Agents of S.W.E.

A SOFTWARE COMPANY

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

Norme di Progetto

Versione	0.0.7
Approvazione	?
Redazione	Luca Violato Bogdan Stanciu Marco Favaro Marco Chilese
Verifica	? ?
Stato	Work in Progress
Uso	Interno
Destinato a	Agents of S.W.E. Prof. Tullio Vardanega Prof. Riccardo Cardin

agentsofswe@gmail.com

Indice

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 Programmatori 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 JavaScript$_G$, HTML$_G$ e CSS$_G$.
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$_G$ e la validazione$_G$.

2.3.3.1 Convenzioni per i nomi:

- I Programmatori devono adottare come notazione per la definizione di cartelle, file, metodi, funzioni e variabili il CamelCase$_G$.
  Di seguito un esempio di corretta nomenclatura:

```
1 INSERIRE ESEMPIO
```

- 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: $Y.K$, dove Y rappresenta la versione principale, K la versione parziale della relativa versione principale.
  I numeri di versione del tipo $Y.0$, dalla 1.0, 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[1] 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[2].
ECMAScript 6 viene stardardizzato da **ECMA**$_{\text{G}}$ [3] nel giugno 2015 con la sigla **ECMA-262**[4].
Come stile di codifica si adottano le line guida proposte da **Airbnb JavaScript**

[1] `http://docs.grafana.org/plugins/developing/development/`
[2] Linguaggio divenuto standard ISO: ISO/IEC 16262:2011, e relativo aggiornamento ISO/IEC 22275:2018.
[3] `http://www.ecma-international.org/`
[4] `https://www.ecma-international.org/ecma-262/6.0/`

Style Guide[5]. Per la verifica dell'adesione a tali norme, i Programmatori devono utilizzare, come suggerito dalla documentazione proposta da *Grafana*, **ESLint**$_G$ [6]. 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`:

```
testDatasource() {
  return this.getServerStatus()
  .then(status => {
    return this.doSomething(status);
  })
}
```

Invece che:

```
testDatasource() {
  var self = this;
  return this.getServerStatus()
    .then(function(status) {
  return self.doSomething(status);
  })
}
```

4. Utilizzare l'oggetto *Promise*:

```
metricFindQuery(query) {
  if (!query) {
    return Promise.resolve([]);
  }
}
```

Invece che:

```
metricFindQuery(query) {
  if (!query) {
    return this.$q.when([]);
  }
}
```

[5] https://github.com/airbnb/javascript
[6] https://eslint.org/

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.

Identazione

Norma 1 L'identazione 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:

```
function() {
  if (condition && condition) {
    ...
  }

  if (
   veryLongCondition
   && longCondition
   && condition
    ) {
    doSomething();
  }
}
```

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

```
function1() {
  if (condition) {
    doSomething():
  }

  return toReturn;
}

function2(){
  ...
}
```

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:

```
if (condition) return true;

if (condtion) {
  return true;
}
```

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 }
```

2.3.3.4 HTML

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_{G} 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 LaTeXper 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 ufficiosa.

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". É 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_{G}.
- **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**: TODO
- **Collegamenti esterni**: TODO

3.1.6 Documenti Correnti

Sono descritti brevemente i documenti formali da consegnare:

- **Analisi dei Requisiti**: TODO
- **Glossario**: TODO
- **Norme di Progetto**: TODO
- **Piano di Progetto**: TODO
- **Piano di Qualifica**: TODO
- **Studio di Fattibilità**: TODO

3.1.7 Ambiente

Per uniformare e strutturare al meglio la scrittura dei documenti il `gruppo` ha adottato il formato LaTeX.
TexMaker[7] 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 Qualità

3.2.1 Descrizione

3.2.2 Classificazione dei processi

3.2.3 Procedure

3.3 Versionamento

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

3.3.1 Controllo di versione

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

3.3.1.1 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.

3.3.1.2 Processo di implementazione L'implementazione dei documenti avviene tramite gli strumenti utilizzati nel paragrafo –INSERIRE PARAGRAFO PARTE DI FAVARO–

[7] `http://www.xm1math.net/texmaker/`

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 $_{G}$ commit di essi. Il commit riporta un *cambiamento* al file,con un messaggio allegato ad esso che ne descrive le modifiche apportate o un commando apposito per chiudere alcune task con tale commit. Dopo di che, il commit viene $_{G}$ pushato nel branch appropriato, a seconda dei criteri descritti nel prossimo paragrafo.

3.3.1.3 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 **realse**. 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 $_{G}$ merge sul branch di **develop**;

5. **Hotfix**: branch molto simili a quelli di release, con l'obbiettivo 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.

3.3.1.4 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.

3.3.2 Configurazione versionamento

3.3.2.1 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.

3.3.2.2 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 commando per i sistemi MacOS & Linux.

3.4 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 cosi l'automazione di tutto il processo.

3.4.1 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.

3.4.1.1 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 cioè 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*.

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

3.4.2 Ciclo di vita delle tasks

3.4.2.1 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 obbiettivo in comune. Inoltre ad ognuna di essere 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.

3.4.2.2 Completamento Il completamento di una task avviene in diversi modi, a seconda delle impostazioni di project board. Se la project board e 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*.

3.4.2.3 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.

3.4.2.4 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.

3.4.2.5 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

4.1.1.1 Comunicazioni Interne

4.1.1.2 Comunicazioni Esterne

4.1.2 Gestione Riunioni

4.1.2.1 Riunioni Interne

4.1.2.2 Riunioni Esterne

4.1.2.3 Verbale di 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 rappresentate del progetto$_G$, 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$_G$ 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$_G$;
- Redigere l'organigramma del gruppo e il *Piano di Progetto*$_G$.

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$_G$ del gruppo, fornendo strumenti utili e occupandosi dell'operatività delle risorse. Ha dunque il compito di gestire l'ambiente lavorativo.
Tra le sue responsbilità specifiche figurano:

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

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, affinchè possa avvenire una corretta progettazione$_G$.
Ha il compito di:

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

4.2.4 Progettista

Il progettista è responsabile delle attività di progettazione$_G$ attraverso la gestione di aspetti tecnici del progetto$_G$.
Più nello specifico si occupa di:

- Definire l'Architettura$_G$ del prodotto$_G$, applicando quanto più possibile norme di *best practice*$_G$, prestando attenzione alla manutenibilità del prodotto$_G$;
- 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$_G$.
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$_G$ e la validazione$_G$ del codice;
- Redarre il *Manuale Utente*.

4.2.6 Verificatore

Il verificatore, figura presente per l'intera durata del progetto$_G$, è responsabile delle attività di verifica$_G$.
Nello specifico egli:

- Verifica l'applicazione ed il rispetto delle *Norme di Progetto*$_G$;
- Segnala al *Responsabile di Progetto* l'emergere di eventuali discordanze tra quanto presentato nel *Piano di Progetto*$_G$ 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$_G$ 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

4.3.1.1 Struttura Repository

4.3.1.2 Tipi di files

4.3.1.3 Norme delle Commit

4.3.2 Gestione degli Strumenti di Coordinamento

4.3.2.1 Tasks

4.3.2.2 Tickets

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$_G$ scelto dal gruppo. E' un sistema open source creato da Linus Torvalds 2005. Presenta un'interfaccia a riga di comando, tuttavia esistono svariati *tools*$_G$ che ne forniscono una GUI.

4.4.2.2 GitHub

Github è un *Respository Manager*$_G$ che usa Git come sistema di versioning, offre inoltre servizi di *issue tracking*$_G$.

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*$_G$.

4.4.4 Diagrammi di Gantt

Lo strumento scelto dal gruppo per la realizzazione dei diagrammi di Gantt$_G$ 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*$_G$, 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*$_G$ 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$_G$. 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 Chilese	Verificatore	Prima 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

Tabella 1: Changelog del documento