MaaS - MongoDB as an admin Service

matrioska.io.go@gmail.com

Norme di Progetto V3.0.0

Nome del documento	Norme di Progetto
Versione del Documento	3.0.0
Data Creazione	05/03/2016
Redazione	Petrov Andrei, Zamberlan Sebastiano
Verifica	D'Amico Roberto
Approvazione	Santi Guido
Uso	*Interno*
Destinatari	*matrioska.io*

Registro delle modifiche

Versione	Autore	Ruolo	Data	Descrizione
3.0.0	Santi Guido	Responsabile	14/08/2016	Approvazione
2.2.0	D'Amico Roberto	Verificatore	13/08/2016	Verifica finale
2.1.0	Zamberlan Sebastiano	Amministratore	16/07/2016	Aggiunta la sezione Codifica e le sue sottosezioni
2.0.0	D'Amico Roberto	Responsabile	11/05/2016	Approvazione
1.9.2	Zamberlan Sebastiano	Verificatore	11/05/2016	Verifica Finale del documento
1.9.1	Berselli Marco	Verificatore	10/05/2016	Verifica del documento trovati errori di punteggiatura
1.9.0	Petrov Andrei	Amministratore	09/05/2016	Aggiunto il paragrafo Dettagli Script ai fini di Verifica e Gestione di Documentazione
1.8.3	Santi Guido	Verificatore	09/05/2016	Trovati errori non gravi nei paragrafi Analisi dei Requisiti, Specifica Tecnica, Definizione di Prodotto, Strategia di Progettazione, Test di unità, Test di integrazione, Tracy e Visual Studio Code
1.8.2	Petrov Andrei	Amministratore	08/05/2016	Correzione di errori nei paragrafi Incontri esterni, Ticketing e Verbali trovata la parola proponente senza la prima lettera maiuscola
1.8.1	Petrov Andrei	Amministratore	07/05/2016	Correzione di errori mediante lo script creato appositamente. Trovati nei paragrafi Ruolo di Progetto e Analista la prola Committenteo anzichè Committente.
1.8.0	Petrov Andrei	Amministratore	06/05/2016	Aggiunta sottosezione riguardante ZenHub.io nelle sezione strumenti Processi di Supporto

Versione	Autore	Ruolo	Data	Descrizione
1.7.0	Petrov Andrei	Amministratore	06/05/2016	Aggiunta appendice riguardante script introdotti per l'attività di verifica
1.6.0	Petrov Andrei	Amministratore	05/05/2016	Aggiunto Visual Studio Code alla sezione Strumenti del Processo Primario
1.5.0	Petrov Andrei	Amministratore	05/05/2016	Aggiunti punti per la lista di controllo
1.4.6	D'Amico Roberto	Verificatore	04/05/2016	Integrazione errori di spelling frequenti nella Lista di Controllo
1.4.5	D'Amico Roberto	Verificatore	03/05/2016	Verifica spelling parole mediante script
1.4.4	D'Amico Roberto	Verificatore	03/05/2016	Verifica di dettaglio Processi di Supporto
1.4.3	D'Amico Roberto	Verificatore	03/05/2016	Verifica di dettaglio Processi Primari
1.4.2	D'Amico Roberto	Verificatore	02/05/2016	Integrazione lista di controllo
1.4.1	D'Amico Roberto	Verificatore	01/05/2016	Controllo contenuto aggiunto
1.4.0	Petrov Andrei	Amministratore	01/05/2016	Aggiunti informazioni sugli script nella sezione strumenti Processi di Supporto
1.3.0	Petrov Andrei	Amministratore	30/04/2016	Aggiunta sezione Verifica e Metriche per i processi e prodotti nel Processo di Verifica
1.2.0	Petrov Andrei	Amministratore	30/04/2016	Aggiunta sezione Lista comandi Git in Appendice
1.1.5	Petrov Andrei	Amministratore	30/04/2016	Aggiunta sezione Formulario UML in Appendice per raccogliere gli elementi base del linguaggio
1.1.4	D'Amico Roberto	Progettista	29/04/2016	Correzione errori riscontrati in verifica
1.1.3	Zamberlan Sebastiano	Verificatore	29/04/2016	Verifica delle sezioni aggiunte internamente alla sezione Attività Processo Sviluppo

Versione	Autore	Ruolo	Data	Descrizione
1.1.2	Petrov Andrei	Amministratore	28/04/2016	Aggiunti gli strumenti WebStorm e Atom.io alla sezione Strumenti Processo Sviluppo
1.1.1	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Programmazione internamente alla sezione Norme Processo Sviluppo
1.1.0	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Condivisione su GitHub internamente alla sezione Norme Processo Sviluppo
1.0.9	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Nomi File internamente alla sezione Norme Processo Sviluppo
1.0.9	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Definizione di classe internamente alla sezione Norme Processo Sviluppo
1.0.8	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Gestione Flusso di Lavoro su GitHub internamente alla sezione Procedure Processo Sviluppo
1.0.7	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Approccio di Progettazione internamente alla sezione Attività Processo Sviluppo
1.0.6	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Strategia di Progettazione internamente alla sezione Attività Processo Sviluppo
1.0.5	Petrov Andrei	Amministratore	28/04/2016	Aggiunta la sezione Progettazione internamente alla sezione Attività Processo Sviluppo
1.0.4	Santi Guido	Progettista	21/04/2016	Modifica a Processi di Supporto: miglioramento dello stile espositivo
1.0.3	Santi Guido	Progettista	21/04/2016	Modifica a Processi Primari: miglioramento dello stile espositivo

Versione	Autore	Ruolo	Data	Descrizione
1.0.2	Berselli Marco	Amministratore	20/04/2016	Modifica del paragrafo riguardante LaTeX. Inserimento del limite di larghezza sulle righe
1.0.1	Santi Guido	Progettista	20/04/2016	Correzione degli errori riscontrati in RR: Inserimento Processi Organizzativi
1.0.0	Petrov Andrei	Responsabile	19/03/2016	Approvazione del documento
0.5.3	Berselli Marco	Verificatore	18/03/2016	Verifica finale del documento
0.5.2	Santi Guido	Amministratore	15/03/2016	Correzione errori ortografici
0.5.1	Berselli Marco	Verificatore	15/03/2016	Verifica del documento
0.5.0	D'Amico Roberto	Amministratore	14/03/2016	Inserimento Processi Organizzativi
0.4.2	Santi Guido	Amministratore	14/03/2016	Correzione errori ortografici
0.4.1	Berselli Marco	Verificatore	11/03/2016	Verifica del documento
0.4.0	D'Amico Roberto	Amministratore	11/03/2016	Inserimento Processi di Supporto
0.3.0	Santi Guido	Amministratore	10/03/2016	Inserimento Processi Primari
0.2.0	Santi Guido	Amministratore	07/03/2016	Inserimento dell'introduzione
0.1.0	Santi Guido	Amministratore	07/03/2016	Stesura dello scheletro del documento

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il seguente documento definisce tutte le norme che devono essere seguite dai membri del *team*$_g$ durante lo sviluppo del *progetto*$_g$ *MaaS*$_g$. Tali norme applicate rigorosamente garantiscono la piena coerenza nei vari aspetti del *progetto*$_g$, anche in quelli sviluppati da più persone. Nel dettaglio sono specificate le regole con le quali:

- Progettare e sviluppare il prodotto *software*$_g$;
- Produrre documentazione ufficiale;
- Eseguire le opportune verifiche;
- Organizzare il lavoro e la spartizione dei ruoli fra i membri del *team*$_g$.

1.2 Scopo del prodotto

L'obiettivo del prodotto *software*$_g$ è adattare il *framework*$_g$ *MaaP*$_g$ (MongoDB as an admin Platform), sviluppato dal gruppo SteakHolders durante l'attività accademica di *progetto*$_g$ nel corso di Ingegneria del Software dell'a.a. 2013/2014, per offrire il prodotto come *servizio web*$_g$.

1.3 Glossario

Con il presente viene consegnato anche un *"Glossario v3.0.0"* con lo scopo di facilitare la lettura dei documenti formali. Ogni acronimo o termine tecnico accompagnato con una g a pedice sarà quindi integrato con ulteriori informazioni da ricercarsi nel suddetto Glossario.

1.4 Riferimenti

1.4.1 Normativi

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

2 Processi Primari

2.1 Processi di Sviluppo

In questa sezione vengono esposti i procedimenti che il *team*$_g$ deve seguire al fine di realizzare il prodotto software. Ciò comprende tutta la parte di pre-analisi, la progettazione e la realizzazione.

2.1.1 Attività

2.1.1.1 Studio di Fattibilità

Prima dell'inizio della realizzazione di un *progetto*$_g$, il *team*$_g$ deve valutare attentamente le specifiche richieste. In particolare, il Responsabile di Progetto deve incaricare uno o più analisti con il compito di redigere lo *Studio di Fattibilità*. Questo è il primo documento che deve essere creato, ancora prima di cominciare la successiva *Analisi dei Requisiti*. Esso deve contenere uno studio approfondito del dominio applicativo e tecnologico del *progetto*$_g$ e un'attenta valutazione dei rischi basata sulla conoscenza da parte degli analisti delle capacità e punti deboli del proprio *team*$_g$.

2.1.1.2 Analisi dei Requisiti

Dopo aver svolto lo Studio di Fattibilità è compito degli analisti produrre l'*Analisi dei Requisiti* interpretando sia il *capitolato*$_g$ d'appalto sia le informazioni raccolte dagli incontri con il *Proponente*$_g$. Questo documento deve raccogliere tutti i casi d'uso e i requisiti del *software*$_g$ cercando di renderli di facile comprensione e ordinandoli secondo lo standard. Per rendere questa attività più automatizzata il gruppo utilizza il software Tracy, al cui interno andranno inseriti tutti i requisiti e dei casi d'uso trovati.

2.1.1.3 Progettazione

L'*Analisi dei Requisiti* produce la Specifica dei Requisiti, un documento che raccoglie le esigenze, i vincoli e le funzionalità individuate dagli analisti. Sulla base di tale documento i progettisti possono iniziare l'analisi di dettaglio e avanzare scelte progettuali più vicine alla realizzazione del prodotto software.

2.1.1.3.1 Specifica Tecnica

I Progettisti devono progettare l'architettura di alto livello del prodotto software e delle singole componenti. Inoltre, devono provvedere alla progettazione di opportuni test d'integrazione.

- ***Diagrammi UML*$_g$**: facendo riferimento allo standard UML 2.0
  - *Diagrammi delle classi*$_g$: i diagrammi delle classi devono essere presenti sia per l'architettura di alto livello che di dettaglio;
  - *Diagrammi dei package*$_g$: devono essere presenti sia per l'architettura di alto livello che di dettaglio. Sono fondamentali per la definizione di moduli in ambiente Javascript;

  - *Diagrammi di attività*$_g$: i diagrammi di attività devono essere presenti sia per l'architettura di alto livello che di dettaglio. Sono un ottimo strumento per studiare l'interazione delle parti ad alto livello;
  - *Diagrammi di sequenza*$_g$: nel caso in cui un'unità progettuale sia complessa è necessario fornire il relativo diagramma di sequenza per semplificare al programmatore la comprensione del contesto.

- ***Design Patterns***$_g$: devono essere presenti durante il periodo di progettazione ed implementazione. Sono un ottimo strumento per raccogliere l'esperienza e la conoscenza da un lato della comunità di sviluppo e dall'altro del gruppo di sviluppo;

- **Tracciamento componenti**

  - Tracciamento Requisito-Componente.

- ***Test di Integrazione***$_g$: devono essere presenti durante fase di codifica. Sono necessari per integrare le componenti sviluppate in isolamento.

2.1.1.3.2 Definizione di Prodotto

I Progettisti devono produrre la *Definizione di Prodotto* dove viene descritta la progettazione di dettaglio del sistema ampliando quanto scritto nella *Specifica Tecnica*. Lo scopo di questo documento è quello di definire dettagliatamente ogni singola unità di cui è composto il sistema in modo da semplificare l'attività di codifica e allo stesso tempo di non fornire alcun grado di libertà al Programmatore. Inoltre devono essere progettati i relativi test di unità che vengono descritti nel *Piano di Qualifica*.

- **Diagrammi *UML***$_g$:

  - *Diagrammi delle classi*$_g$;
  - *Diagrammi di attività*$_g$;
  - *Diagrammi di sequenza*$_g$.

- **Classe-Requisito**:

  - La classe deve essere associata ad uno o più requisiti.

- **Test di Unità**:

  - Ogni classe deve specificare la batteria di test per testare i propri metodi;
  - I test devono essere significativi;
  - I test devono rilevare la presenza di problemi.

2.1.1.3.3 Strategia di Progettazione

Il gruppo ha fissato come strategia di progettazione la *Progettazione Orientata agli Oggetti*$_g$. Questa strategia pone l'attenzione sulle entità e le loro caratteristiche rispetto alle funzioni coinvolte nel sistema software.
I concetti più importanti della Progettazione Orientata agli Oggetti sono:

- Oggetto: ogni entità che contribuisce alla soluzione;
- Classe: descrizione deneralizzata di un oggetto;
- Incapsulamento: raggruppamento dei dati e delle operazioni sui dati sotto un comune spazio di nomi;
- Ereditarietà: scomposizione gerarchica delle classi, affinché ogni classe abbia un'unica responsabilità;
- Polimorfismo: comportamento osservabile a Run-Time di un oggetto. Un oggetto può avere più forme.

2.1.1.3.4 Approccio di Progettazione

È stato deciso di utilizzare un approccio top-down per la progettazione di alto livello e un approccio bottom-up per la progettazione di dettaglio.
Vantaggi del approccio top-down:

- Scomposizione del problema: il problema iniziale viene suddiviso in sotto-insiemi;
- Si inizia la progettazione con un modello generico;
- È ideale nel caso in cui un sistema deve essere ricostruito dall'inizio.

Vantaggi del approccio bottom-up:

- Integrazione di componenti singole per comporre componenti di più alto livello;
- Favorisce il riuso.

I due approcci vengono utilizzati in modo misto per favorire la flessibilità di progettazione e agevolare la comprensibilità del dominio applicativo.

2.1.1.3.5 Test di unità

I *Progettisti* devono definire i test d'unità utili alla verifica delle unità atomiche di progettazione, unità assegnabili al singolo *programmatore* per la loro implementazione. Infatti ogni unità deve essere corretta rispetto alla sua specifica.

2.1.1.3.6 Test di integrazione

I *Progettisti* devono definire i test di integrazione per verificare che la comunicazione tra i moduli sia corretta, dove necessario. Se la comunicazione tra due o più moduli non è corretta si deve riprendere la progettazione di alto livello e correggere gli errori riscontrati.

2.1.1.3.7 Test di sistema

Questi test devono verificare che ogni requisito funzionale sia sia stato implementato nel prodotto software sviluppato.

2.1.1.4 Codifica

Questa attività ha come scopo quello di implementare il codice derivante dalle attività di progettazione di alto e basso livello, ovvero di quanto affermato all'interno della Specifica Tecnica e della Definizione di Prodotto.
Oltre a questo è necessaria la stesura di un'opportuna documentazione sul codice per assicurarne la comprensibilità e manutenibilità.

2.1.1.4.1 Formattazione del codice

Per quanto riguarda la formattazione del codice javascript il gruppo ha scelto di seguire le norme proposte dal progetto *jQuery*. Questa scelta è stata motivata dalla facilità di lettura e dalla possibilità di automatizzare la formattazione del codice mediante il programma *JSHint*. La pagina di riferimento per tali norme è la seguente: http://contribute.jquery.org/style-guide/js/.

2.1.1.4.2 Versionamento del codice

La versione del codice dovrà comparire esclusivamente all'interno dell'apposita sezione nell'intestazione del file, e dovrà essere conforme al formato:

X.Y

dove:

- **X**: indice di versione principale. Indica la versione di rilascio del codice. Questo indice viene aggiornato quando il codice raggiunge una nuova versione stabile. Un aggiornamento di questo indice necessita l'azzeramento del valore di Y;
- **Y**: indice di versione secondario. Questo indice viene aggiornato dopo modifiche rilevanti nel file.

L'avanzamento di versione di un file deve corrispondere al raggiungimento di determinati obbiettivi di stabilità e/o funzionalità del codice. La versione 1.0 dovrà corrispondere alla prima versione del file completo, stabile e funzionante. Sarà la prima versione che potrà essere sottoposta ai necessari test per verificarne l'effettivo funzionamento delle funzionalità.

2.1.1.4.3 Intestazione file di codice

Ogni file di codice dovrà iniziare con una intestazione nel seguente formato:

```
/*
* Name : [ Nome del file ]
```

```
* Package: [ Nome del package ]
*
* History:
* Version Date Programmer
* ================================================
* 1.2 gg/mm/aaaa [ Nome Cognome ]
* [ Description ]
* ------------------------------------------------
* 1.1 gg/mm/aaaa [ Nome Cognome ]
* [ Description ]
* ------------------------------------------------
* ...
* ================================================
*/
```

dove:

- **Name** è il nome del file, compresa anche l'estensione;
- **Package** è il nome del package all'interno del quale si trova il file corrente;
- **History** è il diario delle modifiche del file. Ogni modifica è descritta mediante i seguenti campi:
  - **Version** è la versione del file;
  - **Date** è la data della modifica;
  - **Programmer** è il nome del programmatore che ha effettuato la modifica;
  - **Description** è una breve descrizione che spiega le modifiche apportate nella modifica.

2.1.2 Procedure

2.1.2.1 Flusso di Lavoro su GitHub

Per poter contribuire in modo flessibile al lavoro di tutti ogni sviluppatore deve rispettare le norme fissate e seguire la seguente procedura:

- **Creare un nuovo *branch*$_g$**;
- **Aggiungere un *commit*$_g$**;
- ***Pull Request*$_g$**;
- **Discussione e/o Revisione**;
- ***Merge*$_g$**.

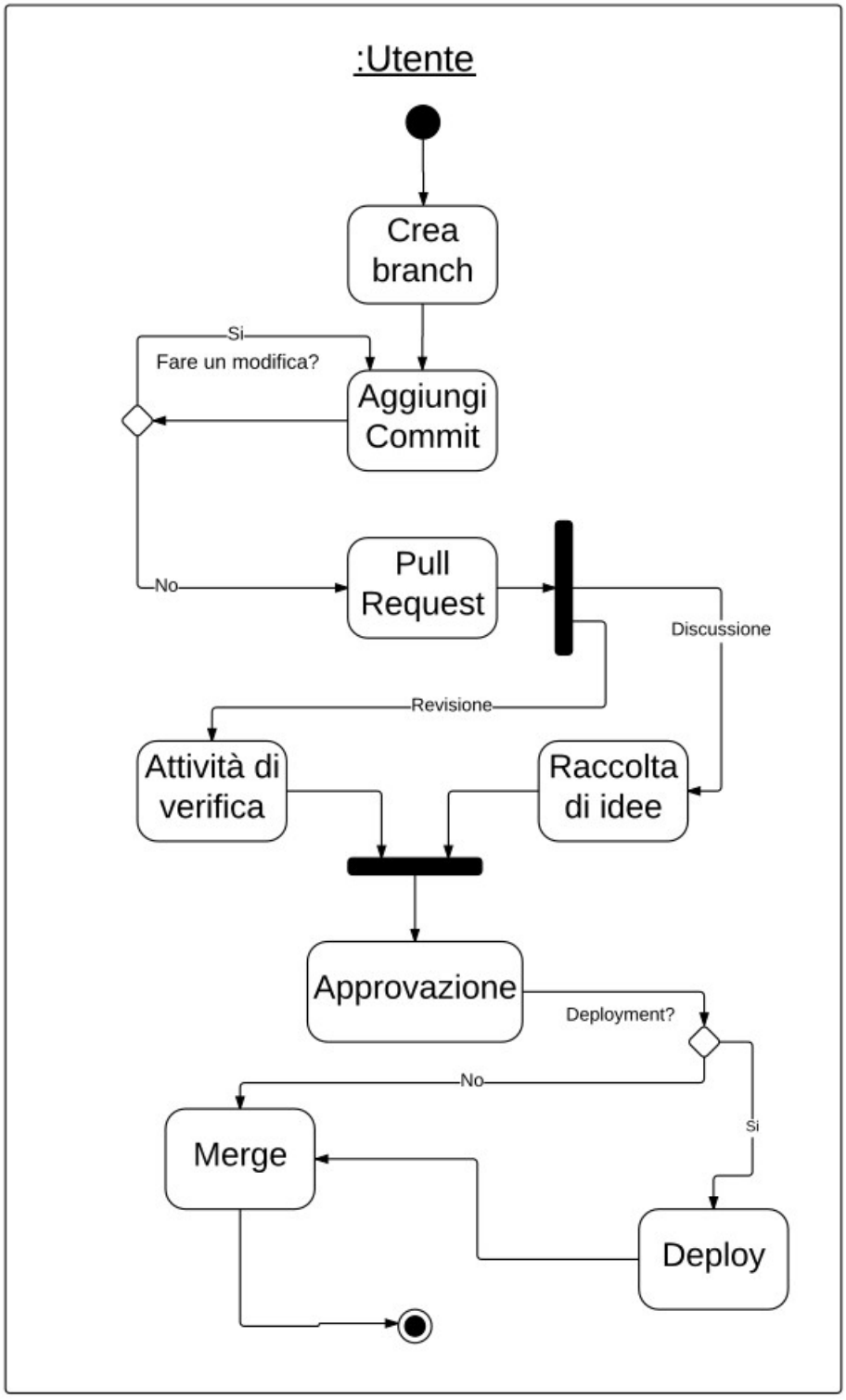

Fig 1: Come contribuire alla repository del gruppo di lavoro

2.1.3 Norme

2.1.3.1 Composizione dei Casi d'Uso

I casi d'uso vengono rappresentati in modo univoco con la seguente codifica:

UC[Codice]

Codice: Sequenza numerica, con i numeri divisi da punti, necessaria a definire la gerarchia (esempio: 2.1.3).

Ogni *Caso d'Uso*$_g$ deve essere descritto dalle seguenti informazioni:

- **Titolo:** argomento trattato;
- **Diagramma UML:** necessario in caso di scenari complessi;
- **Attori Coinvolti:** lista degli attori con un ruolo nel *caso d'uso*$_g$;
- **Scopo e descrizione:** dettagli aggiuntivi su quale aspetto del software stiamo trattando;
- **Pre-Condizione:** condizione necessaria affinché il *caso d'uso*$_g$ abbia luogo;
- **Flusso principale:** sezione in cui si descrive il flusso dei casi d'uso figli;
- **Scenari alternativi:** se presenti, descrivono gli eventi in casi diversi da quello previsto;
- ***Estensioni e/o inclusioni*$_g$:** elenco di tutte le possibili *estensioni*$_g$ e di eventuali casi d'uso già descritti altrove;
- **Post-Condizione:** condizione finale sempre vera raggiunta al termine del *caso d'uso*$_g$.

2.1.3.2 Composizione dei Requisiti

I requisiti vengono rappresentati in modo univoco con la seguente codifica:

R-[Importanza][Tipo][ID]

- **Importanza:** indica se il requisito è:
  1. requisito desiderabile;
  2. requisito opzionale;
  3. requisito obbligatorio.
- **Tipo:** indica se il requisito è di tipo:

  - F: funzionale;
  - Q: di qualità;
  - P: prestazionale;
  - V: di vincolo.

- **Identificativo:** è un valore numerico che identifica univocamente ogni requisito.

Inoltre un requisito è accompagnato da:

- Una descrizione sommaria;
- Se è presente, il requisito padre dal quale dipende;
- Un valore che indica se è stato effettivamente implementato o no.

2.1.3.3 Definizione di classe

Ogni definizione di classe deve comprendere i seguenti punti:

- **Scopo**: definizione del contesto di modellazione, qual è l'obiettivo;
- **Funzionalità modellate**: una classe offre la possibilità di creare modelli. Questi istanziati danno vita ad oggetti che risolvono un certo problema;
- **Elenco**:
  - **Dei metodi**: operazioni sui campi dati propri della classe;
  - **Degli attributi**: campi dati incapsulati internamente alla classe.

2.1.3.4 Programmazione

La programmazione è una attività individuale; come tale è facile osservare che ogni programmatore ha il proprio stile di programmazione. Il lavoro di gruppo comporta standardizzazione delle attività comuni. Infatti, nel dominio della standardizzazione rientra anche lo stile di programmazione. Per stile si intende l'aspetto del codice prodotto e non lo specifico *paradigma di programmazione*$_g$.
Il *Programmatore* deve rispettare le seguenti convenzioni:

- Identazione: le regole di identazione sono reperibili al seguente indirizzo URL `http://javascript.crockford.com/code.html`;
- I nomi delle variabili, classi, funzioni e metodi devono essere in inglese e senza underscore;
- I commenti devono essere in inglese;
- I nomi delle classi devono avere la prima lettera maiuscola;
- I nomi di variabili, metodi e funzioni devono avere la prima lettera minuscola e le successive iniziali delle parole in maiuscolo;

2.1.3.5 Nomi File

I nomi dei file offre un criterio di standardizzazione interno. Ogni tipologia di file viene caratterizzata da un proprio sistema di nomenclatura.

- **File *Bash*$_g$**
  - Nome tutto in minuscolo;
  - Estensione sh.
- ***Makefile*$_g$**
  - Prima lettera in Maiuscolo e il resto in minuscolo;
  - Nessuna estensione.
- **File Tex**
  - Usare la notazione CamelCase ad esempio MioFile.tex;
  - Vanno evitati qualsiasi tipo di separatori nel nome dei file;
  - **Eccezione**: È possibile specificare caratteri di separazione, internamente al nome dei file se e solo se il file è un verbale (esempio: QuestoVerbale11-11-11.tex). La ragione dell'eccezione è che il nome del verbale comprende anche la data di creazione.

2.1.3.6 Condivisione su GitHub

Per agevolare lo sviluppo e facilitare la comunicazione vengono prese in considerazione le *best practices*$_g$ consigliate da GitHub.
Il *workflow*$_g$ del team è orientato al branch e come modello di gestione viene assunto il modello a repository condivisa.
Devono essere rispettate le seguenti convenzioni:

- **Nome branch**: chiaro ed esplicativo;
- **Organizzazione della directory**: rispecchiare l'assetto iniziale;
- **Nomi File**: aderire alle convenzioni prefissate;
- **Conflitti Merge**: evitare, optare per Pull Request;
- **Merge**: consentito solo in seguito a revisione.

2.1.4 Strumenti

2.1.4.1 Tracy

Tracy è un *software*$_g$ per la gestione di requisiti, casi d'uso, attori e offre funzionalità di esportazione in codice LaTeXdegli elementi prima citati. Il sistema permette quindi di strutturare le gerarchie necessarie al *processo*$_g$ di analisi dei requisiti, fornendo un ambiente stabile che reagisce bene ad eventuali ripensamenti e cancellazioni di funzioni. Il tutto si basa su un *database*$_g$ SQL che garantisce l'integrità dei dati.

2.1.4.2 Lucidchart

Lucidchart è uno strumento sviluppato per la creazione di diagrammi UML. Il *software*$_g$ è accessibile via *web*$_g$ e quindi non richiede il download ne l'installazione. Inoltre permette a diversi utenti di lavorare in contemporanea su un unico *file*$_g$ attraverso un *browser-web*$_g$.

2.1.4.3 WebStorm

WebStorm è una *IDE*$_g$ Javascript, multi-piattaforma sviluppato da JetBrains. Informazioni dettagliate sono disponibili visitando l'*URL*$_g$ https://www.jetbrains.com/webstorm/. Come versione del prodotto software viene accettata ogni versione di WebStorm rilasciata da febbraio 2016 in poi.

2.1.4.4 Atom.io

Atom.io è un *editor di testo*$_g$ semplice e facilmente personalizzabile, sviluppato e manutenuto da *GitHub*$_g$. Informazioni dettagliate sono disponibili visitando l'URL https://Atom.io/.

2.1.4.5 Visual Studio Code

Visual Studio Code (VSC) è un editor di testo open source sviluppato dalla Microsoft. VSC è uno strumento flessibile e caratterizzato da un sistema integrato di debugging di applicazioni basate sulla piataforma NodeJS. Per ulteriori informazioni visitare il sito web https://code.visualstudio.com/.

2.1.4.6 JSHint

JSHint è un programma che aiuta a controllare che il codice javascript rispetti le regole dichiarate nella sezione 2.1.1.4.1. Per ulteriori informazioni visitare il sito web http://jshint.com/.

3 Processi di supporto

3.1 Processo di documentazione

In questa sezione vengono dettati gli standard riguardanti la produzione dei documenti formali necessari al buon procedimento del progetto.

3.1.1 Procedure

3.1.1.1 Ciclo di vita dei documenti

Quando arriva il momento di produrre della documentazione, sia essa necessaria per una revisione o semplicemente per formalizzare dei concetti, il Responsabile di Progetto incarica uno o più membri del *team*$_g$ di procedere con la stesura. Quando il documento è completo, un verificatore che non deve aver contribuito alla sua produzione effettua le opportune verifiche attenendosi alle *Norme di Progetto*. In caso di anomalie il documento torna in lavorazione, se invece risulta conforme esso passa all'attenzione del Responsabile di Progetto che ha due possibilità:

- Respingerlo, facendo ritornare il documento in lavorazione e ricominciando il *processo*$_g$;
- Approvarlo, assumendosi la responsabilità dei contenuti e rendendolo formale.

Una volta approvato il documento, viene definito formale e può quindi essere distribuito in modo ufficiale.

3.1.2 Norme

3.1.2.1 Norme Tipografiche

3.1.2.1.1 Stile del testo

Il testo deve essere chiaro e leggibile senza essere banale o eccessivamente ripetitivo, il *team*$_g$ deve inoltre mantenere un indice di leggibilità Gulpease conforme. Vanno inoltre evitati i tempi futuri come dovrà o sarà preferendo il tempo presente per cosa va fatto e il passato per cosa è stato fatto. È quindi consigliabile utilizzare le seguenti espressioni quando si ha bisogno di esprimere un concetto:

- **Deve**, **Deve essere** per tutte le occasioni di tipo MUST;
- **Dovrebbe**, **È preferibile** per tutte le occasioni di tipo SHOULD;
- **Potrebbe** per tutte le occasioni di tipo MAY.

I seguenti stili vanno inoltre adottati secondo le seguenti modalità:

- **Grassetto:** va applicato ai titoli dei capitoli e agli elementi salienti di un elenco. Viene inoltre usato per mettere in risalto singole parole o concetti;

- **Corsivo:** viene utilizzato per indicare *percorsi*$_g$ di cartelle, *file*$_g$ e nomi dei documenti, per riportare citazioni a fonti esterne e per identificare al meglio i termini del glossario (in questo caso il termine viene concluso con la lettera 'g' a pedice);
- **Maiuscolo:** solo gli acronimi possono avere tutti i caratteri in maiuscolo.

3.1.2.1.2 Punteggiatura

- **Maiuscolo:** solo la prima lettera dopo un punto semplice, interrogativo o esclamativo deve sempre essere maiuscola, ad eccezione di nomi, sigle e altri casi che richiedano il contrario;
- **Punteggiatura:** punti e virgole sono indispensabili per dare al testo ritmo e leggibilità. Si raccomanda di non dimenticare le regole della lingua che si sta utilizzando;
- **Spazi:** è buona norma che gli spazi non precedano mai la punteggiatura ad eccezione di apici, virgolette e parentesi aperte;
- **Parentesi:** le parentesi tonde sono le uniche parentesi utilizzate nel normale testo. Possono essere impiegate ogni qual volta si vuole integrare un concetto con un esempio, un sinonimo o una specificazione non strettamente necessaria al flusso della frase;
- **Apici:** gli apici devono essere utilizzati per racchiudere tra di essi un singolo carattere o simbolo;
- **Virgolette:** le virgolette sono utilizzate per racchiudere nomi di *file*$_g$ o di documenti, citazioni o insiemi di parole costituiti da un ordine preciso (esempio: "MongoDB as an admin Service");
- **Numeri:** i numeri vanno formattati secondo lo standard SI/ISO 31-0 (esempio: 1 234 567,89).

3.1.2.1.3 Altre norme

- **Elenchi:** nei vari documenti sono presenti molti elenchi puntati. Per ogni punto dell'elenco valgono le seguenti regole: la prima lettera deve essere maiuscola e la frase deve terminare con ';' a meno che non si tratti dell'ultimo punto dell'elenco, in quel caso si deve concludere con un '.';
- **Paragrafi:** ogni paragrafo deve avere margine sinistro 1cm più a destra rispetto al titolo;

3.1.2.1.4 Formati ricorrenti

- **Date:** le date devono essere espresse seguendo il formalismo:

  GG/MM/AAAA

  dove:

  – GG: rappresenta il giorno;
  – MM: rappresenta il mese;
  – AAAA: rappresenta l'anno in formato esteso.

  Secondo lo standard, il numero di cifre utilizzate deve essere sempre coerente, per questo in caso di numeri non abbastanza grandi si aggiungeranno zeri a sinistra (esempio 14/03/2016);

- **Orari:** gli orari devono essere espressi seguendo il formalismo:

  HH:MM

  dove:

  – HH: rappresenta il numero di ore trascorse dalla mezzanotte;
  – MM: rappresenta i minuti.

  Secondo lo standard, il numero di cifre utilizzate deve essere sempre coerente, per questo in caso di numeri non abbastanza grandi si aggiungeranno zeri a sinistra (esempio 15:07);

- **Luoghi:** per identificare univocamente un luogo si fa riferimento alle classiche norme postali:

  via/piazza n°civico, città, (sigla provincia)

  dove:

  – via/piazza: la via o la piazza dove è situato il luogo;
  – n°civico: indica l'edifico specifico del quale si sta parlando;
  – città: il comune o la frazione a cui appartiene questo indirizzo;
  – sigla provincia: indica in che provincia si trova il luogo.

- **Riferimenti:** per fare riferimento ad un concetto presente all'interno dello stesso documento basta aprire delle parentesi tonde e riportare il numero identificativo del paragrafo (esempio 3.1.2.1.4);
- ***URL*$_g$:** in caso di riferimenti a siti o *risorse*$_g$ *web*$_g$ è necessario copiare l'apposito indirizzo e dargli una colorazione blu.

3.1.2.2 Componenti Grafiche

3.1.2.2.1 Immagini

Al fine di chiarire alcuni concetti con l'ausilio di una rappresentazione grafica, i documenti possono includere delle immagini, siano essi loghi, diagrammi o screenshot. Le immagini vengono importate attraverso il formato PNG o JPG. Ogni immagine deve avere una breve didascalia che ne spieghi brevemente il senso e deve essere riportata nella lista delle immagini presente negli indici del documento.

3.1.2.2.2 Tabelle

Per rendere moli consistenti di dati più ordinati, i documenti possono includere delle tabelle. Come le immagini, le tabelle devono essere accompagnate da una breve didascalia e figurare nella lista delle tabelle tra gli indici del documento.

3.1.2.2.3 Grafici

Allo scopo di rappresentare i dati presenti nella tabelle in maniera più leggibile i documenti includeranno dei grafici, questi sono stati realizzati in LaTeX mediante specifici pacchetti. Come le immagini e le tabelle, i grafici devono essere accompagnati da una breve didascalia e figurare nella lista dei grafici tra gli indici del documento.

3.1.2.2.4 Diagrammi di Gantt

Per rappresentare la dislocazione temporale delle attività utilizzeremo il Diagramma di Gantt, uno strumento che permette di modellare la pianificazione dei compiti necessari alla realizzazione di un progetto. Costruito partendo da un asse orizzontale a rappresentazione dell'arco temporale totale del progetto, suddiviso in fasi incrementali (ad esempio, giorni, settimane, mesi) e da un asse verticale a rappresentazione delle mansioni o attività che costituiscono il progetto. I diagrammi di Gantt vengono importati in formato JPG dallo strumento *Wrike*$_g$. Ogni diagramma deve avere una breve didascalia che ne spieghi il contesto e deve essere riportato nella lista delle figure presente negli indici del documento.

3.1.2.3 Formattazione documenti

3.1.2.3.1 Frontespizio

La prima pagina di ogni documento deve essere composta dai seguenti elementi:

1. Nome del *progetto*$_g$: centrato e ben leggibile come primo elemento;

2. Logo del *team*$_g$: centrato e riportante il nome del gruppo;
3. Email del *team*$_g$: centrata e con una dimensione di carattere più piccola;
4. Nome del documento: centrato, in maiuscolo e completo di numero di versione;
5. Informazioni generali del documento:
    - **Nome del documento**;
    - **Versione del documento**;
    - **Data di creazione** del documento;
    - **Redazione:** membri del *team*$_g$ che hanno contribuito alla stesura del documento;
    - **Verifica:** membri del *team*$_g$ che hanno verificato il documento;
    - **Approvazione:** nome del responsabile che ha approvato il documento;
    - **Uso:** può essere interno o esterno a seconda dei destinatari del documento;
    - **Destinatari:** elenco dei soggetti a cui è destinato il documento.

3.1.2.3.2 Diario delle modifiche

In seguito al frontespizio è prevista una sezione che raccolga le varie modifiche apportate al documento. Deve quindi essere presente una tabella che abbia per righe tutte le operazioni effettuate sul documento e per colonne:

- **Versione:** versione del documento dopo la modifica;
- **Autore:** membro del *team*$_g$ coinvolto;
- **Ruolo:** ruolo dell'autore all'interno del *team*$_g$;
- **Data:** data della modifica;
- **Descrizione:** breve oggetto della modifica.

3.1.2.3.3 Indici

Per rendere subito chiaro il contenuto del documento in maniera dettagliata, viene posto un indice che elenca tutti i capitoli e sottocapitoli seguiti dalla pagina alla quale la rispettiva sezione inizia. Nella versione digitale dei documenti deve venire implementato un sistema di collegamenti ipertestuali che permetta al lettore di raggiungere con un semplice click la sezione desiderata. Oltre all'indice dei contenuti testuali, devono essere presenti, se necessarie, le liste delle tabelle e delle figure.

3.1.2.3.4 Intero documento

In ogni pagina, ad esclusione del frontespizio, dovranno comparire i seguenti elementi:

- **Logo del *team*$_g$:** nell'intestazione di ogni pagina, a sinistra, deve essere presente il logo di matrioska.io;
- **Sezione:** nell'intestazione di ogni pagina, a destra, deve essere riportato il nome della sezione attuale;
- **Indice:** in caso di *file*$_g$ digitale, in alto a destra deve essere presente un collegamento ipertestuale che riporta all'indice;
- **Nome del documento:** nel piè di pagina, a destra, deve essere presente il nome del documento, completo di numero di versione;
- **Indirizzo email del *team*$_g$:** nel piè di pagina, in centro, deve essere presente per esteso la email del gruppo;
- **Pagina:** a piè di pagina, a destra, deve essere presente il numero di pagina attuale rispetto a quello totale.

3.1.2.4 Tipi di documenti

3.1.2.4.1 Verbali

I verbali sono documenti che non seguono il *ciclo di vita*$_g$ standard come il resto della documentazione. Essi sono una trascrizione di ciò che avviene durante riunioni formali con il Proponente o tra i membri del *team*$_g$. All'inizio di ogni incontro viene nominato un segretario che ha la responsabilità di stilare il verbale durante lo svolgimento della riunione, non sono previste altre versioni o verifiche solo una successiva approvazione da parte del Responsabile di Progetto. Per questi motivi il nome del *file*$_g$ di un verbale non prevede versione ma deve riportare solo la data utilizzando un formato invertito che consenta un corretto ordinamento alfanumerico (esempio: Verbale2016-03-16).

I verbali devono essere costituiti dai seguenti elementi:

1. Informazioni sull'incontro o videoconferenza, ovvero:
   - **Data** dell'incontro;
   - **Ora** alla quale è cominciato l'incontro;
   - **Durata** dell'incontro;
   - **Luogo** dell'incontro oppure ***Software***$_g$ con il quale è stata effettuata la videoconferenza;
   - **Partecipanti**.
2. Ordine del giorno espresso per punti;
3. Verbale dell'incontro, cioè una trascrizione di come si è svolto l'incontro.

3.1.2.4.2 Documenti informali

Sono da considerarsi informali tutti quei documenti che non hanno ancora ottenuto l'approvazione del Responsabile di Progetto. Essi non possono essere considerati ufficiali e ancora meno possono essere distribuiti a persone esterne al *team*$_g$. Durante questo stato di lavorazione il documento dovrebbe essere accessibile solo ai membri del *team*$_g$ che partecipano all'avanzamento del suo *ciclo di vita*$_g$.

3.1.2.4.3 Documenti formali

Un documento diventa formale secondo le norme sopra descritte (sezione 3.1.1.1). Un documento formale può essere distribuito a chi necessario e rappresenta in maniera ufficiale il lavoro del *team*$_g$. In caso venga apportata una modifica a questo tipo di documenti si deve renderli nuovamente informali e il loro ciclo di verifica e approvazione ricomincia.

3.1.2.4.4 Glossario

Il glossario è un documento che necessita di un'introduzione molto più leggera degli altri documenti. Esso infatti deve solo contenere le parole che nei vari documenti vengono poste in corsivo e contrassegnate da una 'g' a pedice, includendo una spiegazione concisa del termine. Per semplificare la consultazione di questo strumento si è deciso di suddividerlo in sezioni contenenti tutti termini che iniziano con una determinata lettera, distribuite in ordine alfabetico.

3.1.2.5 *Versionamento*$_g$ dei documenti

Il numero di versione dei documenti deve rispettare il formalismo:

v.[x].[y].[z]

dove:

- **x**: questo numero deve venire incrementato ogni volta che il documento conclude il suo *ciclo di vita*$_g$ venendo approvato dal Responsabile di Progetto. Ciò non dovrebbe avvenire più di una volta per revisione di avanzamento;
- **y**: questo numero deve venire incrementato ad ogni modifica importante come ad esempio spostamento di capitoli o aggiunta di grandi quantità di contenuto;
- **z**: questo numero deve venire incrementato quando il documento subisce piccole correzioni, aggiunte o comunque modifiche poco rilevanti.

3.1.2.6 Organizzazione cartella documenti

I documenti vengono raggruppati per tipologia. Infatti, ogni tipologia rispecchia il contenuto a lei idonea.

- Interni:
  - Norme di Progetto;
  - Studio di Fattibilità.
- Esterni:
  - Analisi dei Requisiti;
  - Piano di Progetto;
  - Piano di Qualifica;
  - Specifica Tecnica;
  - Definizione di Prodotto;
  - Manuali d'uso;
  - Glossario.
- Lettera di Presentazione;
- Template LaTeX.

3.1.2.7 Organizzazione cartella documento

Ogni documento condivide la stessa struttura per semplificare i controlli automatici sui documenti. Infatti, lo schema individuato è:

- Directory aux-files: contenitore di tutto il compilato LaTeX;
- Directory parti: contenitore delle singole parti di un documento LaTeX;
- Directory img: contenitore di figure che vengono incluse nel singolo documento;
- File indenta.sh: script identatore a 80 caratteri per ogni file Tex;
- File Makefile: file contenitore i trigger di compilazione e verifica automatica dei documenti Tex;
- File pgf-pie.sty: pacchetto LaTeX per la gestione dei grafici. Necessario per evitare la non compattibilità tra alcuni sistemi di compilazione;
- File NomeFile.tex: file di struttura del documento necessario per includere le parti del documento LaTeX.

3.1.3 Strumenti

3.1.3.1 Google Drive & Google Doc

Google Drive è un *servizio web*$_g$ basato sul *cloud*$_g$ per la condivisione, memorizzazione e creazione di contenuto digitale come *file*$_g$ di testo, di presentazione e di molto altro ancora. Inoltre, in seguito al suo lancio sul mercato internazionale ha subito una enorme diffusione di utilizzo. Google Docs è un sotto-servizio incluso in Google Drive. La peculiarità' di questo è la possibilità di editing concorrente dei *file*$_g$ di testo. Questo consente al gruppo sparso geograficamente di trattare i documenti con agilità e maggiore spirito di collaborazione durante le attività di stesura di documenti draft.

3.1.3.2 LaTeX & TeXStudio

Per la stesura dei documenti si è adottato il linguaggio di markup LaTeX. Questo linguaggio permette una semplice organizzazione dei contenuti e offre molte utili funzionalità come la realizzazione di grafici. Con LaTeX è possibile creare un *template*$_g$ in grado di uniformare la formattazione di tutti i documenti. Come *editor*$_g$ di testo è stato adottato TeXStudio, mentre la compilazione in formato .pdf avviene in automatico grazie ad un plugin dell'*editor*$_g$.Si è deciso di adottare una larghezza massima per riga di 80 caratteri per facilitare la verifica dei documenti.

3.1.3.3 Script

Per semplificare alcuni compiti sui documenti vengono creati script per automatizzare il lavoro manuale. La natura degli script è di due tipi:

- **Bash shell**: i file shell hanno estensione sh. Usualmente presentano lavoro ripetitivo e di gestione. Permettono con flessibilità di simulare la manutenzione umana non appena vengono stabilite regole precise di lavoro;
- **Make**: comando famosissimo nel mondo open source. È un comando molto comodo per la generazione di metacomandi. Di solito si trova in una posizione top della directory di lavoro; di primo oppure intermendio livello lungo la gerarchia delle cartelle.

Funzionalità messe a disposizione sono:

- Identazione dei file con estensione Tex a 80 caratteri;
- Compilazione del singolo documento;
- Verifica di Warnings e di Errori del singolo documento;
- Creazione della lista dei termini da integrare nello script con estensione sql per aggiungere termini nuovi al glossario, lo script agisce a livello globale su tutti i documenti in parallelo;
- Calcolo indice Gulpease per ogni documento in parte. Il risultato viene salvato in un file con estensione gulpease;
- Creazione della cartella ConsegnaRevisione contenente i file con estensione pdf dei documenti in questione.

3.2 Processo di Verifica

In questa sezione vengono spiegati le modalità che il *team*$_g$ adotta al fine di verificare i proprio prodotti, siano essi documenti o *software*$_g$.

3.2.1 Attività

3.2.1.1 Analisi Documentazione

L'*analisi statica*$_g$ è una semplice tecnica di verifica che viene utilizzata durante l'intero *ciclo di vita*$_g$ del documento. Il suo scopo è quello di individuare anomalie difformi dalle Norme di *Progetto*$_g$. Può essere applicata attraverso due metodologie differenti:

- **Walkthrough:** Prevede la lettura del documento alla ricerca di anomalie nel testo, senza direttive specifiche sugli errori da ricercare. Ogni errore, difetto, imprecisione o incongruenza nel testo viene poi discusso con i redattori del documento al fine di evitare incomprensioni e capire insieme come risolvere il problema. Tale approccio viene utilizzato soprattutto nella fase iniziale della stesura di un documento, ovvero quando non è ancora chiara la natura dei possibili errori. Attraverso questa tecnica è possibile individuare una serie di errori ricorrenti all'interno di uno o più documenti e inserirli in una lista di controllo. Quando questa lista diviene abbastanza grande si comincia a tentare più spesso un approccio di tipo Inspection;
- **Inspection:** Prevede una lettura mirata del documento alla ricerca di errori precedentemente individuati e inseriti nella lista di controllo. Più questa lista diventa completa, più efficace diventa questa tecnica. È consigliabile avvalersi di strumenti o piccoli script per eseguire questo tipo di controllo.

3.2.1.2 Test

3.2.1.2.1 Test di unità

Il test di unità ha la funzione di verificare l'unità che si sta implementando per rilevare la presenza di errori logici. Un vero test mette in difficoltà il programmatore. Tuttavia il test è limitato, esso non può dimostrare l'assenza di bachi.
La sintassi per specificare i test di unità è:

TU[Codice Test]

3.2.1.2.2 Test di integrazione

La sintassi per specificare i test di integrazione è:

TI[Nome del componente]

3.2.1.2.3 Test di sistema

La sintassi per specificare i test di sistema è:

TS[Codice del requisito]

3.2.1.2.4 Test di validazione

La sintassi per specificare i test di validazione è:

TV[Codice Test]]

Inoltre ogni test è accompagnato dalle seguenti informazioni:

- **Descrizione:** riassunto di cosa verifica il test;
- **Stato:** indica se il test ha avuto successo o meno, oppure se deve ancora essere implementato.

3.2.1.3 Verifica

La verifica è una attività ogni presente nello sviluppo del progetto. Infatti, servono modalità operative chiare e dettagliate per i *Verificatori*, in modo da uniformare le attività di verifica svolte ed aumentare la qualità del lavoro interno al gruppo.

3.2.1.3.1 Metriche errori interni ai processi

Vengono definite delle metriche di valutazione, che i *Verificatori* possono utilizzare per valutare l'urgenza di errore/problema. Queste metriche vogliono introdurre una sensibilità proattiva nella valutazione dei problemi.

Errore trovato	Gravità	Priorità risoluzione	Modalità operative
Indici fuori range	Alta	Urgente	Ticket
Ritardi superiori a 2 giorni per attività	Alta	Urgente	Ticket
Errato tracciamento di requisiti e casi d'uso	Alta	Urgente	Ticket
Errore di progettazione	Alta	Urgente	Ticket

Tab 3: Errori nei processi: gravità e procedure di gestione

3.2.1.3.2 Metriche errori interni ad artefatti

La documentazione è uno strumento di comunicazione delle scelte che il gruppo di sviluppo e i singoli membri hanno attuato durante lo sviluppo del progetto.

Errore trovato	Gravità	Priorità risoluzione	Modalità operative
Errore ortografico o di formattazione	Bassa	Entro la Milestone	Ticket o correzione immediata
Errore sistematico di ortografia o formattazione	Media	Breve	Ticket o aggiunta alla checklist
Compilazione fallita del documento	Alta	Urgente	Ticket o correzione immediata
Errore di concetto nel testo	Alta	Urgente	Ticket
Errore di formalismo UML 2.x	Bassa	Urgente	Ticket
Mancata compilazione del codice	Alta	Urgente	Ticket o correzione immediata
Norme di codifica non rispettate	Medio	Breve	Ticket

Tab 5: Errori nei documenti: gravità e procedure di gestione

3.2.1.3.3 Gravità degli errori

La gravità dell'errore può essere:

- **Bassa**: l'errore ha un impatto marginale sulla pianificazione e prodotto software;
- **Media**: l'errore ha un impatto significativo sulla pianificaizone e prodotto software;
- **Alta**: l'errore ha un impatto decisivo sulla pianificazione e prodotto software.

Ambito	Gravità bassa	Gravità media	Gravità alta
Errore nel prodotto	Impatto su aspetti marginali	Impatto su aspetti visibili	Prodotto inutilizzabile
Errore nei processi	Aumento costi o tempi < 10%	Aumento costi o tempi < 25%	Aumento costi o tempi > 25%

Tab 7: Impatto dell'errore sui processi e prodotti

3.2.1.3.4 Priorità di risoluzione

La priorità di risoluzione può essere:

- **Entro la milestone G**: indica che l'errore deve essere risolto prima della milestone G successiva;
- **Breve**: indica che l'errore deve essere risolto entro 4-5 giorni;
- **Urgente**: indica che l'errore deve essere risolto appena possibile.

3.2.1.3.5 Modalità operative di verifica

Le modalità operative di verifica sono le seguenti:

- **Ticket**: il *Verificatore* deve aprire un ticket assegnato al responsabile con le problematiche riscontrate. Il responsabile si prende in causa del problema;
- **Correzione immediata**: il *Verificatore* procede autonomamente alla correzione dell'errore;
- **Aggiunta alla checklist**: il *Verificatore* aggiunge l'errore riscontrato nella lista di problemi opportuna. Il responsabile controlla la lista con periodicità.

3.2.2 Procedure

3.2.2.1 Procedura per la gestione delle anomalie

Durante il $processo_g$ di verifica è normale che chi di dovere si imbatta in degli errori, imperfezioni o aspetti che possono decisamente essere migliorati, definiamo in modo generale questi inconvenienti come "anomalie". Tutte le anomalie riscontrate tramite qualsivoglia tecnica o metodologia devono essere catalogate riportando:

- **Pagina** nel quale si verifica l'anomalia;
- **Sezione** più specifica possibile nel quale si verifica l'anomalia;

- **Descrizione** che spieghi in cosa l'anomalia consiste e suggerimenti per la correzione.

L'elenco completo delle anomalie andrà poi inoltrato ai redattori. Il verificatore deve assicurarsi di riportare tutte le problematiche presenti nel documento con un solo intervento, in modo che alla prossima versione non ci sia più nulla da correggere. Quando il documento passa la verifica senza che vengano riportate anomalie, esso può dunque essere inoltrato al Responsabile di Progetto per l'accettazione.

3.2.2.1.1 Verifica dei processi

Ai Verificatori è richiesto di effettuare quanto segue:

- **Controllo delle metriche**: alla conclusione di ogni periodo di del progetto, per ogni attività significativa, definita nel *Piano di Progetto*, si calcolano gli indici definiti nella sezione *Metriche per i processi* del *Piano di Qualifica*. Tali indici vengono calcolati utilizzando gli strumenti per la pianificazione;
- **Grafico PDCA**: i dati registrati vengono esportati tramite fogli elettronici di calcolo dal *Verificatore*, e successivamente i dati vengono interpretati. La interpretazione dei dati è una mansione del *Verificatore* e *Responsabile*.

3.2.2.1.2 Verifica della progettazione

Al *Verificatore* è richiesto il controllo dei diagrammi UML prodotti:

- **Diagrammi di caso d'uso**: non è possibile automatizzare i controlli sui diagrammi. I controlli si riferiscono alla sintassi di UML versione 2.0.x, e alla compatibilità concettuale;
- **Diagrammi delle classi**: i controlli si riferiscono alla sintassi di UML versione 2.0.x e alle regole di composizione di diagrammi.

3.2.2.1.3 Verifica della progettazione architetturale

Al *Verificatore* è richiesto il controllo degli indici di accoppiamento afferente ed efferente delle classi. I valori sono calcolati automaticamente per ogni classe e come da Tracy.

3.2.2.1.4 Verifica della progettazione di dettaglio

Al *Verificatore* è richiesto di controllare che il documento *Definizione di Prodotto* contenente l'architettura di dettaglio con i vari moduli del sistema abbiano una dimensione e un accoppiamento che li renda realizzabili per un singolo *Programmatore*. Ogni modulo deve essere associato ad almeno un requisito e il *Verificatore* deve controllare la qualità del tracciamento generato.

3.2.3 Strumenti

3.2.3.1 Hunspell

Hunspell è un *software*$_g$ per il controllo ortografico automatico che supporta la lingua italiana. Si è scelto di utilizzarlo perché è disponibile con un plug-in compatibile con TeXStudio, in questo modo una volta raccolti tutti i *file*$_g$ necessari a produrre il documento vero e proprio è possibile effettuare un controllo, ricavando tutti gli errori ortografici del testo.

3.2.3.2 Indice Gulpease

Si è creato un semplice script che dato un testo come input calcola un'approssimazione dell'indice di leggibilità Gulpease. In caso di risultati fuori norma (vedi Appendice Dettagli script), data la natura artigianale dello strumento, il verificatore può decidere di ignorare questo dato se ritiene il documento conforme.

3.2.3.3 Notifica di anomalie

Per la segnalazione di problemi si utilizza Wrike dato che per ogni documento esiste un *ticket*$_g$ che include i redattori. Se il verificatore rileva anomalie in un testo deve allegare il suo rapporto al relativo *ticket*$_g$ e riportare a "incompleto" lo stato del *processo*$_g$ di redazione. I redattori vengono quindi notificati automaticamente e possono correggere tutto il necessario.

3.2.3.4 ZenHub.io

La gestione dei task derivanti da notifiche di miglioria, di segnalazione di problemi e/o di più particolari notifiche risulta essere poco efficiente se vengono attuati troppi scambi di contesto.
ZenHub è uno strumento di Project Management. Ripensato sotto il punto di vista di uno sviluppatore risulta essere flessibile e personalizzabile nel contesto di gestione di problemi interni alla repository. Infatti, ZenHub è pensato per essere integrato con GitHub al supporto di sviluppo di progetti Open Source.
Il gruppo ha deciso di fare uso del seguente strumento per abbattere i tempi di comunicazione derivanti dallo scambio di contesto dovuto all'utilizzo di molteplici strumenti.

3.2.3.5 Travis CI

Come strumento di build automatico viene utilizzato Travis CI. Travis offre la possibilità di gestire in maniera del tutto trasparente la integrazione continua; la integrazione di funzionalità da più semplici a più complesse. Il modello di esecuzione di Travis è Linux. Questo rende flessibile la gestione della configurazione della compilazione. Travis qualsiasi sia l'esito della build automatica, positiva o negativa, il Responsabile di Progetto è continuamente informato sul suo stato.

3.2.3.6 Validatore W3C

Per validare lo stato di conformità con gli standard di W3C vengono utilizzati i validatori messi a disposizione degli sviluppatori. Segue l'elenco:

- Validatore HTML: https://validator.w3.org/;
- Validatore CSS: https://jigsaw.w3.org/css-validator/

4 Processi Organizzativi

4.1 Processo di Gestione

4.1.1 Attività

4.1.1.1 Comunicazioni

4.1.1.1.1 Comunicazioni interne

Le comunicazioni all'interno del *team*$_g$ vengono gestite tramite una chat di gruppo Facebook. Ciò permette un facile e veloce scambio di informazioni inerenti al progetto. In casi speciali è prevista la possibilità di utilizzare le applicazioni Skype o Hangouts per effettuare una videoconferenza.

4.1.1.1.2 Comunicazioni esterne

Il Responsabile di Progetto, come rappresentante del gruppo, gestisce le comunicazioni con l'università e i *Proponenti*$_g$ del *capitolato*$_g$. Per la corrispondenza email viene utilizzata l'apposita casella di posta elettronica:

matrioska.io.go@gmail.com

Per videoconferenze con i *Proponenti*$_g$ viene utilizzato Hangouts.

4.1.1.2 Incontri

4.1.1.2.1 Incontri interni

Il compito di organizzare gli incontri interni è affidato al Responsabile di Progetto. Inizialmente per la fase di documentazione si sono fissati preventivamente degli incontri settimanali di durata variabile dalle 3-5 ore lungo i 5 giorni lavorativi. Ogni membro del gruppo può richiedere incontri personali con il responsabile per risolvere dubbi o gestire problemi riguardanti impegni personali che possono sovrapporsi con le date degli incontri.

4.1.1.2.2 Incontri esterni

Il Responsabile ha il compito di concordare, con il *Proponente*$_g$ o con i Committenti, gli incontri esterni, seguendo le modalità precedentemente descritte (vedi 4.1.1.1.2). Ogni membro del *team*$_g$ può richiedere al Responsabile di organizzare una videoconferenza o iniziare una corrispondenza email al fine di chiarire alcuni aspetti del progetto. Se la richiesta è ben motivata, ed è necessaria una videoconferenza, va concordata una data con il *Proponente*$_g$ che deve essere poi resa nota a tutti i membri del *team*$_g$.

4.1.1.3 Ticketing

Per la suddivisione dei task tra i membri del gruppo viene utilizzato un sistema *software*$_g$ fornito mediante *servizio web*$_g$. Il servizio garantisce in maniera efficiente che ogni membro del gruppo visualizzi i propri compiti da svolgere, svolti, sospesi e in ritardo.

L'organizzazione dei task segue uno schema organizzativo strutturato per contesti. Ogni task è suddiviso in sotto-task , i quali comprendono attività di verifica e *validazione*$_g$.

4.1.2 Procedure

4.1.2.1 Assegnazione ticket

La gestione dei *ticket*$_g$ è una mansione del Responsabile di Progetto. In seguito alla attività di pianificazione e stima delle *risorse*$_g$ necessarie, il Responsabile individua le *risorse umane*$_g$ adatte a svolgere lo specifico task.
La visualizzazione dei task è possibile per:

1. contesto;
2. persone;
3. tempi di scadenza;
4. tempi di programmazione;
5. task completati;
6. task arretrati;
7. task cancellati.

La procedura di assegnazione segue il seguente flusso di gestione:

1. Il responsabile individua il contesto del *ticket*$_g$;
2. Il responsabile definisce il titolo del *ticket*$_g$;
3. Il responsabile identifica il membro a cui assegnare il *ticket*$_g$;
4. Il sistema *software*$_g$ calcola in automatico la durata del *ticket*$_g$;
5. Il responsabile individua il verificatore;
6. Il responsabile aggiunge eventuali commenti nella sezione di descrizione del *ticket*$_g$;
7. Il responsabile assegna il *ticket*$_g$.

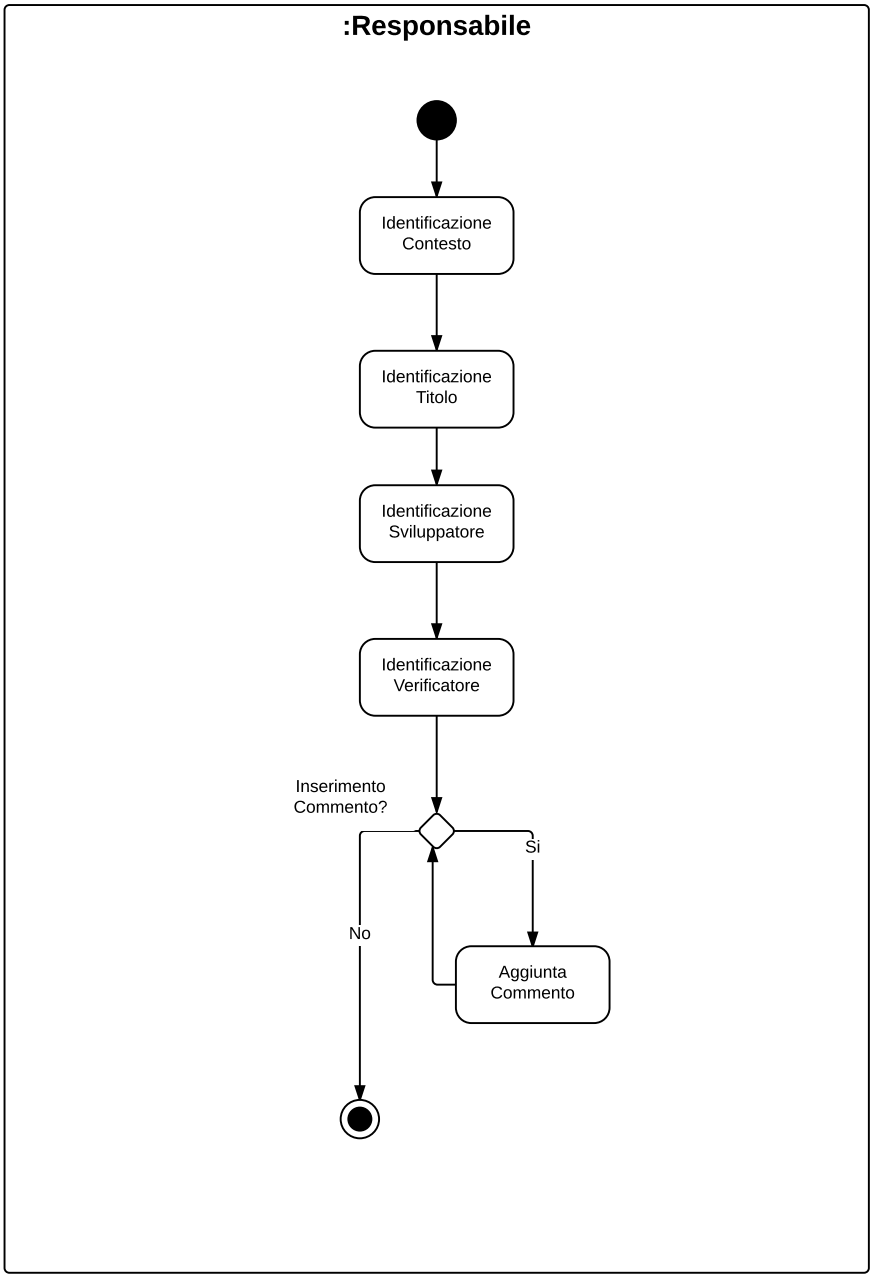

Fig 2: Assegnazione Ticket

I *ticket*$_g$ vengono gestiti in sotto-insiemi e ogni sotto-insieme è caratterizzato da un contesto che localizza il *ticket*$_g$ all'interno del sotto-insieme.

4.1.2.2 Completamento del ticket

Ogni sviluppatore gestisce la propria coda di task assegnati. I task caratterizzati da dipendenze devono mantenere invariate le dipendenze mentre i task senza dipendenze possono essere portati al completamento in parallelo.
L'attività di completamento di un *ticket*$_g$ segue viene suddivisa nella seguente lista di punti:

1. Individuazione dipendenze;
2. Completamento task;
3. Verifica task;
4. Validazione task.

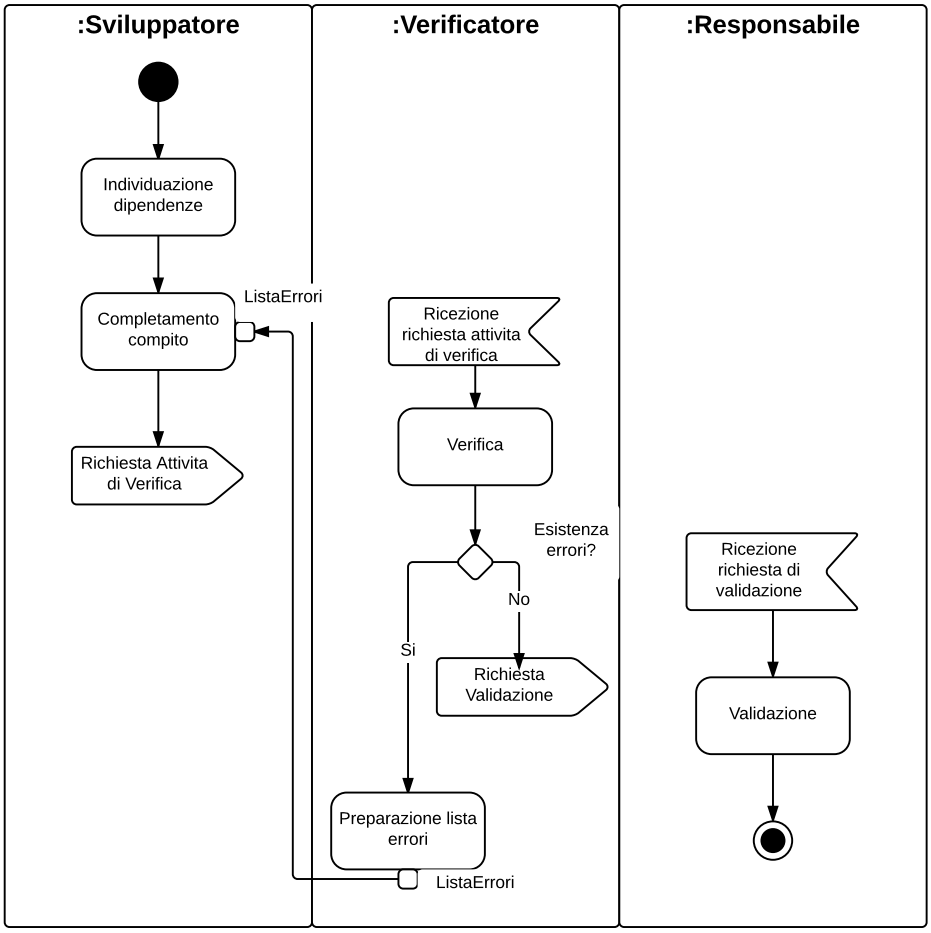

Fig 3: Completamento Task

4.1.3 Norme

4.1.3.1 Regole generali

Ogni *ticket*$_g$ deve presentare la seguente struttura:

1. Contesto di dominio
2. Priorità
   (a) Bassa;
   (b) Normale;
   (c) Alta.
3. Stato
   (a) Attivo;
   (b) Completato;
   (c) Sospeso;
   (d) Cancellato.
4. Incaricato allo svolgimento del compito
   (a) Nome Cognome;
   (b) Ruolo;
   (c) Ore Totali.
5. Lista sotto-task
6. Data
   (a) Inizio;
   (b) Fine;
   (c) Durata.
7. Dipendenze
   (a) Logiche;
   (b) Fisiche.

4.1.3.2 Protocollo di utilizzo

Lo stato determina la percentuale di maturazione del completamento di un *ticket*$_g$. I possibili stati sono:

1. Completato: rappresentato mediante un colore mnemonico verde;
2. Sospeso: rappresentato mediante un colore mnemonico viola;

3. Cancellato: rappresentato mediante un colore mnemonico grigio;
4. Attivo: rappresentato mediante un colore mnemonico bianco.

Più *ticket*$_g$ possono formare un insieme di task programmato a tempo di pianificazione per rappresentare un aggregato strategico il cui intervallo di completamento rappresenta la priorità di implementazione. Il precedente concetto di aggregato prende in prestito il concetto di "sprint" dalla metodologia *Kanban*$_g$.

Ogni membro del gruppo deve svolgere il proprio compito e al termine del quale segnalare la necessità di verifica. Ogni attività di verifica deve essere validata. La *validazione*$_g$ è un atto di feedback per la salute del progetto. Infine, ogni membro deve rispettare le scadenze prefissate. In casi di sovrapposizioni di compiti, il membro del gruppo è tenuto a segnalare l'inconveniente. Ogni azione di notifica relativa alla gestione dei compiti è automatica. Il Responsabile gestisce solo la pianificazione e il controllo del progetto mediante strumenti analitici offerti da Wrike.

4.1.3.3 Ruoli di progetto

Per lo sviluppo del progetto ai membri del *team*$_g$ matrioska.iovengono assegnati vari ruoli, corrispondenti agli omonimi ruoli nel mondo aziendale. Viene garantito che ogni componente del gruppo ricoprirà ogni ruolo almeno una volta e che non si presentino problematiche relative a conflitti di interesse mediante un'attenta fase di pianificazione; sarà poi responsabilità del Responsabile che ogni persona svolga esclusivamente compiti assegnati al suo ruolo e che la pianificazione rispetti le clausole appena esposte. Nel caso vengano a sorgere problemi lo stesso Responsabile avrà il compito di risolvere la situazione.

Segue una descrizione dei vari ruoli con le loro responsabilità e compiti.

4.1.3.3.1 Responsabile

Il Responsabile rappresenta il progetto, in quanto accentra su di sé le responsabilità di scelta e di approvazione, e il *team*$_g$, poiché presenta al *Committente*$_g$ i risultati del lavoro svolto. Detiene il potere decisionale, quindi la responsabilità in merito a:

- coordinamento, pianificazione e controllo delle attività;
- controllo e gestione delle *risorse*$_g$;
- analisi e gestione dei rischi;
- approvazione dei documenti;
- approvazione dell'offerta economica.

Ciò comporta:

- redigere l'organigramma e il *Piano di Progetto*

- assicurarsi che le attività di verifica e *validazione*$_g$ vengano svolte sistematicamente riferendosi alle *Norme di Progetto*;
- garantire che vengano rispettati i ruoli e le competenze assegnate nel *Piano di Progetto*;
- garantire che non vi siano conflitti di interessi tra redattori e Verificatori.

4.1.3.3.2 Amministratore

L'Amministratore si occupa del rendimento e dell'efficienza dell'ambiente di lavoro. Detiene quindi la responsabilità riguardo:

- ricercare strumenti che possano automatizzare il maggior numero possibile di compiti e operazioni;
- gestire l'archiviazione e il versionamento della documentazione di progetto;
- controllare le versioni dei prodotti e gestire le loro configurazioni;
- fornire strumenti e procedure per la segnalazione ed il monitoraggio ai fini del controllo qualità;
- risolvere i problemi legati alle difficoltà di controllo e alla gestione delle *risorse*$_g$ e dei processi. La risoluzione di tali problemi richiede l'adozione di strumenti adeguati.

L'Amministratore inoltre redige le *Norme di Progetto* nelle quali definisce le norme sull'utilizzo degli strumenti, collabora con il Responsabile di Progetto alla stesura del *Piano di Progetto* e redige il capitolo del *Piano di Qualifica* in cui si descrivono metodi e strumenti di verifica.

4.1.3.3.3 Analista

L'Analista è il responsabile delle attività di analisi. Le mansioni di questo ruolo sono le seguenti:

- comprendere appieno la natura e la complessità del problema;
- produrre una specifica di progetto che sia motivata in ogni suo punto e al tempo stesso comprensibile sia dal *Proponente*$_g$, sia dal *Committente*$_g$, sia dai Progettisti.

L'Analista, inoltre, redige il documento *Analisi dei Requisiti* e lo *Studio di Fattibilità*.

4.1.3.3.4 Progettista

Il Progettista è il responsabile delle attività di progettazione. Le mansioni di questo ruolo sono le seguenti:

- effettuare scelte su aspetti progettuali e tecnologici che rendano il prodotto facilmente mantenibile in futuro;

- effettuare scelte su aspetti di progettazione che applichino al prodotto soluzioni note ed ottimizzate;
- produrre una soluzione comprensibile, attuabile e motivata.

Il Progettista redige anche la *Specifica Tecnica*, la *Definizione di Prodotto* e le sezioni relative alle *metriche*$_g$ di verifica della programmazione del *Piano di Qualifica*.

4.1.3.3.5 Programmatore

Il Programmatore è il responsabile delle attività di *codifica*$_g$ e delle componenti di ausilio necessarie per effettuare le prove di verifica. Le mansioni di questo ruolo sono le seguenti:

- implementare in maniera rigorosa le soluzioni descritte dal Progettista;
- scrivere *codice*$_g$ che sia documentato, versionato, manutenibile e che rispetti le *metriche*$_g$ stabilite per la scrittura del *codice*$_g$;
- implementare i test sul *codice*$_g$ prodotto, necessari per le prove di verifica.

Il Programmatore inoltre, redige il *Manuale Utente*.

4.1.3.3.6 Verificatore

Il Verificatore è il responsabile delle attività di verifica. Le mansioni di questo ruolo sono le seguenti:

- controllare la conformità di ogni stadio del *ciclo di vita*$_g$ del prodotto;
- garantire che l'attuazione delle attività sia conforme alle norme stabilite.

Il Verificatore inoltre, redige la sezione del *Piano di Qualifica* che illustra l'esito e la completezza delle verifiche e delle prove effettuate.

4.1.4 Strumenti

4.1.4.1 Facebook

Il noto social network è stato il mezzo più immediato per comunicare e coordinarsi:

- È supportato da qualsiasi dispositivo dotato di connessione a internet;
- Ogni membro del gruppo lo utilizzava ancora prima del progetto;
- Tutte le sue funzionalità, seppur semplici, sono gratuite.

Per questi motivi è stato scelto dal *team*$_g$ come mezzo di comunicazione informale.

4.1.4.2 Wrike

Wrike è uno strumento di project management che unisce le attività di gestione e controllo in un'area di lavoro in tempo reale per promuovere la collaborazione, la discussione e la condivisione di documenti. Lo strumento permette di tracciare con facilità lo stato di progresso del progetto mediante gli strumenti analitici offerti. Infatti è possibile osservare i diagrammi di Gantt, diagrammi PERT e diagrammi di rendimento collettivo e personale. Mediante Wrike è facile avere visibilità a 360 gradi.

4.1.4.3 Google Drive

Google Drive offre molto spazio di archiviazione *cloud*$_g$ senza la minima spesa e permette di condividere *file*$_g$. Comprende anche Google Doc per la modifica contemporanea di testi. Per questi motivi è stato scelto dal *team*$_g$ come piattaforma di archiviazione e condivisione di *file*$_g$.

A Lista di controllo

Verifiche Walkthrough ai documenti hanno permesso la raccolta di informazioni riguardanti le tipologie di errori più frequenti.
La lista di controllo risultante è la seguente:

- **Norme stilistiche**:
  - Elenco puntato: Non inizia con la lettera maiuscola;
  - Elenco puntato: Non termina con il punto e virgola oppure con il punto se è l'ultimo *Elemento*$_g$;
  - Elenco numerato: Non termina con il punto e virgola oppure con il punto se è l'ultimo *Elemento*$_g$;
  - Nome proprio di persona: Non rispetta la norma Cognome Nome;
  - Parole *Proponente*$_g$ e *Committente*$_g$: Non vengono scritte con la maiuscola iniziale.
- **Italiano**:
  - Periodi: Frasi troppo lunghe rendono i concetti di difficile comprensione;
  - Doppie negazioni: Evitare l'utilizzo di doppie negazioni perché complicano la comprensione della frase;
  - Punto e virgola: Evitare l'uso del punto e virgola quando è necessario usare il punto;
  - *Proponente*$_g$ e *Committente*$_g$: Non si deve confondere il loro significato;
  - Nomi dei documenti: Non vengono scritti in corsivo;
  - Parentesi tonde: Viene lasciato uno spazio fra le parentesi e il loro contenuto.
- **LaTeX**:
  - Lettere accentate nelle variabili: Non viene utilizzato il comando apposito;
  - Carattere di spaziatura: Non deve essere utilizzato all'interno dei *Tag*$_g$;
  - Macro LaTeX: Non viene scritta usando l'apposito comando;
  - Doppio tag g a pedice ad un termine da inserire nel Glossario;
  - Non viene rispettato il formatto di identazione a 80 caratteri;
  - Troppi commandi su una singola riga;
  - Le caption delle tabelle devono iniziare con lettera Maiuscola;
  - Uso di pacchetti non standard.
- **UML**:
  - Il sistema non deve mai essere un attore;
  - Controllo ortografico: Deve essere effettuato in modo dettagliato a causa dell'impossibilità di automatizzare i controlli sui diagrammi;

  - Direzione delle frecce non corrette;
  - Consistenza della nomenclatura tra i diagrammi e le descrizioni testuali nei documenti.

  La seguente lista di controllo vuole riassumere invece gli errori più frequenti rilevati durante il Walkthrough del tracciamento dei *Requisiti*$_g$:

- **Tracciamento requisiti**:
  - Ad ogni *caso d'uso*$_g$ deve corrispondere almeno un *Requisito*$_g$;
  - Ad ogni *Requisito*$_g$ deve corrispondere almeno una fonte;
  - La fonte "Capitolato" non deve comparire nei *Requisiti*$_g$ interni;
  - Deve esserci copertura totale del *capitolato*$_g$ nei *Requisiti*$_g$.

B Dettagli Script/Verifica e Gestione di Documentazione

Nella seguente sezione vengono descritti gli scritpt di gestione della documentazione. La gestione comprende attività di stesura, verifica, correzione e miglioramento del contenuto. Vengono descritti le seguenti tipologie di script:

- Identificazione termini non riconosciuti dallo strumento Aspell;
- Creazione lista termini da inserire nel Glossario;
- Correzione termini errati in modalità automatica a partire da una specifica;
- Correzione termini errati da riga di comando;
- Compilazione automatica dei documenti necessari per la Consegna in Revisione;
- Creazione contenuto da presentare in Revisione;
- Gestione anomalie cancellazione file git;
- Gestione di indentazione automatica dei file;
- Calcolo del Indice Gulpease automatico;
- Compilazione automatica documenti LaTeX mezzo server Travis CI;
- Controllo e Verifica automatica per la rilevazione di Errori e Warning in seguito alla compilazione di documenti LaTeX .

B.1 Rilevazione termini non riconosciuti da Aspell

- **Nome**: *aspelling.sh*;
- **Scopo**: Individuazione termini specifi al dominio applicativo di attività di progetto;
- **Input**: Ogni documento LaTeXviene covertito in un documento con estensione TXT;
- **Output**: Lista di termini considerati una sola vola, salvati internamente ad un file con estensione TXT con nome paroleGenerateAspell.txt;
- **Invocazione**: Il comando deve essere invocato dalla cartella riservata *aspelling* mediante il comando ./aspelling.sh.

B.2 Creazione lista termini da glossarizzare

- **Nome**: *glox.sh*;
- **Scopo**: Creazione lista dei termini da inserire nel Glossario oppure verifica di termini mancanti nel Glossario;
- **Input**: Ogni documento LaTeXcontenente il termine da sostituire;

- **Output**: Codice di stato esecuzione script. Codice zero indica che lo script è andato a buon fine, Codice uno indica che lo script non è terminato con successo;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando ./cercaEsostituisci.sh.

B.3 Correzione termini errati a partire da una lista di termini

- **Nome**: *trovaEsostituisci.sh*;
- **Scopo**: Sostituzione in loco del termine da sostituire;
- **Input**: Ogni documento LaTeXcontenente il termine da sostituire;
- **Output**: Codice di stato esecuzione script. Codice zero indica che lo script è andato a buon fine, Codice uno indica che lo script non è terminato con successo;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando ./trovaEsostituisci.sh termine sostituto estensione-file.

B.4 Correzione termini errati da riga di comando

- **Nome**: *trovaEsostituisci.sh*;
- **Scopo**: Sostituzione in loco del termine da sostituire;
- **Input**: Ogni documento LaTeXcontenente il termine da sostituire;
- **Output**: Codice di stato esecuzione script. Codice zero indica che lo script è andato a buon fine, Codice uno indica che lo script non è terminato con successo;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando ./trovaEsostituisci.sh termine sostituto estensione-file.

B.5 Creazione documenti necessari per la consegna

- **Nome**: *Makefile*;
- **Scopo**: Generazione cartella consegna;
- **Input**: Ogni documento PDF, generato da un file LaTeX ;
- **Output**: Codice di stato esecuzione script. Codice zero indica che lo script è andato a buon fine, Codice uno indica che lo script non è terminato con successo;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando make cartella-consegna.

B.6 Compilazione automatica documenti LaTeX

- **Nome**: *compilaTutto.sh*;
- **Scopo**: Generazione documenti PDF;
- **Input**: Ogni documento LaTeX ;
- **Output**: Codice di stato esecuzione script. Codice zero indica che lo script è andato a buon fine, Codice uno indica che lo script non è terminato con successo. Viene lanciato il programma evince che permette una rapida visualizzazione del documento;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando ./compilaTutto.sh.

B.7 Gestione anomalie cancellazione file git

- **Nome**: *Makefile*;
- **Scopo**: Individuazione interferenze di file durante i commit;
- **Input**: File di configurazione git ;
- **Output**: Viene generato una lista di file cancellati durante il periodo di progetto, quando un determinato file è stato cancellato e da chi mediante il tracciamento del commit ed infine il recupero di un file;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando make file'cancellati; make quando-file-sparito; make recupero-file.

B.8 Gestione di identazione automatica dei file

- **Nome**: *Makefile*;
- **Scopo**: Standardizzazione dei file ad una colonna ad 80 caratteri;
- **Input**: Ogni documento LaTeX ;
- **Output**: File LaTeX identati;
- **Invocazione**: Il comando deve essere invocato dalla radice della directory principale mediante il comando make indenta-documenti.

B.9 Calcolo del Indice Gulpease

- **Nome**: *gulpease.sh*;
- **Scopo**: Calcolo del indice di leggibilità lingua italiana e raccolta dati storici di indice per il documento;
- **Input**: Ogni documento LaTeX ;

- **Output**: Viene generato un file di testo in formatto indice di nome gulpease;
- **Invocazione**: Il comando deve essere eseguito dalla cartella indici-documento ./gulpease.sh.

B.10 Compilazione documenti LaTeX via Travis CI

- **Nome**: *.travis.yml*;
- **Scopo**: Compilazione automatica dei documenti LaTeXin casi di grossi conflitti di pacchetti tra i singoli sviluppatori;
- **Input**: Ogni documento LaTeX ;
- **Output**: File PDF;
- **Invocazione**: Il comando vine eseguito in automatico ad ogni push programmata verso la repository.

B.11 Controllo compilazione di non compatibilità pacchetti LaTeX

- **Nome**: *Makefile*;
- **Scopo**: Controllo dei pacchetti LaTeX installati localmente allo spazio di lavoro del sviluppatore;
- **Input**: Ogni file con estensione aux;
- **Output**: File Logs.log;
- **Invocazione**: Il comando deve essere invocato dal contesto corrente di lavoro mediante il comando make verifica.

C Lista comandi Git

La lista dei comandi che segue è una breve guida in sintesi dei comandi più frequentemente utilizzati dal terminale durante la gestione della repository.
Come riferimento viene presa la lista dei comandi consigliata da GitHub.

INSTALL GIT
GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
http://git-scm.com

CONFIGURE TOOLING
Configure user information for all local repositories

`$ git config --global user.name "[name]"`
Sets the name you want attached to your commit transactions
`$ git config --global user.email "[email address]"`
Sets the email you want attached to your commit transactions
`$ git config --global color.ui auto`
Enables helpful colorization of command line output

CREATE REPOSITORIES
Start a new repository or obtain one from an existing URL

`$ git init [project-name]`
Creates a new local repository with the specified name
`$ git clone [url]`
Downloads a project and its entire version history

MAKE CHANGES
Review edits and craft a commit transaction

`$ git status`
Lists all new or modified files to be committed
`$ git diff`
Shows file differences not yet staged
`$ git add [file]`
Snapshots the file in preparation for versioning
`$ git diff --staged`
Shows file differences between staging and the last file version
`$ git reset [file]`
Unstages the file, but preserve its contents
`$ git commit -m "[descriptive message]"`
Records file snapshots permanently in version history

GROUP CHANGES
Name a series of commits and combine completed efforts

`$ git branch`
Lists all local branches in the current repository
`$ git branch [branch-name]`
Creates a new branch
`$ git checkout [branch-name]`
Switches to the specified branch and updates the working directory
`$ git merge [branch]`
Combines the specified branch's history into the current branch
`$ git branch -d [branch-name]`
Deletes the specified branch

Fig 4: Comandi di uso quotidiano

REFACTOR FILENAMES
Relocate and remove versioned files

```
$ git rm [file]
```
Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```
Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```
Changes the file name and prepares it for commit

SUPPRESS TRACKING
Exclude temporary files and paths

```
*.log
build/
temp-*
```
A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```
Lists all ignored files in this project

SAVE FRAGMENTS
Shelve and restore incomplete changes

```
$ git stash
```
Temporarily stores all modified tracked files

```
$ git stash pop
```
Restores the most recently stashed files

```
$ git stash list
```
Lists all stashed changesets

```
$ git stash drop
```
Discards the most recently stashed changeset

REVIEW HISTORY
Browse and inspect the evolution of project files

```
$ git log
```
Lists version history for the current branch

```
$ git log --follow [file]
```
Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```
Shows content differences between two branches

```
$ git show [commit]
```
Outputs metadata and content changes of the specified commit

REDO COMMITS
Erase mistakes and craft replacement history

```
$ git reset [commit]
```
Undoes all commits after `[commit]`, preserving changes locally

```
$ git reset --hard [commit]
```
Discards all history and changes back to the specified commit

SYNCHRONIZE CHANGES
Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```
Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```
Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```
Uploads all local branch commits to GitHub

```
$ git pull
```
Downloads bookmark history and incorporates changes

Fig 5: Comandi di uso eccezionale

D Formulario UML

D.1 Primi passi

Il *Unified Modeling Language* (UML) è un linguaggio per:

- Visualizzare;
- Costruire;
- Documentare.

Gli artefatti di un sistema intensivo software; cioè sistemi in cui il software ha un ruolo centrale. Come linguaggio, UML ha una *sintassi*$_g$ e *semantica*$_g$ ben definita. La parte più visibile della sintassi UML è la sua parte grafica.
Questo *Formulario* ha l'obiettivo di riassumere la notazione UML.
Contenuti inclusi:

- **Cose**:
  - Strutturali;
  - Comportamentali;
  - Raggruppamentali;
  - Annotazionali.
- **Relazioni**:
  - Dipendenza;
  - Associazione;
  - Generalizzazione.
- **Estensibilità**;
- **Diagrammi**.

D.1.1 Cose

D.1.1.1 Cose Strutturali

Le Cose Strutturali rappresentano i nomi dei modelli UML. Queste includono:

- Classi;
- Interfacce;
- Collaborazioni;
- Casi d'Uso;
- Componenti;

- Nodi.

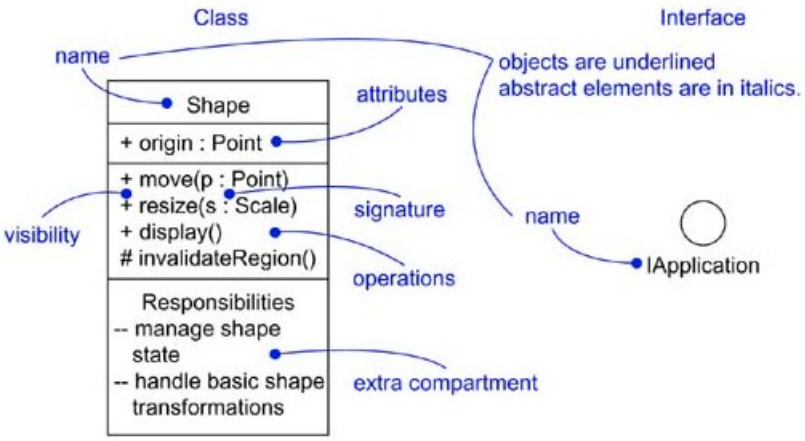

Fig 6: Rappresentazione grafica di una classe a sinistra e di una interfaccia a destra della figura

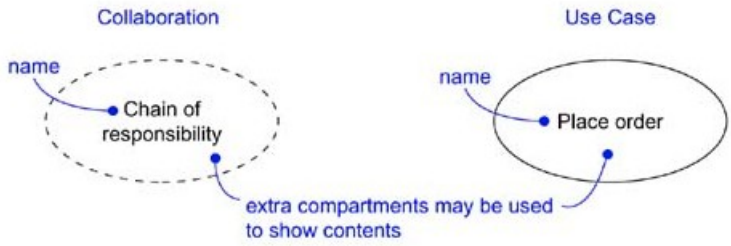

Fig 7: Rappresentazione grafica di una collaborazione a sinistra e di un caso d'uso a destra dell'immagine

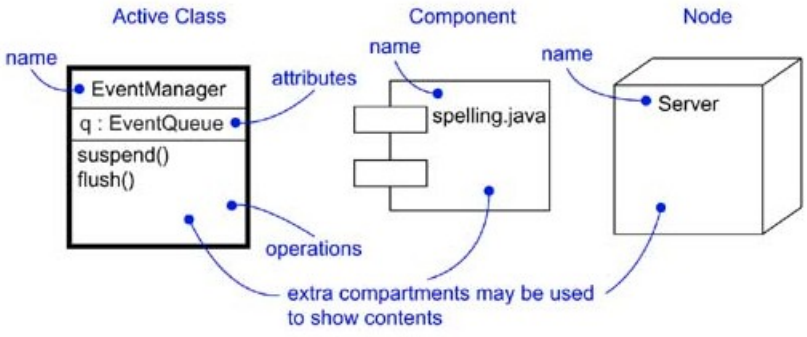

Fig 8: Rappresentazione grafica di una componente al centro, di un nodo computazionale a destra, e di una classe attiva a sinistra della figura

D.1.1.2 Cose Comportamentali

Le Cose Comportamentali rappresentano le parti dinamiche dei Modelli UML. Queste includono:

- Diagrammi di sequenza;
- Diagrammi di stato.

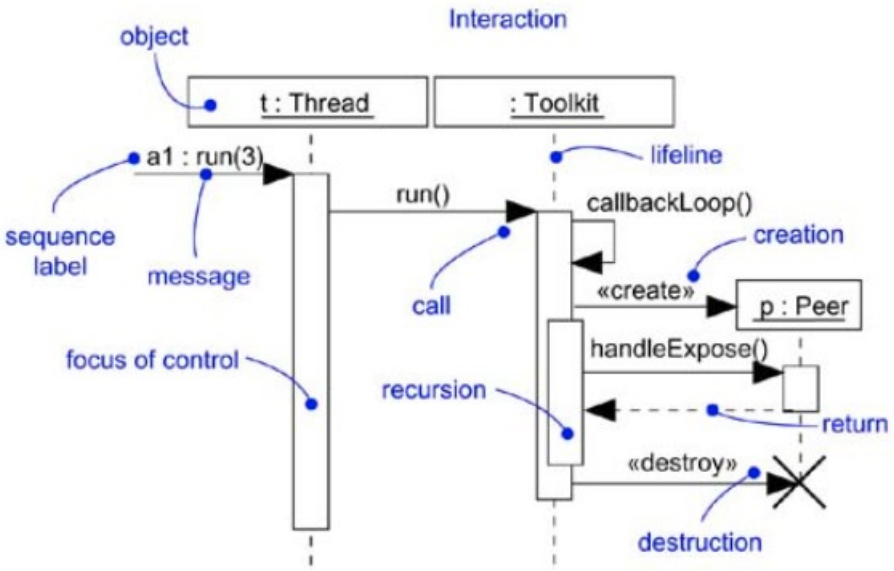

Fig 9: Rappresentazione grafica di un diagramma di sequenza

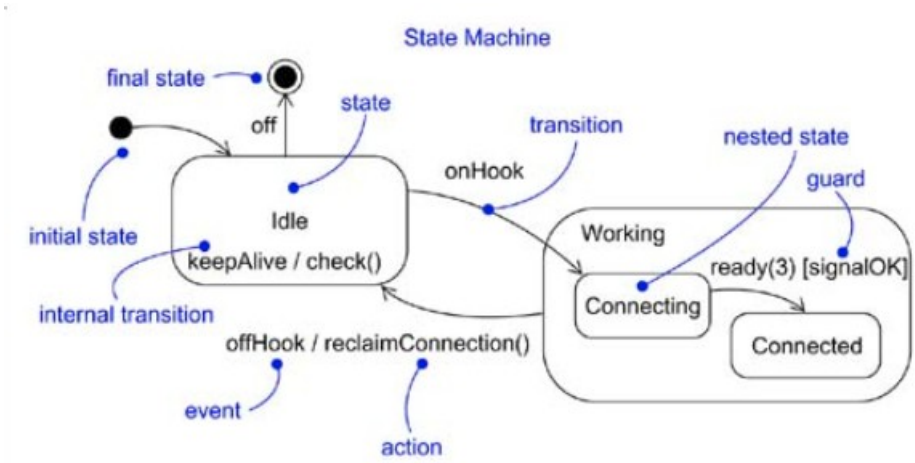

Fig 10: Rappresentazione grafica di un diagramma di macchina a stati

D.1.1.3 Cose Raggruppazionali

Le Cose di Raggruppamento individuano le parti di organizzazione dei Modelli UML. Queste includono i *Package*.

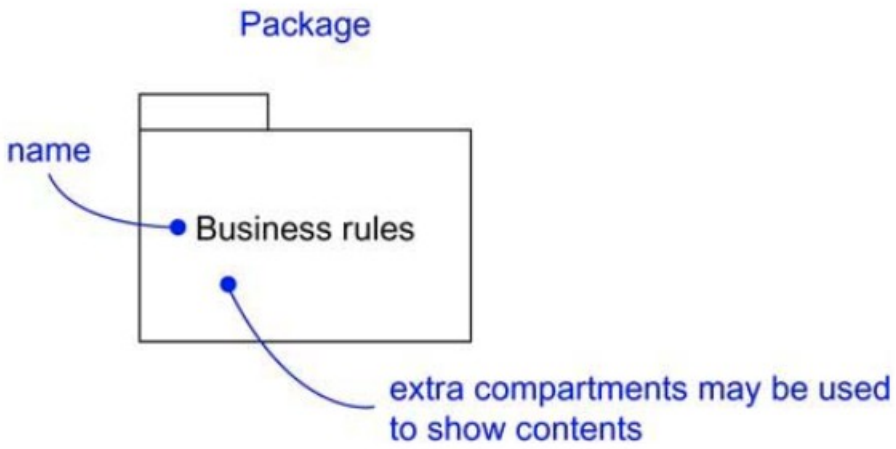

Fig 11: Rappresentazione grafica di un Package

D.1.1.4 Cose Annotazionali

Le Cose Annotazionali individuano le parti di spiegazione dei Modelli UML. Queste includono le note.

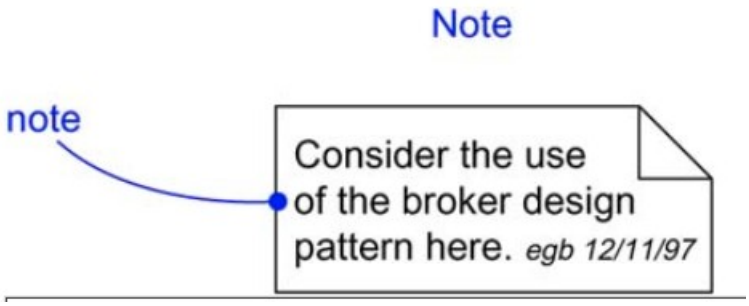

Fig 12: Rappresentazione grafica di una Nota

D.1.2 Relazioni

D.1.2.1 Dipendenza

Una dipendenza è un una relazione semantica tra due cose dove una modifica su una comporta modifiche sull'altra. Essere indipendenti/dipendenti.

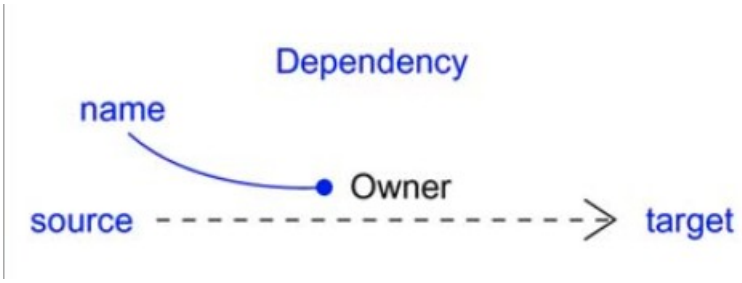

Fig 13: Relazione di dipendenza

D.1.2.2 Associazione

Una associazione è una relazione strutturale che descrive un insieme di link; un link è una connessione tra oggetti.

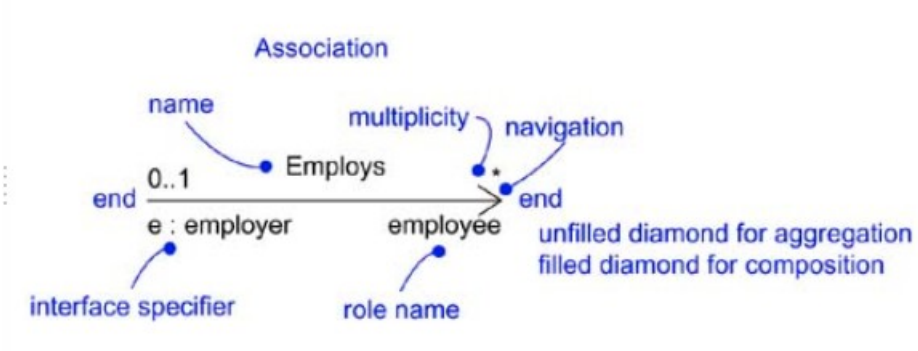

Fig 14: Relazione di associazione

D.1.2.3 Generalizzazione

La generalizzazione è una relazione di specializzazione/generalizzazione in cui gli oggetti dell'elemento specializzato possono sostituire oggetti dell'elemento generalizzante, ovvero il padre.

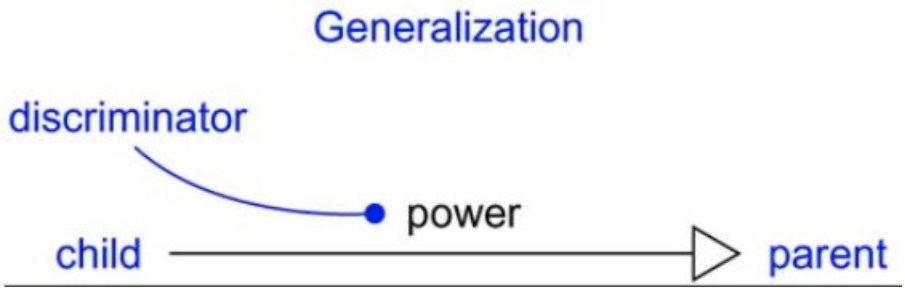

Fig 15: Relazione di generalizzazione

D.1.3 Estensibilità

Il linguaggio UML fornisce tre meccanismi per estendere la sintassi del linguaggio e la semantica:

- **Stereotipi**: rappresentano i nuovi elementi internamente al modello;
- **Valori**: rappresentano i nuovi attributi internamente al modello;
- **Vincoli**: rappresentano la semantica nuova che viene introdotta internamente al modello.

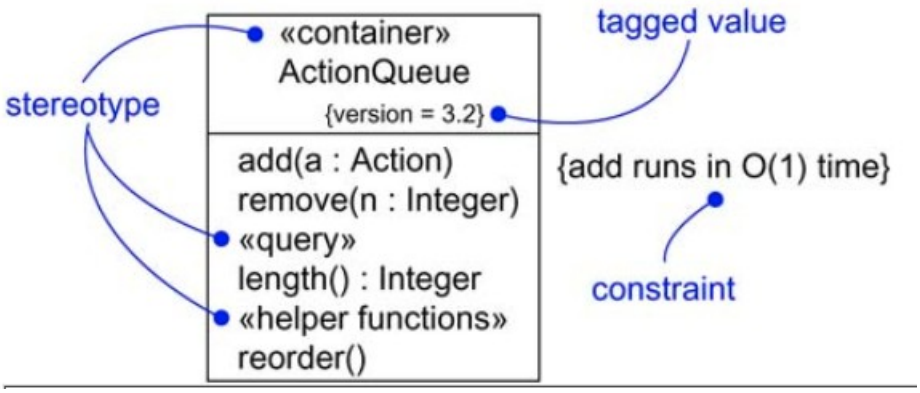

Fig 16: Rappresentazione grafica che abilita l'estensione del linguaggio UML

D.1.4 Diagrammi

Un diagramma è una convenzione grafica per un insieme di elementi, più delle volte renderizzato come un grafo connesso di vertici (cose) e archi (relazioni). Un diagramma è una proiezione del sistema. Il UML include nove tipi di diagrammi.

Tipo	Descrizione
1. Diagramma delle Classi	Diagramma strutturale che presenta un insieme di classi, interfacce, collaborazioni, e le loro relazioni
2. Diagramma degli oggetti	Diagramma strutturale che presenta un insieme di oggetti e le relazioni tra di loro
3. Diagramma dei Casi d'Uso	Diagramma comportamentale che presenta un insieme di casi d'uso e attori e le loro relazioni
4. Diagramma delle Sequenze	Diagramma comportamentale che presenta una interazione, focalizzando l'ordine temporale dei messaggi
5. Diagramma delle Collaborazioni	Diagramma comportamentale che presenta una interazione, focalizzando la organizzazione strutturale degli oggetti che inviano e ricevono messaggi
6. Diagramma a Stati	Diagramma comportamentale che presenta una macchina a stati, focalizzando l'ordine degli eventi di un oggetto

Tipo	Descrizione
7. Diagramma di Attività	Diagramma comportamentale che presenta una macchina a stati, focalizzando il flusso delle attività
8. Diagramma delle Componenti	Diagramma comportamentale che presenta un insieme di componenti e lo loro relazioni
9. Diagramma della distribuzione	Diagramma strutturale che presenta un insieme di nodi e le loro relazioni

Tab 9: Tipologia e descrizione dei diagrammi UML