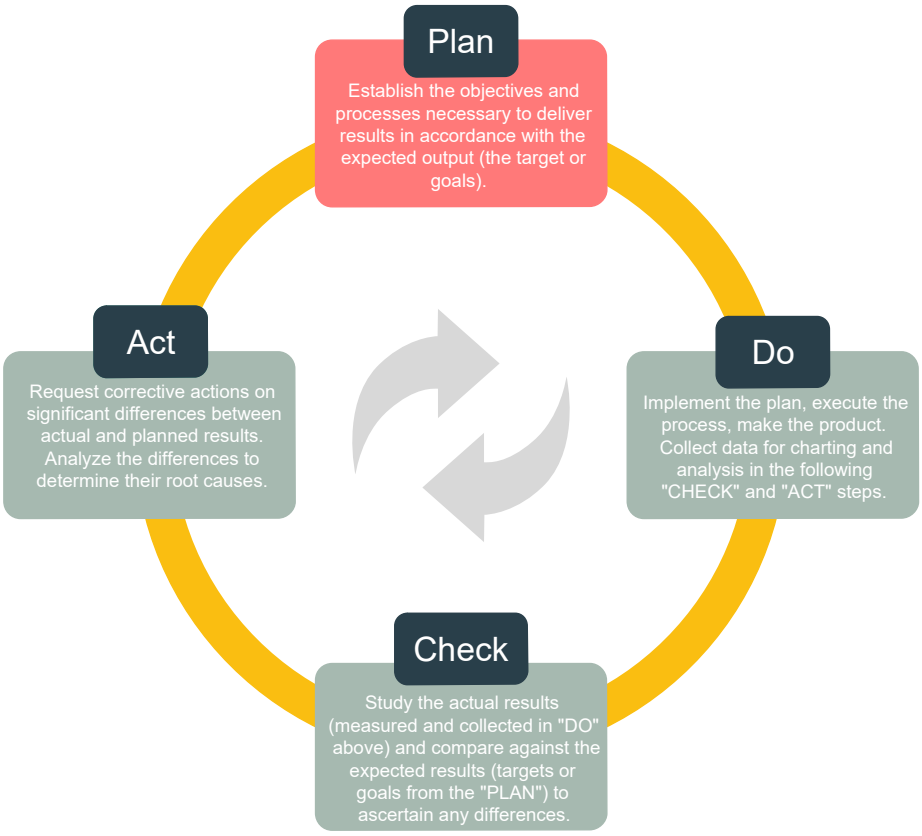

Figura 11: Ciclo di Deming

C Qualità

C.1 SPICE

Il modello ISO/IEC 1554, meglio conosciuto come SPICE (Software Process Improvement and Capability Determination), è lo standard di riferimento per valutare in modo oggettivo la qualità dei processi nello sviluppo del software.
Sono definiti:

- Una serie di sei livelli utilizzati per classificare la capacità e la maturità del processo software. Ogni livello è caratterizzato dal soddisfacimento degli attributi associati:
  1. **Incompleto**: il processo non è stato implementato o non ha raggiunto il successo desiderato;
  2. **Eseguito**: il processo è implementato e ha realizzato il suo obiettivo (conformità); Attributi:
     - Process performance;
  3. **Gestito**: il processo è gestito e il prodotto finale è verificato, controllato e manutenuto (affidabilità); Attributi:
     - Performance management;
     - Work product management;
  4. **Stabilito**: il processo è basato sullo standard di processo (standardizzazione); Attributi:
     - Process definition;
     - Process deployment;
  5. **Predicibile**: il processo è consistente e rispetta limiti definiti (strategico); Attributi:
     - Process measurement;
     - Process control;
  6. **Ottimizzato**: il processo segue un miglioramento continuo per rispettare tutti gli obiettivi di progetto; Attributi:
     - Process innovation;
     - Process optimization;

  Ogni attributo riceve una valutazione nella seguente scala, andando a definire il rispettivo livello di capacità del processo:
  - **N:** non raggiunto (0 - 15%);
  - **P:** parzialmente raggiunto (>15% - 50%);
  - **L:** largamente raggiunto (>50%- 85%);
  - **F:** pienamente raggiunto (>85% - 100%);
- Delle linee guida per effettuare delle **stime**, eseguite tramite:
  - **Processi di misurazione**, descritti nel *PianoDiProgetto_v3.0.0*;
  - **Modello di misurazione**, descritto in questo documento;
  - **Strumenti di misurazione**, descritti nelle *NormeDiProgetto_v3.0.0*;
- Una serie di **competenze** che chi effettua misurazioni deve possedere. La mancanza di esperienza degli elementi del gruppo *SWEight*, fa sì che nessun membro possieda queste skill, rendendo così impossibile la piena adesione allo standard. Tuttavia, ogni componente è chiamato a studiare SPICE e a applicare al meglio le indicazioni descritte in questo documento e nelle *NormeDiProgetto_v3.0.0*, al fine di perseguire un livello di qualità accettabile.

C.2 ISO/IEC 9126

Lo standard ISO/IEC 9126 stabilisce una serie di linee guida mirate al miglioramento delle qualità del software sviluppato.

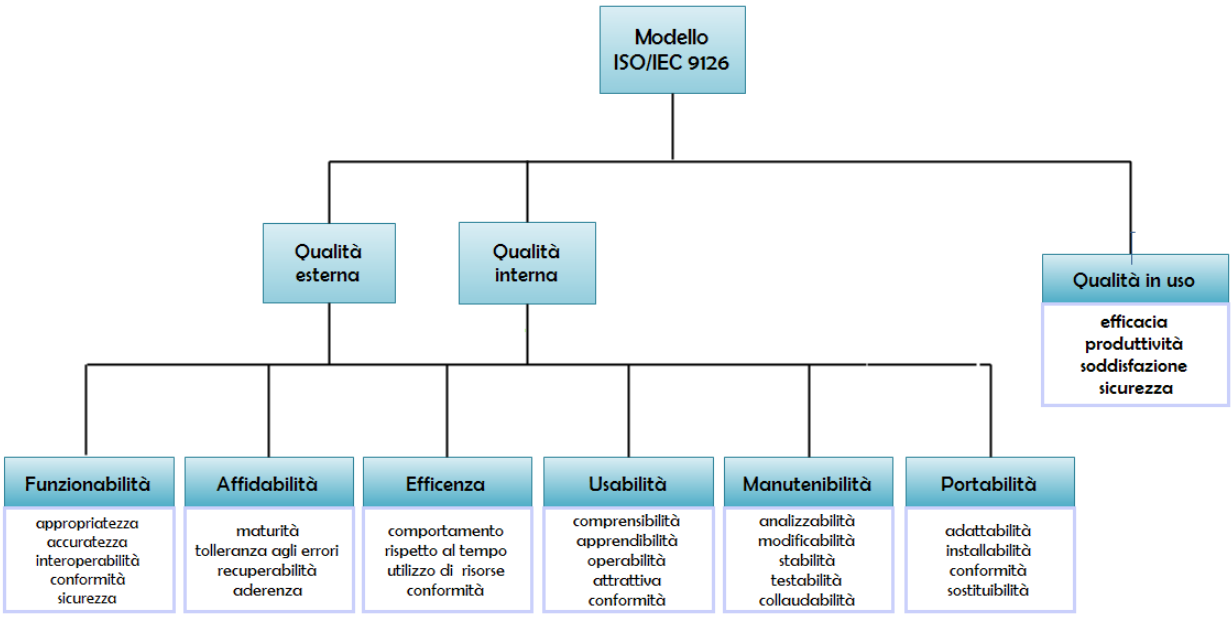

Figura 12: Rappresentazione grafica di ISO/IEC 9126 [Wikipedia]

Come presentato in Figura 12, ISO/IEC 9126 prescrive indicazioni su:

- **Qualità interna:** è misurata sul software non eseguibile, come, ad esempio il codice sorgente. Le misure effettuate permettono di avere una buona previsione della qualità esterna;
- **Qualità esterna:** è misurata tramite l'analisi dinamica su software eseguibile. Le misure effettuate permettono di avere una buona previsione della qualità in uso prodotto;
- **Qualità in uso:** definita in base all'esperienza utente. Sono da perseguire i seguenti obiettivi:
  - Efficacia;
  - Produttività;
  - Soddisfazione;
  - Sicurezza.

ISO/IEC 9126 definisce inoltre una serie di requisiti da soddisfare per produrre software di qualità:

- **Funzionalità:** capacità di un prodotto software di soddisfare le esigenze stabilite (vedi *AnalisiDeiRequisiti_v3.0.0*);
- **Affidabilità:** capacità di un prodotto di mantenere un determinato livello di prestazioni in date condizioni d'uso per un certo periodo;
- **Efficienza:** capacità di un prodotto software di eseguire il proprio compito minimizzando il numero di risorse usate;
- **Usabilità:** capacità del prodotto software di essere utilizzato, capito e studiato dall'utente a cui è rivolto;

- **Manutenibilità:** capacità del prodotto software di evolvere mediante modifiche, correzioni e miglioramenti;
- **Portabilità:** capacità del prodotto software di funzionare ed essere installato su più ambienti hardware e software.