



# Agents of S.W.E.

A SOFTWARE COMPANY

Agents of S.W.E - Progetto "Plugin Grafana"

## Norme di Progetto

<b>Versione</b>	0.0.12
<b>Approvazione</b>	?
<b>Redazione</b>	Luca Violato Bogdan Stanciu Marco Favaro Marco Chilese
<b>Verifica</b>	? ?
<b>Stato</b>	Work in Progress
<b>Uso</b>	Interno
<b>Destinato a</b>	Agents of S.W.E. Prof. Tullio Vardanega Prof. Riccardo Cardin

agentsofswe@gmail.com



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Scopo del Documento . . . . .	1
1.2	Ambiguità e Glossario . . . . .	1
1.3	Riferimenti . . . . .	1
<b>2</b>	<b>Processi Primari</b>	<b>2</b>
2.1	Fornitura . . . . .	2
2.2	Studio di fattibilità . . . . .	2
2.3	Sviluppo . . . . .	2
2.3.1	Analisi dei requisiti . . . . .	2
2.3.1.1	Classificazione dei requisiti . . . . .	2
2.3.1.2	Classificazione dei casi d'uso . . . . .	2
2.3.2	Progettazione . . . . .	2
2.3.3	Codifica . . . . .	2
2.3.3.1	Convenzioni per i nomi: . . . . .	2
2.3.3.2	Convenzioni per la documentazione: . . . . .	3
2.3.3.3	ECMAScript 6: . . . . .	4
<b>3</b>	<b>Processi di supporto</b>	<b>9</b>
3.1	Documentazione . . . . .	9
3.1.1	Descrizione . . . . .	9
3.1.2	Ciclo di vita documentazione . . . . .	9
3.1.3	Template . . . . .	9
3.1.4	Struttura documenti . . . . .	9
3.1.4.1	Prima pagina . . . . .	9
3.1.4.2	Piè di pagina . . . . .	10
3.1.4.3	Nomenclatura . . . . .	10
3.1.4.4	Tabelle . . . . .	10
3.1.4.5	Struttura indice . . . . .	10
3.1.4.6	Registro Modifiche . . . . .	11



3.1.5	Norme tipografiche	11
3.1.6	Documenti Correnti	11
3.1.7	Ambiente	12
3.2	Versionamento	12
3.2.1	Controllo di versione	12
3.2.1.1	Struttura del repository	12
3.2.1.2	Processo di implementazione	13
3.2.1.3	Ciclo di vita dei branch	13
3.2.1.4	Rilascio di versione	13
3.2.2	Configurazione versionamento	14
3.2.2.1	Remoto	14
3.2.2.2	Locale	14
3.3	Gestione di progetto	14
3.3.1	Configurazione strumenti di organizzazione	14
3.3.1.1	Inizializzazione	15
3.3.1.2	Aggiunta milestones	15
3.3.2	Ciclo di vita delle tasks	15
3.3.2.1	Apertura	15
3.3.2.2	Completamento	15
3.3.2.3	Richiesta di revisione	16
3.3.2.4	Chiusura	16
3.3.2.5	Riapertura	16
<b>4</b>	<b>Processi Organizzativi</b>	<b>17</b>
4.1	Processi di Coordinamento	17
4.1.1	Gestione Comunicazioni	17
4.1.1.1	Comunicazioni Interne	17
4.1.1.2	Comunicazioni Esterne	17
4.1.2	Gestione Riunioni	18
4.1.2.1	Riunioni Interne	18
4.1.2.2	Riunioni Esterne	18
4.1.2.3	Verbale di Riunione	18
4.2	Ruoli di Progetto	19
4.2.1	Responsabile di Progetto	19
4.2.2	Amministratore di Progetto	20
4.2.3	Analista	20
4.2.4	Progettista	20
4.2.5	Programmatore	21



---

4.2.6	Verificatore . . . . .	21
4.2.7	Rotazione dei Ruoli . . . . .	21
4.3	Procedure . . . . .	22
4.3.1	Gestione degli Strumenti di Versionamento . . . . .	22
4.3.1.1	Struttura Repository . . . . .	22
4.3.1.2	Uso dei Branch . . . . .	22
4.3.1.3	Tipi di files . . . . .	23
4.3.1.4	Norme delle Commit . . . . .	23
4.3.1.5	Norme dei Merge tra Branch . . . . .	23
4.3.2	Gestione degli Strumenti di Coordinamento . . . . .	24
4.3.2.1	Tasks . . . . .	24
4.3.2.2	Tickets . . . . .	24
4.4	Strumenti . . . . .	25
4.4.1	Sistema Operativo . . . . .	25
4.4.2	Versionamento e Issue Tracking . . . . .	25
4.4.2.1	Git . . . . .	25
4.4.2.2	GitHub . . . . .	25
4.4.3	Comunicazione . . . . .	25
4.4.3.1	Telegram . . . . .	25
4.4.3.2	Slack . . . . .	26
4.4.4	Diagrammi di Gantt . . . . .	26
4.4.5	Diagrammi UML . . . . .	26
4.5	Formazione del Gruppo . . . . .	26
5	Changelog . . . . .	28



# 1 Introduzione

## 1.1 Scopo del Documento

## 1.2 Ambiguità e Glossario

## 1.3 Riferimenti

## 2 Processi Primari

### 2.1 Fornitura

In questa sezione del documento vengono trattate le norme che il team **Agents of S.W.E.** decide e si impegna a rispettare, con lo scopo di proporsi e divenire fornitori nei confronti dell'azienda proponente, *Zucchetti S.p.A.*, e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin nell'ambito della progettazione, sviluppo e consegna del prodotto "*Plugin per Grafana*".

### 2.2 Studio di fattibilità

### 2.3 Sviluppo

#### 2.3.1 Analisi dei requisiti

##### 2.3.1.1 Classificazione dei requisiti

##### 2.3.1.2 Classificazione dei casi d'uso

#### 2.3.2 Progettazione

#### 2.3.3 Codifica

Di seguito vengono definite delle norme che devono essere adottate dai Programatori per garantire una buona leggibilità e manutenibilità del codice. Le prime norme che seguiranno sono le più generali, da adottarsi per ogni linguaggio di programmazione adottato all'interno del progetto, in seguito quelle più specifiche per i linguaggi ECMAScript 6<sub>G</sub> , HTML<sub>G</sub> e CSS<sub>G</sub> .

Ogni norma è caratterizzata da un paragrafo di appartenenza, da un titolo, una breve descrizione, e se il caso lo richiede, un esempio.

Il rispetto delle seguenti norme è fondamentale per garantire uno stile di codifica uniforme all'interno del progetto, oltre che per massimizzare la leggibilità e agevolare la manutenzione, la verifica<sub>G</sub> e la validazione<sub>G</sub> .

##### 2.3.3.1 Convenzioni per i nomi:

- I Programatori devono adottare come notazione per la definizione di cartelle, file, metodi, funzioni e variabili il CamelCase<sub>G</sub> .

Di seguito un esempio di corretta nomenclatura:



```
1 //Cartelle
2 ./thisIsAFolder //OK
3 ./ThisIsAFolder //NO
4
5 //File
6 myFile.extension //OK
7 MyFile.extension //NO
8
9 //Funzioni
10 myFunction() { ... } //OK
11 MyFunction() { ... } //NO
```

- Tutti i nomi devono essere **unici** ed **autoesplicativi**, ciò per evitare ambiguità e limitare la complessità .

### 2.3.3.2 Convenzioni per la documentazione:

- Tutti i nomi ed i commenti al codice vanno scritti in **inglese**;
- Nel codice è possibile utilizzare un commento con denominazione **TODO** in cui si vanno ad indicare compiti da svolgere;
- L'intestazione di ogni file deve essere la seguente:

```
1 /**
2  * File: nomeFile
3  * Type: fileType
4  * Creation date: gg/mm/yyyy
5  * Author: Name Surname
6  * Author e-mail: email@example.com
7  * Version: versionNumber
8  *
9  * Changelog:
10 * #entry || Author || Date || Description
11 */
```

- La versione del file nell'intestazione, deve rispettare la seguente formulazione: *X.Y.Z*, dove *X* rappresenta la versione principale, *Y* la versione parziale della relativa versione principale, e *Z* l'avanzamento rispetto ad *Y*.  
I numeri di versione del tipo *X.0.0*, dalla *1.0.0*, vengono considerate versioni stabili, e quindi versioni da testare per saggiarne la qualità .

**2.3.3.3 ECMAScript 6:** Seguendo le indicazioni presenti nella documentazione<sup>1</sup> dell'azienda fornitrice di *Grafana*, la piattaforma per cui si intende sviluppare il plugin, il team ha deciso di adottare come linguaggio di programmazione principale ECMAScript 6<sup>2</sup>.

ECMAScript 6 viene standardizzato da **ECMA<sub>G</sub>**<sup>3</sup> nel giugno 2015 con la sigla **ECMA-262**<sup>4</sup>.

Come stile di codifica si adottano le linee guida proposte da **Airbnb JavaScript Style Guide**<sup>5</sup>. Per la verifica dell'adesione a tali norme, i Programmatori devono utilizzare, come suggerito dalla documentazione proposta da *Grafana*, **ESLint<sub>G</sub>**<sup>6</sup>. In particolare i Programmatori devono rispettare 5 linee guida proposte dalla documentazione ufficiale di *Grafana*:

1. Se una variabile non viene riutilizzata, deve essere dichiarata come **const**;
2. Utilizzare preferibilmente, per la definizione di variabili, la keyword **let**, anziché **var**;
3. Utilizzare il marcatore freccia (**=>**), in quanto non oscura il **this**:

```
1 testDatasource() {  
2   return this.getServerStatus()  
3   .then(status => {  
4     return this.doSomething(status);  
5   })  
6 }
```

Invece che:

```
1 testDatasource() {  
2   var self = this;  
3   return this.getServerStatus()  
4   .then(function(status) {  
5     return self.doSomething(status);  
6   })  
7 }
```

4. Utilizzare l'oggetto *Promise*:

<sup>1</sup><http://docs.grafana.org/plugins/developing/development/>

<sup>2</sup>Linguaggio divenuto standard ISO: ISO/IEC 16262:2011, e relativo aggiornamento ISO/IEC 22275:2018.

<sup>3</sup><http://www.ecma-international.org/>

<sup>4</sup><https://www.ecma-international.org/ecma-262/6.0/>

<sup>5</sup><https://github.com/airbnb/javascript>

<sup>6</sup><https://eslint.org/>



```
1 metricFindQuery(query) {  
2   if (!query) {  
3     return Promise.resolve([]);  
4   }  
5 }
```

Invece che:

```
1 metricFindQuery(query) {  
2   if (!query) {  
3     return this.$q.when([]);  
4   }  
5 }
```

5. Se si utilizza *Lodash* è meglio essere conseguenti, e preferire in ogni caso le funzioni per gli array native di ES6.

Verranno esaminate di seguito le norme in merito allo stile di codifica che i Programmatori dovranno adottare.

### Indentazione

**Norma 1** L'indentazione è da eseguirsi con tabulazione la cui larghezza sia impostata a due (2) spazi per ogni livello.

Di seguito un esempio da ritenersi corretto:

```
1 function() {  
2   ..let x = 2;  
3   ..if (x > 0)  
4     ....return true;  
5   ..else  
6     ....return false;  
7 }
```

Qualsiasi altro tipo di indentazione è da ritenersi scorretta.

**Norma 2** Dopo la graffa principale va inserito uno (1) spazio. Nel seguente modo:

```
1 function() { ... }
```

**Norma 3** Dopo la keyword di un dato statement (*if*, *while*, etc.) va inserito uno (1) spazio. Per un esempio corretto si veda la norma successiva.

**Norma 4** Prima dell'apertura della graffa negli statement di controllo va inserito uno (1) spazio. Nel seguente modo:

```
1 function() {  
2     if (condition) {  
3         ...  
4     }  
5     while (condition) {  
6         ...  
7     }  
8 }
```

**Norma 5** Negli statement di controllo (`if`, `while`, etc) le condizioni concatenate, o annidate, mediante operatori logici, che diventano eccessivamente lunghe NON vanno espresse in un'unica linea, bensì spezzate in più righe. Nel seguente modo:

```
1 function() {  
2     if (condition && condition) {  
3         ...  
4     }  
5  
6     if (  
7         veryLongCondition  
8         && longCondition  
9         && condition  
10    ) {  
11        doSomething();  
12    }  
13 }
```

**Norma 6** Dopo blocchi, o prima di un nuovo statement va lasciata una riga vuota. Nel seguente modo:

```
1 function1() {  
2     if (condition) {  
3         doSomething():  
4     }  
5  
6     return toReturn;  
7 }  
8  
9 function2(){  
10     ...  
11 }
```

**Norma 7** I blocchi di codice multi-riga devono essere contenuti all'interno di graffe. Blocchi costituiti da una singola riga non è necessario che sia contenuti tra graffe: nel caso non vengano utilizzate, la definizione deve essere *inline*, sulla stessa riga. Nel seguente modo:

```
1 if (condition) return true;
2
3 if (condtion) {
4     return true;
5 }
```

### Commenti al codice

Il codice va commentato nel seguente modo:

- `"//"` se il commento occupa una sola riga;
- `"/** ... */"` se il commento occupa più righe.

Nel seguente modo:

```
1 // single line comment
2 if (condition) return true;
3
4 /**
5  * multi line comment, line 1
6  * multi line comment, line 2
7  */
8 if (condtion) {
9     return true;
10 }
```

### Variabili

**Norma 1** Fare riferimento alle norme 1 e 2, delle 5 linee guida enunciate sopra.

**Norma 2** Non utilizzare dichiarazioni multiple di variabili, dichiarare una variabile per riga.

Nel seguente modo:

```
1 // OK
2 var x = 1;
3 var y = 0;
4
```

```
5 // NO
6 var x = 1, y = 0;
```

### Nomi

**Norma 1** Oltre a quanto enunciato nel secondo punto del paragrafo §2.2.3.1, tutti i nomi di funzioni o variabili composti da una singola lettera, o che indichino temporaneità della variabile sono *vietati*: ogni nome deve essere significativo.

### Norma 2

1. I nomi delle variabili, funzioni ed istanze devono utilizzare il **CamelCase**;
2. I nomi delle classi deve lo stile **capWords**.

Nel seguente modo:

```
1 // OK
2 var thisIsAVariable;
3
4 function thisIsAFunction() { ... }
5
6 class ThisIsAClass() {
7     ...
8 }
9
10 // NO
11 var Variable;
12
13 function Function() { ... }
14
15 class myClass() {
16     ...
17 }
```

## 3 Processi di supporto

### 3.1 Documentazione

#### 3.1.1 Descrizione

Questo capitolo descrive i dettagli su come deve essere redatta e verificata la documentazione durante il ciclo di vita del software. Le norme sono tassativamente valide per tutti i documenti formali.

#### 3.1.2 Ciclo di vita documentazione

Il ciclo di vita previsto della documentazione si può suddividere principalmente in tre processi:

- **Sviluppo:** è il processo di stesura, eseguita dal redattore, dove descrive il ticket<sub>G</sub> assegnato dal responsabile. Una volta terminata la fase di scrittura del documento, il redattore lo segnala al responsabile, che assegnerà a un verificatore il compito di analizzare il lavoro svolto;
- **Verifica:** è il processo eseguito dai verificatori designati dal responsabile, il loro compito è controllare che il redattore abbia scritto il documento nella norma e in maniera corretta grammaticalmente e strutturalmente;
- **Approvato:** è il processo conclusivo, in cui il verificatore ha terminato il suo compito di controllo e comunica al responsabile il termine del lavoro. Il responsabile procederà a confermare il documento e ad eseguire il rilascio.

#### 3.1.3 Template

Il **gruppo** ha deciso di strutturare un template  $\text{\LaTeX}$  per dare uniformità a tutti i documenti. Il template facilita e velocizza la stesura, poiché i redattori devono concentrarsi solo ed esclusivamente al contenuto e non alla layout.

#### 3.1.4 Struttura documenti

Ogni documento segue una determinata struttura, predefinita e accordata dal **gruppo**:

##### 3.1.4.1 Prima pagina

La prima pagina di ogni documento ha la stessa struttura: il logo del **gruppo** centrato in alto, con sotto, sempre centrato, il nome del **gruppo** e il capitolato scelto. Appena sotto è posizionato il titolo del documento e una tabella contenente

informazioni relative al documento, ovvero, la versione, i nome dei redattori e dei verificatori, lo stato (che può essere "confermato" o "work in progress"), l'utilizzo che avrà nel progetto (interno o esterno) e i destinatari.

#### 3.1.4.2 Piè di pagina

Il fondo pagina di tutti i documenti è molto pulito, contiene solamente il numero della pagina e, se presenti, i riferimenti bibliografici alle fonti utilizzate nella pagina corrente.

#### 3.1.4.3 Nomenclatura

Le seguenti regole valgono per tutti i documenti eccetto per la lettera di presentazione. La nomenclatura è un aspetto fondamentale che abbiamo deciso di struttura nel seguente modo:

- **vX.Y.Z** : rappresenta la versione del documento con X , Y e Z numeri non negativi:
  - **X**: rappresenta il numero di pubblicazioni ufficiali del documento in passato; se il valore è 0 significa che il documento non è mai stato pubblicato. Ogni qualvolta viene pubblicato Y e Z vengono azzerati e X viene incrementato di una unità;
  - **Y**: identifica il numero di verifiche avvenute con successo, ogni qualvolta viene effettuata una verifica il valore di Z viene azzerato;
  - **Z**: identifica il numero di volte che il documento è stato modificato prima di una pubblicazione e/o verifica.
- Il formato dei file è .tex durante la fase di sviluppo, mentre dopo l'approvazione da parte del responsabile verrà creato un file con formato .pdf che rappresenta la pubblicazione in via ufficiale.

#### 3.1.4.4 Tabelle

TODO : da definire nel prossimo incontro

#### 3.1.4.5 Struttura indice

L'indice è strutturato nel seguente modo: titolo, argomento, e numero pagina. Ovviamente ogni titolo dell'argomento è un link alla pagina contenente lo stesso.

#### 3.1.4.6 Registro Modifiche

Ogni documento, eccezion fatta per verbali e lettera di presentazione, presenta un registro delle modifiche chiamato "Changelog". È strutturato sotto forma di tabella, che contiene in ordine cronologico tutte le modifiche identificabili dalla versione. Ogni riga contiene la data, il nome di chi ha effettuato la modifica e il relativo ruolo, e infine una breve descrizione della modifica effettuata.

#### 3.1.5 Norme tipografiche

- **Glossario:** i termini contenuti nel glossario si possono identificare dal carattere G maiuscolo e corsivo a pedice della parola interessata, per esempio Norme<sub>G</sub>;
- **Grassetto:** le parole in grassetto identificano il titolo di una sezione, sottosezione, paragrafo e un elemento di un elenco puntato;
- **Nome gruppo:** in qualsiasi documento, quando si fa riferimento al gruppo si è deciso di adottare il seguente font: **gruppo**;
- **Date:** sono scritte seguendo il formato YYYY-MM-DD, dove YYYY rappresenta l'anno, MM il mese e DD il giorno;
- **Elenchi puntati:** ogni elemento di un elenco puntato deve essere seguito da un punto e virgola eccezion fatta per l'ultimo che sarà seguito dal punto;
- **Corsivo:** è utilizzato per citare tecnologie esterne;
- **Collegamenti esterni:** i link esterni sono evidenti per il colore blu, in modo da distinguerli dal testo.

#### 3.1.6 Documenti Correnti

Sono descritti brevemente i documenti formali da consegnare:

- **Analisi dei Requisiti:** ha un utilizzo prettamente esterno, e ha l'obiettivo di esporre e scomporre i requisiti<sub>G</sub> del progetto. Contiene i casi d'uso e i diagrammi di interazione con l'utente. Viene scritto dagli analisti dopo una profonda analisi del capitolato e vari colloqui con il proponente;
- **Glossario:** ha un utilizzo prettamente esterno e ha lo scopo di dare una definizione ai termini più specifici usati nei documenti formali;

- **Norme di Progetto:** è utilizzato internamente, ed espone gli standard e le direttive utilizzate dal **gruppo** per sviluppare il progetto in tutta la sua interezza;
- **Piano di Progetto:** ha un utilizzo esterno, ed espone come il gruppo ha deciso di impiegare le risorse di tempo e umane;
- **Piano di Qualifica:** utilizzo esterno, descrive gli standard e gli obiettivi che il **gruppo** dovrà raggiungere per garantire la qualità di prodotto e processo;
- **Studio di Fattibilità:** documento interno, espone pregi e difetti di tutti i capitolati in esame. Descrive i motivi per cui il **gruppo** ha scelto il capitolato corrente.

### 3.1.7 Ambiente

Per uniformare e strutturare al meglio la scrittura dei documenti il **gruppo** ha adottato il formato  $\text{\LaTeX}$ . *TexMaker*<sup>7</sup> è utilizzato per la stesura, il **gruppo** ha optato per questo editor perchè open-source e integra un controllo ortografico della lingua italiana.

## 3.2 Versionamento

La necessità di più componenti del **gruppo** di cooperare su uno stesso documento, porta alla soluzione di utilizzare un sistema di versionamento distribuito.

### 3.2.1 Controllo di versione

Il sistema di versionamento, utilizzato in questa fase di **RR**, è **git**, con il supporto hosting di **GitHub**.

**3.2.1.1 Struttura del repository** La struttura del repository segue il workflow **gitflow** di **Driessen at nvie**, idealizzato attorno il concetto di **release** del prodotto. Questo produce un framework robusto attorno al quale si possono gestire progetti di grandi dimensioni. I due branch principali sono il **master** e in parallelo ad esso il **develop**. Il **master** viene considerato il branch *main*, dove il codice sorgente della *testa* riflette sempre lo stato di "*production-ready*", mentre il ramo di **develop** è considerato il branch principale dove vengono effettuate le ultime modifiche per il prossimo rilascio del prodotto.

---

<sup>7</sup><http://www.xmlmath.net/texmaker/>



**3.2.1.2 Processo di implementazione** L'implementazione dei documenti avviene tramite gli strumenti utilizzati nel paragrafo §3.1

Quest'ultimi, in fase di compilazione producono dei file di poca rilevanza con estensioni come: *.log*, *.out*, *.idx*, *.aux*, *.gz*, *.aux*, *.toc*, i quali verranno ignorati come da configurazione, ma soprattutto file di più rilevanza come *.pdf*, *.tex*, i quali verranno versionati dal sistema di git preinstallato. Una volta creati/modificati i documenti, si procede con il `git` commit di essi. Il commit riporta un *cambiamento* al file, con un messaggio allegato ad esso che ne descrive le modifiche apportate o un comando apposito per chiudere alcune task con tale commit. Dopo di che, il commit viene `git` pushato nel branch appropriato, a seconda dei criteri descritti nel prossimo paragrafo.

### 3.2.1.3 Ciclo di vita dei branch

1. **Master:** branch *main* del repository, esso rappresenta lo stato di "*production-ready*" del prodotto. Questo branch ha una durata di vita quanto il repository stesso o infinita;
2. **Develop:** branch di sviluppo parallelo al **master** sul quale vengono aggiunte le feature provenienti appunto dai branch **feature**, e dal quale inizia il branch di **release**. Ha una durata di vita quanto il branch master;
3. **Release:** branch di preparazione per un nuovo rilascio o aggiornamento del prodotto. Utilizzato per risolvere piccoli errori e configurare le impostazioni di rilascio. Una volta rilasciato il prodotto, esso si riversa sul branch **master** e **develop**. Ha una durata breve in quanto il rilascio deve essere effettuato il prima possibile;
4. **Feature:** branch usato per sviluppare nuove feature per il prossimo rilascio a breve o lungo tempo. Il suo tempo di vita dura quanto lo sviluppo della nuova feature fintanto che non avviene il `git` merge sul branch di **develop**;
5. **Hotfix:** branch molto simili a quelli di release, con l'obiettivo di risolvere immediatamente un bug del prodotto in produzione o release. Una volta risolto il bug, esso si riversa sui branch **master** e **develop**, aggiornandoli nel minor tempo possibile. Ha un tempo di vita breve, in quanto viene creato per la necessità di risolvere un problema sul prodotto rilasciato.

**3.2.1.4 Rilascio di versione** Il rilascio di una nuova versione del prodotto avviene nel momento in cui si raggiunge un certo numero di feature implementate e

testate. Dal branch **develop** si avvia un processo di verifica e verifica che sfocia in una nuovo branch di **release**, il quale porta il nome della relase e che nella sua ultima fase rilascia la nuova versione sul branch master e applica le modifiche effettuate nel frattempo anche nel branch di **develop**, portando i due allo stesso livello di produzione.

### 3.2.2 Configurazione versionamento

**3.2.2.1 Remoto** La configurazione di **GitHub** avviene nel portale: *www.github.com*, dove si inseriscono le chiavi **G SSH** per ciascun collaboratore del nuovo repository. Una volta creato il repository nel server remoto ed inserite le chiavi **SSH**, si procedere con la configurazione in locale.

**3.2.2.2 Locale** In locale, si devono generare le chiavei **SSH**, le quali permettono il collegamento con li server remoto dove viene gestito il repository. Una volta generate le chiavi, seguendo le varie procedure specifiche per ogni sistema operativo, vengono caricate sul portare apposito del gestore **GitHub**. L'ultima fase prevede la clonazione con il programma apposito tra i seguenti:

- **GitHub Desktop**: gestore di versionamento a interfaccia grafica per sistemi Windows & MacOS;
- **GitKraken**: gestore di versionamento a interfaccia grafica per sistemi Windows, MacOS & Linux;
- **Terminale(Bash)**: gestore di versionamento a riga di comando per i sistemi MacOS & Linux.

## 3.3 Gestione di progetto

La gestione di progetto avviene tramite un sistema di task integrato nel servizio di hosting **GitHub**. Esso permette l'integrazione delle task con il repository stesso, dando la possibilità ai vari **G** commit di chiudere con comandi appositi determinate task, aumentando così l'automazione di tutto il processo.

### 3.3.1 Configurazione strumenti di organizzazione

La configurazione di tutto il processo di organizzazione avviene nel portale di **GitHub**, dove si crea una project board per ogni categoria di processo.

**3.3.1.1 Inizializzazione** L'inizializzazione della project board avviene tramite un'istanza vuota oppure selezionando un template di ciclo di vita fornito da **GitHub** largamente utilizzate in molti progetti, quindi testate ed affidabili. Tra i template forniti abbiamo:

- **Basic Kanban:** presenta le fasi di *ToDo*, *In Progress*, e *Done*;
- **Automated Kanban:** presenta  $G$  trigger predefiniti che permettono lo spostamento di task automatici nei vari cicli di vita, utilizzando il meccanismo di chiusura dei commit;
- **Automated Kanban with Reviews:** tutto ciò che viene incluso nel template "*Automated Kanban*" con l'aggiunta di trigger aggiuntivi per la revisione di nuove componenti;
- **Bug Triage:** template centrato sulla gestione degli errori, fornendo un ciclo di vita per essi che varia tra *ToDo*, *Alta Priorità*, *Bassa Priorità* e *Chiusi*.

**3.3.1.2 Aggiunta milestones** Le milestones sono gruppi di task mirate a un obiettivo comune tra esse. Possono essere aggiunte in qualsiasi momento, sia prima che dopo la creazione di una task.

### 3.3.2 Ciclo di vita delle tasks

**3.3.2.1 Apertura** Da una specifica project board si possono creare le task o le  $G$  issue, le quali possono essere assegnate a uno o più individui che collaborano al repository, inoltre ogni task può far parte di una  $G$  milestone, che raggruppa un insieme di task o issue per il raggiungimento di un obiettivo in comune. Inoltre ad ognuna di esse può essere assegnato un colore che ne identifica il tipo come per esempio: *bug*, *ToDo*, *miglioramenti*, ecc.... Si può creare una nuova task senza l'obbligo di assegnarla a una project board, mantenendo comunque tutte le funzionalità descritte prima.

**3.3.2.2 Completamento** Il completamento di una task avviene in diversi modi, a seconda delle impostazioni di project board. Se la project board è automatizzata, il completamento di una task può avvenire tramite commit utilizzando il codice di chiusura. Questo metodo collega direttamente l'implementazione richiesta alla task. Se la project board non è automatizzata, il completamento dalla task deve essere manuale spostandola nello stato di *Concluso*.



**3.3.2.3 Richiesta di revisione** Accumulate un certo numero di task o di milestones, si avvia la procedura di revisione da parte del verificatore. Questa può essere notificata e pianificata in modo automatico a seconda del livello di automatizzazione della project board, oppure può essere totalmente gestita dal verificatore.

**3.3.2.4 Chiusura** Una volta che le task o le milestones sono state approvate dal verificatore, esse concludono il loro ciclo di vita nello stato di chiusura, le quali verranno spostate manualmente dal verificatore o automaticamente dalla project board se il merge è avvenuto con successo.

**3.3.2.5 Riapertura** Le task in stato di "*Chiusura*" possono essere riaperte e spostate nello stato di "*Apertura*" se esse in un lungo periodo non soddisfanno certi parametri di qualità richiesti.

## 4 Processi Organizzativi

### 4.1 Processi di Coordinamento

#### 4.1.1 Gestione Comunicazioni

In questa sezione vengono descritte le norme che regolano le comunicazioni del gruppo *Agents of S.W.E.*, sia interne, tra i suoi componenti, sia verso entità esterne, come committenti e proponenti.

##### 4.1.1.1 Comunicazioni Interne

Le comunicazioni interne ai membri del gruppo vengono gestite principalmente attraverso un gruppo *Telegram<sub>G</sub>*, presso cui il gruppo, nella sua interezza, discute degli aspetti più generali o collettivi riguardanti il progetto.

Per facilitare una comunicazione più specifica, monotematica, ed efficiente tra alcuni membri del gruppo, e per gestire meglio la stesura dei documenti, sono stati inoltre predisposti svariati canali tematici all'interno dell'app di messaggistica: *Slack<sub>G</sub>*. Tali canali sono:

- **#general**: Per discussioni riguardanti rotazione di ruoli e decisioni degli argomenti principali da discutere nelle riunioni;
- **#normeprogetto**: Per discutere riguardo le regole del *Way of Working* del gruppo, le norme da seguire e, di conseguenza, la stesura in collaborazione del documento *Norme di Progetto<sub>G</sub>*;
- **#pianoprogetto**: Per confrontarsi riguardo il monte ore dei vari ruoli e per facilitare la stesura del documento *Piano di Progetto<sub>G</sub>*;
- **#analisirequisiti**: Per discutere gli *User Case<sub>G</sub>* e i requisiti necessari alla stesura dell'*Analisi dei Requisiti*;
- **#pianoqualifica**: Per discutere strategie da attuare per garantire qualità attraverso *verifica<sub>G</sub>* e *validazione<sub>G</sub>*.

##### 4.1.1.2 Comunicazioni Esterne

Le comunicazioni esterne avvengono attraverso la casella di posta elettronica del gruppo: *agentsofswe@gmail.com*, gestita principalmente dal *Responsabile* del gruppo, ma accessibile da ogni membro e configurata per eseguire un inoltramento automatico delle mail ricevute ad ogni membro del gruppo.

#### 4.1.2 Gestione Riunioni

Durante ogni riunione, interna o esterna, verrà nominato, tra i componenti del gruppo, un segretario che avrà il compito di far rispettare l'ordine del giorno, stilato dal *Responsabile di Progetto*, ed occuparsi della stesura del Verbale di Riunione<sub>G</sub>

##### 4.1.2.1 Riunioni Interne

E' compito del *Responsabile di Progetto* organizzare riunioni interne al gruppo *Agents of S.W.E.*. Ciò prevede, più nello specifico, la stesura dell'ordine del giorno, stabilire data, orario e luogo di incontro, ed assicurarsi, attraverso la comunicazione mediante i mezzi propri del gruppo, che ogni componente sia pienamente a conoscenza della riunione in tutti i suoi dettagli.

D'altro canto ogni membro del gruppo deve presentarsi puntuale agli appuntamenti, e comunicare in anticipo eventuali ritardi o assenze adeguatamente giustificate.

Una riunione non è da ritenersi valida se i partecipanti risultino essere in numero inferiore a cinque.

##### 4.1.2.2 Riunioni Esterne

E' nuovamente compito del *Responsabile di Progetto* organizzare riunioni esterne. Nello specifico egli deve preoccuparsi di contattare l'azienda proponente per fissare l'incontro, tendo conto anche delle preferenze di data e orario espresse dagli altri membri del gruppo. La partecipazione a tali riunioni deve essere, a meno di casi eccezionali, unanime.

Ogni membro del gruppo può, inoltre, esprimere al *Responsabile* una richiesta, adeguatamente motivata, di fissare una riunione esterna. A questo punto sarà compito dello stesso *Responsabile* giudicare come valida o meno la richiesta presentatali ed agire di conseguenza.

##### 4.1.2.3 Verbale di Riunione

Ad ogni riunione, interna o esterna, è compito del Segretario designato redigere il Verbale di Riunione corrispondente, che deve essere poi approvato dal *Responsabile*. Tale Verbale avrà la seguente **Struttura**:

- **Informazioni Generali:** Questa prima sezione composta di:
  - **Luogo;**
  - **Data;**
  - **Ora;**
  - **Membri del team partecipanti;**

– **Segretario.**

- **Ordine del Giorno:** Sotto forma di elenco puntato rappresentante gli argomenti discussi;
- **Resoconto:** Tale sezione rappresenta il riassunto, redatto dal Segretario, punto per punto di tutti gli argomenti di discussione, sia quelli preventivamente presenti nell'ordine del giorno sia eventuali spunti di riflessione maturati autonomamente durante la riunione.

## 4.2 Ruoli di Progetto

Nell'ottica di un lavoro ben organizzato e collaborativo tra i membri del gruppo, ad ogni componente, in ogni momento, è attribuito un ruolo per un periodo di tempo limitato.

Questi ruoli, che corrispondono ad una figura aziendale ben precisa, sono:

- **Responsabile di Progetto;**
- **Amministratore di Progetto;**
- **Analista;**
- **Progettista;**
- **Programmatore;**
- **Verificatore.**

### 4.2.1 Responsabile di Progetto

Detto anche "*Project Manager*", è il rappresentante del progetto<sub>G</sub>, agli occhi sia del committente che del fornitore. Egli risulta dunque essere, in primo luogo, il responsabile ultimo dei risultati del proprio gruppo. Figura di grande responsabilità, partecipa al progetto<sub>G</sub> per tutta la sua durata, ha il compito di prendere decisioni e approvare scelte collettive.

Nello specifico egli ha la responsabilità di:

- Coordinare le attività del gruppo, attraverso la gestione delle risorse umane;
- Approvare i documenti redatti, e verificati, dai membri del gruppo;
- Elaborare piani e scadenze, monitorando i progressi nell'avanzamento del progetto<sub>G</sub> ;
- Redigere l'organigramma del gruppo e il *Piano di Progetto<sub>G</sub>* .

#### 4.2.2 Amministratore di Progetto

L'amministratore è la figura chiave per quanto concerne la produttività. Egli ha infatti come primaria responsabilità il garantire l'efficienza<sub>G</sub> del gruppo, fornendo strumenti utili e occupandosi dell'operatività delle risorse. Ha dunque il compito di gestire l'ambiente lavorativo.

Tra le sue responsabilità specifiche figurano:

- Redigere documenti che normano l'attività lavorativa, e la loro verifica<sub>G</sub> ;
- Redigere le *Norme di Progetto*<sub>G</sub> ;
- Scegliere ed amministrare gli strumenti di versionamento<sub>G</sub> ;
- Ricercare strumenti che possano agevolare il lavoro del gruppo;
- Attuare piani e procedure di gestione della qualità<sub>G</sub> .

#### 4.2.3 Analista

L'analista deve essere dotato di un'ottima conoscenza riguardo al dominio del problema. Egli ha infatti il compito di analizzare tale dominio e comprenderlo appieno, affinché possa avvenire una corretta progettazione<sub>G</sub> .

Ha il compito di:

- Comprendere al meglio il problema, per poi poterlo esporre in modo chiaro attraverso specifici requisiti<sub>G</sub> ;
- Redarre lo *Studio di Fattibilità* e l'*Analisi dei Requisiti*<sub>G</sub> .

#### 4.2.4 Progettista

Il progettista è responsabile delle attività di progettazione<sub>G</sub> attraverso la gestione di aspetti tecnici del progetto<sub>G</sub> .

Più nello specifico si occupa di:

- Definire l'Architettura<sub>G</sub> del prodotto<sub>G</sub> , applicando quanto più possibile norme di *best practice*<sub>G</sub> , prestando attenzione alla manutenibilità del prodotto<sub>G</sub> ;
- Suddividere il problema, e di conseguenza il sistema, in parti di complessità trattabile.



#### 4.2.5 Programmatore

Il programmatore si occupa delle attività di codifica, le quali portano alla realizzazione effettiva del prodotto<sub>G</sub>.

Egli ha dunque il compito di:

- Implementare l'architettura definita dal *Progettista*, prestando attenzione a scrivere codice coerente con ciò che è stato stabilito nelle norme di qualifica;
- Produrre codice documentato e manutenibile;
- Realizzare le componenti necessarie per la verifica<sub>G</sub> e la validazione<sub>G</sub> del codice;
- Redarre il *Manuale Utente*.

#### 4.2.6 Verificatore

Il verificatore, figura presente per l'intera durata del progetto<sub>G</sub>, è responsabile delle attività di verifica<sub>G</sub>.

Nello specifico egli:

- Verifica l'applicazione ed il rispetto delle *Norme di Progetto<sub>G</sub>*;
- Segnala al *Responsabile di Progetto* l'emergere di eventuali discordanze tra quanto presentato nel *Piano di Progetto<sub>G</sub>* e quanto effettivamente realizzato;
- Ha il compito di redarre il *Piano di Qualifica*.

#### 4.2.7 Rotazione dei Ruoli

Come da istruzioni ogni membro del gruppo dovrà ricoprire, per un periodo di tempo limitato, ciascun ruolo, nel rispetto delle seguenti regole:

- Ciascun membro dovrà svolgere esclusivamente le attività proprie del ruolo a lui assegnato;
- Al fine di evitare conflitti di interesse nessun membro potrà ricoprire un ruolo che preveda la verifica<sub>G</sub> di quanto da lui svolto, nell'immediato passato;
- Vista l'ampia differenza di compiti e mansioni tra i vari ruoli, e al fine di valorizzare l'attività collaborativa all'interno del gruppo, ogni componente che abbia ricoperto in precedenza un ruolo ora destinato a qualcun altro dovrà fornire supporto al compagno in caso di necessità, fornendogli consigli e, se possibile, affiancandolo in situazioni critiche.

## 4.3 Procedure

### 4.3.1 Gestione degli Strumenti di Versionamento

Come strumento per la gestione del versionamento dei files si è scelto di utilizzare una *Repository<sub>G</sub>* su *GitHub<sub>G</sub>*. La gestione di tale *Repository<sub>G</sub>* è compito dell'*Amministratore di Progetto*.

#### 4.3.1.1 Struttura Repository

La *Repository<sub>G</sub>* "*Agents-of-S.W.E.*" è caratterizzata da una cartella principale "Documentazione", la cui sottocartella "RR" rappresenta la principale struttura in cui sono organizzati i files su cui il gruppo *Agents of S.W.E.* ha lavorato in vista della RR.

Nello specifico la sottocartella "**RR**" è caratterizzata dalla seguente struttura di folders:

- **Documenti Esterni:**
  - Analisi dei Requisiti;
  - Glossario;
  - Piano di Progetto;
  - Piano di Qualifica;
  - Verbali Esterni.
- **Documenti Interni:**
  - Norme di Progetto;
  - Studio di Fattibilità;
  - Verbali Interni.

Ciascuna cartella contiene a sua volta un file LaTeX ".tex" che assume il nome del corrispondente documento e il corrispettivo file ".pdf" ottenuto attraverso la compilazione.

#### 4.3.1.2 Uso dei Branch

Al fine di agevolare il più possibile il parallelismo, evitando al contempo quanto più possibile eventuali problematiche in fase di *merge<sub>G</sub>*, sono stati creati i seguenti *branch<sub>G</sub>*:

- **master:** Questo branch contiene solamente i documenti che si trovano in stato "**Approvato**", i quali formano dunque la *baseline<sub>G</sub>*;

- **develop**: Questo è il branch di "sviluppo". Esso contiene tutti i documenti che, sebbene siano considerati ultimati per quanto riguarda la loro stesura, sono in attesa di approvazione o in fase di verifica;
- **feature/"nomeDocumento"**: Vi sono quattro branch<sub>G</sub> distinti questo tipo, nello specifico: "feature/analisiRequisiti", "feature/normeProgetto", "feature/pianoProgetto" e "feature/pianoQualifica". Ciascuno di questi è specializzato nell'avanzamento della stesura del documento a cui si riferisce.

#### 4.3.1.3 Tipi di files

All'interno della repository<sub>G</sub> sono presenti solamente i seguenti formati di files:

- *.tex*: files sorgenti LaTeX<sub>G</sub> ;
- *.pdf*: files compilati dai sorgenti;
- *.jpg* o *.png*: immagini.

I files temporanei prodotti dalla compilazione LaTeX<sub>G</sub> sono stati introdotti in un file *.gitignore*, vendono dunque ignorati da Git in sede di *push<sub>G</sub>* .

#### 4.3.1.4 Norme delle Commit

Ogni *commit<sub>G</sub>* , ovvero ciascuna modifica alla repository<sub>G</sub> *Agents-of-S.W.E.* deve essere caratterizzata da una descrizione sensata, eventualmente accompagnata ad un riferimento esplicito ad una *issue<sub>G</sub>* aperta, al fine di agevolare una piena, e non onerosa, comprensione da parte di ogni membro del gruppo.

#### 4.3.1.5 Norme dei Merge tra Branch

Al fine di rispettare la caratterizzazione che si è deciso di dare al branch **master**, e per agevolare un lavoro sistematico ed organizzato del lavoro si sono stabilite le seguenti norme in sede di *merge<sub>G</sub>* tra diversi branch<sub>G</sub> :

- **Merge develop-feature/"nomeDocumento"**: Questo merge avviene solo quando gli incaricati alla stesura del documento a cui si riferisce il branch: *feature/"nomeDocumento"* ritengono ultimata questa prima attività. Il documento in questione, dunque, si considera completo di ogni sua parte essenziale, eccezion fatta per eventuali modifiche, anche cospicue, da apportare a seguito di un'attività di verifica e/o approvazione;
- **Merge master-develop**: Questo merge avviene solo quando ogni documento contenuto nel branch: "develop" è stato verificato ed approvato, e si considera dunque pronto per il rilascio.

### 4.3.2 Gestione degli Strumenti di Coordinamento

#### 4.3.2.1 Tasks

La suddivisione del lavoro in  $tasks_G$  è compito del *Responsabile di Progetto*. Lo strumento scelto per la creazione e gestione di questi  $tasks_G$  è lo stesso  $Git_G$ , il quale mette a disposizione l'utile strumento delle  $issue_G$ .

La creazione di un  $task_G$  da parte del *Responsabile di Progetto*, risulterà dunque essere l'istanziamento di una  $issue_G$  caratterizzata dalle seguenti proprietà:

- *Titolo* significativo;
- *Descrizione* concisa ma caratteristica ed esplicativa del problema da affrontare;
- *Uno o più tags* associati a particolarità del *task* in questione, e/o al/ai documento/i a cui si riferiscono. Tali etichette consentono una rapida catalogazione delle *issue* stesse;
- *Data di scadenza* che rappresenta il termine ultimo entro cui tale *issue* deve essere chiusa.

E' importante far notare che, sebbene l'onere della suddivisione del lavoro, e dunque la creazione e gestione dei  $tasks_G$  e dei conseguenti  $tickets_G$ , ricada sul *Responsabile di progetto*, ciascun membro del gruppo ha la facoltà di creare  $tasks_G$ , a patto che tale compito veda lui come unico assegnatario. Tali  $tasks_G$  dovranno inoltre essere approvati dal *Responsabile* per essere validi.

#### 4.3.2.2 Tickets

I  $tickets_G$  rappresentano l'operazione di assegnazione dei  $tasks_G$ , e quindi in questo caso delle  $issue_G$ , ad uno o più specifici membri del gruppo. Tale operazione è responsabilità unica del *Responsabile di Progetto* a meno che non si tratti di  $tasks_G$  (approvati dal Responsabile) creati da un membro del gruppo, ed assegnati autonomamente a sè stesso.

Questa operazione di *ticketing* può avvenire in due modalità distinte:

- **Proattivamente:** Nel caso in cui l'assegnatario del task in questione sia già noto, ed indicato come tale, alla creazione della *issue* da parte del Responsabile. E' importante notare come questa sia l'unica modalità di ticketing possibile nel caso in cui il creatore del task sia un membro diverso dal *Responsabile di Progetto*;

- **Retroattivamente:** Nel caso in cui uno o più membri del gruppo vengano designati, in un secondo momento, come assegnatari di una issue già precedentemente esistente. Questa modalità di ticketing consente di gestire situazioni in cui non è utile individuare subito un assegnatario, come nel caso di task di importanza marginale e/o scadenze molto permissive, oppure invece si tratti di un compito le cui complessità sono emerse solo in seconda battuta, e necessiti dunque di un maggior apporto lavorativo per rispettarne le scadenze.

## 4.4 Strumenti

### 4.4.1 Sistema Operativo

I sistemi operativi utilizzati dai membri del gruppo sono i seguenti:

- Windows 10 x64;
- Mac OS 10.14.1;
- Majaro Linux - 4.14.85-1 LTS.

### 4.4.2 Versionamento e Issue Tracking

#### 4.4.2.1 Git

*Git* è il sistema di versionamento<sub>G</sub> scelto dal gruppo. E' un sistema open source creato da Linus Torvalds 2005. Presenta un'interfaccia a riga di comando, tuttavia esistono svariati *tools*<sub>G</sub> che ne forniscono una GUI.

#### 4.4.2.2 GitHub

*Github* è un *Respository Manager*<sub>G</sub> che usa Git come sistema di versioning, offre inoltre servizi di *issue tracking*<sub>G</sub>.

### 4.4.3 Comunicazione

#### 4.4.3.1 Telegram

Telegram è una delle maggiori e più note applicazioni di messaggistica istantanea *cross platform*, utilizzabile contemporaneamente su più dispositivi. Oltre alla semplice messaggistica offre servizi quali lo scambio di files, la creazione di gruppi e le chiamate vocali.

#### 4.4.3.2 Slack

Slack è un'applicazione di messaggistica istantanea specializzata nella comunicazione interna tra membri di un gruppo di lavoro. L'applicazione è organizzata in *workspace*, a loro volta suddivisi in canali, i quali consentono di catalogare le conversazione sulla base dell'argomento trattato. Questa struttura, studiata appositamente per l'ambito lavorativo, è stata giudicata come positiva e vantaggiosa dal gruppo, visto che consente di mantenere chat ordinate e monotematiche.

Nonostante preveda anche abbonamenti a pagamento le funzionalità base di Slack sono gratuite, inoltre, come Telegram, risulta essere un'applicazione *cross platform*<sub>G</sub>.

#### 4.4.4 Diagrammi di Gantt

Lo strumento scelto dal gruppo per la realizzazione dei diagrammi di Gantt<sub>G</sub> è "Gantt Project". Le motivazioni che hanno portato a questa scelta sono molteplici, tra queste spiccano il fatto che sia uno strumento gratuito, *open-source*<sub>G</sub>, e *cross platform*. L'elevata accessibilità è stata infatti giudicata come una caratteristica di primaria importanza, considerando i differenti sistemi operativi utilizzati dai componenti del gruppo.

#### 4.4.5 Diagrammi UML

Lo strumento scelto dal gruppo per la realizzazione dei diagrammi UML è "Umbrello", un software *open-source*<sub>G</sub> per il disegno di diagrammi UML 2.0, creato da KDE Team. Anche in questo caso è stata tenuta in grande considerazione l'accessibilità dello strumento. Il software in questione risulta infatti gratuito e disponibile sia su Windows, sia su Mac OS tramite porting, sebbene sia nativo Unix.

Umbrello risulta essere sia *User Friendly*, con un'interfaccia utente chiara e consistente, sia abbastanza potente da essere utilizzato anche per le industrie. Questo, unito alle dimensioni relativamente ridotte del software, e al fatto che non sia richiesta alcuna licenza per l'installazione, hanno fatto propendere il gruppo per questo programma, a discapito di alternative, come Papyrus o Astah, comunque considerate valide.

### 4.5 Formazione del Gruppo

La formazione dei componenti del gruppo **Agents of S.W.E.** è da considerarsi individuale. Ogni membro del team è infatti tenuto a documentarsi autonomamente riguardo le tecnologie coinvolte nello sviluppo del progetto<sub>G</sub>. Tuttavia, nell'ottica



di un ambiente di lavoro sano e collaborativo, nel caso in cui fosse necessario, è compito degli *Amministratori* mettere a disposizione di chi ne avesse bisogno risorse basilari, al fine di agevolare la formazione dei restanti componenti del gruppo.

## 5 Changelog

Versione	Data	Autore	Ruolo	Descrizione
0.0.1	2018-11-23	Luca Violato	Amministratore	Strutturazione del Documento
0.0.2	2018-11-23	Marco Chiese	Verificatore	Stesura §2
0.0.3	2018-12-01	Luca Violato	Amministratore	Strutturazione §4, stesura §4.2 e §4.5
0.0.4	2018-12-03	Bogdan Stanciu	Responsabile	Stesura da §3.3 a §3.3.2.2
0.0.5	2018-12-04	Luca Violato	Amministratore	Stesura §4.4
0.0.6	2018-12-04	Marco Favaro	Analista	Stesura da §3.1 a §3.1.4
0.0.7	2018-12-06	Marco Favaro	Analista	Stesura da §3.1.4 a §3.1.7
0.0.8	2018-12-06	Bogdan Stanciu	Responsabile	Stesura da §3.4 a §3.4.2.5
0.0.9	2018-12-06	Luca Violato	Amministratore	Stesura §4.1
0.0.10	2018-12-07	Marco Favaro	Analista	Stesura §3.1.6
0.0.11	2018-12-07	Luca Violato	Amministratore	Stesura §4.3
0.0.12	2018-12-09	Marco Chiese	Verificatore	Stesura §2

**Tabella 1:** Changelog del documento