

SWE16

# GITKRAFFEN

## Norme di Progetto

*Gruppo GitKraffen – Progetto IronWorks*

gitKraffen.swe16@gmail.com

<b>Versione</b>	3.0.0
<b>Redazione</b>	Iris Balaj Mihai Eni Elia Rebellato
<b>Verifica</b>	Mattia Poloni
<b>Approvazione</b>	Daniel Rossi
<b>Uso</b>	Interno
<b>Distribuzione</b>	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo GitKraffen

### Descrizione

Questo documento descrive le regole, gli strumenti e le convenzioni che il team GitKraffen dovrà rispettare per tutta la realizzazione del progetto IronWorks.

## Registro delle modifiche

Versione	Ruolo	Nominativo	Descrizione	Data
3.0.0	Responsabile	Mattia Poloni	Approvazione	21-05-2018
2.1.0	Programmatore	Mihai Eni	Verifica	21-05-2018
2.0.2	Programmatore	Daniel Rossi	Correzioni sezione Processi di documentazione	20-05-2018
2.0.1	Programmatore	Daniel Rossi	Approfondimento attività di Technology Baseline e Product Baseline in Processi primari	20-05-2018
2.0.0	Responsabile	Federica Ramina	Approvazione	03-05-2018
1.1.0	Verificatore	Mattia Poloni	Verifica Documento	02-05-2018
1.0.1	Amministratore	Daniel Rossi	Aggiunta sezione sulle metriche	01-05-2018
1.0.1	Amministratore	Elia Rebellato	Aggiunta sezione sulla formazione	01-05-2018
1.0.1	Amministratore	Mihai Eni	Aggiunta sezione sulla gestione	27-04-2018
1.0.1	Amministratore	Mihai Eni	Correzione generale documento	26-04-2018
1.0.0	Responsabile	Elia Rebellato	Approvazione	14-03-2018
0.1.0	Verificatore	Iris Balaj	Verifica Documento	13-03-2018
0.0.4	Amministratore	Daniel Rossi	Stesura processi di organizzativi	12-03-2018
0.0.3	Amministratore	Daniel Rossi	Stesura processi di supporto	11-03-2018
0.0.2	Amministratore	Mihai Eni	Stesura processi primari	10-03-2018
0.0.1	Amministratore	Daniel Rossi	Stesura introduzione	09-03-2018
0.0.0	Amministratore	Daniel Rossi	Creazione template documento	09-03-2018

## Indice

<b>1</b>	<b>Introduzione</b>	<b>8</b>
1.1	Scopo del documento . . . . .	8
1.2	Scopo del prodotto . . . . .	8
1.3	Ambiguità . . . . .	8
1.4	Riferimenti . . . . .	8
1.4.1	Normativi . . . . .	8
1.4.2	Informativi . . . . .	8
<b>2</b>	<b>Processi Primari</b>	<b>9</b>
2.1	Fornitura . . . . .	9
2.1.1	Scopo . . . . .	9
2.1.2	Aspettative . . . . .	9
2.1.3	Descrizione . . . . .	9
2.1.4	Attività . . . . .	9
2.1.4.1	Studio di fattibilità . . . . .	9
2.1.4.2	Piano di progetto . . . . .	9
2.1.4.3	Piano di qualifica . . . . .	10
2.1.4.4	Collaudo e consegna del prodotto . . . . .	10
2.2	Sviluppo . . . . .	10
2.2.1	Scopo . . . . .	10
2.2.2	Aspettative . . . . .	11
2.2.3	Descrizione . . . . .	11
2.2.4	Attività . . . . .	11
2.2.4.1	Analisi dei requisiti . . . . .	11
2.2.4.1.1	Scopo . . . . .	11
2.2.4.1.2	Aspettative . . . . .	11
2.2.4.1.3	Descrizione . . . . .	11
2.2.4.1.4	Diagrammi . . . . .	12
2.2.4.1.5	Qualità dei requisiti . . . . .	12
2.2.4.2	Technology baseline . . . . .	12
2.2.4.2.1	Scopo . . . . .	12
2.2.4.2.2	Attività . . . . .	12
2.2.4.3	Progettazione . . . . .	12
2.2.4.3.1	Scopo . . . . .	12
2.2.4.3.2	Aspettative . . . . .	13
2.2.4.3.3	Descrizione . . . . .	13
2.2.4.3.4	Qualità dell'architettura . . . . .	13
2.2.4.3.5	Design pattern . . . . .	13
2.2.4.3.6	Diagrammi . . . . .	13
2.2.4.4	Product baseline . . . . .	14
2.2.4.4.1	Scopo . . . . .	14
2.2.4.4.2	Attività . . . . .	14
2.2.4.5	Codifica . . . . .	14

2.2.4.5.1	Scopo . . . . .	14
2.2.4.5.2	Aspettative . . . . .	14
2.2.4.5.3	Descrizione . . . . .	14
2.2.4.5.4	Nomenclatura dei file . . . . .	14
2.2.4.5.5	Stile di codifica . . . . .	15
2.2.4.5.6	Intestazioni e commenti . . . . .	15
2.2.4.5.7	Versionamento . . . . .	16
2.2.5	Strumenti . . . . .	16
2.2.5.1	IntelliJ IDEA . . . . .	16
2.2.5.2	WebStorm . . . . .	17
2.2.5.3	Swego . . . . .	18
<b>3</b>	<b>Processi di supporto</b>	<b>19</b>
3.1	Documentazione . . . . .	19
3.1.1	Scopo . . . . .	19
3.1.2	Aspettativa . . . . .	19
3.1.3	Descrizione . . . . .	19
3.1.4	Nomenclatura dei documenti . . . . .	19
3.1.4.1	Versionamento documentazione . . . . .	19
3.1.5	Archiviazione documenti e formato . . . . .	19
3.1.6	Ciclo di vita documento . . . . .	20
3.1.7	Classificazione documento . . . . .	20
3.1.7.1	Verbale . . . . .	21
3.1.8	Documenti presenti . . . . .	21
3.1.9	Attività . . . . .	22
3.1.9.1	Analisi dei requisiti . . . . .	22
3.1.9.1.1	Casi d'uso . . . . .	22
3.1.9.1.2	Requisiti . . . . .	23
3.1.9.2	Glossario . . . . .	23
3.1.9.2.1	Sezione . . . . .	23
3.1.9.2.2	Indice . . . . .	23
3.1.9.2.3	Inizio sezione alfabetica . . . . .	23
3.1.9.2.4	Termine . . . . .	24
3.1.9.2.5	Eccezioni . . . . .	24
3.1.10	Struttura documento . . . . .	24
3.1.10.1	Frontespizio . . . . .	24
3.1.10.2	Diario delle modifiche . . . . .	24
3.1.10.3	Indice . . . . .	25
3.1.10.4	Contenuto del documento . . . . .	25
3.1.11	Norme grafiche . . . . .	25
3.1.11.1	Tabelle . . . . .	25
3.1.11.2	Immagine . . . . .	25
3.1.12	Norme tipografiche . . . . .	25
3.1.12.1	Stile del testo . . . . .	25
3.1.12.2	Titoli . . . . .	26
3.1.12.3	Elenchi puntati . . . . .	26

3.1.12.4	Elenchi numerati . . . . .	26
3.1.12.5	Formati comuni . . . . .	26
3.1.13	Strumenti . . . . .	27
3.1.13.1	L <sup>A</sup> T <sub>E</sub> X . . . . .	27
3.1.13.2	TexStudio . . . . .	27
3.1.13.3	Lucidchart . . . . .	28
3.1.13.4	Papyrus . . . . .	28
3.2	Gestione della configurazione . . . . .	29
3.2.1	Versionamento . . . . .	29
3.2.2	Controllo della configurazione . . . . .	29
3.2.3	Stato della configurazione . . . . .	29
3.2.4	Rilasci e consegne . . . . .	29
3.3	Gestione della qualità . . . . .	30
3.3.1	Scopo . . . . .	30
3.3.2	Aspettative . . . . .	30
3.3.3	Standard di riferimento . . . . .	30
3.3.3.1	Qualità di processo - ISO/IEC 15504 . . . . .	30
3.3.3.2	Qualità di prodotto - ISO/IEC 9126 . . . . .	30
3.3.4	Metriche . . . . .	30
3.3.4.1	Nomenclature metriche . . . . .	30
3.3.4.2	Metriche per i processi . . . . .	31
3.3.4.2.1	Schedule variance . . . . .	31
3.3.4.2.2	Budget variance . . . . .	31
3.3.4.2.3	Requirement stability index . . . . .	31
3.3.4.2.4	Violazioni dello stile di codifica . . . . .	31
3.3.4.2.5	Errori frequenti nella documentazione . . . . .	31
3.3.4.2.6	Test di unità implementati . . . . .	32
3.3.4.2.7	Test di integrazione implementati . . . . .	32
3.3.4.2.8	Test di sistema implementati . . . . .	32
3.3.4.3	Metriche per i documenti . . . . .	32
3.3.4.3.1	Gulpease . . . . .	32
3.3.4.4	Metriche per il software . . . . .	33
3.3.4.4.1	Copertura requisiti obbligatori . . . . .	33
3.3.4.4.2	Copertura requisiti desiderabili . . . . .	33
3.3.4.4.3	Complessità ciclomatica . . . . .	33
3.3.4.4.4	Accoppiamento tra classi . . . . .	33
3.3.4.4.5	Nested block depth . . . . .	34
3.3.4.4.6	Variabili non utilizzate e non definite . . . . .	34
3.3.4.4.7	Numero di parametri per metodo . . . . .	34
3.3.4.4.8	Numero di funzioni di interfaccia per package . . . . .	34
3.3.4.4.9	Complessità pesata dei metodi . . . . .	34
3.3.4.4.10	Numero di campi dati per classe . . . . .	34
3.3.4.4.11	Copertura del codice . . . . .	34
3.3.4.4.12	Bugs per lines of code . . . . .	35
3.4	Verifica . . . . .	36
3.4.1	Scopo . . . . .	36

3.4.2	Aspettative . . . . .	36
3.4.3	Descrizione . . . . .	36
3.4.4	Attività . . . . .	36
3.4.4.1	Analisi . . . . .	36
3.4.4.1.1	Analisi statica . . . . .	36
3.4.4.1.2	Analisi dinamica . . . . .	37
3.4.4.1.3	Verifica documentazione . . . . .	37
3.4.4.1.4	Lista anomalie documentazione . . . . .	37
3.4.4.1.5	Verifica del software . . . . .	37
3.4.5	Procedure . . . . .	38
3.4.5.1	Controllo qualità di prodotto . . . . .	38
3.4.5.2	Controllo qualità processo . . . . .	38
3.4.5.3	Gestione anomalie . . . . .	38
3.4.5.4	Gestione modifiche . . . . .	38
3.4.5.5	Test . . . . .	38
3.4.5.5.1	Test di unità . . . . .	38
3.4.5.5.2	Test di integrazione . . . . .	39
3.4.5.5.3	Test di regressione . . . . .	39
3.4.5.5.4	Test di sistema . . . . .	39
3.4.5.5.5	Codice indentificativo . . . . .	39
3.5	Validazione . . . . .	40
3.5.1	Scopo . . . . .	40
3.5.2	Aspettative . . . . .	40
3.5.3	Descrizione . . . . .	41
3.5.4	Procedure . . . . .	41
3.5.4.1	Validazione documentazione . . . . .	41
3.5.4.2	Validazione documentazione . . . . .	41
3.5.4.3	Test . . . . .	41
3.5.4.3.1	Test di validazione . . . . .	41
3.5.4.3.2	Codice identificativo . . . . .	41
3.5.5	Collaudo . . . . .	42
3.6	Strumenti . . . . .	42
3.6.1	Metriche . . . . .	42
3.6.2	Verifica ortografica . . . . .	42
3.6.3	Analisi statica . . . . .	42
3.6.4	Analisi dinamica . . . . .	42
<b>4</b>	<b>Processi Organizzativi</b>	<b>43</b>
4.1	Scopo . . . . .	43
4.2	Aspettative . . . . .	43
4.3	Descrizione . . . . .	43
4.4	Ruoli di progetto . . . . .	43
4.4.1	Amministratore di progetto . . . . .	43
4.4.2	Responsabile di progetto . . . . .	44
4.4.3	Analista . . . . .	44
4.4.4	Progettista . . . . .	44

4.4.5	Verificatore . . . . .	44
4.4.6	Programmatore . . . . .	44
4.5	Gestione di progetto . . . . .	45
4.5.1	Gestione delle comunicazioni . . . . .	45
4.5.1.1	Comunicazioni interne . . . . .	45
4.5.1.2	Comunicazioni esterne . . . . .	45
4.5.2	Gestione degli incontri . . . . .	45
4.5.2.1	Incontri interni . . . . .	45
4.5.2.2	Incontri esterni . . . . .	45
4.5.3	Gestione degli strumenti di coordinamento . . . . .	45
4.5.3.1	Ticketing . . . . .	45
4.5.4	Gestione degli strumenti di versionamento . . . . .	46
4.5.4.1	Repository . . . . .	46
4.5.4.2	Struttura delle repository . . . . .	46
4.5.4.3	Tipi di file e .gitignore . . . . .	46
4.5.4.4	Norme sui commit . . . . .	46
4.5.5	Gestione dei rischi . . . . .	46
4.5.5.1	Codice identificativo . . . . .	46
4.5.6	Formazione personale . . . . .	47
4.6	Strumenti . . . . .	47
4.6.1	Sistema operativo . . . . .	47
4.6.2	Slack . . . . .	47
4.6.3	Trello . . . . .	48
4.6.4	Git . . . . .	49
4.6.5	GitHub . . . . .	49

## Elenco delle figure

1	Strumenti: IntelliJ . . . . .	16
2	Strumenti: WebStorm . . . . .	17
3	Strumenti: Swego . . . . .	18
4	Ciclo di vita documentazione . . . . .	20
5	Strumenti: TexStudio . . . . .	27
6	Strumenti: Lucidchart . . . . .	28
7	Strumenti: Papyrus . . . . .	28
8	Test attuabili in fase di verifica del software . . . . .	40
9	Strumenti: Slack . . . . .	48
10	Strumenti: Trello . . . . .	49
11	Strumenti: Github . . . . .	50

# 1 Introduzione

## 1.1 Scopo del documento

Questo documento ha lo scopo di definire strumenti, regole e convenzioni adottate dal gruppo GitKraffen durante l'intero svolgimento del progetto. Tutti i componenti del gruppo devono visionare il documento per garantire quanto scritto, in modo tale da mantenere omogeneità e coesione in ogni aspetto del progetto.

Nel caso siano necessarie modifiche o aggiunte di ciò che riporta il documento in questione, è obbligatorio mantenere aggiornato ogni componente del gruppo.

Il documento si svilupperà in modo incrementale, ossia al progressivo maturare delle esigenze di progetto, sempre garantendo che ogni attività da svolgere sia stata precedentemente normata.

## 1.2 Scopo del prodotto

Lo scopo del *prodotto<sub>G</sub>* è creare un software, sotto forma di *applicazione web<sub>G</sub>* che permetta la creazione di diagrammi di robustezza, in grado di generare automaticamente il relativo codice *Java<sub>G</sub>* ed *SQL<sub>G</sub>*.

L'utente, interagendo con il sistema, sarà in grado di:

- realizzare un diagramma di robustezza a proprio piacimento;
- generare codice *Java* ed *SQL* a partire dai diagrammi di robustezza.

L'*editor* sarà fruibile dall'utente attraverso un *browser<sub>G</sub>* desktop idoneo all'utilizzo delle tecnologie *HTML5<sub>G</sub>*, *CSS<sub>G</sub>* e *Javascript<sub>G</sub>*.

## 1.3 Ambiguità

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti viene fornito il *Glossario v3.0.0*, contenente la definizione dei termini in corsivo marcati con una G pedice.

## 1.4 Riferimenti

### 1.4.1 Normativi

- *Capitolato<sub>G</sub>*: <http://www.math.unipd.it/tullio/IS-1/2017/Progetto/C5.pdf>;
- **Verbale di incontro interno**: con i componenti del gruppo del 08-03-2018;
- **Verbale di incontro esterno**: con il *proponente<sub>G</sub>* Zucchetti S.p.A del 12-03-2018.

### 1.4.2 Informativi

- Per una dettagliata guida *L<sup>A</sup>T<sub>E</sub>X* fare riferimento a Guida ai comandi *L<sup>A</sup>T<sub>E</sub>X*. v1.0.0;
- Per una dettagliata guida *Git* fare riferimento a Guida ai comandi *Git* v1.0.0.



## 2 Processi Primari

### 2.1 Fornitura

#### 2.1.1 Scopo

Questo *processo<sub>G</sub>* ha lo scopo di trattare le norme e i termini che ogni membro del gruppo GitKraffen è tenuto a rispettare per diventare fornitori della *proponente* Zucchetti S.p.A. e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin per quanto concerne il prodotto IronWorks.

#### 2.1.2 Aspettative

Durante lo svolgimento dell'intero progetto il gruppo intende instaurare con l'azienda Zucchetti S.p.A., in particolare col referente *Gregorio Piccoli (Zucchetti S.p.a.)*, un costante rapporto collaborativo in modo tale da:

- Concordare la qualifica del prodotto;
- Determinare vincoli sui processi;
- Determinare vincoli sui requisiti;
- Determinare gli aspetti necessari per soddisfare le richieste del *proponente*;
- Fare una stima dei vari costi.

#### 2.1.3 Descrizione

Al fine di garantire un riscontro efficace sul lavoro svolto, il gruppo GitKraffen si impegna a mantenersi in contatto col *proponente* per tutta la durata del progetto.

#### 2.1.4 Attività

##### 2.1.4.1 Studio di fattibilità

Il *Responsabile* di Progetto ha il compito di coordinare le attività del gruppo e stabilire riunioni preventive tra i membri, al fine di facilitare la collaborazione e lo scambio di opinioni sui capitoli. Il documento è redatto dall'*Analista* sulla base dei seguenti punti:

- **Dominio tecnologico e applicativo** : viene valutato il *capitolato* prendendo in considerazione la richiesta della conoscenza delle tecnologie da parte dei membri del gruppo, valutando anche i rischi che potrebbero insorgere basandosi su esperienze passate;
- **Rapporto costi/benefici** : viene analizzata la *qualità<sub>G</sub>* dei requisiti obbligatori, il costo in base ai risultati prefissati e l'interesse dei componenti del gruppo sulle tematiche che riporta il *capitolato*;
- **Individuazione dei rischi** : viene analizzata in dettaglio la realizzazione del progetto, cercando quali punti potrebbero far insorgere delle problematiche. Ad esempio la scarsa conoscenza dei requisiti necessari o la mancata conoscenza delle tecnologie.

##### 2.1.4.2 Piano di progetto

Il Responsabile, con la collaborazione dell'Amministratore, dovrà stilare un piano da seguire e rispettare durante la realizzazione del progetto.

Il documento dovrà contenere:

- **Analisi dei Rischi:** vengono analizzati dettagliatamente tutti gli eventuali rischi che potrebbero insorgere durante lo svolgimento del progetto e studiati i vari metodi per affrontarli, esaminando la probabilità che possano verificarsi;
- **Pianificazione:** vengono progettate tutte le attività da svolgere nel corso del progetto, stabilendo precise scadenze temporali;
- **Preventivo e Consuntivo:** viene stimato per ogni fase il quantitativo necessario di lavoro in base a ciò che è stato concordato nella pianificazione, potendo così proporre un preventivo riguardo il costo totale del progetto. Al termine di ogni attività si redige un consuntivo per tracciare l'andamento effettivo rispetto a quanto è stato preventivato.

#### 2.1.4.3 Piano di qualifica

Il Verificatore dovrà scegliere i metodi da adottare per la *verifica<sub>G</sub>* e la *validazione<sub>G</sub>* per gli elaborati creati dal gruppo. Il documento dovrà contenere:

- **Visione generale della strategia di *verifica*:** Si devono stabilire e motivare le procedure di controllo sulla *qualità* del prodotto e dei processi, tenendo presente le risorse che si hanno a disposizione;
- **Gestione della revisione:** si devono stabilire le modalità di comunicazione per mantenere ogni componente del gruppo aggiornato sulle anomalie e le procedure di controllo per la *qualità* di *processo*;
- **Pianificazione del collaudo:** ovvero definire le metodologie di collaudo del prodotto realizzato in modo dettagliato;
- **Misure e metriche:** si devono stabilire delle metriche oggettive per i documenti, i processi e il software;
- **Resoconto delle attività di *verifica*:** al termine di ogni attività bisogna riportare le metriche calcolate e un resoconto sulla *verifica* di tale attività.

#### 2.1.4.4 Collaudo e consegna del prodotto

Al fine di consegnare il prodotto il team dovrà dare eseguire una dimostrazione del prodotto in presenza del proponente e del committente attraverso un collaudo. Al fine di rendere tale collaudo positivo, sono imprescindibili la correttezza, completezza ed affidabilità del sistema software, affinché durante il collaudo si possa dare prova dei seguenti punti:

- l'esecuzione di tutti i test di validazione descritti nel documento *Piano di Qualifica v3.0.0* abbiano esito positivo;
- tutti i requisiti obbligatori descritti nel documento *Analisi dei Requisiti v3.0.0* siano stati soddisfatti;
- alcuni dei requisiti desiderabili e opzionali nel documento *Analisi dei Requisiti v3.0.0* siano stati soddisfatti.

Concluso il collaudo il Responsabile di progetto comunicherà al committente il consuntivo finale e gli consegnerà il prodotto software su un supporto fisico.

## 2.2 Sviluppo

### 2.2.1 Scopo

Questo *processo* contiene ogni attività e tutti i compiti svolti dal gruppo GitKraffen necessari per portare a termine il software richiesto dal *proponente*.

### 2.2.2 Aspettative

Le aspettative prefissate dal gruppo per ottenere una corretta *implementazione<sub>G</sub>* di tale *processo* sono:

- realizzare un prodotto finale conforme e soddisfacente in base alle richieste del *proponente*;
- realizzare un prodotto finale che soddisfi i test di verifica;
- realizzare un prodotto finale che soddisfi i test di validazione;
- fissare i vincoli tecnologici;
- fissare gli obiettivi di *sviluppo<sub>G</sub>*;
- fissare i vincoli di design.

### 2.2.3 Descrizione

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

- Realizzare un prodotto finale conforme e soddisfacente in base alle richieste del *proponente*;
- Analisi dei Requisiti;
- Progettazione;
- Codifica.

### 2.2.4 Attività

#### 2.2.4.1 Analisi dei requisiti

##### 2.2.4.1.1 Scopo

Fornire un elenco chiaro di tutti i requisiti necessari per soddisfare lo scopo del *capitolato*. I requisiti possono essere estrapolati da più fonti:

- verbali di riunioni interne o esterne;
- *capitolato* d'appalto;
- casi d'uso.

I requisiti marcati come **interno** sono requisiti che non richiedono una decisione da parte del gruppo, ma vengono decise dall'Analista.

Il risultato dell'attività è il documento chiamato *Analisi dei Requisiti v3.0.0*; esso è redatto dagli Analisti e contiene una lista dei requisiti e dei casi d'uso. I requisiti individuati permetteranno la definizione dei test di superamento dei requisiti del software in *sviluppo*.

##### 2.2.4.1.2 Aspettative

L'obiettivo consiste nel creare la documentazione formale nel quale vengono stilati tutti i requisiti necessari per soddisfare le richieste del *proponente*.

##### 2.2.4.1.3 Descrizione

Nel documento *Analisi dei Requisiti v3.0.0* sono specificati tutti i requisiti analizzati con i metodi precedentemente riportati. Il tracciamento dei requisiti avviene tramite l'utilizzo di un software riportato nella sezione strumenti.

La tecnica utilizzata per l'analisi e la ricerca dei requisiti è quella dei casi d'uso.

#### 2.2.4.1.4 Diagrammi

Nel documento Analisi dei Requisiti è richiesto l'utilizzo di diagrammi *UML<sub>G</sub>*, illustrando i casi d'uso, mettendo in particolare evidenza gli attori e i servizi del sistema. L'utilizzo di questi diagrammi rispetta la notazione del linguaggio *UML v2.0*.

#### 2.2.4.1.5 Qualità dei requisiti

Nell'Analisi dei Requisiti si valutano i requisiti avanzati dal *proponente* e vengono schematizzati i requisiti che descrivono il sistema. La specifica dei requisiti deve avere le seguenti *qualità* essenziali:

- **Diagrammi dei casi d'uso:** descrivono i casi d'uso, mettendo in particolare evidenza gli attori e i servizi del sistema;
- **Correttezza:** ogni requisito sia realmente richiesto e utile alla determinazione del sistema finale;
- **Completezza:** ogni requisito deve essere esaurito completamente nella sua definizione al fine di trattare ogni possibile iterazione tra gli attori e il sistema, e tra il sistema e se stesso;
- **Verificabilità:** ogni requisito deve essere verificabile al fine di controllare la sua effettiva *implementazione* nel prodotto finale;
- **Consistenza:** nessun requisito deve essere ridondante, ambiguo, o in contrasto con altri requisiti;
- **Estensibilità:** la codifica dei requisiti fa capo ad una convenzione condivisa dal gruppo, questo permette di mantenere estensibilità anche in caso di modifiche;
- **Tracciabilità:** ogni requisito deve essere tracciabile al fine di verificare la sua fonte o il requisito padre.

#### 2.2.4.2 Technology baseline

##### 2.2.4.2.1 Scopo

Lo scopo di questa attività è il raffinamento dei requisiti individuati durante il periodo di sviluppo, scelta delle tecnologie e produzione di una dimostrazione.

##### 2.2.4.2.2 Attività

Le attività da eseguire sono le seguenti:

- **Raffinamento requisiti:** l'*Analista* deve affinare, ed eventualmente ampliare, i requisiti analizzati nel periodo di sviluppo;
- **Scelta tecnologie:** il *Progettista* deve effettuare un'analisi delle tecnologie disponibili ed individuare quelle più idonee alla realizzazione del prodotto software, queste scelte saranno motivate e dimostrate attraverso la *Proof of Concept*;
- **Proof of Concept:** La *Proof of Concept* richiede la realizzazione di una applicazione web che dimostri che le tecnologie scelte siano idonee alla realizzazione di tale applicativo, oltre che dimostrare la conoscenza di esse da parte del team.

#### 2.2.4.3 Progettazione

##### 2.2.4.3.1 Scopo

Questa attività definisce tutte le caratteristiche essenziali del prodotto software richiesto, in funzione dei requisiti specificati nel documento *Analisi dei Requisiti v3.0.0*.

#### 2.2.4.3.2 Aspettative

Il *processo* ha come risultato la stesura dei documenti citati sopra. Ciò permetterà coerenza ed affidabilità in funzione del prodotto finale.

#### 2.2.4.3.3 Descrizione

L'attività di progettazione deve rispettare vincoli e requisiti che si sono stabiliti tra il gruppo GitKraffen ed il *proponente*.

#### 2.2.4.3.4 Qualità dell'architettura

Conclusa l'Analisi dei Requisiti con il conseguente delineamento delle funzionalità di cui dovrà essere dotato il prodotto, si procede con la realizzazione di un'architettura che possieda le seguenti caratteristiche:

- **Sufficienza:** Ogni requisito individuato è stato implementato;
- **Robustezza:** l'architettura è stata pensata per sopportare diversi livelli di "stress" dovuti all'accesso o alla presenza contemporanea di più utenti;
- **Affidabilità:** l'architettura persegue l'usabilità finale del prodotto gestendo ogni possibile situazione di rischio;
- **Manutenibilità:** l'architettura è predisposta all'evoluzione nel tempo, dovrà quindi essere possibile eseguire modifiche evitando di usare tempo superiore al necessario;
- **Modularità:** l'architettura persegue un design modulare al fine di rendere ogni compartimento logicamente ben definito e chiaro.

In particolare, riguardo alla modularità, il *Progettista* deve scomporre il sistema in *moduli<sub>G</sub>*, fornendo una descrizione precisa della struttura modulare e delle relazioni che esistono tra essi. Questo porta ai seguenti vantaggi:

- maggiore leggibilità e riusabilità del codice;
- semplificazione dell'individuazione e della correzione degli errori;
- possibilità di realizzazione di prototipi.

Per perseguire le *qualità* sopra elencate, è richiesta l'*implementazione* di *design pattern<sub>G</sub>* ove necessario. Di particolare importanza è il rispetto delle metriche per la progettazione architeturale definite nelle sezioni successive.

#### 2.2.4.3.5 Design pattern

Successivamente alla conclusione dell'*Analisi dei Requisiti*, in questa sezione saranno descritti i *design pattern* utilizzati per la realizzazione dell'architettura; questi rappresentano soluzioni progettuali adottati per problemi ricorrenti. Ogni *design pattern* deve essere accompagnato da una descrizione ed un diagramma, che ne esponga il significato e la struttura.

#### 2.2.4.3.6 Diagrammi

La progettazione deve utilizzare le seguenti tipologie di diagrammi *UML 2.0*:

- **Diagrammi delle classi:** descrivono il tipo degli oggetti che compongono il sistema e le relazioni statiche esistenti tra loro;
- **Diagramma dei package:** raggruppamento di un numero arbitrario di elementi *UML* in una *unità<sub>G</sub>* di livello più alto;

- **Diagrammi delle attività:** descrivono le operazioni di un *processo*, organizzano più entità in un insieme di azioni secondo un determinato flusso;
- **Diagrammi di sequenza:** Descrivono la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento.

L'utilizzo di questi diagrammi rispetta la notazione del linguaggio *UML* v2.0.

#### 2.2.4.4 Product baseline

##### 2.2.4.4.1 Scopo

Lo scopo di questa attività è lo studio dei pattern su cui basare l'architettura del prodotto software, oltre che gli strumenti con cui realizzarla.

##### 2.2.4.4.2 Attività

Le attività da eseguire sono le seguenti:

- **Architetture:** attività mirata allo studio dei pattern architetturali nonché alla scelta di uno di essi;
- **Design pattern:** attività mirata allo studio dei design pattern che durante la fase di progettazione saranno utilizzati per realizzare il diagramma delle classi;
- **Diagrammi:** realizzazione dei diagrammi delle classi e diagrammi di sequenza;
- **Framework:** individuazione di un *Framework* compatibile con il progetto da sviluppare;
- **Implementazione design pattern:** verifica dei design pattern che possono essere implementati e loro integrazione nelle architetture scelte.

#### 2.2.4.5 Codifica

##### 2.2.4.5.1 Scopo

In questa attività verrà realizzato concretamente il progetto attraverso implementazione dell'architettura delineata nella fase di progettazione. La fase in questione ha quindi l'obiettivo di realizzare quello che sarà il software finale completo di tutte le funzionalità progettate nelle fasi precedenti.

##### 2.2.4.5.2 Aspettative

Ciò che ci aspettiamo da questa fase è la realizzazione del software conforme a quanto concordato con il cliente.

##### 2.2.4.5.3 Descrizione

Dovranno essere quindi seguite le linee guida stabilite nel documento *Norme di Progetto v3.0.0* e le regole definite nel documento *Piano di Qualifica v3.0.0* per la stesura del codice. Durante questo *processo* inoltre dovranno essere realizzati parallelamente i documenti *Manuale Utente<sub>G</sub>* e *Manuale Sviluppatore<sub>G</sub>* che consentiranno agli utenti finali di usufruire del *prodotto* in tutte le sue funzionalità e di arricchirlo ulteriormente. Questa attività è comprensiva anche di strumenti di *verifica* e di *validazione* utili a verificare la correttezza del software realizzato.

##### 2.2.4.5.4 Nomenclatura dei file

I *file<sub>G</sub>* sorgente del progetto saranno organizzati in cartelle in modo da ottenere un albero ordinato per argomenti e sezioni. I nomi dei file saranno univoci e scritti in minuscolo, nel caso di lunghezza eccessiva si può utilizzare il carattere *underscore* come separatore per una più facile comprensione.

#### 2.2.4.5.5 Stile di codifica

Ogni membro del gruppo è tenuto a rispettare delle norme che garantiscono uniformità nel codice del progetto:

- **Indentazione:** verranno usati 4 spazi;
- **Operatori:** prima e dopo l'operatore dovrà esserci uno spazio;
- **Parentesi graffe:** l'apertura del blocco di codice complesso va definita dalla parentesi graffa scritta in linea (non a capo), mentre la chiusura si troverà in una nuova riga successiva all'ultima del blocco;
- **Nomi di variabili:** per i nomi verrà utilizzato lo stile *camelCase<sub>G</sub>*;
- **Nomi di variabili globali:** le variabili globali dovranno essere scritte in stile *uppercase<sub>G</sub>*;
- **Nomenclatura:** gli elementi definiti seguiranno le seguenti norme:
  - ogni elemento dovrà avere un nome chiaro, univoco e privo di ambiguità, definire la funzione svolta e non abbreviato o grammaticalmente scorretto;
  - i metodi che agiscono su oggetti definiti dovranno essere nominati come segue: *verboNome*.

#### 2.2.4.5.6 Intestazioni e commenti

I file sorgente dovranno contenere prima delle righe di codice un blocco commentato contenente quanto segue:

- **file:** nome del file;
- **Version:** versione del file;
- **Type:** tipo del file;
- **Data:** data creazione file;
- **Author:** nome e cognome autore, nel caso di più autori separarli con le virgole;
- **E-mail:** indirizzo del gruppo;
- **License:** tipo di licenza;
- **Registro modifiche:** registro delle modifiche scritte seguendo la seguente struttura:  
Autore || Data || breve descrizione.

Tutti i metodi dovranno essere commentati prima del codice del metodo stesso come segue:

- Descrizione delle funzionalità del metodo;
- @param TipoParametro NomeParametro - Descrizione.

Durante la scrittura del commento il Programmatore dovrà tener presente quanto segue:

- è molto importante commentare le righe di codice più complesse in modo da facilitare la loro comprensione;
- il commento della riga o del blocco di codice deve essere posto sopra per facilitarne la lettura e la comprensione;
- ad ogni aggiornamento del metodo o della riga di codice dovrà essere corrisposta una modifica al corrispondente commento.

### 2.2.4.5.7 Versionamento

La versione del file va inserita come descritto sopra nell'intestazione del file stesso rispettando il seguente formato:

**A.B**

Il valore **A** definir  l'ultima versione stabile, quindi la principale. Ogni incremento comporta l'azzeramento delle sottoversioni (quindi di B) La sottoversione **B** invece indica un avanzamento parziale del codice, viene incrementata ad ogni modifica o verifica rilevante.

La versione *A.0* definir  il raggiungimento degli obiettivi obbligatori in modo stabile.

## 2.2.5 Strumenti

Di seguito sono elencati e descritti tutti gli strumenti utilizzati dal gruppo durante la realizzazione del progetto.

### 2.2.5.1 IntelliJ IDEA

*IntelliJ IDEA* viene utilizzato per la codifica in *Java* e *Javascript*. Questo *IDE<sub>G</sub>* offre piena compatibilit  con *Linux<sub>G</sub>*, *Windows<sub>G</sub>* e *macOS<sub>G</sub>*, oltre ad essere un potente editor con molte funzionalit  integrate.

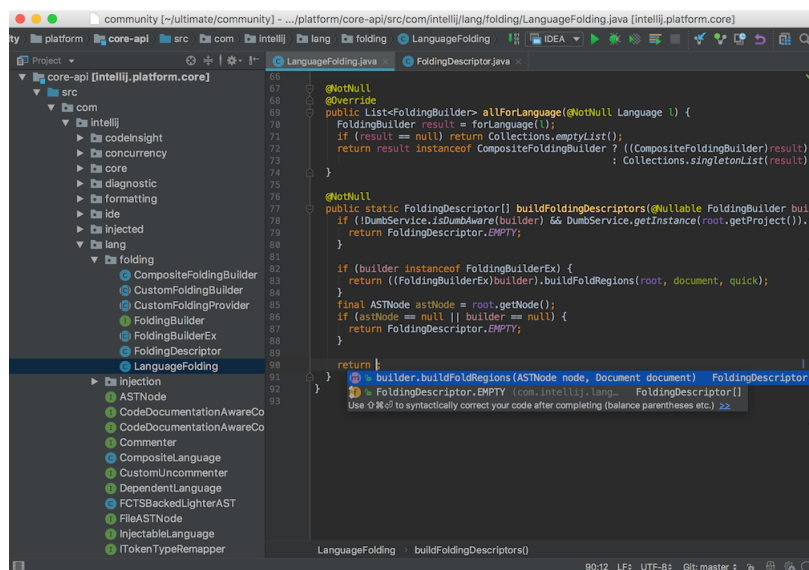


Figura 1: Strumenti: IntelliJ



### 2.2.5.2 WebStorm

*WebStorm* viene utilizzato per la codifica in *Javascript*. Questo *IDE* è ottimizzato principalmente per il web. Offre piena compatibilità con *Linux*, *Windows* e *macOS*.

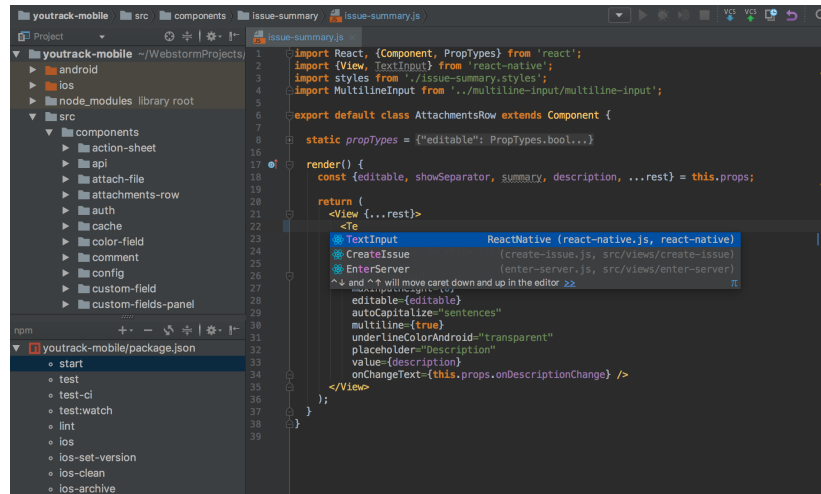
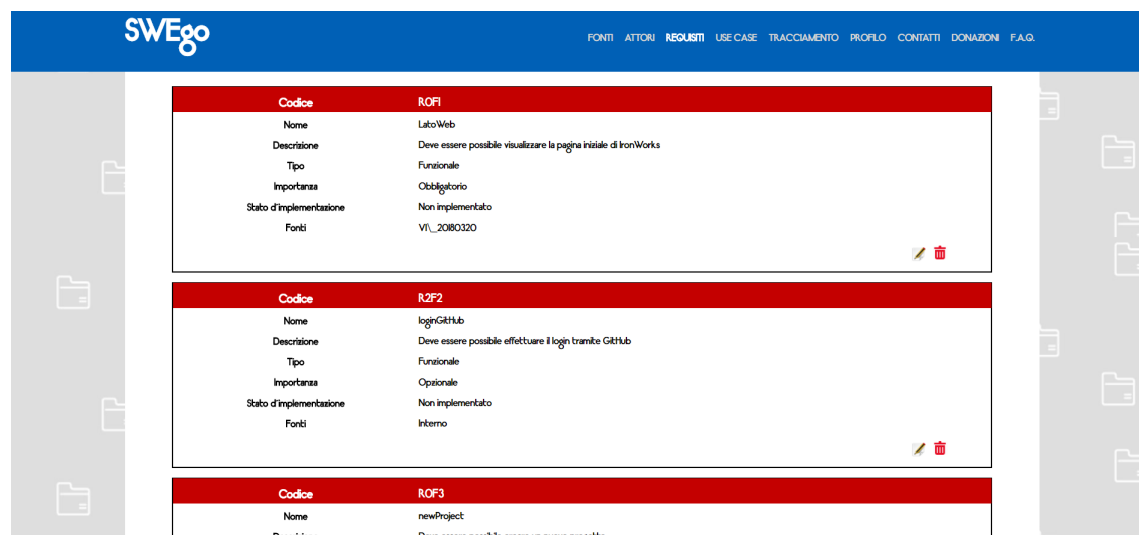


Figura 2: Strumenti: WebStorm

### 2.2.5.3 Swego

*Swego* è un software che permette di inserire i *casi d'uso<sub>G</sub>* ed i *requisiti<sub>G</sub>*, oltre ai vari *attori*, e scaricare una cartella con i file *.tex* utili a creare il documento *LaTeX* dell'*Analisi dei Requisiti*.



Codice	RQF1
Nome	LatoWeb
Descrizione	Deve essere possibile visualizzare la pagina iniziale di IronWorks
Tipo	Funzionale
Importanza	Obbligatorio
Stato d'implementazione	Non implementato
Fonti	VI_20180320

Codice	RQF2
Nome	loginGitHub
Descrizione	Deve essere possibile effettuare il login tramite GitHub
Tipo	Funzionale
Importanza	Opzionale
Stato d'implementazione	Non implementato
Fonti	Interno

Codice	RQF3
Nome	newProject
Descrizione	Deve essere possibile creare un nuovo progetto

Figura 3: Strumenti: Swego

## 3 Processi di supporto

### 3.1 Documentazione

#### 3.1.1 Scopo

Questo *processo* racchiude le linee guida su come redarre e mantenere la documentazione durante il ciclo di vita del software.

#### 3.1.2 Aspettativa

Le aspettative sulla correttezza della documentazione sono le seguenti:

- disporre di un modello formale di come la documentazione debba essere realizzata;
- scelta di convenzioni al fine di rendere coerente e corretta la documentazione prodotta;
- predisposizione di una documentazione il più precisa possibile.

#### 3.1.3 Descrizione

In questa sezione riporteremo tutte le convenzioni accettate dal gruppo ed adottate per la realizzazione di una documentazione precisa e coerente, le codifiche a seguire sono da considerarsi per documenti formali.

#### 3.1.4 Nomenclatura dei documenti

La denominazione di ciascun documento seguirà la seguente codifica:

Esempio: **NomeDelDocumento\_vX.Y.Z**

- **NomeDelDocumento**: rappresenta il nome del documento, questo non dovrà contenere spazi tra una parola e l'altra e la prima lettera di ciascuna parola andrà in maiuscolo;
- **\_v**: tra il nome del documento e la codifica di versione, verrà inserito il carattere *underscore* seguito da una *v*, abbreviazione di versione;
- **X.Y.Z**: numeri interi positivi potenzialmente uguali a 0 che definiscono la versione corrente di un documento.

##### 3.1.4.1 Versionamento documentazione

Ogni documento seguirà questa codifica per il *versionamento*<sub>G</sub>:

- **X**: indica l'ultima versione approvata dal Responsabile del documento;
- **Y**: indica il numero di modifiche importanti nel documento nella fase di sviluppo, come la creazione o la modifica di una sezione di un certo documento alla versione X;
- **Z**: indica ogni modifica del documento anche minima del contenuto del documento.

#### 3.1.5 Archiviazione documenti e formato

Ogni documento sarà inserito in una apposita cartella denominata con lo stesso nome del documento stesso, al suo interno sarà presente il documento in formato *.tex* per tutta la durata del ciclo di vita; nel momento in cui il *Responsabile* approverà il documento sarà presente oltre che in formato *.tex* anche una sua versione in formato *.pdf*.

### 3.1.6 Ciclo di vita documento

Ogni documento durante il suo ciclo di vita potrà passare di versione in versione attraverso i suddetti stadi:

- **Sviluppo**: primo stadio del ciclo di vita del documento in cui viene creato o modificato in base alla necessità del contesto, questi opereranno sotto il controllo del *Responsabile* di Progetto che assegnerà loro contenuti e problemi da trattare. Il passaggio allo stadio successivo deve essere approvato dal *Responsabile* di Progetto;
- **Verifica**: secondo stadio del ciclo di vita del documento, un *Verificatore* si occuperà di eseguire le procedure previste al fine di controllare le procedure previste dalle *Norme di Progetto v3.0.0* e dal *Piano di Qualifica v3.0.0*, rileveranno incongruenze, lacune, problematiche eventuali e le riporteranno al *Responsabile* di Progetto, il quale potrà approvare il documento o rimandarlo nello stadio di sviluppo per le eventuali correzioni necessarie;
- **Approvazione**: una volta che il *Responsabile* di Progetto approva il documento si effettuerà un incremento di versione dello stesso.



Figura 4: Ciclo di vita documentazione

### 3.1.7 Classificazione documento

Ogni documento potrà essere di tipo:

- **Interno**: Il documento sarà ad uso esclusivo del *team<sub>G</sub>*;
- **Esterno**: Il documento sarà condiviso con *committente* e *proponente*.

Inoltre ogni documento potrà essere classificato come:

- **Formale**: la versione di un documento approvato dal *Responsabile* di Progetto contenuta nella *repository<sub>G</sub>*;

- **Informale:** Una qualsiasi versione di un documento in fase di sviluppo contenuta nella Repository oppure nella cartella condivisa attraverso il servizio *cloud<sub>G</sub> Dropbox<sub>G</sub>*;
- **Verbale:** documento redatto da un segretario in occasione di incontri esterni o interni.

### 3.1.7.1 Verbale

Un verbale è un documento redatto da un segretario in occasione di incontri interni al gruppo o con altre entità esterne. I verbali non subiscono modifiche successive alla loro prima redazione, pertanto non prevedono versionamento. Ogni verbale deve essere approvato dal *Responsabile* di Progetto e deve indicare nel seguente ordine e con il formato indicato:

- **Luogo:** Città (Provincia), Via, Sede;
- **Data:** dd-mm-yyyy;
- **Ora inizio-Ora fine:** hh:mm-hh:mm 24h;
- **Partecipanti del gruppo;**
- **Partecipanti esterni**(se presenti).

Tutte le informazioni sopra elencate andranno riportate in una sezione detta *Informazioni Generali*, sarà poi presente una seconda sezione detta *Argomenti trattati* dove si elencheranno i principali argomenti di discussione dell'incontro.

La terza sezione del verbale dipenderà dal tipo di incontro:

- **Interno:** la sezione detta *Riassunto discussione*, riporterà una lista completa e ben dettagliata di tutti gli argomenti trattati nella discussione dal *team* e le scelte maturate in merito;
- **Esterno:** la sezione detta *Domande risposte* riporterà una lista di domande poste ai partecipanti esterni e le risposte ad esse relative.

Ogni verbale ha un codice univoco identificativo con il seguente formalismo:

- **X:** identifica uno dei seguenti tipi di verbale:
  - **E:** Verbale Esterno;
  - **I :** Verbale Interno.
- **Y:** identifica la data nel formato YYYY/MM/DD.

Per facilitare il tracciamento delle decisioni emerse da ogni incontro, sia interno che esterno, è richiesta una tabella riassuntiva alla fine di ogni verbale. Tali decisioni sono tracciate con il seguente formalismo:

Esempio: **CodiceVerbale.{X}**

Dove:

- **CodiceVerbale:** codice del verbale, come sopra indicato;
- **X:** numero progressivo che identifica le decisioni prese, partendo da 1.

### 3.1.8 Documenti presenti

Riportiamo di seguito i documenti in ordine alfabetico con classificazione di uso Interno od Esterno e lo scopo degli stessi:

- Analisi dei Requisiti, uso **Esterno**: lo scopo del documento è di analizzare i requisiti del progetto e fornirne una esposizione chiara. Contiene l'analisi dei casi d'uso e i diagrammi *UML* di interazione tra le diverse parti del sistema;

- Glossario, uso **Esterno**: lo scopo del documento è di vanificare ogni dubbio nella persona che leggerà un documento in cui viene però usato un linguaggio tecnico. Il documento sarà quindi il raggruppamento di definizioni atte a spiegare con chiarezza termini o concetti;
- Norme di Progetto, uso **Interno**: lo scopo del documento è di definire convenzioni, codifiche e procedure condivise dal *team* di lavoro nell'esecuzione delle diverse attività;
- Piano di Progetto, uso **Esterno**: lo scopo del documento è di quantificare risorse disponibili e come esse vengano impiegate nelle diverse attività secondo uno scadenziario, con l'obiettivo di misurare i progressi e monitorare i costi;
- Piano di Qualifica, uso **Esterno**: lo scopo del documento è di illustrare come il *team* intenda perseguire i requisiti di *qualità* del progetto;
- Studio di Fattibilità, uso **Esterno**: lo scopo del documento è spiegare cosa abbia portato il *team* alla scelta di tale *capitolato*, avendo cura di spiegare anche per quale motivo si è deciso di scartare gli altri capitolati disponibili con una breve descrizione e una lista di pro e contro.
- Manuale Utente, uso **Esterno**: lo scopo del documento è di spiegare le funzionalità e le modalità di utilizzo dell'applicazione *Ironworks*;
- Manuale sviluppatore, uso **Esterno**: lo scopo del documento è di fornire a coloro che eventualmente dovranno arricchire con nuove funzionalità il prodotto, i mezzi e le linee guida con cui procedere.

### 3.1.9 Attività

#### 3.1.9.1 Analisi dei requisiti

Questo documento scritto dall'*Analista* conterrà:

- **Casi d'uso**: saranno diversificati attraverso una descrizione generale e un diagramma che ne permettano un'immediata comprensione;
- **Requisiti**: saranno descritti tutti i requisiti accordati con il *Proponente*, specificando tipo di requisito e livello di necessità;
- **Tracciamento dei requisiti**: deve essere possibile seguire il tracciamento di ogni requisito fino alla fonte da cui esso deriva; ciò sarà realizzato attraverso tabelle di tracciamento.

##### 3.1.9.1.1 Casi d'uso

Ogni caso d'uso sarà definito attraverso l'uso dei seguenti elementi:

- Codice identificativo;
- Titolo;
- Diagramma *UML*;
- Attori primari;
- Attori secondari;
- Scopo;
- Descrizione;
- Precondizione;
- Postcondizione;

- Flusso base degli eventi;
- Inclusioni (se presenti);
- Esclusioni (se presenti).

Il codice identificativo sopra citato seguirà la seguente codifica:

Esempio: **UC{codice\_univoco}.{codice\_livello}**

Dove:

- **codice\_univoco**: ogni caso d'uso sarà provvisto di un identificativo unico;
- **codice\_livello**: numero intero positivo progressivo che identifica i sottocasi di un singolo caso d'uso.

### 3.1.9.1.2 Requisiti

Ogni requisito sarà definito attraverso l'uso dei seguenti elementi:

- Codice identificativo;
- Descrizione;
- Fonti.

Il codice identificativo sopra citato seguirà la seguente codifica:

Esempio: **R{X}{Y}{codice\_univoco}.{codice\_livello}**

Dove:

- **X**: si riferisce ad uno dei seguenti livelli di necessità:
  - **0**: Obbligatorio;
  - **1**: Desiderabile;
  - **2**: Opzionale.
- **Y**: si riferisce ad uno dei seguenti tipi di requisito:
  1. **Funzionale**: funzione o servizio offerto dal sistema;
  2. **Prestazionale**: specifica prestazionale richiesta;
  3. **Qualitativo**: condizione di validità per una funzione o servizio;
  4. **Di vincolo**: vincolo imposto dal proponente.
- **codice\_univoco**: ogni requisito sarà provvisto di un identificativo unico;
- **codice\_livello**: numero intero positivo progressivo che identifica i sottorequisiti di un singolo requisito.

### 3.1.9.2 Glossario

#### 3.1.9.2.1 Sezione

Il Glossario dovrà essere diviso in sezioni tante quante sone le lettere dell'alfabeto; se una sezione risultasse priva di termini questa non verrà inserita. I termini saranno inseriti nelle sezioni alfabetiche corrispondenti.

#### 3.1.9.2.2 Indice

L'indice sarà formato dalle sezioni alfabetiche.

#### 3.1.9.2.3 Inizio sezione alfabetica

Ogni sezione alfabetica dovrà iniziare in una propria pagina, tra una sezione alfabetica e l'altra si dovrà quindi inserire il comando *newpage*.

#### 3.1.9.2.4 Termine

Ogni termine dovrà essere inserito nella sezione corrispondente come sottosezione di essa avendo cura di seguire la seguente codifica:

- Il termine stesso;
- Una descrizione il più chiara possibile del termine.

#### 3.1.9.2.5 Eccezioni

Dato la specificità dei documenti Manuale Sviluppatore e Manuale Utente si è deciso di inserire alla fine di essi un glossario in cui viene riportata una breve spiegazione dei termini tecnici più specifici utilizzati in tali documenti. Questo perché crediamo che la mole di riferimenti tra manuali e glossario potrebbe portare ad una inefficienza nella consultazione di quest'ultimo.

### 3.1.10 Struttura documento

Per facilitare l'estensibilità delle modifiche della struttura del documento è stato creato un *Template<sub>G</sub>* riutilizzabile per ogni documento ufficiale.

#### 3.1.10.1 Frontespizio

Il frontespizio è la prima pagina di un documento formale e conterrà:

- Logo del gruppo;
- Nome del documento;
- Nome del gruppo GitKraffen
- Nome del progetto IronWorks;
- Mail del gruppo gitKraffen.swel6@gmail.com;
- Versione;
- Redazione;
- Verifica;
- Approvazione;
- Uso;
- Distribuzione;
- Descrizione del documento.

#### 3.1.10.2 Diario delle modifiche

Successivamente al frontespizio, dopo la prima pagina, è presente il diario di modifica, una tabella in cui vengono riportate in ordine cronologico tutte le modifiche a partire da quella più recente fino alla più vecchia. Questa tabella non sarà presente ove non siano richieste ulteriori modifiche oltre la prima scrittura. La tabella sarà formata dai seguenti campi:

- **Descrizione:** breve descrizione delle modifiche eseguite;
- **Autore:** l'autore delle modifiche, nome e cognome;
- **Ruolo:** il ruolo ricoperto dall'autore;
- **Data:** data della modifica;
- **Versione:** la nuova versione alla luce delle operazioni eseguite.



### 3.1.10.3 Indice

L'indice del documento inizia nelle pagine successive a quelle del diario di modifica, ogni sezione e sottosezione sarà etichettata con un numero progressivo a partire dal numero 1, e permetterà il collegamento ipertestuale direttamente alla pagina corrispondente. Inoltre saranno presenti altri due elenchi:

- **Tabelle:** elenco delle eventuali tabelle presenti nel documento, escluse il diario delle modifiche;
- **Immagini:** elenco delle eventuali immagini presenti nel documento, escluso il logo del Gruppo.

L'elenco delle immagini o delle tabelle non saranno presenti se non sono presenti immagini o tabelle;

### 3.1.10.4 Contenuto del documento

Il resto del documento sarà dedicato al contenuto del documento stesso, ogni pagina seguente avrà la seguente codifica:

- In alto a destra il nome del documento trattato nella pagina stessa;
- Una riga che divide l'intestazione dal contenuto;
- Il contenuto effettivo della pagina;
- Una riga che suddivide il contenuto dal piè di pagina;
- In basso a sinistra l'indirizzo e-mail del gruppo;
- In basso a destra il numero di pagina e il numero totale delle pagine.

### 3.1.11 Norme grafiche

#### 3.1.11.1 Tabelle

Ogni tabella dovrà essere indicata attraverso:

- **Numero:** indice progressivo a partire da 1 che individui la tabella univocamente;
- **Titolo:** breve descrizione che riassume il contenuto della tabella.

Inoltre ogni tabella dovrà essere corredata da un'eventuale descrizione ove non fosse presente al suo interno per una maggiore semplicità di comprensione.

#### 3.1.11.2 Immagine

Ogni immagine dovrà essere indicata attraverso:

- **Numero:** indice progressivo a partire da 1 che individui l'immagine univocamente;
- **Titolo:** breve descrizione che descriva al meglio il contenuto dell'immagine.

### 3.1.12 Norme tipografiche

#### 3.1.12.1 Stile del testo

- **Glossario:** ogni parola contenuta nel *Glossario* deve essere marcata, alla sua prima occorrenza in ogni documento, in carattere corsivo e con una G maiuscola a pedice;
- **Corsivo:** il corsivo deve essere utilizzato per le seguenti occorrenze:

- citazioni;
  - parole inserite nel Glossario;
  - attività del progetto;
  - ruoli nel progetto;
  - riferimenti ad altri documenti;
  - parole particolari solitamente poco usate.
- **Maiuscolo:** le uniche parole che è consentito scrivere interamente in maiuscolo sono gli acronimi.

### 3.1.12.2 Titoli

Nel definire i titoli delle sezioni si dovrà seguire questa codifica:

- la prima parola del titolo dovrà avere la propria iniziale in maiuscolo;
- tutte le altre parole dovranno essere in minuscolo;
- se un nome proprio di cosa dovesse essere contenere nel proprio nome una o più lettere maiuscole queste potranno essere inserite;
- non devono essere scritte né in grassetto né in corsivo;
- non deve essere riportata la lettera **G** del Glossario.

### 3.1.12.3 Elenchi puntati

Gli elenchi puntati vengono rappresentati graficamente da un *pallino* nel primo livello, da un *trattino* nel secondo e da un *asterisco* nel terzo. Gli elenchi puntati servono ad esprimere in modo sintetico un concetto, evitando frasi lunghe e discorsive. Ogni voce di un elenco puntato deve terminare con un punto e virgola, ad eccezione dell'ultima, che va terminata con un punto.

### 3.1.12.4 Elenchi numerati

Gli elenchi numerati vengono rappresentati graficamente da un numero a partire dal numero 1. Gli elenchi numerati servono ad identificare un concetto con un indice numerico. Ogni riga non necessita di segni di interpunzione ad indicarne la chiusura, la riga successiva inizierà a partire dal numero seguente.

### 3.1.12.5 Formati comuni

Per le seguenti tipologie di concetto vengono applicati i seguenti formalismi:

- **Date:**

**GG-MM-AAAA**

- **GG:** rappresenta il giorno rappresentato utilizzando due cifre;
- **MM:** rappresenta il mese rappresentato utilizzando due cifre;
- **AAAA:** rappresenta l'anno rappresentato utilizzando quattro cifre.

- **Orari:**

**HH:MM**

- **HH:** rappresenta l'ora e può assumere valori da 0 a 23;
- **MM:** rappresenta i minuti e può assumere valori da 0 a 59.

- **Nomi ricorrenti:**

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

### 3.1.13 Strumenti

#### 3.1.13.1 L<sup>A</sup>T<sub>E</sub>X

Per la produzione della documentazione si è fatto uso del formato L<sup>A</sup>T<sub>E</sub>X perche permette l'utilizzo di costrutti grafici standard ed ha un migliore stile di codifica.

#### 3.1.13.2 TexStudio

TexStudio è consigliato per la stesura dei documenti, con i pregi di essere open source e contenere un dizionario per il controllo dell'ortografia in varie lingue, tra le quali l'italiano.

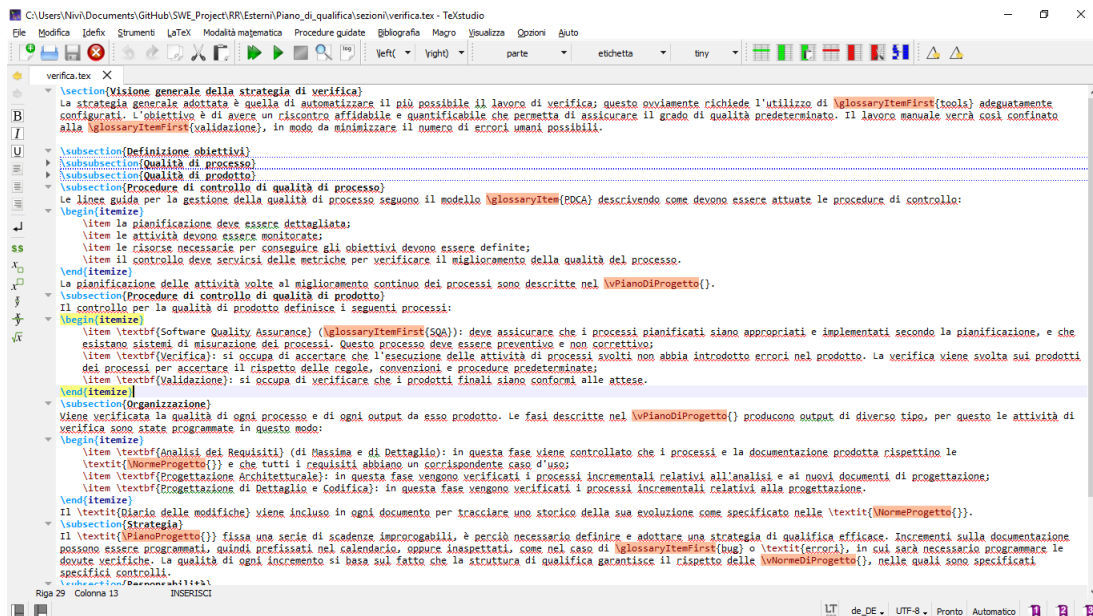


Figura 5: Strumenti: TexStudio

### 3.1.13.3 Lucidchart

Per la produzione di schemi illustrativi abbiamo sfruttato **Lucidchart** in quanto ha librerie più ampie rispetto agli altri editor.

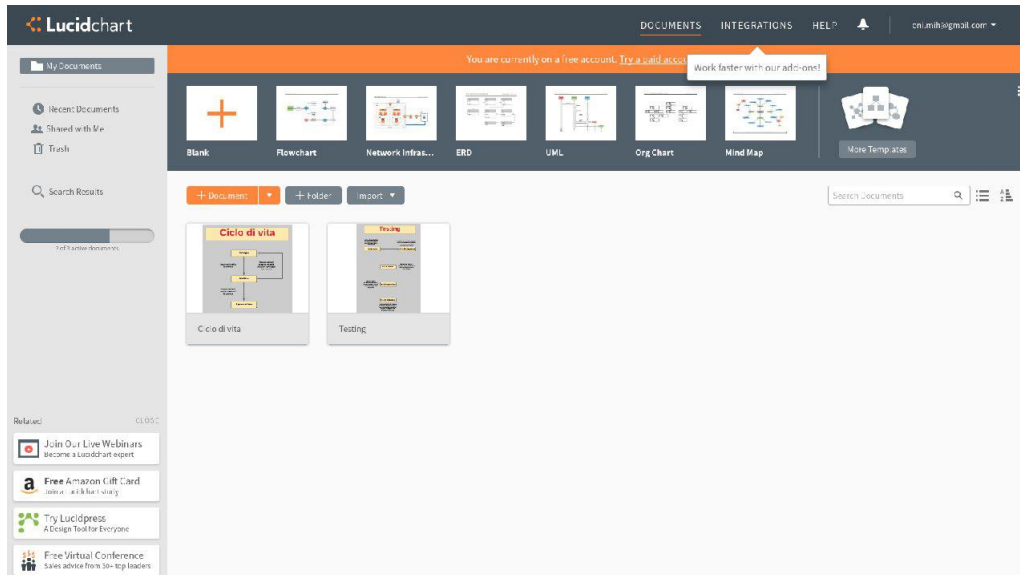


Figura 6: Strumenti: Lucidchart

### 3.1.13.4 Papyrus

Per la produzione dei diagrammi di classe abbiamo sfruttato **Papyrus** in quanto mette a disposizione varie librerie che aiutano la creazione di UML.

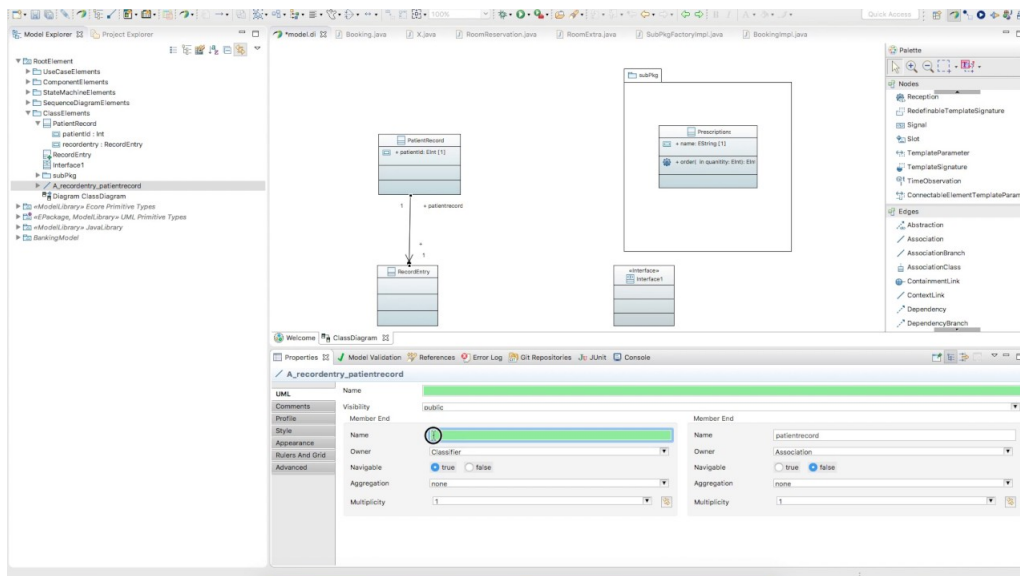


Figura 7: Strumenti: Papyrus

## 3.2 Gestione della configurazione

### 3.2.1 Versionamento

Ogni documento o componente del codice sorgente soggetto a versionamento nonché la documentazione formale è versionata mediante la tecnologia *Git*, nello specifico utilizzando il servizio gratuito GitHub. Per la condivisione di materiale informale e/o non versionabile si predilige invece una cartella condivisa sullo spazio cloud offerto da *Dropbox*.

### 3.2.2 Controllo della configurazione

Per identificare e tracciare le richieste di modifica viene utilizzata la tecnologia *Trello<sub>G</sub>*, questa permette di identificare le operazioni di modifica e assegnarle alle risorse disponibili, per analizzare e valutare le modifiche effettuate e approvare o meno quelle proposte, viene utilizzato il sistema delle *Pull request<sub>G</sub>* fornito dal servizio gratuito *GitHub<sub>G</sub>*. Le *Pull request* riportano nel titolo la ragione della modifica effettuata, nonché la o le *commit* precedenti ad esse, sarà cura del verificatore approvare le modifiche e chiudere la *Pull request*.

### 3.2.3 Stato della configurazione

Successivamente ad ogni revisione, l'ultima commit effettuata viene marcata con un'etichetta di versione (essa costituirà la nuova *baseline* di riferimento). L'esito delle revisioni vengono riportate in appendice al *Piano di Qualifica v3.0.0* con le relative correzioni da apportare ai prodotti oggetto della revisione.

### 3.2.4 Rilasci e consegne

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

### 3.3 Gestione della qualità

#### 3.3.1 Scopo

Il *processo* di *verifica* ha lo scopo di individuare potenziali errori presenti nelle attività del *processo* esaminato, i quali potrebbero essere stati prodotti a seguito del suo *sviluppo* a partire dall'ultima approvazione. Il fine ultimo è di fornire una valutazione sulla correttezza di ciò che si sta verificando.

#### 3.3.2 Aspettative

Una corretta *implementazione* di tale *processo* permette di individuare:

- una procedura di *verifica*;
- le condizioni affinché l'oggetto in esame superi tale *verifica*;
- gli errori più comuni, catalogati e descritti.

#### 3.3.3 Standard di riferimento

##### 3.3.3.1 Qualità di processo - ISO/IEC 15504

La qualità di processo è un fattore determinante per la qualità di prodotto. Si è deciso di perseguire quindi la qualità seguendo questi due modelli:

- $SPICE_G$ , definito nello standard ISO/IEC 15504, ai fini di una valutazione oggettiva dei processi, per darne un giudizio sulla maturità e individuare azioni migliorative;
- $PDCA_G$ , per il controllo delle attività di processo ripetibili e misurabili e per la manutenzione dei processi stessi, incrementandone la qualità.

##### 3.3.3.2 Qualità di prodotto - ISO/IEC 9126

Oltre alla qualità di processo, sono necessari degli obiettivi rivolti direttamente alla qualità del prodotto, per massimizzarne l'efficacia. A tal fine, lo standard ISO/IEC 9126 classifica la qualità del software e definisce delle metriche per la sua misurazione.

#### 3.3.4 Metriche

Vengono adottate delle metriche per rendere misurabili e valutabili i processi, i documenti e il software prodotto. La visione serve al gruppo per monitorare l'andamento dei processi e la qualità del prodotto.

##### 3.3.4.1 Nomenclature metriche

Ogni metrica ha un codice univoco identificativo con il seguente formalismo:

$$M\{W\}\{Y\}$$

Dove:

- **W**: identifica se la metrica si riferisce a processi, documenti o software e può assumere i seguenti valori:
  - **P**: una metrica per processi;
  - **D**: una metrica per documenti;
  - **S**: una metrica per software.
- **Y**: codice numerico progressivo a partire da 1.

### 3.3.4.2 Metriche per i processi

#### 3.3.4.2.1 Schedule variance

- **Codice:** MP1;
- **Descrizione:** Indica se si è in linea, in anticipo o in ritardo rispetto alla schedulazione delle attività di progetto pianificate.

$$SV = BCWP - BCWS$$

Dove *BCWP* indica il valore delle attività realizzate alla data corrente e *BCWS* rappresenta il costo pianificato per realizzare le attività di progetto alla data corrente. È un indicatore di efficacia soprattutto nei confronti del Cliente. Se  $SV > 0$  significa che il progetto sta procedendo con maggior velocità a quanto pianificato, viceversa se negativo.

#### 3.3.4.2.2 Budget variance

- **Codice:** MP2;
- **Descrizione:** Indica se alla data corrente si è speso di più o di meno rispetto a quanto previsto.

$$BV = BCWS - ACWP$$

Dove *BCWS* indica il costo pianificato per realizzare le attività di progetto alla data corrente e *ACWP* rappresenta il costo effettivamente sostenuto alla data corrente. È un indicatore che ha un valore unicamente contabile e finanziario. Se  $BV > 0$  significa che il progetto sta spendendo il proprio budget con minor velocità di quanto pianificato, viceversa se negativo.

#### 3.3.4.2.3 Requirement stability index

- **Codice:** MP3;
- **Descrizione:** Indica la percentuale dei requisiti che il gruppo si è prefissato di portare a termine rimasti invariati nel tempo. Un valore elevato di tale metrica indica un'attività di analisi attenta e corretta. Viene calcolata tramite la seguente formula, dove ogni voce fa riferimento ai requisiti:

$$RSI[\%] = 1 - \frac{\#aggiunti + \#tolti + \#modificati}{\#totali\ iniziali}$$

#### 3.3.4.2.4 Violazioni dello stile di codifica

- **Codice:** MP4;
- **Descrizione:** Calcola il numero di occorrenze di violazioni dello stile di codifica, definito in 2.2.4.4.5, all'interno del codice. Un numero elevato indica poca cura nel processo di codifica e, di conseguenza, codice poco affidabile.

#### 3.3.4.2.5 Errori frequenti nella documentazione

- **Codice:** MP5;
- **Descrizione:** Calcola il numero di errori frequenti, descritti nella sezione 3.4.4.1.4, individuati durante una verifica tramite Inspection. Un riscontro elevato indica poca cura nella stesura dei documenti.

#### 3.3.4.2.6 Test di unità implementati

- **Codice:** MP6;
- **Descrizione:** Indica la percentuale di test di unità effettivamente implementati. Un valore elevato di tale metrica indica un livello soddisfacente di test sulle componenti base del sistema. Viene calcolata tramite la seguente formula:

$$TUI[\%] = \frac{\#test\ di\ unità\ implementati}{\#test\ di\ unità\ totali}$$

#### 3.3.4.2.7 Test di integrazione implementati

- **Codice:** MP7;
- **Descrizione:** Indica la percentuale di test di integrazione effettivamente implementati. Un valore elevato di tale metrica indica un livello soddisfacente di test sulle componenti base del sistema. Viene calcolata tramite la seguente formula:

$$TII[\%] = \frac{\#test\ di\ integrazione\ implementati}{\#test\ di\ integrazione\ totali}$$

#### 3.3.4.2.8 Test di sistema implementati

- **Codice:** MP8;
- **Descrizione:** Indica la percentuale di test di sistema effettivamente implementati. Un valore elevato di tale metrica indica un livello soddisfacente di test sulle componenti base del sistema. Viene calcolata tramite la seguente formula:

$$TSI[\%] = \frac{\#test\ di\ sistema\ implementati}{\#test\ di\ sistema\ totali}$$

#### 3.3.4.3 Metriche per i documenti

Per potere garantire la qualità dei documenti è indispensabile che essi siano leggibili, è stato quindi utilizzato il seguente indice per misurare la leggibilità dei testi in lingua italiana:

##### 3.3.4.3.1 Gulpease

- **Codice:** MD1;
- **Descrizione:** 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. Permette di misurare la complessità dello stile di un documento. L'indice di Gulpease considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere. La formula per il suo calcolo è:

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

I risultati sono compresi tra 0 e 100, dove il valore 100 indica la leggibilità più alta e 0 la leggibilità più bassa.



#### 3.3.4.4 Metriche per il software

Nessun membro del gruppo ha conoscenza del linguaggio *NodeJS<sub>G</sub>* e delle sue particolarità come l'aspetto *funzionale<sub>G</sub>*. Tali differenze con i linguaggi studiati nel percorso universitario si sono tradotte nella difficoltà di individuare metriche non incentrate sulla visione ad oggetti del codice. Infine, si è osservata l'assenza di strumenti per la misurazione di metriche tradizionali come la coesione e l'instabilità dei package. Di seguito vengono elencate le metriche per il software prodotto:

##### 3.3.4.4.1 Copertura requisiti obbligatori

- **Codice:** MS1;
- **Descrizione:** Questa metrica permette di tracciare il numero di requisiti obbligatori coperti dal prodotto software. Viene calcolata in forma percentuale utilizzando la seguente formula:

$$\text{Copertura req. obbligatori}[\%] = \frac{\# \text{req. obbligatori soddisfatti}}{\# \text{req. obbligatori totali}}$$

##### 3.3.4.4.2 Copertura requisiti desiderabili

- **Codice:** MS2;
- **Descrizione:** Questa metrica permette di tracciare il numero di requisiti desiderabili coperti dal prodotto software. Viene calcolata in forma percentuale utilizzando la seguente formula:

$$\text{Copertura req. desiderabili}[\%] = \frac{\# \text{req. desiderabili soddisfatti}}{\# \text{req. desiderabili totali}}$$

##### 3.3.4.4.3 Complessità ciclomatica

- **Codice:** MS3;
- **Descrizione:** La complessità ciclomatica è una metrica software che indica la complessità di un programma misurando il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. Nel grafo sopracitato 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. Questo indice può essere applicato indistintamente a singole funzioni, moduli, metodi o package di un programma. Si vuole utilizzare tale metrica per limitare la complessità durante la fase di sviluppo. Durante il testing è utile per determinare il numero di casi di test necessari, infatti l'indice di complessità è un limite superiore al numero di test necessari per raggiungere la copertura completa del modulo testato. Viene calcolata da *IntelliJ IDEA*.

##### 3.3.4.4.4 Accoppiamento tra classi

- **Codice:** MS4;
- **Descrizione:** 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. Un valore elevato di questa metrica non è desiderabile poiché evidenzia una bassa modularità del codice. Viene calcolata da *IntelliJ IDEA*.

#### 3.3.4.4.5 Nested block depth

- **Codice:** MS5;
- **Descrizione:** Rappresenta il massimo numero di livelli di annidamento delle strutture di controllo nei metodi. Un valore elevato di tale indice implica un'alta complessità del codice. Viene calcolata da *IntelliJ IDEA*.

#### 3.3.4.4.6 Variabili non utilizzate e non definite

- **Codice:** MS6;
- **Descrizione:** La presenza di variabili non utilizzate viene considerata *pollution<sub>G</sub>* pertanto non viene tollerata. Tali occorrenze vengono rilevate analizzando l'*Abstract syntax tree<sub>G</sub>* (AST) confrontando le variabili dichiarate e quelle di inizializzazione.

#### 3.3.4.4.7 Numero di parametri per metodo

- **Codice:** MS7;
- **Descrizione:** Un numero elevato di parametri per un metodo potrebbe evidenziare un metodo troppo complesso.

#### 3.3.4.4.8 Numero di funzioni di interfaccia per package

- **Codice:** MS8;
- **Descrizione:** Un numero elevato di funzioni d'interfaccia in un package evidenzia un possibile errore di progettazione.

#### 3.3.4.4.9 Complessità pesata dei metodi

- **Codice:** MS9;
- **Descrizione:** Rappresenta la somma pesata dei metodi di una classe, dove il peso di un metodo è dato dalla sua complessità ciclomatica. Tale metrica, se si attesta su valori bassi, garantisce una buona riusabilità dei metodi, mentre valori elevati indicano una dimensione eccessiva della classe. Viene calcolata da *IntelliJ IDEA*.

#### 3.3.4.4.10 Numero di campi dati per classe

- **Codice:** MS10;
- **Descrizione:** Calcola il numero totale di campi dati per ogni classe, escludendo i campi dati statici ereditati. Tale metrica è utile per valutare il grado di manutenibilità e comprensibilità del codice della classe. Viene calcolata da *IntelliJ IDEA*.

#### 3.3.4.4.11 Copertura del codice

- **Codice:** MS11;
- **Descrizione:** Indica in percentuale il codice che è stato testato con esito positivo tramite i test eseguiti, un valore elevato di tale metrica diminuisce la probabilità di errori, andando a garantire un livello maggiore di affidabilità del software. Viene calcolata tramite la seguente formula:

$$Copertura\ codice[\%] = \frac{\#test\ passati}{\#totale\ test\ richiesti}$$

#### 3.3.4.4.12 Bugs per lines of code

- **Codice:** MS12;
- **Descrizione:** Questa metrica misura il numero di bugs trovati su un certo quantitativo di linee di codice. L'aumentare del sorgente implica un incremento delle probabilità di nascondere errori, per questo è bene mantenere il codice più chiaro e semplice possibile. Con la crescita del prodotto è utile monitorare il rapporto tra i difetti trovati e il codice incrementale, tale indice dovrebbe restare costante o diminuire nel tempo. Il gruppo fissa questa metrica ad un massimo di 60, considerando il fatto che nessun membro ha conoscenze dello stack tecnologico utilizzato, l'obiettivo è di giungere alla *Revisione di Accettazione* con valori compresi tra 0 e 20. Lo sfioramento di tali valori determina l'intervento del *Responsabile* di progetto che dovrà individuare tempestivamente la causa del problema.

## 3.4 Verifica

### 3.4.1 Scopo

Il processo di verifica è finalizzato al controllo delle attività svolte dal gruppo per garantire uno sviluppo efficace ed efficiente del prodotto richiesto.

### 3.4.2 Aspettative

Una corretta *implementazione* di tale *processo* permette di individuare:

- obiettivi di qualità;
- sistemi oggettivi di misurazione della qualità;
- pianificazione efficiente dei test.

### 3.4.3 Descrizione

Il *processo* prevede la successione di due principali attività:

- **Analisi:** si propone di attuare un'analisi dettagliata del documento *codice sorgente<sub>G</sub>* posto sotto *verifica* attraverso due tecniche:
  - analisi statica;
  - analisi dinamica.
- **Test:** attività in cui si definiscono tutti i test da effettuare sul sistema.

### 3.4.4 Attività

#### 3.4.4.1 Analisi

##### 3.4.4.1.1 Analisi statica

Grazie all'analisi statica si possono individuare errori all'interno dei documenti e nel *codice sorgente*. Si realizza attraverso l'uso di due tecniche:

- **Walkthrough:**  
consiste in una lettura critica del documento *codice sorgente* al fine di individuare potenziali errori o anomalie, con la consapevolezza di non conoscere la natura degli stessi. Ogni potenziale errore verrà discusso con gli autori per evitare incomprensioni e per pianificare la correzione degli stessi. Essendo una tecnica non efficiente, in quanto il coinvolgimento di persone che non ricoprono ruoli di *verifica* può portare ad un rallentamento del *processo*, dovuto ai più comuni problemi organizzativi, sarà utilizzata per lo più nella prima parte del progetto, ossia quando i membri non avranno una completa conoscenza delle *Norme di Progetto* e del *Piano di Qualifica*. Attraverso l'uso reiterato dell'attività *Walkthrough* si otterrà una **lista di controllo** contenente gli errori più comuni riscontrati fino a quel momento. Non appena sarà redatta una lista di controllo sufficientemente ampia si migrerà verso una tecnica di analisi detta *inspection* a sfavore della tecnica *Walkthrough*;
- **Inspection:**  
consiste nell'ispezione mirata dei documenti/*codice sorgente* cercando gli errori segnalati nella lista di controllo, nel minor tempo possibile e con la massima efficienza. Grazie all'ampliamento della lista di controllo, attraverso l'inserimento degli errori individuati ad ogni iterazione dell'analisi, oltre che a quelli più comuni individuati con la tecnica *Walkthrough*, e all'incremento di esperienza del *Verificatore*, il *processo* diventerà via via più veloce ed efficiente avendo inoltre un costo inferiore in quanto solitamente svolto da svolto da una sola persona.

#### 3.4.4.1.2 Analisi dinamica

L'analisi dinamica si applica solamente al prodotto software e deve essere svolta durante l'esecuzione dello stesso, con lo scopo di verificarne il corretto funzionamento attraverso l'uso di *test<sub>G</sub>*. Questi devono permettere l'identificazione di errori o anomalie, con l'obiettivo di catalogarli e pianificare una loro correzione. I test devono essere ripetibili, ossia dato lo stesso input e lo stesso ambiente, si ottiene lo stesso output. Affinché la ripetibilità sia possibile per ogni test vengono definiti i seguenti parametri:

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

#### 3.4.4.1.3 Verifica documentazione

Si consiglia al *Verificatore* di suddividere la *verifica* della documentazione nelle seguenti parti:

- **Contenuto:** si proceda in prima analisi alla *verifica* che il contenuto del documento sia coerente al contesto e concetto che si voleva esprimere in ogni sua parte;
- **Norme tipografiche e ortografiche:** successivamente si proceda con il controllo degli errori di natura tipografica e ortografica catalogati nella *Lista anomalie documentazione* nella sezione *Errori tipografici e ortografici*, avendo comunque occhio per eventuali errori non segnalati;
- **Costrutti  $\LaTeX$ :** successivamente si proceda con la compilazione del documento *.tex* e si verifichi che non siano presenti errori catalogati nella *Lista anomalie documentazione* nella sezione *Errori  $\LaTeX$* , avendo comunque occhio per eventuali errori non segnalati.

#### 3.4.4.1.4 Lista anomalie documentazione

Di seguito la lista delle anomalie riscontrate con maggiore frequenza durante l'analisi della documentazione:

- **Comandi  $\LaTeX$  deprecati:** l'utilizzo di costrutti obsoleti non più utilizzati;
- **Caratteri *underscore*:** Utilizzo di caratteri *Underscore* senza prima anteporre una *backslash*;
- **Date:** Formato delle date errato o non conforme alle *Norme di Progetto v3.0.0*;
- **Elenchi puntati:** Elenchi puntati non conformi alle norme, in particolare le frasi spesso non terminano con il punto e virgola;
- **Accenti errati:** Avverbi che necessitano di accenti acuti scritti invece con accenti gravi.

#### 3.4.4.1.5 Verifica del software

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

### 3.4.5 Procedure

#### 3.4.5.1 Controllo qualità di prodotto

Le linee guida per la gestione della qualità di processo seguono il modello *PDCA* descrivendo come devono essere attuate le procedure di controllo:

- la pianificazione deve essere dettagliata;
- le attività devono essere monitorate;
- le risorse necessarie per conseguire gli obiettivi devono essere definite;
- il controllo deve servirsi delle metriche per verificare il miglioramento della qualità del processo.

La pianificazione delle attività volte al miglioramento continuo dei processi sono descritte nel *Piano di Progetto v3.0.0*.

#### 3.4.5.2 Controllo qualità processo

Il controllo per la qualità di prodotto definisce i seguenti processi:

- **Software Quality Assurance (*SQA<sub>G</sub>*)**: deve assicurare che i processi pianificati siano appropriati e implementati secondo la pianificazione, e che esistano sistemi di misurazione dei processi. Questo processo deve essere preventivo e non correttivo;
- **Verifica**: si occupa di accertare che l'esecuzione delle attività di processi svolti non abbia introdotto errori nel prodotto. La verifica viene svolta sui prodotti dei processi per accertare il rispetto delle regole, convenzioni e procedure predeterminate;
- **Validazione**: si occupa di verificare che i prodotti finali siano conformi alle attese.

#### 3.4.5.3 Gestione anomalie

Il Verificatore durante il controllo stende una lista di anomalie che poi comunicherà al Responsabile. Successivamente il Responsabile si occuperà di assegnare la correzione delle anomalie trovate.

#### 3.4.5.4 Gestione modifiche

Ogni volta che durante la verifica si presenta un problema, che necessità di una modifica, verrà inoltrata una richiesta di modifica al Responsabile con la seguente struttura:

- **autore**: il nome e cognome di chi fa la richiesta;
- **documento/file**: indica quale documento/file;
- **motivazione**: spiega perché il motivo della richiesta.

Il Responsabile controlla se esiste un problema e nel caso ci fosse assegna la modifica, inoltre aggiunge alla richiesta se è stata approvata oppure rifiutata.

#### 3.4.5.5 Test

Di seguito riportiamo i vari tipi di test possibili in base alla situazione:

##### 3.4.5.5.1 Test di unità

Il *test di unità* verifica che ogni minima componente software autosufficiente funzioni correttamente. Il test di unità verificherà il corretto funzionamento di parti di programma permettendo così una migliore individuazione degli errori. Questo test dimostrerà se l'*unità* funziona correttamente, semplificando modifiche ed eventuali integrazioni.

#### 3.4.5.5.2 Test di integrazione

Il *test di integrazione* è la diretta estensione del *test di unità*. Il test *verifica* il corretto funzionamento dell'interazione delle *unità*, dopo che esse hanno superato con successo il test di *unità*. Il test avviene attraverso l'esecuzione combinata di più parti, man mano che il *processo* avanza aumenterà il numero di *interfacce* tra le diverse *unità* testate, con la creazione di un gruppo di *moduli e interfacce*, l'obiettivo finale è di eseguire il test su tutte le *unità* e i *moduli* che fanno parte di uno stesso gruppo contemporaneamente.

#### 3.4.5.5.3 Test di regressione

Il *test di regressione* viene eseguito ad ogni modifica dell'*implementazione* del sistema. Verranno eseguiti nuovamente tutti i test disponibili con lo scopo di accertarsi che ogni componente precedentemente funzionante funzioni correttamente dopo le modifiche effettuate.

#### 3.4.5.5.4 Test di sistema

Il *test di sistema* viene attuato quando si pensa di avere una versione definitiva del prodotto e si vuole validarla. Viene quindi verificato se sono stati rispettati i requisiti fissati e che siano stati completamente incorporati dal prodotto finale.

#### 3.4.5.5.5 Codice identificativo

Ogni Test ha la seguente struttura:

- Codice identificativo;
- Descrizione;
- Stato.

Ogni test ha un codice univoco identificativo con il seguente formalismo:

**T{tipologia}{codice identificativo}**

Dove:

- **lettera**: identifica uno dei seguenti test:
  - **U**: test di *unità*;
  - **I**: test di integrazione;
  - **S**: test di sistema;
  - **R**: test di regressione.
- **codice identificativo**: assume i valori seguenti in base al tipo di test:
  - **codice numerico**: per i test di unità, integrazione e regressione, è un numero progressivo a partire da 1;
  - **codice requisito**: per i test di sistema. Gli viene assegnato il codice univoco associato ad ogni requisito nel documento *Analisi dei Requisiti v3.0.0*.

Infine lo Stato del test può assumere uno dei seguenti valori:

- implementato;
- superato
- non implementato;
- non eseguito;
- non superato

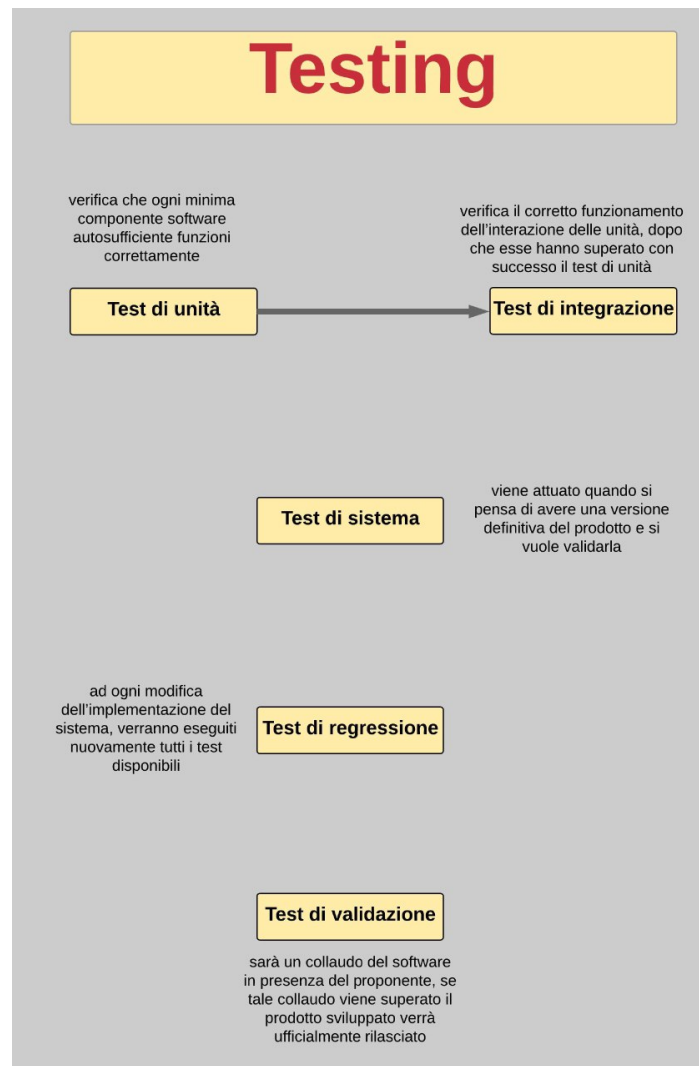


Figura 8: Test attuabili in fase di verifica del software

## 3.5 Validazione

### 3.5.1 Scopo

Il *processo* di *validazione* ha lo scopo di verificare la conformità del prodotto software a quanto pianificato, questo avverrà dopo alcuni cicli di *verifica* affinché la *validazione* sia di esito positivo con buona probabilità.

### 3.5.2 Aspettative

Una corretta esecuzione della *validazione* permette di realizzare:

- un metodo di *validazione*;
- le condizioni affinché il prodotto sia valido;
- la corrispondenza positiva tra prodotto finito e richieste avanzate.



### 3.5.3 Descrizione

Consiste nel controllo della validità e della correttezza dei risultati ottenuti attraverso i *test di validazione*, rispetto ai requisiti e alle caratteristiche richieste dal *proponente*. Il Verificatore dovrà eseguire nuovamente tutti i test, controllando attentamente i risultati dei singoli test, soprattutto quelli di *validazione*. Il Responsabile deve esaminare i risultati ottenuti per determinare se è possibile proseguire e quindi approvare il documento o software.

### 3.5.4 Procedure

#### 3.5.4.1 Validazione documentazione

1. viene assegnata ad un *Verificatore* il compito di eseguire la verifica su di un documento;
2. viene calcolato l'indice *Gulpease* del documento;
3. se i risultati rilevati vengono ritenuti soddisfacenti, il documento viene approvato.

#### 3.5.4.2 Validazione documentazione

1. esecuzione dei test di sistema;
2. esecuzione dei test di validazione;
3. se i risultati rilevati vengono ritenuti soddisfacenti, il documento viene approvato.

#### 3.5.4.3 Test

##### 3.5.4.3.1 Test di validazione

Il test di validazione sarà un collaudo del software in presenza del *proponente*, se tale collaudo viene superato il prodotto sviluppato verrà ufficialmente rilasciato.

##### 3.5.4.3.2 Codice identificativo

Ogni Test ha la seguente struttura:

- Codice identificativo;
- Descrizione;
- Stato.

Ogni test di validazione ha un codice univoco identificativo con il seguente formalismo:

**TV { codice requisito }**

Dove:

- **codice requisito:** identifica il codice univoco associato ad ogni requisito nel documento *Analisi dei Requisiti v3.0.0*.

Infine lo Stato del test può assumere uno dei seguenti valori:

- implementato;
- superato
- non implementato;
- non eseguito;
- non superato

### 3.5.5 Collaudo

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

## 3.6 Strumenti

### 3.6.1 Metriche

### 3.6.2 Verifica ortografica

Per quanto riguarda la correttezza verrà usa il controllore ortografico, strumento implementato da *TexStudio* che sottolinea in rosso le parole errate secondo il dizionario della lingua italiana.

### 3.6.3 Analisi statica

- **JSHINT:** OpenSource per rilevare gli errori e i potenziali problemi per *javascript*; <http://www.jshint.com/>
- **Closure Compiler:** permette di compilare il codice *javascript*; <https://closure-compiler.appspot.com/home>

### 3.6.4 Analisi dinamica

- **Mocha:** framework con funzionalità per l'esecuzione di test *javascript* <http://mochajs.org/>
- **Karma:** ambiente di testing utile ad effettuare test su browser e dispositivi mobili <http://karma-runner.github.io/0.13/index.html>

## 4 Processi Organizzativi

### 4.1 Scopo

Lo scopo di questo *processo* è la creazione del documento *Piano di Progetto*, utile ai membri del gruppo per organizzare e gestire i ruoli di ogni suo componente.

### 4.2 Aspettative

Le aspettative del *processo* sono:

- produzione del documento *Piano di Progetto*;
- definizione ruoli dei membri del gruppo;
- definizione di un piano per l'esecuzione dei compiti programmati.

### 4.3 Descrizione

Orari di lavoro:

- dalle 9.00 alle 13.00;
- dalle 14.00 alle 18.00.

Viene trattata la gestione dei seguenti argomenti:

- ruoli di progetto;
- comunicazioni e incontri;
- strumenti di coordinamento e *versionamento*;
- rischi.

### 4.4 Ruoli di progetto

Tutti i ruoli saranno ricoperti da ciascun membro del gruppo a rotazione, in modo che ogni componente possa assumere almeno una volta ciascun ruolo. Nel documento *Piano di Qualifica v3.0.0* vengono organizzate e pianificate le attività da assegnare ai vari ruoli, di seguito elencati.

#### 4.4.1 Amministratore di progetto

L'Amministratore di Progetto controlla e amministra l'ambiente di lavoro con piena responsabilità sulla capacità operativa e sull'efficienza. Le sue responsabilità in particolare sono:

- ricerca di strumenti che migliorino l'ambiente di lavoro e che lo automatizzino dove possibile;
- gestione del *versionamento*;
- risoluzione dei problemi di gestione dei processi;
- controllo della *qualità* sul *prodotto*.

#### 4.4.2 Responsabile di progetto

Il *Responsabile* di Progetto è il punto di riferimento sia per il *committente*, che per il fornitore. Approva inoltre le scelte prese dal gruppo e se ne assume responsabilità. I compiti di questo ruolo sono:

- approvazione della documentazione;
- approvazione dell'offerta economica;
- gestione delle risorse umane;
- pianificazione e coordinamento delle attività di progetto;
- analisi e gestione dei rischi.

#### 4.4.3 Analista

L'Analista si occupa dell'interpretazione ed analisi dei problemi e del dominio applicativo. Il suo ruolo è fondamentale per il successo del progetto, nonostante ne partecipi per un periodo di tempo limitato (soprattutto nella parte iniziale di *Analisi dei Requisiti*). Le sue responsabilità sono:

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

#### 4.4.4 Progettista

Il Progettista gestisce gli aspetti tecnici e tecnologici ed è responsabile delle scelte architetture del progetto. I suoi compiti sono:

- effettuare scelte efficienti, ottimizzate e attuabili sugli aspetti tecnici del progetto;
- rendere facilmente mantenibile e scalabile il progetto.

#### 4.4.5 Verificatore

Il Verificatore è la persona incaricata di garantire una esaustiva *verifica* delle attività di progetto. Deve conoscere molto bene le normative di progetto. Le sue responsabilità sono:

- controllo che tutte le attività seguano le normative di progetto prestabilite.

#### 4.4.6 Programmatore

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

- implementare le decisioni del *Progettista*;
- scrivere un codice robusto e facilmente mantenibile, secondo le *Norme di Progetto*;
- *versionamento* del codice *prodotto*;
- realizzazione degli strumenti per la *verifica* e *validazione* del software.

## 4.5 Gestione di progetto

### 4.5.1 Gestione delle comunicazioni

#### 4.5.1.1 Comunicazioni interne

Le comunicazioni interne sono gestite tramite un'applicazione chiamata *Slack*, creata per gestire al meglio gruppi di lavoro. Essa offre infatti più servizi rispetto a una normale applicazione di messaggistica istantanea, come ad esempio la suddivisione di canali per lo scambio di informazioni tra i membri del gruppo.

#### 4.5.1.2 Comunicazioni esterne

Le comunicazioni esterne sono affidate al *Responsabile* di Progetto, che utilizza l'apposita casella di posta elettronica:

**gitKraffen.swe16@gmail.com**

Il *Responsabile* di Progetto, inoltre, terrà informati tutti i membri del gruppo sulle discussioni con le componenti esterne tramite i canali di comunicazione interna.

### 4.5.2 Gestione degli incontri

#### 4.5.2.1 Incontri interni

Il *Responsabile* di Progetto ha il compito di organizzare gli incontri interni utilizzando i canali di comunicazione interni. Ogni componente ha il diritto di richiedere l'organizzazione di un incontro al *Responsabile* di Progetto, il quale poi deve decidere se accettare o rifiutare la proposta. Una volta accertata la partecipazione, l'incontro sarà calendarizzato. Inoltre, un membro del gruppo verrà incaricato dal *Responsabile* di Progetto di stendere il verbale dell'incontro avvenuto, secondo le norme.

#### 4.5.2.2 Incontri esterni

Il *Responsabile* di Progetto ha il compito di gestire e organizzare gli incontri esterni con il *proponente* o *committente*. Come per gli incontri interni, ogni componente del gruppo ha diritto a richiedere l'organizzazione di un incontro esterno. Il *Responsabile* di Progetto deciderà poi se organizzare o meno l'incontro, dopo aver accertato i motivi della richiesta. Nel caso venga deciso di contattare l'entità esterna e quest'ultima sia d'accordo, il *Responsabile* di Progetto comunicherà a tutti i membri del gruppo le informazioni riguardanti data, ora e luogo dell'incontro. Come nel caso degli incontri interni, un componente del gruppo sarà incaricato di stendere il verbale dell'incontro.

### 4.5.3 Gestione degli strumenti di coordinamento

#### 4.5.3.1 Ticketing

Per dividere il carico di lavoro in *task<sub>G</sub>*, che saranno divisi tra tutti i membri del gruppo viene utilizzata la piattaforma *Trello<sub>G</sub>*. *Trello* permette di avere un quadro completo delle attività, con le relative scadenze fissate e dello stato di ogni singolo *task* (completo o no). Il compito di suddividere il carico di lavoro in *task* spetta al *Responsabile* di Progetto. La procedura per l'assegnazione di un *task* è la seguente:

- inserire un titolo al *task* e la data di inizio e fine;
- inserire una breve descrizione del compito assegnato;
- inserire il nome del membro del gruppo che deve svolgere l'attività, oppure lasciare vuoto in caso il compito non sia da assegnare a un componente in particolare;

- il membro del gruppo specificato (o un membro del gruppo che si occupa di quelle mansioni) prende in carico il compito.

#### 4.5.4 Gestione degli strumenti di versionamento

##### 4.5.4.1 Repository

Per il *versionamento* e il salvataggio dei *file* vengono utilizzate due *repository* (una per i documenti, l'altra che contiene i *file* implementativi del progetto) su *GitHub<sub>G</sub>*. L'*Amministratore di Progetto* si deve occupare della creazione delle *repository* e, successivamente, di inserire tutti i membri del gruppo, che dovranno possedere un account personale.

##### 4.5.4.2 Struttura delle repository

I membri del gruppo sono tenuti a rispettare le seguenti norme sull'organizzazione dei *file*.  
Struttura della *repository* contenente i documenti:

- layout<sub>Latex</sub>: cartella contenente tutti i *file* del *Template* e il layout dei documenti *L<sup>A</sup>T<sub>E</sub>X*;
- cartelle per ogni revisione: a loro volta contengono due sottocartelle per i documenti interni ed esterni al gruppo. All'interno è presente una suddivisione dei singoli documenti in cartelle dove in ciascuna di esse sono contenuti tutti i relativi *file L<sup>A</sup>T<sub>E</sub>X*, i *file* pdf e una cartella *img* che contiene le immagini utilizzate nei documenti.

La struttura della *repository* contenente i *file* implementativi sarà definita in modo dettagliato nelle successive revisioni.

##### 4.5.4.3 Tipi di file e .gitignore

Nelle cartelle che contengono tutti i documenti saranno presenti solo *file* con formato .tex, .pdf, .jpg, .png. Le estensioni dei *file* generati automaticamente sono stati aggiunti a *.gitignore*, sono quindi ignorati da *Git*.

##### 4.5.4.4 Norme sui commit

Ogni volta che vengono effettuate modifiche sui *file* della *repository* devono essere specificate le motivazioni. Questo avviene tramite il comando *commit* accompagnato da un messaggio riassuntivo e da una descrizione in cui vanno specificate le liste dei *file* coinvolti e delle modifiche effettuate.

#### 4.5.5 Gestione dei rischi

Il *Responsabile* di Progetto ha il compito di rilevare i rischi indicati nel *Piano di Progetto v3.0.0*. Nel caso ne vengano individuati di nuovi, dovrà aggiungerli nell'analisi dei rischi. La procedura da seguire nella gestione dei rischi è la seguente:

- individuare problemi non calcolati e monitorare rischi già rilevati;
- registrare riscontri previsti e aggiungere nuovi rischi individuati nel *Piano di Progetto v3.0.0*;
- ridefinire, se necessario, le strategie di progetto.

##### 4.5.5.1 Codice identificativo

Ogni rischio è strutturato come segue:

- Codice

- Descrizione
- Probabilità
- Gravità

Per ciascun rischio è assegnato un codice identificativo seguendo questo formalismo:

**R{tipologia}{numero}**

Dove:

- **tipologia**: identifica uno dei seguenti tipi di rischio:
  - **T**: tecnologico;
  - **P**: personale;
  - **O**: organizzativo;
  - **S**: strumentale;
  - **R**: dei requisiti.
- **numero**: incrementale, comincia da 1 e al cambio di tipologia torna a 1.

#### 4.5.6 Formazione personale

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

- Materiale elencato nella sezione *Riferimenti informativi* di ogni documento;
- Siti web e relativa documentazione delle tecnologie riportati in forma di link in nello studio di fattibilità.
- Membri del gruppo più esperti;

### 4.6 Strumenti

#### 4.6.1 Sistema operativo

Il gruppo lavora sui seguenti sistemi operativi:

- Windows 10 Pro x64;
- Ubuntu 16.04 *LTS* x64;
- Windows 10 Home x64.

#### 4.6.2 Slack

*Slack* è uno strumento utile per le comunicazioni interne in un gruppo di lavoro. L'applicazione funziona come un comune sistema di messaggistica, con la possibilità di integrare numerosi servizi tra i quali *GitHub* e *Trello*. Gli utenti possono inoltre suddividere gli argomenti di discussioni in vari canali, in modo che sia organizzata al meglio la comunicazione. L'applicazione fornisce anche una versione desktop gratuita e necessita la registrazione di un dominio per il proprio *team*.

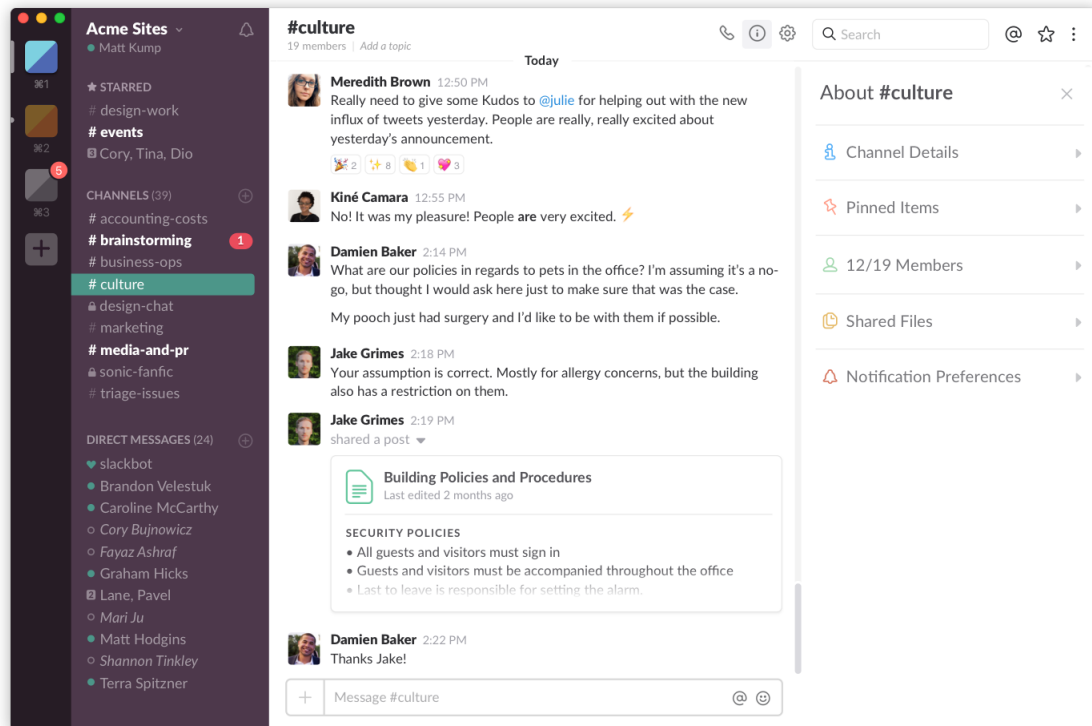


Figura 9: Strumenti: Slack

#### 4.6.3 Trello

*Trello* è uno strumento online per la collaborazione e il *project management*, che aiuta a coordinare il lavoro da eseguire e dà una visione generale sull'andamento del progetto. *Trello* ha diverse funzionalità, tra le più importanti ci sono:

- controllo del flusso dei movimenti in tempo reale;
- gestione del carico di lavoro;
- gestione delle scadenze temporali;
- discussione sui blocchi di lavoro;
- classificazione delle attività per urgenza/importanza.



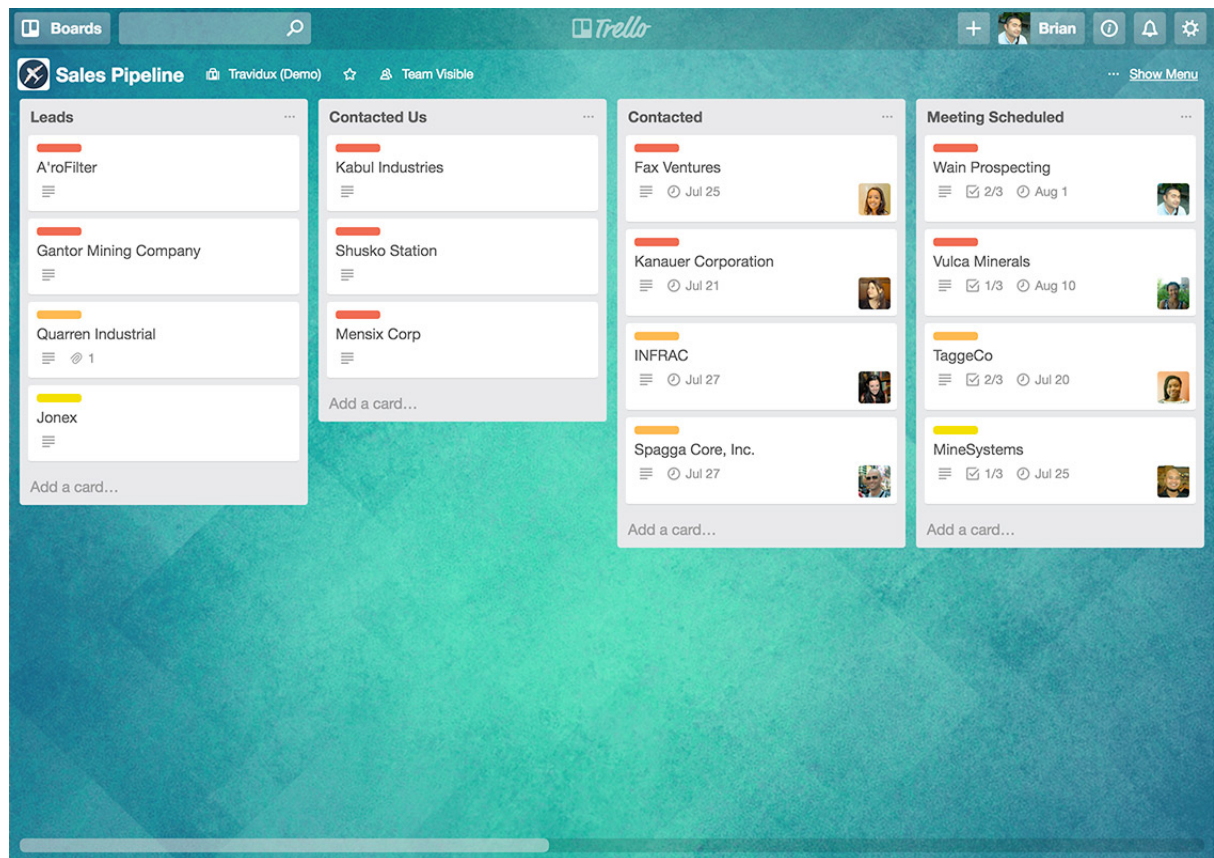


Figura 10: Strumenti: Trello

#### 4.6.4 Git

*Git* è un sistema software di controllo di versione distribuito, creato da Linus Torvalds nel 2005. La versione utilizzata è maggiore o uguale alla 2.16.

#### 4.6.5 GitHub

*GitHub* è un servizio web di *hosting* per lo *sviluppo* di progetti software, che usa il sistema di controllo di versione *Git*. *GitHub* offre diversi piani per *repository* private sia a pagamento, sia gratuiti, molto utilizzati per lo *sviluppo* di progetti OpenSource.

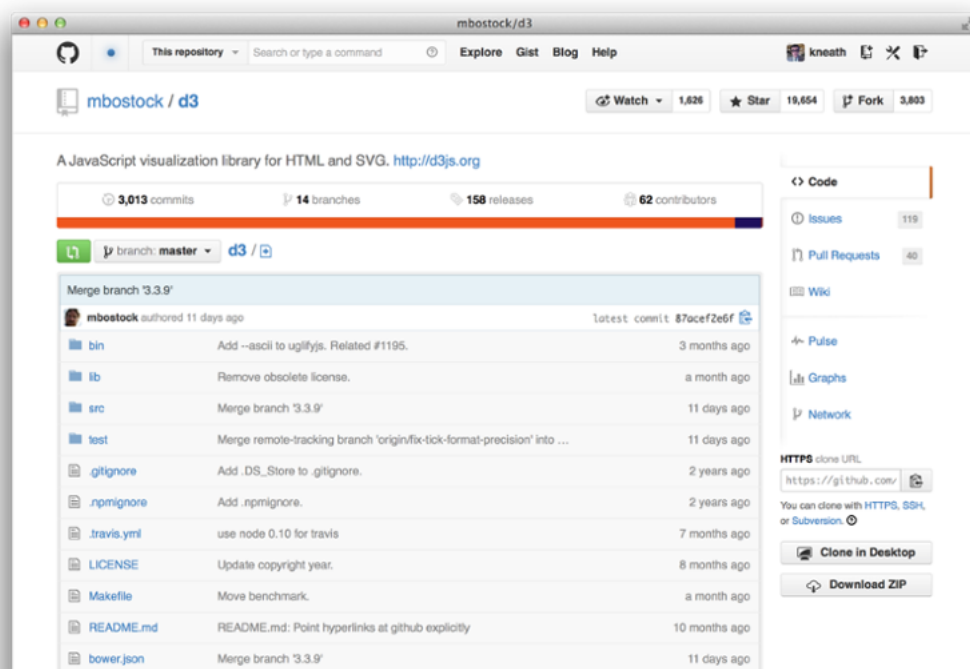


Figura 11: Strumenti: Github