



Agents of S.W.E.

A SOFTWARE COMPANY

Agents of S.W.E - Progetto "G&B"

Norme di Progetto

Versione	3.0.4
Approvazione	Luca Violato
Redazione	Luca Violato Bogdan Stanciu Marco Favaro Marco Chilese Matteo Slanzi
Verifica	Carlotta Segna Diego Mazzalovo
Stato	Approvato
Uso	Interno
Destinato a	Agents of S.W.E. Prof. Tullio Vardanega Prof. Riccardo Cardin

agentsofswe@gmail.com

Registro delle Modifiche

Versione	Data	Ruolo	Autore	Descrizione
3.0.4	2019-05-02	Amministratore	Diego Mazza-lovo	Aggiunta libreria <i>JSBayes-Viz</i> a tecnologie.
3.0.3	2019-04-26	Amministratore	Diego Mazza-lovo	Fix A
3.0.2	2019-04-26	Analista	Carlotta Segna	Inserimento metrica MT-PC08
3.0.1	2019-04-25	Responsabile	Matteo Slanzi	Correzioni §2.1
3.0.0	2019-04-12	Responsabile	Luca Violato	Approvazione documento per il rilascio RQ
2.1.0	2019-04-10	Verificatore	Diego Mazza-lovo	Verifica documento
2.0.9	2019-04-09	Amministratore	Carlotta Segna	Spostamento Metriche in appendice, sezione F
2.0.8	2019-04-02	Responsabile	Matteo Slanzi	Ampliamento §2.2.3.7
2.0.7	2019-04-02	Verificatore	Marco Chiese	Stesura §2.2.3.7 , aggiornamento §1.3
2.0.6	2019-04-02	Responsabile	Matteo Slanzi	Aggiunta sezione strumenti §2.2.4 in Sviluppo §2.2
2.0.5	2019-04-01	Responsabile	Matteo Slanzi	Aggiunte attività di Fornitura §2.1
2.0.4	2019-03-22	Progettista	Marco Chiese	Modifica struttura §4
2.0.3	2019-03-20	Progettista	Marco Chiese	Stesura §3.4 e §3.5



Versione	Data	Ruolo	Autore	Descrizione
2.0.2	2019-03-19	Progettista	Marco Chilese	Aggiornamento §3.3.1. Aggiornamento §2.2.3.5.
2.0.1	2019-03-18	Progettista	Marco Chilese	Spostamento §2.1.1 da §2.2 a §2.1. Spostamento §4.4.2 da §2 a §4.4. Aggiornamento §1.3
2.0.0	2019-03-07	Responsabile	Diego Mazzalovo	Approvazione documento per il rilascio RP
1.2.7	2019-03-06	Amministratore	Marco Chilese	Stesura §3.4.3.1
1.2.6	2019-03-06	Amministratore	Marco Chilese	Aggiornamento §1.3 e §3.2
1.2.5	2019-03-05	Responsabile	Diego Mazzalovo	Correzioni ortografiche
1.2.4	2019-03-04	Amministratore	Marco Chilese	Avanzamento §2.2.2.3
1.2.3	2019-02-28	Analista	Marco Favaro	Correzioni §2.2.3.6, stesura §2.2.4.4
1.2.2	2019-02-28	Programmatore	Matteo Slanzi	Stesura §2.2.3.6
1.2.1	2019-02-25	Amministratore	Marco Chilese	Correzioni §2.2.2.3
1.2.0	2019-02-24	Progettista	Carlotta Segna	Verifica Documento
1.1.2	2019-02-21	Analista	Marco Favaro	Normato strumento GitLab e sistemati riferimenti
1.1.1	2019-02-14	Progettista	Marco Chilese	Correzioni §2.2.2.3
1.1.0	2019-02-14	Programmatore	Carlotta Segna	Verifica del Documento



Versione	Data	Ruolo	Autore	Descrizione
1.0.12	2019-02-14	Progettista	Marco Chilese	Stesura §2.2.3.3
1.0.11	2019-02-09	Verificatore	Marco Chilese	Ampliamento §4.1.2.3, stesura §4.4.2
1.0.10	2019-02-07	Verificatore	Carlotta Segna	Aggiornamento §4.1.2.3
1.0.9	2019-02-05	Verificatore	Carlotta Segna	Inserimento §2.2.3.4 e §2.2.3.5
1.0.8	2019-02-04	Analista	Marco Favaro	Risanamento terminato §3.2
1.0.7	2019-02-04	Verificatore	Marco Chilese	Ultimazione §2.2.2.3 e §2.2.2.4
1.0.6	2019-02-03	Verificatore	Marco Chilese	Ampliamento §2.2.2.3 e §2.2.2.4
1.0.5	2019-02-03	Verificatore	Carlotta Segna	Stesura §D
1.0.4	2019-01-30	Analista	Marco Favaro	§2.1.1 spostata sotto §2.2, Risanamento §3.2
1.0.3	2019-01-27	Amministratore	Luca Violato	Integrazione §4.3 e §4.2.1, Sezione spostata sotto §4
1.0.2	2019-01-26	Verificatore	Marco Chilese	Stesura §2.2.2.1, §2.2.2.3, §2.2.2.4, §2.2.2.5, §2.2.2.2
1.0.1	2019-12-25	Verificatore	Marco Chilese	Prima stesura §2.2.2
1.0.0	2018-12-01	Responsabile	Bogdan Stanciu	Approvazione per il rilascio
0.2.0	2018-11-30	Verificatore	Carlotta Segna	Verifica del documento
0.1.7	2018-11-29	Analista	Marco Favaro	Stesura §2.1.1



Versione	Data	Ruolo	Autore	Descrizione
0.1.6	2018-11-28	Analista	Marco Favaro	Avanzamento revisione documento
0.1.5	2018-11-27	Analista	Marco Chilesse	Avanzamento revisione documento
0.1.4	2018-11-27	Amministratore	Luca Violato	Stesura §1
0.1.3	2018-11-27	Analista	Marco Favaro	Avanzamento revisione documento
0.1.2	2018-11-27	Amministratore	Luca Violato	Modifica e integrazione §3.3.1
0.1.1	2018-11-26	Analista	Marco Favaro	Stesura §2.2.1, §3.2
0.1.0	2018-11-26	Verificatore	Carlotta Segna	Verifica del documento
0.0.16	2018-11-26	Responsabile	Bogdan Stanciu	Sistemazione sintassi paragrafi
0.0.15	2018-11-26	Analista	Marco Favaro	Sistemazione elenchi §2.2.1
0.0.14	2018-11-26	Analista	Marco Favaro	Stesura §A, §B e §C
0.0.13	2018-11-26	Analista	Marco Favaro	Inserimento indice e valuta
0.0.12	2018-11-25	Verificatore	Marco Chilesse	Stesura §2
0.0.11	2018-11-25	Amministratore	Luca Violato	Modifiche §4.3
0.0.10	2018-11-25	Analista	Marco Favaro	Stesura §3.1.6
0.0.9	2018-11-24	Amministratore	Luca Violato	Stesura §4.1
0.0.8	2018-11-24	Responsabile	Bogdan Stanciu	Stesura da §4.3 a §4.3.2.5



Versione	Data	Ruolo	Autore	Descrizione
0.0.7	2018-11-24	Analista	Marco Favaro	Stesura da §3.1.4 a §3.1.7
0.0.6	2018-11-23	Analista	Marco Favaro	Stesura da §3.1 a §3.1.4
0.0.5	2018-11-22	Amministratore	Luca Violato	Stesura §4.4
0.0.4	2018-11-22	Responsabile	Bogdan Stanciu	Stesura da §4.3 a §4.3.2.2
0.0.3	2018-11-21	Amministratore	Luca Violato	Strutturazione §4, stesura §4.2.1 e §4.5
0.0.2	2018-11-21	Verificatore	Marco Chilese	Stesura §2
0.0.1	2018-11-21	Amministratore	Luca Violato	Strutturazione del Documento

Tabella 1: Registro delle Modifiche

Indice

1	Introduzione	12
1.1	Scopo del Documento	12
1.2	Ambiguità e Glossario	12
1.3	Riferimenti	12
1.3.1	Referimenti Normativi	12
1.3.2	Referimenti Informativi	13
2	Processi Primari	14
2.1	Fornitura	14
2.1.1	Studio di Fattibilità	14
2.1.2	Rapporti di Fornitura con la Proponente	14
2.1.3	Collaudo e Consegna del Prodotto	15
2.2	Sviluppo	15
2.2.1	Analisi dei Requisiti	15
2.2.1.1	Classificazione dei Requisiti	16
2.2.1.2	Classificazione dei Casi d'Uso	17
2.2.2	Progettazione	18
2.2.2.1	Descrizione	18
2.2.2.2	Obiettivi della Progettazione	19
2.2.2.3	Sviluppo	19
2.2.2.4	Integrazione	21
2.2.2.5	Diagrammi	21
2.2.3	Codifica	22
2.2.3.1	Convenzioni per i Nomi	22
2.2.3.2	Convenzioni per la Documentazione	22
2.2.3.3	<i>ECMAScript 6</i>	23
2.2.3.4	<i>CSS</i>	29
2.2.3.5	<i>HTML</i>	30
2.2.3.6	<i>AngularJS</i>	32
2.2.3.7	<i>NodeJS</i>	35
2.2.4	Strumenti	36
2.2.4.1	<i>WebStorm</i>	36
2.2.4.2	<i>NPM</i>	36
2.2.4.3	<i>ESLint</i>	36
2.2.4.4	<i>Webpack</i>	36
2.2.4.5	Diagrammi UML	37

3	Processi di Supporto	39
3.1	Documentazione	39
3.1.1	Descrizione	39
3.1.2	Ciclo di Vita della Documentazione	39
3.1.3	Template	39
3.1.4	Struttura Documenti	39
3.1.4.1	Prima Pagina	40
3.1.4.2	Nomenclatura	40
3.1.4.3	Struttura Indice	40
3.1.4.4	Struttura Tabelle	40
3.1.4.5	Elenco Tabelle	41
3.1.4.6	Elenco Figure	41
3.1.4.7	Elenco Riferimenti	41
3.1.4.8	Piè di Pagina	41
3.1.4.9	Registro delle Modifiche	41
3.1.5	Norme Tipografiche	41
3.1.6	Documenti Correnti	42
3.1.7	Ambiente	43
3.2	Qualità	43
3.2.1	Introduzione	43
3.2.2	Classificazione Processi	43
3.2.3	Classificazione Metriche	43
3.2.4	Controllo Qualità di Processo e Metriche	44
3.3	Configurazione	44
3.3.1	Controllo di Versione	44
3.3.1.1	Struttura del Repository	45
3.3.1.2	Processo di Implementazione	47
3.3.1.3	Uso dei Branch	47
3.3.1.4	Ciclo di Vita dei Branch	47
3.3.1.5	Norme delle Commit	48
3.3.1.6	Norme dei Merge tra Branch	48
3.3.1.7	Rilascio di Versione	49
3.3.2	Configurazione Versionamento	49
3.3.2.1	Remoto	49
3.3.2.2	Locale	49
3.4	Verifica	49
3.4.1	Descrizione	49

3.4.2	Analisi Statica	50
3.4.3	Analisi Dinamica	50
3.4.3.1	Classificazione dei Test	50
3.4.3.2	Test di Unità	51
3.4.3.3	Test di Integrazione	51
3.4.3.4	Test di Sistema	51
3.4.3.5	Test di Validazione	51
3.4.4	Verifica dei Diagrammi UML	51
3.4.5	Strumenti	52
3.5	Validazione	52
3.5.1	Descrizione	52
3.5.2	Procedure	52
3.5.2.1	Validazione Software	52
4	Processi Organizzativi	54
4.1	Processi di Coordinamento	54
4.1.1	Gestione Comunicazioni	54
4.1.1.1	Comunicazioni Interne	54
4.1.1.2	Comunicazioni Esterne	54
4.1.2	Gestione Riunioni	55
4.1.2.1	Riunioni Interne	55
4.1.2.2	Riunioni Esterne	55
4.1.2.3	Verbale di Riunione	55
4.2	Processi di Pianificazione	56
4.2.1	Ruoli di Progetto	56
4.2.1.1	<i>Responsabile di Progetto</i>	56
4.2.1.2	<i>Amministratore di Progetto</i>	57
4.2.1.3	<i>Analista</i>	57
4.2.1.4	<i>Progettista</i>	58
4.2.1.5	<i>Programmatore</i>	58
4.2.1.6	<i>Verificatore</i>	58
4.2.1.7	Rotazione dei Ruoli	59
4.2.2	Gestione degli Strumenti di Coordinamento	59
4.2.2.1	Task	59
4.2.2.2	Ticket	60
4.3	Gestione di Progetto	60
4.3.1	Configurazione Strumenti di Organizzazione	60
4.3.1.1	Inizializzazione	60

4.3.1.2	Aggiunta di Milestone	61
4.3.2	Ciclo di Vita delle Task	61
4.3.2.1	Apertura	61
4.3.2.2	Completamento	61
4.3.2.3	Richiesta di Revisione	61
4.3.2.4	Chiusura	62
4.3.2.5	Riapertura	62
4.4	Strumenti	62
4.4.1	Sistema Operativo	62
4.4.2	Ambiente di Test del Prodotto	62
4.4.3	Versionamento e Issue Tracking	63
4.4.3.1	<i>Git</i>	63
4.4.3.2	<i>GitHub</i>	63
4.4.3.3	<i>GitLab</i>	63
4.4.4	Comunicazione	63
4.4.4.1	<i>Telegram</i>	63
4.4.4.2	<i>Slack</i>	63
4.4.5	Diagrammi di Gantt	64
4.5	Formazione del Gruppo	64
A	PDCA	65
B	ISO/IEC 9126:2001	67
C	CMMI	69
D	Modello a V	71
E	CI/CD	72
E.1	CI	72
E.2	CD	73
F	Metriche	74
F.1	Metriche per la Qualità di Processo	74
F.1.1	Gestione dei Task	74
F.1.2	PR02: Gestione dei Costi	74
F.1.3	PR03: Verifica del Software	75
F.1.4	PR04: Gestione dei Rischi	76
F.1.5	PR05: Gestione dei Test	76



F.1.6	PR06: Versionamento e Build	78
F.2	Metriche per la Qualità di Prodotto	78
F.2.1	Leggibilità	78
F.2.2	Funzionalità	78
F.2.3	Affidabilità	79
F.2.4	Efficienza	80
F.2.5	Usabilità	80
F.2.6	Manutenibilità	81
F.2.7	Portabilità	81

Elenco delle tabelle

1	Registro delle Modifiche	5
---	--------------------------	---

Elenco delle figure

1	Rappresentazione del Processo di CI/CD presso GitLab. Immagine dal Sito Web del produttore: https://docs.gitlab.com/ee/ci/	20
2	Rappresentazione del Funzionamento di Telegraf e InfluxDB. Immagine dal Sito Web del Produttore: https://www.influxdata.com/time-series-platform/telegraf	21
3	Validazione del Software	53
4	Modello PDCA. Immagine dal sito web https://www.mindtools.com/pages/article/newPPM_89.htm	66
5	ISO/IEC9126:2001. Immagine dal sito web https://it.wikipedia.org/wiki/ISO/IEC_9126	68
6	Capability Maturity Model Integration - CMMI. Immagine dal sito https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration	70
7	Modello a V. Immagine dal sito web https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L16.pdf	71
8	Rappresentazione del processo di CI/CD https://docs.gitlab.com/ee/ci/	72

1 Introduzione

1.1 Scopo del Documento

Il documento *Norme di Progetto v3.0.0* si propone di definire le regole che i membri del gruppo **Agents of S.W.E.** sono tenuti a rispettare durante tutto lo svolgimento del progetto "*G&B*", al fine di garantire quanto più possibile l'uniformità del materiale prodotto.

Le norme presenti in questo documento verranno prodotte incrementalmente, al progressivo maturare delle esigenze e delle attività di progetto. Di conseguenza il documento, allo stato corrente, non è da considerarsi completo.

Il documento sarà perciò aggiornato e sottoposto a più revisioni.

1.2 Ambiguità e Glossario

I termini che potrebbero risultare ambigui all'interno del documento sono siglati tramite pedice rappresentante la lettera G, tale terminologia trova una sua più specifica definizione nel *Glossario v3.0.0* che viene fornito tra i Documenti Esterni.

1.3 Riferimenti

1.3.1 Referimenti Normativi

- **Capitolato d'Appalto C3:**

- <https://www.math.unipd.it/~tullio/IS-1/2018/Progetto/C3.pdf>;

- **ISO/IEC 9126:2001:**

- <https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L13.pdf>;

- **Modello a V:**

- <https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L16.pdf>;
 - <https://www.hintsw.com/it/safety-engineering/pianificazione-e-concezione-del-sw/modello-a-v-di-sviluppo-del-sw.html>.

- **Metriche:**

- <https://www.qasymphony.com/blog/64-test-metrics>;
 - <https://www.softwaretestinghelp.com/software-test-metrics-and-measurements/>.

- **UML 2.0:**

- <https://www.omg.org/spec/UML/2.0/>.

1.3.2 Referimenti Informativi

- **Standard ISO/IEC 12207:1995:**
https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
- **Presentazione Capitolato:**
<https://www.math.unipd.it/~tullio/IS-1/2018/Progetto/C3p.pdf>;
- **Materiale Didattico del Corso di Ingegneria del Software:**
 - **Gestione di Progetto:**
<https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L06.pdf>;
 - **Regole del Progetto Didattico:**
<https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/P01.pdf>;
 - **Regolamento Organigramma:**
<https://www.math.unipd.it/~tullio/IS-1/2018/Progetto/RO.html>.
- **Sviluppo:**
 - <https://www.influxdata.com/time-series-platform/telegraf/>;
 - <https://www.influxdata.com/time-series-platform/influxdb/>;
 - <https://jestjs.io/>;
 - <https://www.jetbrains.com/webstorm/>;
 - <https://webpack.js.org/>.
 - <https://angularjs.org/>;
 - <http://codecov.io>;
 - <https://jquery.com/>;
 - <https://nodejs.org/it/>.

All'interno del documento eventuali ulteriori riferimenti normativi o informativi vengono contrassegnati da apice¹ e riportati a piè pagina.

1

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 "*G&B*".

2.1.1 Studio di Fattibilità

Lo *Studio di Fattibilità v1.0.0* riguarda l'analisi dei capitolati, la specificazione di pro e contro di ognuno di essi e le motivazioni che hanno portato all'esclusione di questi, tranne per il capitolato C3, ovvero *G&B*, che è stato scelto del gruppo e, per questo, verranno redatte le motivazioni che hanno portato alla scelta. La stesura di questo documento è effettuata dagli analisti. Tale documento è così strutturato:

- **Informazioni del Capitolato:** sezione che contiene il nome del fornitore e il nome del capitolato;
- **Descrizione Capitolato e Obiettivo Finale:** descrizione obiettivo finale del capitolato e ambito di utilizzo del prodotto;
- **Dominio Tecnologico:** definisce il dominio tecnologico, ovvero le tecnologie richieste dall'azienda;
- **Valutazione del Capitolato:** definisce aspetti positivi, negativi, e criticità, motivando la scelta del capitolato.

2.1.2 Rapporti di Fornitura con la Proponente

Una volta scelto il capitolato, si intende instaurare un rapporto quanto più costante e profittevole con la proponente *Zucchetti S.p.A.* allo scopo di:

- stabilire un accordo in merito allo sviluppo, al mantenimento, al funzionamento e alla consegna del prodotto;
- realizzare un prodotto che soddisfi totalmente i requisiti obbligatori concordati e quanto più possibile quelli desiderabili;
- stimare i costi;

- concordare la qualifica del prodotto.

A seguito della consegna del prodotto, salvo ulteriori accordi, non seguirà l'attività di manutenzione.

2.1.3 Collaudo e Consegna del Prodotto

Prima di consegnare il prodotto, il team dovrà eseguire una dimostrazione del prodotto in presenza del proponente e del committente attraverso un collaudo. Il gruppo, prima del collaudo, deve assicurarsi della correttezza, completezza ed affidabilità delle componenti software, affinché durante il collaudo si possano dimostrare i seguenti punti:

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

Concluso e superato il collaudo finale, il *Responsabile* di progetto presenterà al committente il consuntivo finale e consegnerà il prodotto software su un supporto fisico comprensivo di:

- utilità di installazione e istruzioni per l'uso;
- codice sorgenti completi e utilità di compilazione;
- documentazione completa e finale;
- eventuali utilità di collaudo.

2.2 Sviluppo

2.2.1 Analisi dei Requisiti

È un documento stilato dagli Analisti con lo scopo di riportare tutti i requisiti_G del capitolato. Le informazioni possono essere recuperate da più fonti, tra cui:

- **Riunioni Esterne:** incontri con il cliente con lo scopo di approfondire aspetti critici;

- **Riunioni Interne:** riunioni del gruppo dove l'obiettivo è discutere i casi d'uso del capitolato e studiare il dominio di utilizzo;
- **Documentazione:** analisi della documentazione offerta dall'azienda.

Il risultato dello studio sarà un documento verificato che contiene:

- Descrizione generale del prodotto;
- Argomentazioni precise ed affidabili per i *Progettisti*;
- Diagrammi UML che rappresentano i casi d'uso;
- Funzionalità finali accordate con il cliente;
- Stima dei costi.

2.2.1.1 Classificazione dei Requisiti

Il requisito si può definire come:

- Capacità necessaria a un utente per raggiungere un obiettivo;
- Capacità del sistema per soddisfare un obbligo da contratto;
- Descrizione documentata di una capacità.

La classificazione dei requisiti aiuta a mettere ordine e a facilitare la comprensione e il mantenimento futuro del sistema. Si possono dividere in:

- **Attributi di Prodotto:** specificano "*cosa*" bisogna svolgere e definiscono i requisiti funzionali, prestazionali e di qualità;
- **Attributi di Processo:** specificano "*come*" bisogna svolgere definendo norme contrattuali e realizzative.

Ogni requisito, nel documento, deve seguire le seguenti regole di identificazione:

R[Importanza][Tipo][Identificativo]

- **R:** sta per appunto "requisito";
- **Importanza:** specifica quanto è importante il requisito, può assumere tre valori:
 - **O:** indica che il requisito è obbligatorio, e deve essere per forza soddisfatto per il corretto funzionamento di base del sistema;

- **D**: indica che il requisito è desiderabile, se viene soddisfatto aumenta la completezza del sistema, se non viene soddisfatto non crea alcuna penalizzazione;
- **F**: indica un requisito opzionale (F sta per "facoltativo").
- **Tipo**: specifica la tipologia e può assumere i seguenti valori:
 - **F**: requisito funzionale;
 - **P**: requisito prestazionale;
 - **Q**: requisito qualitativo;
 - **V**: requisito di vincolo.
- **Identificativo**: numero progressivo che identifica il requisito, strutturato come segue:

[codice padre].[codice figlio]

2.2.1.2 Classificazione dei Casi d'Uso

Un caso d'uso consiste nel valutare ogni requisito con lo scopo di definire gli attori (utenti esterni) all'interno di un scenario che hanno un obiettivo finale comune. Il gruppo, nel documento, oltre a fornire una spiegazione verbosa e dettagliata dei casi d'uso, fornirà una rappresentazione grafica nel linguaggio UML. Si è stabilito la seguente classificazione dei casi d'uso:

UC[codice padre].[codice identificativo]

dove:

- **Codice Padre**: indica il codice identificativo del caso d'uso che ha generato il corrente, se non è stato generato da nessun caso d'uso allora viene tralasciato;
- **Codice Identificativo**: indica il codice univoco del caso d'uso corrente, è rappresentato da una cifra.

Ogni caso d'uso avrà le seguenti informazioni:

- **Identificativo**: codice univoco del caso d'uso;
- **Attori**: identifica gli attori principali e attori secondari del caso d'uso;
- **Precondizioni**: identifica le condizioni sempre vere prima degli eventi del caso d'uso;

- **Postcondizioni:** identifica le condizioni sempre vere dopo gli eventi del caso d'uso;
- **Scenario:** sequenza di passi che descrivono interazioni tra gli attori e il sistema, possono creare scenari secondari a seconda del comportamento dell'utente, per esempio dalla log-in passare alla registrazione.

2.2.2 Progettazione

2.2.2.1 Descrizione

L'attività di Progettazione consiste nel descrivere una soluzione che sia soddisfacente per gli stakeholder_G. È compito dei *Progettisti* svolgere tale attività, definendo l'architettura del prodotto finale, mantenendo chiare e riusabili le componenti, restando nei costi prefissati.

La progettazione viene svolta in due periodi distinti: nel periodo di Progettazione Architetturale verranno studiati i design pattern per la realizzazione dell'architettura. Nel periodo di Progettazione di Dettaglio e Codifica la progettazione diventa atomica per ogni componente del sistema, in modo che i *Programmatori* possano sviluppare il codice in maniera parallela e focalizzandosi su specifiche task.

L'architettura definita dovrà soddisfare le seguenti qualità:

- **Sufficienza:** soddisfare i requisiti definiti nel documento *Analisi dei Requisiti v3.0.0*;
- **Modularità:** essere divisa in parti chiare e distinte, ognuna con la sua specifica funzione;
- **Robustezza:** essere in grado di gestire malfunzionamenti in modo da rimanere operativa di fronte a situazioni erronee improvvise;
- **Flessibilità:** permettere modifiche a costo contenuto nel caso in cui i requisiti dovessero evolversi o se ne dovessero aggiungere di nuovi;
- **Riusabilità:** essere costruita in modo da poter permettere il riutilizzo di alcune sue parti;
- **Efficienza:** soddisfare i requisiti utilizzando il minor numero di risorse possibili;
- **Affidabilità:** garantire che i servizi esposti siano sempre disponibili;
- **Disponibilità:** la manutenzione deve essere effettuata in tempi rapidi, evitando di rendere indisponibile il servizio;

- **Sicurezza:** essere sicura rispetto ad intrusioni e malfunzionamenti;
- **Semplicità:** prediligere la semplicità, contenendo solo il necessario;
- **Coesione:** raggruppare le parti con un obiettivo comune, in modo da avere maggiore manutenibilità e riusabilità;
- **Basso accoppiamento:** parti distinte devono essere poco dipendenti tra loro, in modo da poter essere facilmente manutenibili.

2.2.2.2 Obiettivi della Progettazione

La Progettazione serve per garantire che il prodotto sviluppato soddisfi le proprietà ed i requisiti delineati durante l'attività di analisi, ponendosi come obiettivi:

- Costruire un'architettura logica del prodotto;
- Garantire la qualità di prodotto sviluppato, perseguendo la correttezza per costruzione;
- Rendere chiara e comprensibile l'architettura progettata per gli stakeholder;
- Organizzare e ripartire compiti implementativi, riducendo la complessità del problema originale fino alle singole componenti facilitandone la codifica da parte dei singoli *Programmatori*;
- Mantenere bassa la dipendenza tra le varie parti del prodotto favorendo la modularità ed il riuso;
- Ottimizzare l'uso delle risorse.

2.2.2.3 Sviluppo

Lo sviluppo di $G\mathcal{E}B$ avviene seguendo il modello incrementale, spiegato nel dettaglio nel documento *Piano di Progetto v3.0.0* alla sezione §4.

La proponente nel capitolato d'appalto pone un singolo vincolo riguardo le tecnologie da utilizzare:

- *JavaScript*: in particolare nella sua declinazione nota come *ECMAScript 6_G*.

Tuttavia, ai fini della realizzazione del prodotto finale, il team **Agents of S.W.E.** ha definito l'uso di ulteriori tecnologie, analizzate di seguito, per scopi diversi:

- *Telegraf_G* : è un agente per la raccolta e la rilevazione periodica di metriche² e dati. Si connette ad un database_G e vi salva tali dati;

²Metriche d'uso di un elaboratore. Per esempio: percentuale d'uso della CPU, pressione di memoria, etc.

- *InfluxDB_G* : è un database_G per le serie temporali. Verrà utilizzato per il salvataggio dei dati raccolti da *Telegraf*;
- *JSBayes_G* : libreria suggerita dal proponente per la semplice definizione di reti bayesiane ed il relativo calcolo probabilistico di eventi;
- *JSBayes-Viz_G* : libreria la quale utilizza la libreria precedentemente citata *JSBayes*, al fine di creare una mappa grafica della rete bayesiana creata contenente i relativi nodi con ognuno le proprie probabilità indicate;
- *GitLab_G* : piattaforma open-source per la gestione di repository *Git*. Il team utilizzerà un repository salvato su tale piattaforma in quanto permette la gestione di una pipeline di CI/CD_G , evitando quindi l'uso di altri strumenti. Il processo di CI/CD si configura come rappresentato nell'immagine e viene spiegato nell'appendice [E](#) ;
- *jQuery_G* : è una libreria *JavaScript* per applicazioni web. Viene utilizzato all'interno del progetto per le *AJAX_G* request.

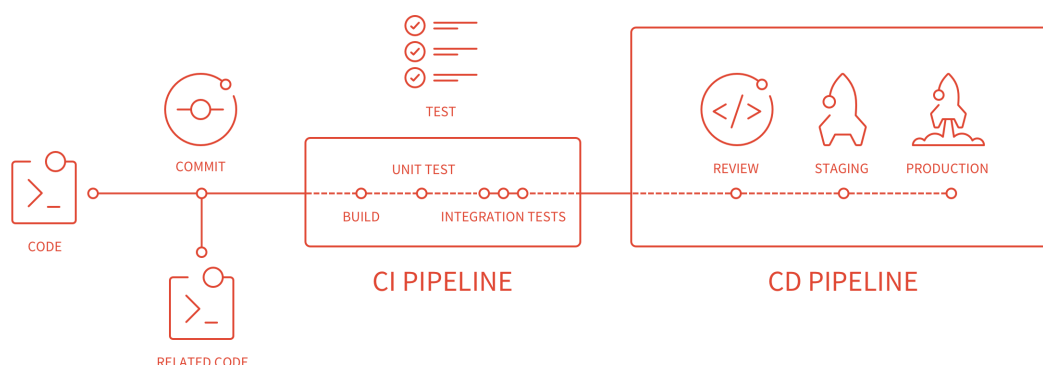


Figura 1: Rappresentazione del Processo di CI/CD presso GitLab. Immagine dal Sito Web del produttore: <https://docs.gitlab.com/ee/ci/>

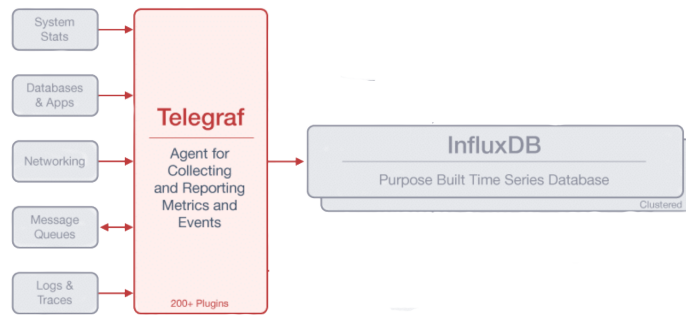


Figura 2: Rappresentazione del Funzionamento di Telegraf e InfluxDB. Immagine dal Sito Web del Produttore: <https://www.influxdata.com/time-series-platform/telegraf>

2.2.2.4 Integrazione

L'attività di integrazione sarà effettuata utilizzando il servizio disponibile su *GitLab*, piattaforma per repository utilizzata dal gruppo per il versionamento dei file sorgente.

Questo servizio implementa un modello di integrazione continua: ad ogni commit_G nel repository_G, *GitLab* creerà automaticamente la build ed andrà ad eseguire i test di unità. In questo modo è possibile rilevare tempestivamente eventuali errori.

Per quanto riguarda il code coverage_G si utilizzerà un servizio messo a disposizione da *GitLab*.

2.2.2.5 Diagrammi

Per rendere più chiare le scelte progettuali adottate, si farà largo uso di diagrammi UML 2.0_G. Essi potranno essere di diversi tipi:

- **Diagrammi dei Casi d'Uso:** dedicati alle funzionalità offerte dal sistema;
- **Diagrammi delle Classi:** dedicati alla descrizione degli oggetti che fanno parte del sistema, e le loro dipendenze;
- **Diagrammi dei Package:** dedicati alla descrizione delle dipendenze tra gli oggetti raggruppati in un package;
- **Diagrammi di Sequenza:** dedicati alla descrizione della collaborazione nel tempo tra un gruppo di oggetti;
- **Diagrammi di Attività:** dedicati alla descrizione della logica procedurale.

2.2.3 Codifica

Di seguito vengono definite delle norme che devono essere adottate dai *Program-matori* 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 il linguaggio principale *ECMAScript 6_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 .

2.2.3.1 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 //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 .

2.2.3.2 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: nameFile
3  * Type: fileType
4  * Creation date: yyyy-mm-gg
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.

2.2.3.3 *ECMAScript 6*

Seguendo le indicazioni presenti nella documentazione dell'azienda fornitrice di *Grafana*, la piattaforma per cui si intende sviluppare il plug-in, il team ha deciso di adottare come linguaggio di programmazione principale *ECMAScript 6*³.

ECMAScript 6 viene standardizzato da *ECMA_G* nel giugno 2015 con la sigla "ECMA-262".

Come stile di codifica si adottano le linee guida proposte da *Airbnb JavaScript Style Guide*. Per la verifica dell'adesione a tali norme, i *Programmatori* devono utilizzare, come suggerito dalla documentazione proposta da *Grafana*, *ESLint_G*.

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()
```

³Linguaggio divenuto standard ISO: ISO/IEC 16262:2011, e relativo aggiornamento ISO/IEC 22275:2018.


```
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*:

```
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_G*, per coerenza, lo si preferisca alle array function native di *ES6*.

Verranno esaminate di seguito le norme in merito allo stile di codifica che i *Programmatici* 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 siano contenuti tra graffe: nel caso non vengano utilizzate, la definizione deve essere inline, cioè sulla stessa riga. Nel seguente modo:

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

Commenti al Codice

Norma 1

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 }
```

Norma 2

Il codice sviluppato va adeguatamente commentato con i commenti di documentazione appositi:

```
1 /**
2  * Function or class description ...
3  */
```

Il team fa riferimento a quanto indicato dal sito *@use JSDoc*.

In particolari vengono ritenuti necessari, qualora utilizzato, il riferimento alle seguenti parti:

- **@extends**: da utilizzare nell'intestazione della classe se la stessa ne estende un'altra;
- **@param**: da utilizzare per descrivere l'uso di un parametro di una funzione;
- **@return**: da utilizzare per descrivere il tipo di valore ritornato da una funzione;
- **@module**: da utilizzare per indicare un modulo importato;
- **@throws**: da utilizzare per descrivere quali eccezioni possono essere lanciate.

Di seguito un esempio di utilizzo:

```
1 **
2 * Class representing a dot.
3 * @extends Point
4 */
5 class Dot extends Point {
6     /**
7      * Create a dot.
8      * @param {number} x - The x value.
```



```
9      * @param {number} y - The y value.
10     * @param {number} width - The width of the dot, in pixels.
11     */
12     constructor(x, y, width) {
13         // ...
14     }
15
16     /**
17     * Get the dot's width.
18     * @return {number} The dot's width, in pixels.
19     */
20     getWidth() {
21         // ...
22     }
23
24     /**
25     * Set the dot's width.
26     * @param {number} width - The width of the dot, in pixels.
27     * @throws {Error} Will throw an error if the argument is lower than 0.
28     */
29     setWidth(w) {
30         if(w < 0) throw new Error('Negative width');
31         this.width = w;
32     }
33 }
```

Variabili

Norma 1

Fare riferimento alle norme 1 e 2, all'inizio della sezione §2.2.3.3.

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 avere 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 }
```

2.2.3.4 CSS

Per la parte front-end_G del plug-in il gruppo **Agents of S.W.E.** ha scelto di utilizzare i linguaggi *CSS3_G* ed *HTML5_G*.

Qui di seguito verrà descritta la Style Guide resa disponibile da *Airbnb CSS-in-JavaScript Style Guide*.

Indentazione

Norma 1

Tra blocchi adiacenti dello stesso livello deve essere lasciata una linea bianca, tra l'una e l'altra. In questo modo il codice risulterà essere più leggibile.

Di seguito, un esempio della corretta indentazione:

```
1 //OK
2 {
3   bigBang: {
4     display: 'inline-block',
5     '::before': {
6       content: "''",
7     },
8   },
9   universe: {
10    border: 'none',
11  },
12 }
13
14 //NO
15 {
16   bigBang: {
17     display: 'inline-block',
18
19     '::before': {
20       content: "''",
21     },
22   },
23
24   universe: {
25     border: 'none',
26   },
27 }
```

Per le restanti regole si farà riferimento allo standard imposto dalla World Wide Web Consortium. Per essere considerato corretto il file *.css* dovrà essere validato tramite il sito reso disponibile dall'organizzazione, precedentemente nominata. La validazione potrà essere effettuata tramite il sistema messo a disposizione dalla W3C.

2.2.3.5 *HTML*

Insieme al *CSS3*, per lo sviluppo della parte front-end del progetto, il gruppo ha scelto di utilizzare *HTML5*. Per la stesura della Style Guide verrà utilizzata la documentazione resa disponibile da W3C.

Di seguito saranno riportate le principali norme da seguire:

Norma 1

Il tipo di documento deve sempre essere dichiarato all'inizio del documento in maiuscolo, in questo modo:

```
1 <!DOCTYPE html>
```

Norma 2

I nomi degli elementi ed attributi devono essere sempre scritti in minuscolo:

```
1 <tag attr1="value"/> <!-- OK -->
2
3 <TAG ATTR1="value"/> <!-- NO -->
```

Norma 3

Tutti gli elementi *HTML* devono essere chiusi, anche se vuoti:

```
1 <tag /> <!-- OK -->
2 <tag></tag> <!-- OK -->
3
4 <tag> <!-- NO -->
```

Norma 4

Il valore degli attributi deve essere racchiuso dentro doppio apice:

```
1 <tag attr1="value"/> <!-- OK -->
2
3 <tag attr1='value' /> <!-- NO -->
```

Norma 5

Intorno al simbolo di uguaglianza (=) non devono essere presenti spazi:

```
1 <tag attr1="value"/> <!-- OK -->
2
3 <tag attr1 = 'value' /> <!-- NO -->
```

Norma 6

Devono essere usati due (2) spazi e non le tabulazioni. Gli spazi bianchi vanno uti-

lizzati solamente per ampi blocchi di codice, per una maggiore leggibilità;

```
1 <!-- OK -->
2 <rootTag>
3   ..<childTag>
4     ....<subChildTag/>
5   ..</childTag>
6 </rootTag>
7
8 <!-- NO -->
9 <rootTag>
10   <childTag>
11     <subChildTag/>
12   </childTag>
13 </rootTag>
```

Norma 7

Nonostante i tag `html`, `body` e `head` possano essere omessi, vanno inseriti affinché siano compatibili con tutti i browser:

```
1 <!-- OK -->
2 <html>
3   <head>
4     ...
5   </head>
6   <body>
7     ...
8   </body>
9 </html>
```

Norma 8

Le classi che verranno dichiarate all'interno dell'*HTML*, al fine di andare poi a ridefinire il *CSS*, dovranno essere quelle rese disponibili dalla documentazione di *Grafana*.

Equivalentemente al *CSS*, anche il codice *HTML* dovrà essere validato. Questa validazione avverrà tramite il sito reso disponibile dalla W3C, al seguente [link](#).

2.2.3.6 *AngularJS*

Insieme al *CSS* e all'*HTML* il team ha deciso di sviluppare la parte front-end appoggiandosi al framework *JavaScript Angularjs_G*.

AngularJS è un framework per applicazioni web, sviluppato con lo scopo di affrontare le difficoltà incontrate con lo sviluppo di applicazioni su singola pagina di tipo client-side.

La caratteristica principale è l'introduzione di funzionalità chiamate direttive, che permettono di creare componenti *HTML* personalizzati e riusabili con lo scopo di associare a ogni direttiva un comportamento, così nascondendo la complessità della struttura DOM_G . Le direttive sono riconoscibili poiché integrate nei tag con la dicitura "ng-" e susseguite dal nome (scelto dallo sviluppatore). Di seguito sono riportate alcune norme da seguire:

Norma 1

Verrà seguito il pattern Model View Controller(MVC) $_G$ che propone la scomposizione in tre componenti:

- **Model:** responsabile per il mantenimento e l'elaborazione dei dati;
- **View:** responsabile per la visualizzazione dei dati all'utente;
- **Controller:** si occupa di controllare l'interazione tra Model e View.

Norma 2

Il controller non deve essere sovraccaricato di funzionalità che non gli appartengono. In particolare il controller non deve:

- manipolare il DOM, questo renderà i controller difficili da testare. Questo compito è riservato alle direttive;
- formattare l'input, a questo scopo vengono usati i form *Angular*;
- formattare l'output, sarà compito dei filtri;
- condividere dati con altri controller;
- implementare funzionalità generali, le funzionalità che non riguardano direttamente l'interazione tra dati e la visualizzazione, deve essere fatta nei servizi.

Norma 3

Il nome dei moduli seguono la notazione lowerCamelCase;

```
1 <div ng-app="myModule">...</div>
```

Norma 4

Quando è presente un modulo b che è submodule di a , si possono concatenare usando la notazione $a.b$;

Norma 5

Il nome dei controller viene assegnato in base alla loro funzionalità e viene usata la notazione UpperCamelCase;

```
1 <div ng-controller="MyController">
2   {{ greeting }}
3 </div>
```

Norma 6

I nomi delle restanti direttive seguono la notazione camelCase;

```
1 <select class="myClass" ng-model="myModel">
```

Norma 7

Le direttive *AngularJS* vengono scritte alla fine degli attributi di un tag:

```
1 <!-- OK -->
2 <p>Nome: <input type="text" ng-model="utente.nome"></p>
3
4 <!-- NO -->
5 <p>Nome: <input ng-model="utente.nome" type="text"></p>
```

Norma 8

Il prefisso $\$$ è riservato ad *AngularJS*, non deve essere usato per nomi di variabili, proprietà o metodi;

Norma 9

I controller non devono essere definiti globali;

Norma 10

Usare prefissi personalizzati per le direttive per evitare collisioni con librerie di terze parti;

Norma 11

In *AngularJS* viene adottato il two-way binding, ogni modifica al modello dati si riflette sulla view e ogni modifica alla view viene riportata sul modello dati.

Questa sincronizzazione avviene senza la necessità di scrivere codice particolare, è

sufficiente associare il modello allo scope all'interno del controller ed utilizzare la direttiva `ng-model` nella view.

2.2.3.7 *NodeJS*

NodeJS è una runtime di *JavaScript* Open source multiplatforma orientato agli eventi per l'esecuzione di codice *JavaScript* Server-side, costruita sul motore *JavaScript V8* di *Google Chrome*. Molti dei suoi moduli base sono scritti in *JavaScript*, e gli sviluppatori possono scrivere nuovi moduli in *JavaScript*.

In particolare, tale tecnologia viene utilizzata dal team **Agents of S.W.E.** per l'implementazione del server che permette il funzionamento del *plug-in*.

Le norme di codifica da seguire per *NodeJS* sono le stesse di *ECMAScript 6* con ulteriori norme specifiche riportate di seguito:

Norma 1

Iniziare un nuovo progetto usando il comando `NPM init` come riportato di seguito:

```
1      $ mkdir my-project
2      $ cd my-project
3      $ npm init
```

Norma 2

Il codice deve essere partizionato in componenti ognuno nella propria cartella, garantendo così che ogni unità sia piccola e semplice.

Norma 3

Per i moduli più complessi e composti da più file, è possibile includere l'intero set di file con un'unica chiamata alla funzione `requireG`. In questo caso è però necessario definire un file descrittore all'interno del modulo che permette al motore `V8G` di conoscere quali sono i file da includere.

Norma 4

Richiedere i moduli all'inizio di ogni file, evitando di richiederli all'interno di funzioni, permettendo così di individuare facilmente le dipendenze all'inizio del file.

Norma 5

Usare `async-awaitG` o `promiseG` per la gestione di errori asincroni, che consente di avere un codice più compatto e familiare.

Norma 6

Evitare l'uso di funzioni anonime, nominare ogni funzione permetterà una più facile comprensione in fase di lettura.

2.2.4 Strumenti

2.2.4.1 *WebStorm*

Come ambiente di sviluppo integrato comune il team ha scelto di utilizzare l'IDE *WebStorm*, facente parte della famiglia di prodotti di JetBrains.

WebStorm è una cross-platform, sviluppata principalmente per lo sviluppo web, *JavaScript* e *TypeScript*.

La configurazione della piattaforma di sviluppo dovrà essere equivalente per tutti i componenti del gruppo. All'interno della piattaforma dovrà essere installato il plug-in *ESLint*, la cui funzione verrà illustrata in seguito.

2.2.4.2 *NPM*

Npm è un gestore di pacchetti per *JavaScript*, gestisce inoltre i pacchetti di default presenti nell'ambiente *Node.js*_G.

Risulta essere strutturato in due parti: un client a riga di comando ed un database, contenente pacchetti sia pubblici che privati. Lo scopo di *node.js* è incoraggiare il riuso del codice tra team ed il suo utilizzo è mirato alla gestione delle dipendenze.

Il suo utilizzo è necessario per poter eseguire e testare il codice *ES6*.

2.2.4.3 *ESLint*

Il plug-in *ESLint*, scaricabile dall'apposita sezione all'interno dell'IDE utilizzato, ha la funzione di segnalare errori che vengono effettuati durante la stesura del codice, in modo da evitare bug e rendere il codice più coerente.

Si occupa inoltre di far aderire il codice allo standard *Airbnb* che il team ha deciso di adottare, segnalando le varie difformità.

Il plug-in è reperibile tra i pacchetti resi disponibili da *NPM*.

Le regole utilizzate all'interno del plug-in saranno quelle esplicitate nella sezione §2.2.3.3, riguardanti lo stile e le regole da rispettare per la stesura del codice con *EcmaScript6*.

2.2.4.4 *Webpack*

Webpack è un module bundler_G per applicazioni *javascript*. Nella pratica lo scopo di *webpack* è creare un pacchetto di assets leggibile e utilizzabile dal browser a partire

da un insieme di file sorgenti strutturati su diversi file e con dipendenze complesse. Gli aspetti fondamentali sono:

- **Entry Point:** rappresenta il punto di partenza da cui iniziare la build e creare lo schema delle dipendenze, di default è `./src/index.js` ma può essere modificato nei file di configurazione;
- **Output:** rappresenta la destinazione in cui verranno generati i file leggibili dal browser, di default è `./dist` e il file principale è `./dist/main.js`;
- **Loaders:** di default *webpack* esegue la build solamente di file *javascript* e *JSON*, tramite questa configurazione *webpack* può essere adattato ad altri tipi di file, così da permettere la conversione anche di essi. I loaders possono avere due proprietà:
 - La proprietà `test` che identifica quali file dovrebbero essere trasformati;
 - La proprietà `use` che identifica quali loaders utilizzare per eseguire la trasformazione.
- **Plugins:** a differenza dei loaders, i plugins operano in aspetti più generali riguardanti l'ottimizzazione dei pacchetti e la gestione delle risorse. Per utilizzare un plugin bisogna eseguire una `'require'` dalla sorgente ed aggiungerlo mediante la `'new'` nell'array apposito chiamato `'plugin'`.
- **Mode:** viene settato in base all'ambiente in cui lanciato *webpack*, così da importare ottimizzazioni specifiche. Può assumere i seguenti valori:
 - Development;
 - Production;
 - None.

Se non viene specificato viene settato come valore di default `'production'`.

2.2.4.5 Diagrammi UML

Lo strumento scelto dal gruppo per la realizzazione dei diagrammi UML è *Draw.io*, una applicazione online per il disegno di diagrammi UML 2.0. Anche in questo caso è stata tenuta in grande considerazione l'accessibilità dello strumento. Il software in questione risulta infatti gratuito ed essendo online non genera problemi per l'uso su più piattaforme.

Draw.io risulta essere sia User Friendly, con un'interfaccia utente chiara e consistente, sia abbastanza potente da essere utilizzato anche scopi più evoluti. Questo, unito



alle dimensioni relativamente ridotte del software, e al fatto che sia gratuito, hanno fatto propendere il gruppo per questa applicazione online, a discapito di alternative, come *Papyrus* o *Astah*, comunque considerate valide.

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 della 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 sviluppa il task_G assegnato dal *Responsabile*. Una volta terminata la fase di scrittura del documento, il redattore lo segnalerà al *Responsabile*, il quale assegnerà ad un *Verificatore* il compito di analizzare il lavoro svolto;
- **Verifica:** è il processo eseguito dai *Verificatori* designati dal responsabile. Il loro compito consiste nel controllare che il redattore abbia scritto il documento adottando le regole indicate nel documento *Norme di Progetto v3.0.0* ed in maniera grammaticalmente e strutturalmente corretta;
- **Approvazione:** 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 ed ad eseguirne il rilascio.

3.1.3 Template

Il gruppo ha deciso di strutturare un template $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ per dare uniformità a tutti i documenti. Il template facilita e velocizza la stesura, poiché i redattori devono concentrarsi solo ed esclusivamente sul contenuto e non sul 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 ed il capitolato scelto. Appena sotto è posizionato il titolo del documento ed 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) ed i destinatari.

3.1.4.2 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 strutturare 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. Dopo l'approvazione da parte del responsabile, verrà creato un file con formato *.pdf* che rappresenta la pubblicazione ufficiale.

3.1.4.3 Struttura Indice

In tutta la documentazione, fatta eccezione per i verbali, dopo la pagina di presentazione è presente l'indice del documento. La struttura è standard: numero e titolo del capitolo, eventuali sottosezioni e paragrafi con indicato a fianco la pagina del contenuto. Ogni titolo è un link alla pagina del contenuto.

3.1.4.4 Struttura Tabelle

Il gruppo ha deciso di standardizzare il formato delle tabelle come descritto:

- Intestazione a sfondo blu e scritte bianche;
- Corpo con righe alternate di colore bianco e grigio, per facilitare la lettura.

3.1.4.5 Elenco Tabelle

In tutti i documenti è presente un elenco delle tabelle utilizzate, ove presenti. È posizionato appena dopo l'indice.

3.1.4.6 Elenco Figure

In tutti i documenti è presente un elenco delle figure utilizzate, ove presenti. È posizionato appena dopo l'elenco delle tabelle.

3.1.4.7 Elenco Riferimenti

In tutti i documenti, se necessario, è presente un elenco dei riferimenti alle fonti utilizzate per stilare il documento in esame. È posizionato all'inizio del documento, dopo l'elenco delle tabelle e delle figure.

3.1.4.8 Piè di Pagina

Tutte le pagine, escluso il frontespizio, hanno a destra il numero della pagina ed il numero di pagine totali, così rappresentate: Pagina x di n .

3.1.4.9 Registro delle Modifiche

Ogni documento, eccezion fatta per verbali e lettera di presentazione, presenta un registro delle modifiche chiamato "Registro delle Modifiche". È strutturato sotto forma di tabella, che contiene in ordine cronologico tutte le modifiche identificabili dalla versione. Ogni riga contiene la versione del documento, la data, il nome di chi ha effettuato la modifica, con il suo corrispettivo ruolo, ed 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 ;
- **Nome Gruppo:** in qualsiasi documento, quando si fa riferimento al gruppo si è deciso di adottare il seguente font: **Agents of S.W.E.**;
- **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. La prima lettera di ogni item dovrà essere maiuscola;
- **Stile testo:**

- **Corsivo:** è utilizzato per citare tecnologie esterne, per riferimenti a documentazione (es. *Analisi dei Requisiti*, *Piano di Progetto*, ecc.), estensione dei file (es. *.pdf*, *.tex*, ecc.) e per identificare i ruoli del progetto (*Responsabile*, *Analista*, ecc.);
- **Grassetto:** le parole in grassetto identificano il titolo di una sezione, sottosezione, paragrafo e un elemento di un elenco puntato;
- **URI:** i link esterni sono evidenti per il colore blu.

- **Formati:**

- **Date:** sono scritte seguendo il formato YYYY-MM-DD, dove YYYY rappresenta l'anno, MM il mese e DD il giorno;
- **Valuta:** si è adottato il formato XXX.YY con Y che denota le cifre decimali, X le cifre intere ed il punto (.) come delimitatore tra decimale e parte intera.

3.1.6 Documenti Correnti

Sono descritti brevemente i documenti formali da consegnare:

- ***Analisi dei Requisiti v3.0.0:*** ha un utilizzo prettamente esterno, inoltre ha l'obiettivo di esporre e scomporre i requisiti_G del progetto. Contiene i casi d'uso ed i diagrammi di interazione con l'utente. Viene redatto dagli *Analisti* dopo una profonda analisi del capitolato ed eventuali colloqui con la proponente;
- ***Glossario v3.0.0:*** ha un utilizzo prettamente esterno ed ha lo scopo di dare una definizione ai termini più specifici usati nei documenti formali;
- ***Norme di Progetto v3.0.0:*** 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 v3.0.0:*** ha un utilizzo esterno, ed espone come il gruppo ha deciso di impiegare le risorse di tempo ed umane;
- ***Piano di Qualifica v3.0.0:*** ha un utilizzo esterno, descrive gli standard e gli obiettivi che il gruppo dovrà raggiungere per garantire la qualità di prodotto e processo;

- **Manuale Utente v1.0.0:** ha un utilizzo esterno, descrive le funzionalità del prodotto e spiega, in modo intuitivo e preciso, le modalità di utilizzo del plug-in *G&B*;
- **Manuale Sviluppatore v1.0.0:** ha un utilizzo esterno, descrive tutte le informazioni necessarie agli sviluppatori che intenderanno estendere o migliorare il plug-in *G&B*.

3.1.7 Ambiente

Per uniformare e strutturare al meglio la scrittura dei documenti il gruppo ha adottato il formato \LaTeX . *TexMaker* è utilizzato per la stesura, si è optato per questo editor perchè open-source ed integra un controllo ortografico della lingua italiana. Per la costruzione dei diagrammi è stato optato per l'utilizzo di script in *Python*, scritti e testati dal gruppo, per uniformare la procedura in tutta la documentazione.

3.2 Qualità

3.2.1 Introduzione

Il contenuto di questa sezione descrive le metriche e i criteri di qualità di processo e prodotto che vengono utilizzati nel documento *Piano di Qualifica v3.0.0*.

3.2.2 Classificazione Processi

Per garantire una corretta struttura, il gruppo ha deciso di introdurre una nomenclatura per identificare i processi descritti nel documento. I processi saranno identificati così:

PR[num]

dove:

- **num:** rappresenta il numero identificativo del processo formato da due cifre intere a partire da 1, unico per tutto il documento.

3.2.3 Classificazione Metriche

Per garantire una corretta struttura, il gruppo ha deciso di introdurre una nomenclatura per identificare le metriche utilizzate. Si potranno quindi identificare così:

MT[mcat][cat][num]

- **mcats**: identifica le metriche in base a quale macrocategoria andranno a misurare. Può assumere i seguenti valori:
 - **PC**: per indicare le metriche di processo;
 - **PD**: per indicare le metriche di prodotto;
 - **TS**: per indicare le metriche di test.
- **cat**: identifica la categoria di appartenenza, se esiste, altrimenti è vuota. Per le metriche di prodotto può assumere i seguenti valori:
 - **D**: per indicare i documenti;
 - **S**: per indicare il software.
- **num**: identificativo univoco formato da due cifre intere a partire da 1.

3.2.4 Controllo Qualità di Processo e Metriche

La qualità di processo è raggiunta tramite l'utilizzo di metodi e modelli che garantiscono il corretto procedimento delle fasi di sviluppo del processo. Il team sfrutta le potenzialità del metodo PDCA, descritto nell'appendice [A](#), ottenendo miglioramenti continui nelle qualità di processo e verifica in modo da avere una maggiore qualità nel prodotto risultante. Inoltre verrà utilizzato lo standard ISO/IEC 15504, comunemente conosciuto con l'acronimo SPICE, che misurerà il livello di maturità dei processi.

Le metriche utilizzate per la valutazione dell'efficacia e dell'efficienza degli stessi sono presentate nell'appendice [F](#).

3.3 Configurazione

3.3.1 Controllo di Versione

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

Il sistema di versionamento, utilizzato nella fase di RR, è *Git*, con il supporto hosting di *GitHub*. Per quanto riguarda la fase di codifica il gruppo ha deciso di appoggiarsi al supporto hosting di *GitLab*, le motivazioni principali riguardano i servizi inclusi in materia di integrazione e pipeline di sviluppo spiegati nelle sezioni [§2.2.2.3](#) e [§2.2.2.4](#).

Successivamente alla fase di RP, il team ha deciso di unire le due repository convogliando quindi documentazione e codice in un unico repository *Git* supportato dalla piattaforma di hosting *GitLab*, al fine di semplificare la gestione del processo.

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" ed 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 incremento del prodotto. Più precisamente, per quanto riguarda i documenti, il repository utilizzato è "Agents-of-S.W.E." ed è caratterizzata da due cartelle principali: "Documentazione" e "Codice".

Per quanto riguarda la prima, essa contiene le sottocartelle "RR", "RP" ed "RQ" che rappresentano la principale struttura in cui sono organizzati i file su cui il gruppo **Agents of S.W.E.** ha lavorato in vista dell'ultima consegna prestabilita.

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

- **Documenti Esterni:**

- *Analisi dei Requisiti v1.0.0;*
- *Glossario v1.0.0;*
- *Piano di Progetto v1.0.0;*
- *Piano di Qualifica v1.0.0;*
- *Verbali Esterni.*

- **Documenti Interni:**

- *Norme di Progetto v1.0.0;*
- *Studio di Fattibilità v1.0.0;*
- *Verbali Interni.*

- **Corrispondenza;**

- **Utility.**

Nello specifico la sottocartella "RP" è caratterizzata dalla seguente struttura di folder:

- **Documenti Esterni:**

- *Analisi dei Requisiti v2.0.0;*
- *Glossario v2.0.0;*
- *Piano di Progetto v2.0.0;*
- *Piano di Qualifica v2.0.0;*
- *Verbali Esterni.*

- **Documenti Interni:**

- *Norme di Progetto v2.0.0;*
- *Verbali Interni.*

- **Corrispondenza;**

- **Utility.**

Nello specifico la sottocartella "RQ" è caratterizzata dalla seguente struttura di folder:

- **Documenti Esterni:**

- *Analisi dei Requisiti v3.0.0;*
- *Glossario v3.0.0;*
- *Piano di Progetto v3.0.0;*
- *Piano di Qualifica v3.0.0;*
- *Manuale Utente v1.0.0;*
- *Manuale Sviluppatore v1.0.0;*

- **Documenti Interni:**

- *Norme di Progetto v3.0.0;*
- *Verbali Interni.*

- **Corrispondenza;**

- **Utility.**

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

La cartella "Corrispondenza" contiene una copia delle mail inviate all'azienda proponente attraverso l'indirizzo `agentsofswe@gmail.com`, la cartella "Utility" invece contiene gli Script_G realizzati dal gruppo.

La cartella "Codice" contiene le sottocartelle corrispondenti alle revisioni.

3.3.1.2 Processo di Implementazione

L'implementazione dei documenti avviene tramite gli strumenti utilizzati nel paragrafo §3.1.7

Quest'ultimi, in fase di compilazione producono dei file di poca rilevanza con estensioni come *.log*, *.out*, *.idx*, *.aux*, *.gz*, *.toc*, i quali verranno ignorati come da configurazione. I file con maggior rilevanza, come *.pdf*, *.tex*, verranno versionati dal sistema di git preinstallato.

Una volta creati e/o modificati i documenti, si procede con il 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. Dopodiché il commit viene pushato_G nel branch appropriato, a seconda dei criteri descritti nel prossimo paragrafo.

3.3.1.3 Uso dei Branch

Al fine di agevolare il più possibile il parallelismo, evitando al contempo quanto più possibile eventuali problematiche in fase di *merge_G*, sono stati creati i seguenti branch_G:

- **master**: questo branch contiene solamente i documenti e file che si trovano in stato "Approvato", i quali formano dunque la baseline_G;
- **develop**: questo è il branch di "sviluppo". Esso contiene tutti i documenti e file 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 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;
- **feature/revisione"nomeFile"**: questi branch vengono sfruttati qualora un documento o file presente nel branch develop, in sede di verifica, dovesse necessitare di modifiche non immediate.

3.3.1.4 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 "features", e dal quale inizia il branch di "release". Ha la stessa durata di vita del 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 nel 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 termine. Il suo tempo di vita dura quanto lo sviluppo della nuova feature fintanto che non avviene il merge_G sul branch "develop";
5. **Hotfix:** branch molto simili a quello 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.

3.3.1.5 Norme delle Commit

Ogni commit_G, ovvero ciascuna modifica alle repository, deve essere caratterizzata da una descrizione sensata, eventualmente accompagnata ad un riferimento esplicito ad una issue_G aperta, al fine di agevolare una piena, e non onerosa, comprensione da parte di ogni membro del gruppo.

3.3.1.6 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 tra diversi branch:

- **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 develop-feature/revisione"nomeDocumento":** questo merge avviene qualora le modifiche necessarie al documento in questione siano risolte con successo;

- **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.

3.3.1.7 Rilascio di Versione

Il rilascio di una nuova versione del prodotto avviene nel momento in cui si raggiunge un certo numero di features implementate e testate. Dal branch "develop" si avvia un processo di verifica, che sfocia in una nuovo branch di "release", il quale porta il nome della release e che nella sua ultima fase rilascia la nuova versione sul branch "master" e applica le modifiche effettuate nel frattempo anche nel branch "develop", portando i due allo stesso livello di produzione.

3.3.2 Configurazione Versionamento

3.3.2.1 Remoto

La configurazione di *GitLab* avviene nel rispettivo portale, www.gitlab.com, dove si inseriscono le chiavi *SSH_G* 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 chiavi *SSH*, che permettono il collegamento con il server remoto dove viene gestito il repository. Una volta generate le chiavi, seguendo le varie procedure specifiche per ogni sistema operativo, vengono caricate sul portale apposito del gestore *GitHub*. L'ultima fase prevede la clonazione con uno dei seguenti metodi:

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

3.4 Verifica

3.4.1 Descrizione

In questa sezione si analizzano i metodi e gli strumenti utilizzati per verificare codice e documenti durante la loro realizzazione.

3.4.2 Analisi Statica

L'analisi statica è una tecnica che si può applicare sia alla documentazione che al codice, essa permette di verificare il lavoro svolto individuando anomalie e/o errori. Tale tipo di verifica può svolgersi in due modi:

- **Walkthrough:** tecnica applicata quando non si conoscono le tipologie di errori o problemi che si stanno cercando, è prevista quindi una lettura da cima a fondo del codice o documento per trovare anomalie di qualsiasi tipo. Viene sempre svolto per la verifica dei contenuti dei documenti;
- **Inspection:** tecnica applicata quando si è conoscenza della tipologia di errori o problemi, è prevista quindi una lettura attenta e mirata alla ricerca di tale tipologia di errori.

3.4.3 Analisi Dinamica

Il processo di analisi dinamica consiste nella realizzazione ed esecuzione di una serie di test sul codice del software prodotto. Questa tecnica non è applicabile per trovare errori nella documentazione.

3.4.3.1 Classificazione dei Test

Per garantire una corretta struttura, il gruppo ha deciso di introdurre una nomenclatura per identificare i test implementati. Si potranno quindi identificare così:

T[Tipo][Priorità]-[Codice]

Dove:

- **Tipo:** indica il tipo di test ed è identificato da una lettera a scelta tra:
 - **V:** validazione;
 - **S:** sistema;
 - **I:** integrazione;
 - **U:** unità.
- **Priorità:** indica la priorità, identificato da una lettere a scelta tra:
 - **0:** obbligatorio;
 - **1:** desiderabile;

– 2: opzionale.

- **Codice:** indica il codice del test, rispettando una struttura gerarchica.

L'esito del test, può assumere i seguenti valori:

- "N.I." (non implementato);
- "N.S." (non superato);
- "S." (superato).

3.4.3.2 Test di Unità

Il test di unità verifica che ogni minima componente software funzioni correttamente. Il test di unità verificherà il corretto funzionamento di parti di programma permettendo una migliore individuazione degli errori.

3.4.3.3 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.

3.4.3.4 Test di Sistema

Categoria di test atta a verificare il soddisfacimento di tutti i requisiti, al fine di garantire che tutte le funzionalità richieste siano presenti.

3.4.3.5 Test di Validazione

Categoria di test atta a verificare se il software sviluppato sia conforme alle attese degli stakeholder. Prevede il collaudo del prodotto in presenza del proponente e in caso di superamento di tale collaudo, consegue il rilascio ufficiale del prodotto sviluppato.

3.4.4 Verifica dei Diagrammi UML

I verificatori devono controllare che i diagrammi UML prodotti rispettino lo standard *UML 2.x* e che siano corretti semanticamente.

3.4.5 Strumenti

- **Software:** per la verifica del codice *JavaScript* è stato scelto *Jest*;
- **Documenti:** per la verifica dei documenti si utilizzano le funzionalità di *TexMaker*;
- **Gestione processi e feedback:** è stato scelto di utilizzare il sistema integrato di issues presente su *GitLab*;
- **Analisi dinamica locale:** è stato scelto *ESLint* come soluzione iniziale per risolvere tutte le necessità di analisi del codice locale;
- **Analisi dinamica remota:** è stato scelto *Codecov.io* servizio di code coverage_G basato su repository presenti in piattaforme di versionamento quali *GitHub*_G e *GitLab*. Ad ogni esecuzione in *GitLab* della pipeline di CI/CD, vengono generati da *Jest* i report di coverage da cui *Codecov.io* elabora la copertura del codice.

3.5 Validazione

3.5.1 Descrizione

La fase di validazione ha il compito di verificare che il prodotto sia conforme a quanto pianificato e sia in grado di gestire e minimizzare gli effetti dei possibili errori.

3.5.2 Procedure

3.5.2.1 Validazione Software

I passi da compiere per compiere l'attività di validazione del software sono i seguenti:

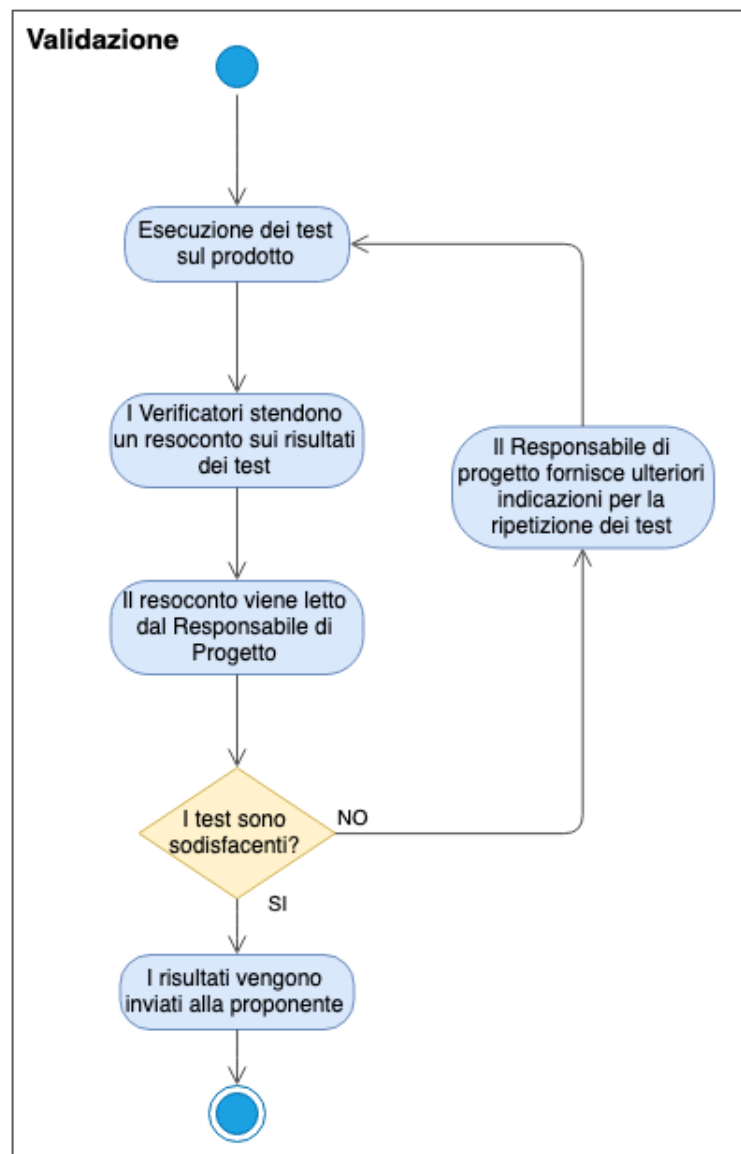


Figura 3: Validazione del Software

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_G*, presso il quale vengono discusse le tematiche riguardanti gli 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_G*. 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 v3.0.0*;
- **#pianoprogetto**: per confrontarsi riguardo il monte ore dei vari ruoli e per facilitare la stesura del documento *Piano di Progetto v3.0.0*;
- **#analisirequisiti**: per discutere gli *Use Case_G* e i requisiti necessari alla stesura dell'*Analisi dei Requisiti v3.0.0*;
- **#pianoqualifica**: per discutere strategie da attuare per garantire qualità attraverso verifica e validazione.

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_G.

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 incontri qualora sia necessario, tenendo conto anche delle preferenze di date 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 presentatagli 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;
- **Tracciamento delle Decisioni:** consiste in una tabella per il tracciamento delle decisioni effettuate durante l'incontro, in modo tale da avere un prospetto più chiaro ed immediato di ciò che è stato trattato.
Ogni decisione sarà identificata da un codice nel formato VER-DATA.X dove VER indica la parola verbale, DATA rappresenta la data del verbale nel formato formato YYYY-MM-DD e X rappresenta un numero sequenziale della decisione presa.

4.2 Processi di Pianificazione

4.2.1 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.*

4.2.1.1 *Responsabile di Progetto*

Detto anche "*Project Manager*", è il rappresentante 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 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;
- Redigere l'organigramma del gruppo e il *Piano di Progetto v3.0.0*.

4.2.1.2 *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 responsabilità specifiche figurano:

- Redigere documenti che normano l'attività lavorativa, e la loro verifica;
- Redigere le *Norme di Progetto v3.0.0*;
- 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 .

4.2.1.3 *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_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 v1.0.0* e l'*Analisi dei Requisiti v3.0.0*.

4.2.1.4 *Progettista*

Il *Progettista* è responsabile delle attività di progettazione attraverso la gestione degli aspetti tecnici del progetto.

Più nello specifico si occupa di:

- Definire l'Architettura_G del prodotto_G, applicando quanto più possibile norme di best practice_G e prestando attenzione alla manutenibilità del prodotto;
- Suddividere il problema, e di conseguenza il sistema, in parti di complessità trattabile.

4.2.1.5 *Programmatore*

Il *Programmatore* si occupa delle attività di codifica, le quali portano alla realizzazione effettiva del prodotto.

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 Progetto v3.0.0*;
- Produrre codice documentato e manutenibile;
- Realizzare le componenti necessarie per la verifica e la validazione del codice;
- Redarre il *Manuale Utente v1.0.0*.

4.2.1.6 *Verificatore*

Il *Verificatore*, figura presente per l'intera durata del progetto, è responsabile delle attività di verifica.

Nello specifico egli:

- Verifica l'applicazione ed il rispetto delle *Norme di Progetto v3.0.0*;
- Segnala al *Responsabile di Progetto* l'emergere di eventuali discordanze tra quanto presentato nel *Piano di Progetto v3.0.0* e quanto effettivamente realizzato;
- Ha il compito di redarre il *Piano di Qualifica v3.0.0*.

4.2.1.7 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 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.

4.2.2 Gestione degli Strumenti di Coordinamento

4.2.2.1 Task

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

La creazione di un task da parte del *Responsabile di Progetto*, risulterà dunque essere l'istanziamento di una issue 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 task e dei conseguenti ticket_G , ricada sul *Responsabile di progetto*, ciascun membro del gruppo ha la facoltà di creare task, a patto che tale compito veda lui come unico assegnatario. Tali task dovranno inoltre essere approvati dal *Responsabile* per essere validi.

4.2.2.2 Ticket

I tickets rappresentano l'operazione di assegnazione dei task, e quindi in questo caso delle issue, ad uno o più specifici membri del gruppo. Tale operazione è responsabilità unica del *Responsabile di Progetto* a meno che non si tratti di task (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 scadezze 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.

4.3 Gestione di Progetto

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

4.3.1 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.

4.3.1.1 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 testati ed affidabili. Tra i template forniti abbiamo:

- **Basic Kanban:** presenta le fasi di *ToDo*, *In Progress*, e *Done*;

- **Automated Kanban:** presenta trigger_G 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 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.

4.3.1.2 Aggiunta di Milestone

Le milestone_G sono gruppi di task mirate ad un obiettivo comune tra esse. Possono essere aggiunte in qualsiasi momento, sia prima che dopo la creazione di una task.

4.3.2 Ciclo di Vita delle Task

4.3.2.1 Apertura

Da una specifica project board si possono creare le task o le issue_G, le quali possono essere assegnate ad uno o più individui che collaborano al repository, inoltre ogni task può far parte di una milestone, che raggruppa un insieme di task o issue per il raggiungimento di un obiettivo comune.

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 ad una project board, mantenendo comunque tutte le funzionalità descritte precedentemente.

4.3.2.2 Completamento

Il completamento di una task avviene in diversi modi, a seconda delle impostazioni della 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".

4.3.2.3 Richiesta di Revisione

Accumulate un certo numero di task o di milestone, 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*.

4.3.2.4 Chiusura

Una volta che le task o le milestone 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.

4.3.2.5 Riapertura

Le task in stato di "Chiusura" possono essere riaperte e spostate nello stato di "Apertura" se esse non soddisfanno tutti i parametri di qualità richiesti.

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.x;
- Manjaro Linux - 4.14.85-1 LTS.

4.4.2 Ambiente di Test del Prodotto

Il team **Agents Of S.W.E.** ha deciso, per avere un ambiente comune su cui testare il prodotto in via di sviluppo, di dotarsi di un server noleggiato online attraverso la nota piattaforma di hosting DigitalOcean.

Il server in oggetto è raggiungibile all'indirizzo IP 142.93.102.115 ha le seguenti caratteristiche:

- **SO:** Ubuntu 18.04 LTS;
- **CPU:** 1x Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz;
- **RAM:** 2GiB ECC DIMM;
- **SSD:** 24GiB.

Il team ha provveduto ad installare all'interno del server i seguenti pacchetti:

- *Grafana*;

- *InfluxDB*;
- *Telegraf*;
- *Apache_G* .

4.4.3 Versionamento e Issue Tracking

4.4.3.1 *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.

4.4.3.2 *GitHub*

Github è un Respository Manager_G che usa *Git* come sistema di versioning, offre inoltre servizi di issue tracking_G . Il seguente sistema è stato rimosso dopo la fase di RP.

4.4.3.3 *GitLab*

Gitlab è un Respository Manager che usa *Git* come sistema di versioning, a differenza di *GitHub* offre servizi di integrazione test e pipeline di sviluppo.

4.4.4 Comunicazione

4.4.4.1 *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.

4.4.4.2 *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 .

4.4.5 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.

4.5 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.

A PDCA

Il PDCA è un modello studiato per il miglioramento continuo della qualità di un prodotto a lungo raggio. Fondamento di questo metodo è promuovere un tipo di cultura orientata al miglioramento continuo dei processi e all'utilizzo ottimale delle risorse. Il nome di fatto è un acronimo qui scomposto:

Plan-Do-Check-Act

Difatti questo metodo è scomposto in quattro fasi qui descritte:

- **Plan:** fase di pianificazione, nella quale vengono stabiliti obiettivi metrici da raggiungere e processi da utilizzare, in base ai risultati attesi;
- **Do:** in questa fase vengono attuate le soluzioni ed i piani precedentemente definiti, eseguendo il programma. Inoltre, vengono raccolti i dati per la fase successiva;
- **Check:** test e controllo, si studiano i risultati ottenuti dall'esecuzione del programma (rilevati nel "DO") confrontandoli con i risultati attesi (trovati nel "PLAN");
- **Act:** si agisce sulla base dei risultati ottenuti e dei risultati attesi, al fine di migliorare i processi o il prodotto, non conformi alle attese. Finita questa fase, si procede con una nuova iterazione ripartendo dalla fase di Plan, potendo definire obiettivi più stringenti laddove rispettati nell'iterazione appena terminata.

In sostanza il PDCA è un modello che fissati gli obiettivi iniziali e soddisfatti, si passa a fissare nuovi obiettivi da raggiungere per alzare il livello della qualità.

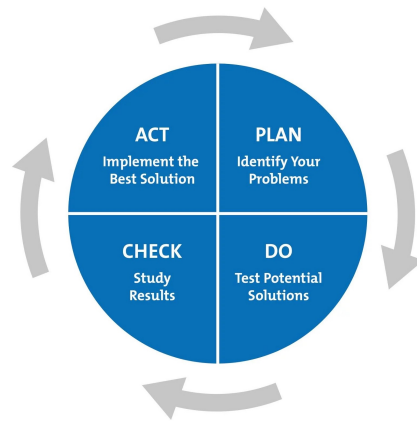


Figura 4: Modello PDCA. Immagine dal sito web https://www.mindtools.com/pages/article/newPPM_89.htm

B ISO/IEC 9126:2001

L'ISO/IEC 9126:2001 individua un'insieme di normative e linee guida al fine di descrivere un modello di qualità del software. È scomposto in quattro parti, ciascuna delle quali si occupa di un diverso ambito.

- **Modello di Qualità:** insieme di caratteristiche di qualità che sono in grado di descrivere i principali fattori di qualità di un prodotto software, si possono classificare in: funzionalità, affidabilità, efficienza, usabilità, manutenibilità, portabilità;
- **Qualità Esterne:** insieme di metriche che misurano i comportamenti del software sulla base di test, dell'operatività e dal monitoraggio durante la sua esecuzione;
- **Qualità Interne:** insieme di metriche che si applicano alla qualità del software "non eseguibile", in sostanza al codice sorgente, durante la fase di sviluppo e pianificazione; l'obiettivo di queste metriche è scovare eventuali difetti, e prevedere la qualità finale attraverso misurazioni e strumenti che simulano il comportamento finale del prodotto;
- **Qualità in Uso:** rappresenta il punto di vista dell'utente; il livello corrente si raggiunge nel momento in cui tutte le qualità precedenti sono soddisfatte e quando l'utente è abilitato ad ottenere specificati obiettivi con:
 - **Efficacia:** capacità del software di raggiungere obiettivi specificati con accuratezza e completezza;
 - **Produttività:** capacità di mettere in grado l'utente di spendere una quantità di risorse appropriate per compiere una determinata azione in uno specificato ambito d'uso;
 - **Sicurezza:** capacità del prodotto di raggiungere accettabili livelli di rischio per il software, apparecchiatura, persone;
 - **Soddisfazione:** capacità del prodotto di soddisfare le attese degli utenti.

Le quattro parti sono legate strettamente da dipendenze importanti, il soddisfacimento di ognuna dipende dalla precedente. La seguente figura spiega al meglio il legame.

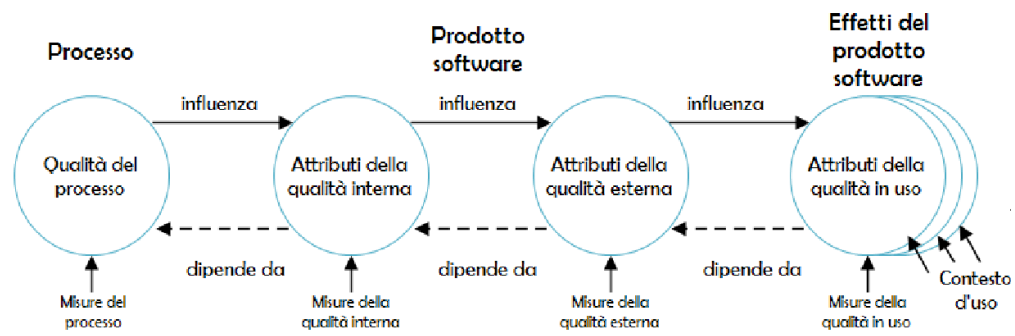


Figura 5: ISO/IEC9126:2001. Immagine dal sito web https://it.wikipedia.org/wiki/ISO/IEC_9126

C CMMI

Il Capability Maturity Model Integration è un approccio al miglioramento dei processi organizzativi al fine di migliorare le prestazioni. Fondamento del CMMI è la valutazione dei processi e la classificazione in livelli di maturità degli stessi al fine di giudicare un sistema, o un'intera azienda. Possiamo definire ogni singolo elemento del CMMI:

- **Capability:** misura la capacità del sistema di raggiungere gli scopi a esso assegnato, può essere in quattro livelli:
 - **Incompleto:** approccio incompleto per raggiungere l'obiettivo;
 - **Initial:** approccio iniziale per soddisfare gli obiettivi, non è una serie soddisfacente di pratiche necessarie;
 - **Managed:** completa e semplice serie di pratiche che soddisfano l'intento generale;
 - **Definita:** si basa su pratiche standardizzate che organizzano e personalizzano il sistema di lavoro e di gestione.
- **Maturity:** misura quanto l'azienda è governata dal suo sistema di processi; la maturità può essere classificata in cinque livelli:
 - **Initial:** il lavoro non è organizzato ed è imprevedibile e reattivo, porta spesso a ritardi e sforamento del budget;
 - **Managed:** il lavoro è gestito, controllato e misurato a livello di progetto, si può considerare ancora un livello basato sulla ricerca reattiva di una eventuale soluzione;
 - **Defined:** il lavoro è proattivo piuttosto che reattivo, gli standard a livello organizzativo gestiscono programmi e portfogli;
 - **Quantitatively Managed:** il lavoro è misurato e controllato, l'organizzazione è guidata con obiettivi di miglioramento delle prestazioni prevedibili e allineati per soddisfare gli stakeholder;
 - **Optimizing:** il lavoro è stabile e flessibile, l'organizzazione è focalizzata al miglioramento continuo ed è studiare per ruotare e cambiare a seconda del contesto; la stabilità offre una piattaforma che permette innovazione e agilità.
- **Model:** criteri per la valutazione del grado di qualità dei processi di validazione;

- **Integration:** architettura di integrazione delle diverse discipline e tipologie di attività delle aziende:
 - **CMMI-DEV:** sviluppo prodotti;
 - **CMMI-SVC:** gestione dei servizi;
 - **CMMI-ACQ:** acquisto prodotti e servizi esterni.

Il CMMI si può definire un framework che ottiene benefici maggiori in aziende di medie-grandi dimensione, circa il 50% delle aziende che hanno tra i 1000 e i 2000 dipendenti si collocano al livello massimo di maturità, mentre il 70% delle aziende con in media 25 dipendenti si collocano al livello 2.

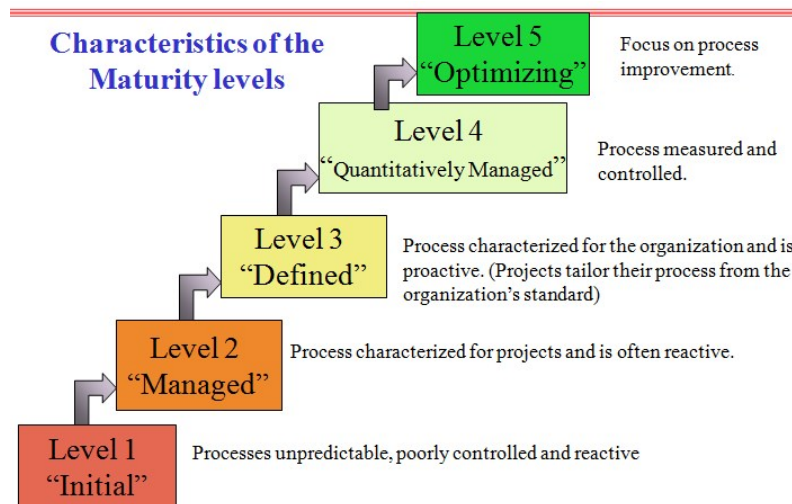


Figura 6: Capability Maturity Model Integration - CMMI. Immagine dal sito https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

D Modello a V

Il Modello a V (o V-Model) è un modello di sviluppo del software. Per la precisione si tratta di un'estensione del modello a cascata.

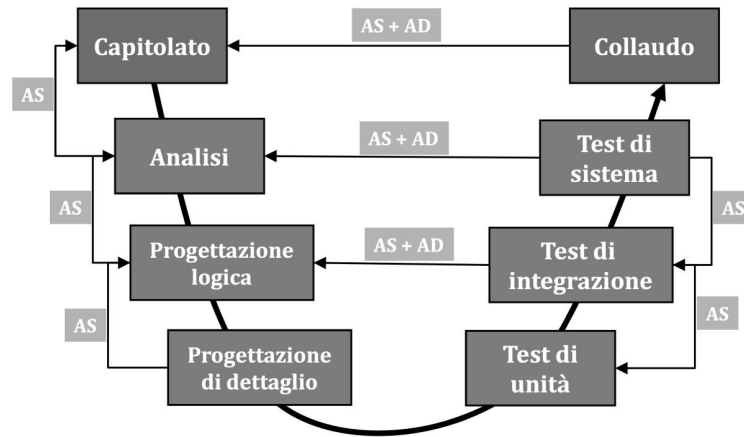


Figura 7: Modello a V. Immagine dal sito web <https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L16.pdf>

Il modello a V illustra le relazioni tra ogni fase del ciclo di vita di sviluppo e la fase di testing ad essa associata. L'asse orizzontale rappresenta la completezza del progetto (da sinistra a destra); l'asse verticale il livello di astrazione (livello di astrazione minore in basso, livello di astrazione maggiore in alto).

E CI/CD

CI e CD sono due acronimi particolarmente usati nelle pratiche di sviluppo software moderno. Esse collaborano tra di loro per il raggiungimento di uno sviluppo software rapido ed efficace.

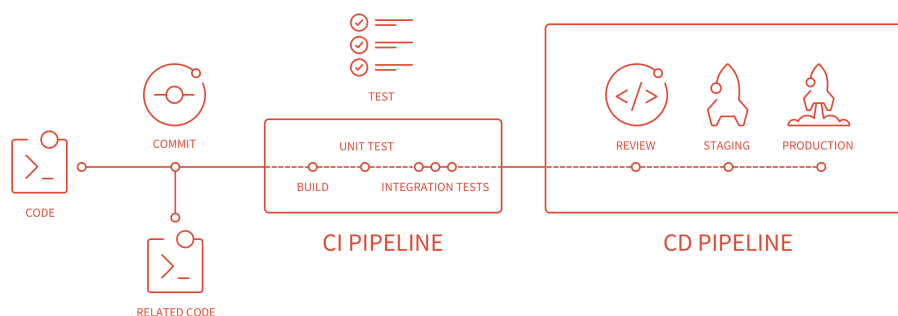


Figura 8: Rappresentazione del processo di CI/CD <https://docs.gitlab.com/ee/ci/>

E.1 CI

La CI (Continuous integration) è una filosofia di sviluppo e set di pratiche con l'obiettivo di guidare un team di sviluppo a implementare cambiamenti e testing frequenti sul repository del codice. L'obiettivo della CI è di stabilire un modo coerente e automatico per costruire e testare applicazioni. Con la coerenza del processo di integrazione, i team hanno maggiori probabilità di modificare il codice più frequentemente, il che porta a una migliore collaborazione e qualità del software, riducendone i tempi.

Per implementare questa filosofia ci sono dei costi:

- Il team di sviluppo dovrà scrivere test automatici per ogni nuova feature, miglioramento o bug fix;
- Viene impostato un server dedicato, il quale monitora il repository principale e testa il codice con i test precedentemente scritti;
- Gli sviluppatori devono mergiare le modifiche effettuate con quelle degli altri sviluppatori il più frequentemente possibile, almeno una volta al giorno.

Dai precedenti costi si ottiene:

- Vengono spediti meno bug in produzione poiché le regressioni vengono acquisite in anticipo dai test automatici;

- Costruire la versione è facile poiché tutti i problemi di integrazione sono stati risolti in anticipo;
- Meno cambi di contesto, poiché gli sviluppatori sono allertati non appena la build si interrompe, dando la possibilità di sistemare prima di procedere con la successiva task;
- I costi dei test sono ridotti drasticamente. Il server dedicato può eseguire centinaia di test in pochi secondi;
- Il team spende meno tempo a testare il prodotto e si concentra di più a migliorare la qualità.

E.2 CD

La CD (Continuous delivery) è un'estensione della continuous integration per poter assicurare di poter rilasciare rapidamente nuove modifiche ai clienti in modo sostenibile. Ciò significa che oltre ad aver automatizzato i test, si ha anche automatizzato il processo di rilascio e si può distribuire l'applicazione in qualsiasi momento. Affinché sia veramente efficace, è necessario distribuire la produzione il prima possibile per assicurarsi di rilasciare piccolo batch che siano facili da risolvere in caso di problemi. Per implementare efficacemente la CD bisogna seguire i seguenti punti:

- Serve una solida base nella continuous integration e la suite di test deve coprire una certa percentuale della codebase;
- Il rilascio deve essere automatizzato. Il trigger può essere manuale, ma una volta avviata la distribuzione non dovrebbe essere necessario l'intervento umano;
- Il team dovrà adottare le flag di funzionalità in modo che le funzionalità incomplete non influiscano sui clienti durante la produzione.

Dalle precedenti si ottiene:

- La complessità della distribuzione del software è portata via. Il team non deve più prepararsi per il rilascio;
- Si può rilasciare molto più frequentemente, accelerando così il ciclo di feedback con i clienti;
- C'è meno pressione sulle decisioni per piccoli cambiamenti, incoraggiando dunque la iterazione più velocemente.

F Metriche

F.1 Metriche per la Qualità di Processo

F.1.1 Gestione dei Task

È rilevante l'adempimento dei task assegnati entro i tempi prestabiliti. Per far ciò, nella fase di pianificazione, si sono scelte delle date entro le quali è preferibile e consigliabile completare i task assegnati. Tuttavia è comunque accettabile l'adempimento di un task a priorità minore oltre i limiti prefissati, se e solo se il ritardo è dovuto al completamento di un task a priorità maggiore.

- **MTPC01 Schedule Variance (SV)**

È un indice che consente di rilevare se l'andamento del progetto è in linea con quanto pianificato nella baseline_G. Può essere utile al cliente per verificare quantitativamente se il team di sviluppo del progetto è in linea con i tempi.

Calcolo:

$$SV = BCWP - BCWS$$

dove:

- **BCWP**: valore delle attività realizzate alla data odierna (in giorni);
- **BCWS**: costo pianificato per realizzare le attività alla data odierna (in giorni).

Se $SV > 0$ significa che il progetto sta producendo con maggiore velocità prevista.

F.1.2 PR02: Gestione dei Costi

Per la gestione dei costi del progetto il gruppo ha deciso di utilizzare l'indice Budget Variance (BV),

- **MTPC02 Budget Variance (BV)**

È un indice calcolato che permette di rilevare la differenza tra i costi previsti e quelli reali alla data odierna.

Calcolo:

$$BV = BCWS - ACWP$$

dove:

- **BCWS**: costo pianificato per realizzare le attività alla data odierna (in euro);
- **ACWP**: costo effettivo sostenuto per completare le attività alla data odierna (in euro).

Se $BV > 0$ significa che il progetto sta risparmiando sui costi prestabiliti, se $BV = 0$ significa che il progetto sta mantenendo i costi prefissati, se $BV < 0$ significa che il progetto sta superando il budget imposto.

- **MTPC03 Estimated At Completion (EAC)**

Indice che rappresenta la stima dei costi mancanti. Viene calcolato man mano che il progetto procede ed è fondamentale per attività di pianificazione.

Calcolo:

$$EAC = ACWP + ETC$$

dove:

- **ACWP**: costo effettivo sostenuto per completare le attività alla data odierna (in euro);
- **ETC**: valore stimato per la realizzazione delle attività mancanti (in euro).

F.1.3 PR03: Verifica del Software

Questo processo ha lo scopo di verificare che i requisiti software precedentemente stabiliti vengano rispettati. È inoltre suo compito verificare che vengano soddisfatti tutti i requisiti precedentemente fissati nel documento Analisi dei Requisiti v3.0.0. Verranno utilizzati i seguenti indici:

- **MTPC04 Function Coverage**: verificare che una funzione sia chiamata;
- **MTPC05 Statement Coverage**: verificare che ogni statement del codice sia eseguito, e non ci sia quindi codice che non verrà mai preso in considerazione;
- **MTPC06 Branch Coverage**: verificare che tutti i possibili percorsi delle strutture di controllo (del tipo `if`, `case`, ecc.) siano stati eseguiti.
- **MTPC07 Condition Coverage**: verificare che ogni condizione booleana sia considerata sia vera che falsa;
- **MTPC08 Code Coverage**: verifica la percentuale di codice coperto da test rispetto al totale delle righe di codice.

F.1.4 PR04: Gestione dei Rischi

Il suo scopo è quello del continuo monitoraggio, della continua identificazione, scoperta dei rischi che incorrono o che posso incorrere durante lo svolgimento del progetto e la loro risoluzione qualora si verifichino.

- **MTPC09 Rischi non Preventivati**

Indice numerico incrementale a partire da 0. Indica il numero di rischi non preventivati che si verificano e vengono considerati durante la corrente fase del progetto. La misurazione avviene incrementando il valore per ogni rischio (non individuato precedentemente) che viene rilevato, il valore viene azzerato per ogni fase del progetto.

F.1.5 PR05: Gestione dei Test

- **MTTS10 Percentuale di Test Passati**

Indica la percentuale di test passati, utile per misurare l'avanzamento qualitativo. La misurazione avviene:

$$PTP = \frac{TP}{TT} * 100$$

dove PTP è la percentuale finale di test passati, TP sono i test passati e TT i test totali eseguiti.

- **MTTS11 Percentuale di Test Falliti**

Indica la percentuale di test falliti. La misurazione avviene:

$$PTF = \frac{TF}{TT} * 100$$

dove PTF è la percentuale finale di test falliti, TF sono i test passati e TT i test totali eseguiti.

- **MTTS12 Percentuale di Difetti Sistemati**

Indica la percentuale di bug/difetti sistemati, indice utile per misurare l'avanzamento. La misurazione avviene:

$$PDS = \frac{DS}{DT} * 100$$

dove PDS è la percentuale calcolata, DS i difetti sistemati e DT i difetti totali.

- **MTTS13 Tempo Medio di Risoluzione degli Errori**

Indice calcolato sul tempo medio della risoluzione di bug creati dal team durante lo sviluppo. Utile per considerare abilità e tempo di assorbimento di un bug nel sistema. La misurazione avviene:

$$TMRE = \frac{TRE}{NE}$$

dove TMRE è il tempo medio calcolato, TRE il tempo totale per la risoluzione degli errori e NE il numero degli errori.

- **MTTS14 Numero Medio di Bug Trovati per Test**

Indica il numero medio di bug trovati eseguendo i test. È un indice utile per verificare la qualità dei test e del sistema in generale. La misurazione avviene:

$$MBT = \frac{NB}{NT}$$

dove MBT è la media calcolata, NB il numero di bug rilevati e NT il numero dei test.

- **MTTS15 Copertura dei Test Eseguiti**

Valore che rappresenta la percentuale di test che sono stati eseguiti rapportato al numero di test da eseguire. La misurazione avviene:

$$PTE = \frac{TE}{TT} * 100$$

Dove TE indica i test eseguiti i test eseguiti, mentre TT i test totali.

- **MTTS16 Copertura dei Requisiti**

Valore che rappresenta la percentuale dei requisiti coperti rispetto al totale. La misurazione avviene:

$$CR = \frac{RC}{RT} * 100$$

Dove RC indica i requisiti coperti, mentre RT i requisiti totali.

F.1.6 PR06: Versionamento e Build

- **MTPC17 Media Commit per Settimana**

Calcolo della media dei commit effettuati settimanalmente. Essendo le repository presenti su due sistemi di versionamento differenti, verrà calcolata la media per ognuna di essi. Per il calcolo della media verrà utilizzato un bot.

- **MTPC18 Percentuali Build Superate**

Calcolo della media delle build effettuate sulla repository contenente il codice per lo sviluppo del prodotto. Il calcolo avviene tramite un bot connesso al sistema di versionamento *GitLab*.

F.2 Metriche per la Qualità di Prodotto

Le seguenti metriche sono utilizzate per misurare qualitativamente il prodotto.

F.2.1 Leggibilità

- **MTPDD19 Indice di Gulpease**

Il gruppo ha deciso di utilizzare l'*indice di Gulpease_G* per misurare la leggibilità di un testo. È stato sviluppato appositamente uno script in *Python* per automatizzare la procedura e allo stesso modo velocizzarla. La procedura verrà utilizzata a documento terminato e completo così da valutare il lavoro svolto dai redattori.

- **MTPDD20 Correttezza Ortografica**

La correttezza ortografica è un aspetto importante che non accetta errori, i documenti al momento della pubblicazione sono corretti. Verranno utilizzati appositi strumenti che supporteranno la correzione, ovvero il software *TexMaker* il quale integra un segnalatore automatico degli errori grammaticali.

- **Correttezza Logica e Semantica**

Non essendo disponibili sistemi automatici al fine di controllare la correttezza logica e semantica, la comprensione totale del prodotto letto identificherà anche tale correttezza, in quanto un documento viene considerato leggibile solamente se è corretto.

F.2.2 Funzionalità

Con *Funzionalità* si intendono le qualità riguardanti le funzioni offerte dal software.

- **MTPDS21 Soddisfacimento Requisiti Obbligatori**

Indicatore percentuale che verifica che tutti i requisiti obbligatori siano soddisfatti. Condizione necessaria al fine di rispettare il contratto. La misurazione avviene:

$$PRO = \frac{ROS}{ROT} * 100$$

dove PRO è la percentuale di requisiti obbligatori soddisfatti, ROS i requisiti obbligatori soddisfatti e ROT i requisiti obbligatori totali.

- **MTPDS22 Soddisfacimento Requisiti Opzionali Accettati**

Indicatore percentuale che verifica se tutti i requisiti opzionali scelti siano soddisfatti. Condizione necessaria al fine di rispettare il contratto. La misurazione avviene:

$$PRP = \frac{RPS}{RPT} * 100$$

dove PRP è la percentuale di requisiti opzionali accettati, RPS i requisiti opzionali scelti soddisfatti e RPT il numero di requisiti accettati.

F.2.3 Affidabilità

Con *Affidabilità* si intende la garanzia di funzionamento del software sotto determinate condizioni d'uso.

- **MTPDS23 Densità di Failure**

Indicatore percentuale utilizzato per calcolare la percentuale di testing che si è conclusa in failure. La misurazione avviene:

$$DF = \frac{FR}{TE} * 100$$

in cui DF è la percentuale della densità di failure, FR sono il numero di test falliti durante il testing e TE il numero di test totali.

- **MTPDS24 Tolleranza agli Errori**

Indicatore percentuale delle funzionalità che sono in grado di gestire correttamente ed efficientemente gli errori. La misurazione avviene:

$$TE = \frac{FE}{ON} * 100$$

in cui TE è la percentuale di tolleranza agli errori, FE sono i test falliti durante il testing e ON sono i test eseguiti che eseguono operazioni non corrette che possono causare failure.

F.2.4 Efficienza

Con *Efficienza* si intendono le prestazioni raggiungibili sotto specifiche condizioni di utilizzo.

- **MTDS25 Tempo di Risposta Medio**

Il tempo di risposta medio è dovuto a varie componenti:

- Carico medio del server in un range di tempo;
- Complessità rete bayesiana presa in carico;
- Complessità media della libreria JSBayes;

- **MTDS26 Tempo di Risposta di Picco**

Il tempo di risposta di Picco massimo, è inteso il tempo massimo di risposta in situazioni limite.

F.2.5 Usabilità

Con *Usabilità* si intende il livello di comprensione del prodotto da parte dell'utilizzatore.

- **MTPDS27 Tempo Medio di Comprensione**

Indica il tempo medio che l'utente impiega per comprendere cosa può svolgere il sistema. È misurato in minuti ed è rilevato attraverso test a persone esterne al team di sviluppo.

- **MTPDS28 Tempo Medio di Apprendimento**

Indica il tempo medio che l'utente impiega per riuscire a utilizzare a pieno il software e tutte le sue funzionalità. È misurato in minuti ed è rilevato tramite test a persone esterne al team di sviluppo.

F.2.6 Manutenibilità

Con *Manutenibilità* si intende il livello di semplicità richiesto al fine di eseguire interventi di modifica, correzione o adattamento.

- **MTPDS29 Percentuale Commenti/Codice**

Indica le righe di commenti presente rispetto al codice. È calcolato per ogni procedura e non per l'intera codebase. La misurazione avviene:

$$PC = \frac{RC}{RT} * 100$$

dove PC è la percentuale calcolata, RC il numero di righe di commento e RT il numero di righe totale.

F.2.7 Portabilità

Con *Portabilità* si intende la capacità del software di funzionare in diversi sistemi, che siano essi software o hardware.