



# Norme di Progetto

*Gruppo DigitalCookies — Progetto SWEDesigner*

[digitalcookies.group@gmail.com](mailto:digitalcookies.group@gmail.com)

## Informazioni sul documento

<b>Versione</b>	3.0.0
<b>Redazione</b>	Carlo Sindico, Christian Cabrera, Alberto Giudice, Alessia Bragagnolo, Alberto Rossetti
<b>Verifica</b>	Alessia Bragagnolo
<b>Approvazione</b>	Saverio Follador
<b>Uso</b>	Interno
<b>Distribuzione</b>	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo DigitalCookies

## Descrizione

Questo documento descrive le regole, gli strumenti e le convenzioni adottate dal gruppo DigitalCookies durante la realizzazione del progetto SWEDesigner.

## Registro delle modifiche

Versione	Data	Collaboratori	Ruolo	Descrizione
3.0.0	19-05-2017	Saverio Follador	Responsabile	Approvazione del documento
2.1.0	19-05-2017	Alessia Bragagnolo	Verificatore	Verifica del documento
2.0.9	18-05-2017	Christian Cabrera	Verificatore	Suddivisione sezioni test nel processo di Verifica e Validazione
2.0.8	18-05-2017	Alberto Rossetti	Amministratore	Aggiunta sezione 2.1.4.4 Collaudo e Consegna del prodotto
2.0.7	18-05-2017	Davide Albertini	Amministratore	Modifica sezione 2.2.4.3.6 e aggiunta sezione 2.2.4.3.4
2.0.6	17-05-2017	Carlo Sindico	Progettista	Aggiunte notazioni metodi per la Definizione di Prodotto in sezione 3.1.4.3.1
2.0.5	17-05-2017	Christian Cabrera	Verificatore	Aggiunte metriche per i processi in sezione 2
2.0.4	16-05-2017	Davide Albertini	Amministratore	Aggiunte sezioni 2.2.4.1.4, 2.2.4.1.5, 2.2.4.2.5
2.0.3	16-05-2017	Alberto Rossetti	Amministratore	Spostamento sezione Definizione di Prodotto in sezione 3.1.4.3

2.0.2	16-05-2017	Alberto Rossetti	Amministratore	Spostamento sezione Specifica Tecnica in sezione 3.1.4.2
2.0.1	16-05-2017	Alberto Rossetti	Amministratore	Spostamento semantica dei requisiti in sezione 3.1.4.1
2.0.0	01-05-2017	Alessia Bragagnolo	Responsabile	Approvazione del documento
1.2.0	01-05-2017	Davide Albertini	Verificatore	Verifica del documento
1.1.7	28-04-2017	Christian Cabrera	Amministratore	Aggiornamento degli strumenti per la validazione nella sezione 3.4
1.1.6	28-04-2017	Alberto Rossetti	Amministratore	Ampliamento sezione 3.2.4.1 con nuove metriche
1.1.5	27-04-2017	Alberto Rossetti	Amministratore	Ampliamento sezione 3.2.5.5 relativa ai test, con descrizione semantica dei codici identificativi dei test
1.1.4	26-04-2017	Alberto Rossetti	Amministratore	Stesura sezione processo di validazione
1.1.3	25-04-2017	Christian Cabrera	Amministratore	Aggiunte sezioni manuale utente e manuale sviluppatore nella sezione 2.2.4.3

1.1.2	25-04-2017	Alberto Rossetti	Amministratore	Ampliamento contenuti sullo stile di codifica nella sezione 2.2.4.3.5
1.1.1	24-04-2017	Christian Cabrera	Amministratore	Aggiunta descrizione semantica dei package e delle classi nella sezione 2.2.4.2
1.1.0	21-04-2017	Carlo Sindico	Verificatore	Verifica del documento
1.0.3	20-04-2017	Christian Cabrera	Amministratore	Stilate liste di controllo per gli errori frequenti nella sezione 3.2.4.2.1
1.0.2	19-04-2017	Alberto Giudice	Amministratore	Aggiunta descrizione semantica dei rischi nella sezione 4.1.5.5
1.0.1	18-04-2017	Christian Cabrera	Amministratore	Creazione paragrafo dedicato alle metriche nella sezione 3.2.4.1
1.0.0	02-03-2017	Davide Albertini	Responsabile	Approvazione
0.2.0	02-03-2017	Christian Cabrera	Verificatore	Verifica sezione processi di supporto e organizzativi
0.1.0	01-03-2017	Alberto Giudice	Verificatore	Verifica sezione introduzione e sezione processi primari

---

0.0.6	28-02-2017	Alessia Bragagnolo	Amministratore	Fine sezione processi organizzativi
0.0.5	28-02-2017	Alberto Giudice	Amministratore	Inizio stesura sezione processi organizzativi
0.0.4	27-02-2017	Alberto Giudice	Amministratore	Stesura sezione processi di supporto
0.0.3	27-02-2017	Carlo Sindico	Amministratore	Stesura sezione processi primari
0.0.2	24-02-2017	Alessia Bragagnolo	Amministratore	Stesura sezione introduzione
0.0.1	24-02-2017	Carlo Sindico	Amministratore	Creazione del template

---

## Indice

<b>1</b>	<b>Introduzione</b>	<b>10</b>
1.1	Scopo del documento . . . . .	10
1.2	Scopo del prodotto . . . . .	10
1.3	Ambiguità . . . . .	10
1.4	Riferimenti . . . . .	11
1.4.1	Normativi . . . . .	11
1.4.2	Informativi . . . . .	11
<b>2</b>	<b>Processi primari</b>	<b>12</b>
2.1	Fornitura . . . . .	12
2.1.1	Scopo . . . . .	12
2.1.2	Aspettative . . . . .	12
2.1.3	Descrizione . . . . .	12
2.1.4	Attività . . . . .	12
2.1.4.1	Studio di fattibilità . . . . .	12
2.1.4.2	Piano di Progetto . . . . .	13
2.1.4.3	Piano di Qualifica . . . . .	13
2.1.4.4	Collaudo e Consegna del Prodotto . . . . .	14
2.2	Sviluppo . . . . .	14
2.2.1	Scopo . . . . .	14
2.2.2	Aspettative . . . . .	14
2.2.3	Descrizione . . . . .	14
2.2.4	Attività . . . . .	15
2.2.4.1	Analisi dei requisiti . . . . .	15
2.2.4.1.1	Scopo . . . . .	15
2.2.4.1.2	Aspettative . . . . .	15
2.2.4.1.3	Descrizione . . . . .	15
2.2.4.1.4	Diagrammi . . . . .	15
2.2.4.1.5	Qualità dei requisiti . . . . .	16
2.2.4.2	Progettazione . . . . .	16
2.2.4.2.1	Scopo . . . . .	16
2.2.4.2.2	Aspettative . . . . .	16
2.2.4.2.3	Descrizione . . . . .	16
2.2.4.2.4	Diagrammi . . . . .	17
2.2.4.2.5	Qualità dell'architettura . . . . .	17
2.2.4.3	Codifica . . . . .	18
2.2.4.3.1	Scopo . . . . .	18
2.2.4.3.2	Aspettative . . . . .	18
2.2.4.3.3	Descrizione . . . . .	18
2.2.4.3.4	Nomenclatura dei file . . . . .	18
2.2.4.3.5	Stile di codifica . . . . .	19

---

2.2.4.3.6	Intestazione e commenti . . . . .	19
2.2.4.3.7	Versionamento . . . . .	20
2.2.4.3.8	Ricorsione . . . . .	21
2.2.5	Strumenti . . . . .	21
2.2.5.1	Trender . . . . .	21
2.2.5.2	Astah . . . . .	22
2.2.5.3	IntelliJ IDEA . . . . .	23
<b>3</b>	<b>Processi di supporto</b>	<b>25</b>
3.1	Documentazione . . . . .	25
3.1.1	Scopo . . . . .	25
3.1.2	Aspettative . . . . .	25
3.1.3	Descrizione . . . . .	25
3.1.4	Attività . . . . .	25
3.1.4.1	Analisi dei Requisiti . . . . .	25
3.1.4.1.1	Casi d'uso . . . . .	26
3.1.4.1.2	Requisiti . . . . .	26
3.1.4.2	Specifica Tecnica . . . . .	27
3.1.4.2.1	Definizione dei Package . . . . .	27
3.1.4.2.2	Definizione delle Classi . . . . .	28
3.1.4.3	Definizione di Prodotto . . . . .	29
3.1.4.3.1	Definizione dei metodi . . . . .	29
3.1.4.4	Manuale Utente . . . . .	30
3.1.4.5	Manuale Manutentore . . . . .	30
3.1.5	Procedure . . . . .	31
3.1.5.1	Approvazione dei documenti . . . . .	31
3.1.6	Template . . . . .	31
3.1.7	Struttura dei documenti . . . . .	31
3.1.7.1	Prima pagina . . . . .	31
3.1.7.2	Registro delle modifiche . . . . .	32
3.1.7.3	Indice . . . . .	32
3.1.7.4	Contenuto principale . . . . .	32
3.1.7.5	Note a piè di pagina . . . . .	33
3.1.8	Versionamento . . . . .	33
3.1.9	Norme tipografiche . . . . .	34
3.1.9.1	Stile del testo . . . . .	34
3.1.9.2	Elenchi puntati . . . . .	34
3.1.9.3	Formati comuni . . . . .	34
3.1.9.4	Sigle . . . . .	35
3.1.10	Elementi grafici . . . . .	36
3.1.10.1	Tabelle . . . . .	36
3.1.10.2	Immagini . . . . .	36
3.1.11	Classificazione dei documenti . . . . .	36

---

3.1.11.1	Documenti informali . . . . .	36
3.1.11.2	Documenti formali . . . . .	37
3.1.11.3	Verbali . . . . .	37
3.1.12	Strumenti . . . . .	38
3.1.12.1	L <sup>A</sup> T <sub>E</sub> X . . . . .	38
3.1.12.2	TexStudio . . . . .	38
3.1.12.3	Lucidchart . . . . .	39
3.2	Verifica . . . . .	40
3.2.1	Scopo . . . . .	40
3.2.2	Aspettative . . . . .	40
3.2.3	Descrizione . . . . .	40
3.2.4	Attività . . . . .	40
3.2.4.1	Metriche . . . . .	40
3.2.4.1.1	Metriche per i processi . . . . .	41
3.2.4.1.2	Metriche per i prodotti . . . . .	43
3.2.4.1.3	Strumenti . . . . .	46
3.2.4.2	Analisi . . . . .	46
3.2.4.2.1	Analisi statica . . . . .	46
3.2.4.2.2	Analisi dinamica . . . . .	48
3.2.5	Procedure . . . . .	49
3.2.5.1	Controllo qualità di prodotto . . . . .	49
3.2.5.2	Controllo qualità di processo . . . . .	49
3.2.5.3	Gestione anomalie . . . . .	50
3.2.5.4	Gestione modifiche . . . . .	50
3.2.5.5	Test . . . . .	50
3.2.5.5.1	Test di unità . . . . .	50
3.2.5.5.2	Test di integrazione . . . . .	50
3.2.5.5.3	Test di sistema . . . . .	51
3.2.5.5.4	Test di regressione . . . . .	51
3.2.5.6	Codice identificativo . . . . .	51
3.3	Validazione . . . . .	52
3.3.1	Scopo . . . . .	52
3.3.2	Aspettative . . . . .	52
3.3.3	Descrizione . . . . .	52
3.3.4	Attività . . . . .	53
3.3.4.1	Analisi dinamica . . . . .	53
3.3.5	Procedure . . . . .	53
3.3.5.1	Test . . . . .	53
3.3.5.1.1	Test di validazione . . . . .	53
3.3.5.2	Codice identificativo . . . . .	53
3.4	Strumenti . . . . .	54
3.4.1	Verifica ortografica . . . . .	54
3.4.2	Validazione W3C . . . . .	54



---

3.4.3	Analisi statica . . . . .	54
3.4.4	Analisi dinamica . . . . .	54
<b>4</b>	<b>Processi organizzativi</b>	<b>56</b>
4.1	Gestione . . . . .	56
4.1.1	Scopo . . . . .	56
4.1.2	Aspettative . . . . .	56
4.1.3	Descrizione . . . . .	56
4.1.4	Ruoli di progetto . . . . .	56
4.1.4.1	Amministratore di Progetto . . . . .	57
4.1.4.2	Responsabile di Progetto . . . . .	57
4.1.4.3	Analista . . . . .	57
4.1.4.4	Progettista . . . . .	57
4.1.4.5	Verificatore . . . . .	58
4.1.4.6	Programmatore . . . . .	58
4.1.5	Procedure . . . . .	58
4.1.5.1	Gestione delle comunicazioni . . . . .	58
4.1.5.1.1	Comunicazioni interne . . . . .	58
4.1.5.1.2	Comunicazioni esterne . . . . .	58
4.1.5.2	Gestione degli incontri . . . . .	59
4.1.5.2.1	Incontri Interni . . . . .	59
4.1.5.2.2	Incontri Esterni . . . . .	60
4.1.5.3	Gestione degli strumenti di coordinamento . . . . .	61
4.1.5.3.1	Ticketing . . . . .	61
4.1.5.4	Gestione degli strumenti di versionamento . . . . .	62
4.1.5.4.1	Repository . . . . .	62
4.1.5.4.2	Struttura del repository . . . . .	63
4.1.5.4.3	Tipi di file e .gitignore . . . . .	63
4.1.5.4.4	Norme sui commit . . . . .	63
4.1.5.5	Gestione dei rischi . . . . .	64
4.1.5.5.1	Codice identificativo . . . . .	64
4.1.6	Strumenti . . . . .	65
4.1.6.1	Sistema operativo . . . . .	65
4.1.6.2	Slack . . . . .	65
4.1.6.3	Wrike . . . . .	66
4.1.6.4	Git . . . . .	67
4.1.6.5	GitHub . . . . .	67

## Elenco delle figure

1	Trender . . . . .	22
2	Astah Desktop per Windows . . . . .	23
3	IntelliJ IDEA Desktop per Windows . . . . .	24



---

4	TexStudio Desktop per Windows . . . . .	39
5	Lucidchart . . . . .	39
6	Organizzazione di un incontro interno . . . . .	60
7	Organizzazione di un incontro esterno . . . . .	61
8	Assegnazione di un ticket . . . . .	62
9	Slack Desktop per Windows . . . . .	66
10	Wrike Web application per Windows . . . . .	67

# 1 Introduzione

## 1.1 Scopo del documento

Questo documento ha lo scopo di definire le regole, gli strumenti e le convenzioni adottate dal gruppo DigitalCookies durante l'intero svolgimento del progetto. In quest'ottica, questo documento deve essere visionato da tutti i componenti del gruppo, che sono obbligati ad applicare quanto scritto al fine di mantenere omogeneità e coesione in ogni aspetto del progetto.

In caso di modifiche o aggiunte al presente documento è obbligatorio informare ogni membro del gruppo.

## 1.2 Scopo del prodotto

Lo scopo del *prodotto<sub>G</sub>* è creare un software di costruzione di diagrammi *UML<sub>G</sub>* con relativa generazione di codice *Java<sub>G</sub>*. Il codice potrà essere generato dall'utente a partire dai diagrammi UML delle *classi<sub>G</sub>* e da una versione modificata del diagramma delle *attività<sub>G</sub>*.

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

- delineare la struttura delle classi utilizzando lo standard UML;
- definire il corpo dei metodi delle classi sfruttando una versione modificata del diagramma delle attività;
- generare un applicativo scritto in codice Java a partire dai diagrammi sopracitati.

L'utente potrà inoltre sfruttare la *libreria<sub>G</sub>* fornita con il prodotto per generare con facilità diagrammi relativi al dominio dei giochi di carte.

L'*editor<sub>G</sub>* sarà fruibile dall'utente attraverso un *browser<sub>G</sub>* desktop idoneo all'utilizzo delle tecnologie *HTML5<sub>G</sub>*, *CSS3<sub>G</sub>* e *JavaScript<sub>G</sub>*.

## 1.3 Ambiguità

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

## 1.4 Riferimenti

### 1.4.1 Normativi

- **ISO/IEC 12207**  
[https://en.wikipedia.org/wiki/ISO/IEC\\_12207](https://en.wikipedia.org/wiki/ISO/IEC_12207) (sezioni §2, §3.2, §3.3);
- **Capitolato:**<sub>G</sub>  
<http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C6p.pdf> (ultima consultazione effettuata in data 27-02-2017);
- **Verbale di incontro interno** con i componenti del gruppo del 25-02-2017;
- **Verbal di incontro esterno** con il *proponente*<sub>G</sub> Zucchetti S.p.A. del 23-02-2017 e del 23-03-2017;
- **Verbale di incontro interno** con i componenti del gruppo del 21-04-2017;
- **Verbale di incontro interno** con i componenti del gruppo del 27-04-2017;
- **Verbale di incontro interno** con i componenti del gruppo del 25-05-2017.

### 1.4.2 Informativi

- Per una dettagliata guida  $\text{\LaTeX}$  fare riferimento a *Guida ai comandi  $\text{\LaTeX}$  v1.0.0*;
- Per una dettagliata guida *Git*<sub>G</sub> fare riferimento a *Guida ai comandi Git v1.0.0*;

## 2 Processi primari

### 2.1 Fornitura

#### 2.1.1 Scopo

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

#### 2.1.2 Aspettative

Nel Registro delle modifiche: la localizzazione delle modifiche effettuate aumenterà specificandola in modo numerico e non narrativo; occorrerà anche migliorarne la tracciabilità rispetto alle relative cause. so dell'intero progetto il gruppo intende instaurare con Zucchetti S.p.A., in particolare nella figura del referente Dr. Gregorio Piccoli, un rapporto di costante collaborazione al fine di:

- Determinare aspetti chiave per soddisfare i bisogni del proponente;
- Determinare vincoli sui processi e sui requisiti;
- Stimare i costi;
- Concordare la qualifica del prodotto.

#### 2.1.3 Descrizione

Il gruppo intende mantenere un costante dialogo con il proponente per avere un riscontro efficace sul lavoro svolto.

#### 2.1.4 Attività

##### 2.1.4.1 Studio di fattibilità

È compito del *Responsabile* di Progetto organizzare riunioni preventive tra i membri del gruppo al fine di permettere lo scambio di opinioni sui capitolati proposti. Il documento è redatto dall'*Analista<sub>G</sub>* sulla base dei seguenti punti:

- **Dominio tecnologico e applicativo:** si valuta il capitolato prendendo in considerazione la conoscenza attuale delle tecnologie richieste da parte dei membri del gruppo, valutando anche eventuali esperienze passate con problematiche simili;

- **Rapporto costi/benefici:** si analizzano la quantità di requisiti obbligatori, il costo in rapporto ai risultati previsti e l'interesse del gruppo rispetto alle tematiche del capitolato;
- **Individuazione dei rischi:** si analizzano i punti critici della realizzazione, quali ad esempio mancanza di conoscenze adeguate o difficoltà nell'individuazione di requisiti. Si analizzano inoltre eventuali problematiche che possono sorgere in corso d'opera.

#### 2.1.4.2 Piano di Progetto

Il *Responsabile*, aiutato dagli *Amministratori*, dovrà redigere un piano da seguire nella realizzazione del progetto. Il documento dovrà contenere:

- **Analisi dei rischi:** si analizzano nel dettaglio i rischi che potrebbero insorgere nel corso del progetto e i modi per affrontarli, capendo la probabilità che essi accadano e il livello di gravità ad essi associato;
- **Pianificazione:** si pianificano le attività da svolgere nel corso del progetto, fornendo delle scadenze temporali precise;
- **Preventivo e Consuntivo<sub>G</sub>:** sulla base della pianificazione si stima la quantità di lavoro necessaria per ogni fase, proponendo così un preventivo per il costo totale del progetto. Alla fine di ogni attività si redige inoltre un consuntivo di periodo per tracciare l'andamento rispetto a quanto preventivato.

#### 2.1.4.3 Piano di Qualifica

I *Verificatori* dovranno scegliere una strategia da adottare per la *verifica<sub>G</sub>* e la *validazione<sub>G</sub>* del materiale prodotto dal gruppo. Il documento dovrà contenere:

- **Visione generale:** si stabiliscono gli standard adottati per il controllo sulla *qualità<sub>G</sub>* di processo e di prodotto, tenendo in considerazione le risorse a disposizione. Vengono inoltre classificate le possibili anomalie, le responsabilità e l'organizzazione delle attività di verifica;
- **Obiettivi di qualità:** si stabiliscono gli obiettivi da raggiungere per la qualità di processo e di prodotto, assegnando dei range alle metriche descritte in sezione 3.2.4.1;
- **Specifica dei test:** si devono definire nel dettaglio i test da eseguire sul prodotto;
- **Resoconto delle attività di verifica:** alla fine di ogni attività si devono riportare le metriche calcolate e un resoconto sulla verifica di tale attività;
- **Esiti delle revisioni:** ad ogni revisione si devono descrivere le modifiche apportate in base alle segnalazioni del *commitente<sub>G</sub>*.

#### 2.1.4.4 Collaudo e Consegna del Prodotto

Prima di poter consegnare il prodotto, il gruppo deve effettuare il collaudo in presenza del proponente e del committente. Prima di tale collaudo il gruppo deve assicurarsi della correttezza, completezza ed affidabilità delle componenti software, in modo che durante il collaudo possano essere dimostrati i seguenti punti:

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

In seguito al superamento del collaudo finale, il *Responsabile* di Progetto presenta al committente il consuntivo finale e consegna il prodotto su un supporto fisico.

## 2.2 Sviluppo

### 2.2.1 Scopo

Questo processo contiene tutte quelle attività e quei compiti svolti dal gruppo nel produrre il software finale richiesto dal proponente.

### 2.2.2 Aspettative

Per una corretta implementazione di tale processo le aspettative sono le seguenti:

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

### 2.2.3 Descrizione

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

- Analisi dei requisiti
- Progettazione
- Codifica

## 2.2.4 Attività

### 2.2.4.1 Analisi dei requisiti

#### 2.2.4.1.1 Scopo

Individuare ed elencare, evitando ambiguità, tutti i requisiti del capitolato. I requisiti possono essere estrapolati da più fonti:

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

Il risultato dell'attività è un accordo tra il proponente Zucchetti S.p.A. e il gruppo DigitalCookies riguardo i requisiti e le funzionalità del prodotto, che vengono descritti nel documento *Analisi dei Requisiti v3.0.0* e che rispettano le norme individuate nella sezione 3.1.4.1.

#### 2.2.4.1.2 Aspettative

Obiettivo dell'attività è l'individuazione di tutti i requisiti richiesti dal proponente.

#### 2.2.4.1.3 Descrizione

Durante l'Analisi dei Requisiti il gruppo analizza le fonti e si confronta con il proponente per individuare i casi d'uso e i requisiti. Il tracciamento dei requisiti avviene tramite il software *Trender<sub>G</sub>*.

#### 2.2.4.1.4 Diagrammi

L'Analisi dei requisiti deve utilizzare le seguenti tipologie di diagrammi *UML*:

- **Diagrammi dei casi d'uso:** illustrano i casi d'uso, in particolare mettono in evidenza gli attori e i servizi del sistema.

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



#### 2.2.4.1.5 Qualità dei requisiti

In seguito all'analisi dei requisiti in cui si valutano le necessità del proponente del software, vengono schematizzati i requisiti che descrivono il sistema. La specifica dei requisiti deve avere le seguenti qualità essenziali:

- **Assenza di ambiguità:** ogni requisito ha una sola interpretazione formale;
- **Correttezza:** ogni requisito è realmente richiesto e o necessario per gli utenti del sistema finale;
- **Completezza:** ogni funzione richiesta al software e il suo comportamento rispetto ad ogni possibile input è specificato;
- **Verificabilità:** è possibile verificare che il sistema realizzi ogni requisito, questo richiede la non ambiguità dei requisiti;
- **Consistenza:** nessun requisito è in contraddizione con gli altri;
- **Modificabilità:** la struttura e lo stile dei requisiti sono tali da consentire facili modifiche, preservando consistenza e completezza;
- **Tracciabilità:** l'origine di ciascun requisito è chiara e può essere referenziata nello sviluppo futuro.

#### 2.2.4.2 Progettazione

##### 2.2.4.2.1 Scopo

Questa attività definisce, in funzione dei requisiti specificati nell'*Analisi dei Requisiti*, tutte le caratteristiche essenziali del prodotto software richiesto. Essa ha come obiettivo la realizzazione dell'architettura del sistema, che viene descritta nei documenti *Specific Tecnica v2.0.0* e *Definizione di Prodotto v1.0.0*, secondo le norme individuate rispettivamente nelle sezioni 3.1.4.2 e 3.1.4.3.

##### 2.2.4.2.2 Aspettative

Il processo ha come risultato la realizzazione dell'architettura del sistema. L'architettura deve rispettare le qualità descritte nella sezione 2.2.4.2.5.

##### 2.2.4.2.3 Descrizione

Precedentemente alla realizzazione dell'architettura il gruppo decide le tecnologie da

utilizzare, elencandone vantaggi e svantaggi nel documento *Specifica Tecnica v2.0.0*. Durante l'attività di progettazione il gruppo realizza l'architettura del sistema rispettando i requisiti ed i vincoli stabiliti con il proponente. In particolare il gruppo individua e descrive le classi e i metodi che andranno a comporre il prodotto e le loro interazioni.

#### 2.2.4.2.4 Diagrammi

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

- **Diagrammi delle classi:** illustrano una collezione di elementi che rappresenta un modello come classi e tipi, con all'interno i loro contenuti e le loro relazioni;
- **Diagramma dei package:** raggruppamenti di classi in una *unità<sub>G</sub>* ad un livello più alto;
- **Diagrammi delle attività:** rappresentano il flusso di operazioni relativo ad un'attività; vengono utilizzati in particolare per descrivere la logica di un algoritmo;
- **Diagrammi di sequenza:** descrivono una determinata *sequenza<sub>G</sub>* di azioni dove tutte le scelte sono già state fatte. Nel diagramma non compaiono scelte, nè flussi alternativi.

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

#### 2.2.4.2.5 Qualità dell'architettura

In seguito alla specifica dei requisiti che consolida le funzionalità richieste, viene realizzata l'architettura del sistema. Tale architettura deve perseguire le seguenti caratteristiche:

- **Sufficienza:** l'architettura soddisfa tutti i requisiti;
- **Modularità:** l'architettura è suddivisa in parti ben distinte;
- **Robustezza:** l'architettura può sopportare diversi ingressi dall'utente e dall'ambiente;
- **Affidabilità:** l'architettura garantisce una buona usabilità del prodotto quando viene implementata;
- **Manutenibilità:** l'architettura può subire modifiche a costi contenuti.

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

- maggiore leggibilità e riusabilità del codice;
- semplificazione dell'individuazione e della correzione degli errori;

- possibilità di realizzazione di prototipi.

Per perseguire le qualità sopra elencate, è richiesta l'implementazione di *design pattern<sub>G</sub>* ove necessario. Di particolare importanza è il rispetto delle metriche per la progettazione architetturale definite in 3.2.4.1.

#### **2.2.4.3 Codifica**

##### **2.2.4.3.1 Scopo**

Questa attività ha come scopo l'effettiva realizzazione del prodotto software richiesto. In questa fase si concretizza la soluzione attraverso la programmazione, in modo da ottenere il prodotto software finale.

##### **2.2.4.3.2 Aspettative**

Obiettivo dell'attività è la creazione di un prodotto software conforme alle richieste prefissate con il proponente.

##### **2.2.4.3.3 Descrizione**

La codifica dovrà seguire le indicazioni presenti nel documento *Definizione di Prodotto* per la realizzazione delle singole componenti presenti. La scrittura del codice dovrà inoltre rispettare gli obiettivi di qualità definiti all'interno del *Piano di Qualifica v3.0.0* per poter garantire una buona qualità al codice.

L'implementazione concreta dell'architettura porta lo sviluppatore a dover installare software di terze parti per procedere con la stesura del codice per realizzare un prodotto che dovrà esser compreso dagli utenti che ne usufruiranno e che dovrà esser mantenuto attivo dopo il proprio rilascio. Proprio per questo durante l'attività di codifica verranno scritti i documenti *Manuale Utente<sub>G</sub>* e *Manuale Manutentore<sub>G</sub>*, descritti rispettivamente nelle sezioni 3.1.4.4 e 3.1.4.5.

L'attività di codifica comprende infine l'implementazione dei test che sono stati definiti nelle attività precedenti, in modo tale che, durante i processi di verifica e di validazione, sia possibile eseguirli per verificare la correttezza del software prodotto o per rilevarne dei *bug<sub>G</sub>*.

##### **2.2.4.3.4 Nomenclatura dei file**

I file contenenti il codice sorgente devono essere suddivisi in cartelle, in modo tale che la loro struttura rispecchi l'architettura del sistema. I nomi dei file devono essere univoci

e vanno scritti con lo stile *lowercase<sub>G</sub>*. È possibile utilizzare dei punti per dividere nomi troppo lunghi, per una maggiore chiarezza.

#### 2.2.4.3.5 Stile di codifica

Al fine di garantire uniformità nel codice del progetto ciascun membro del gruppo è tenuto a rispettare le seguenti norme:

- **Indentazione:** è richiesto l'utilizzo di esattamente quattro spazi;
- **Operatori:** gli operatori vanno preceduti e seguiti da uno spazio;
- **Parentesi graffe:** per i costrutti complessi che sono delimitati da parentesi graffe è richiesto che la parentesi di apertura venga scritta in linea e quella di chiusura sia scritta a capo dell'ultimo *statement<sub>G</sub>*;
- **Nomi di variabili:** le variabili vanno scritte con lo stile *camelCase<sub>G</sub>*;
- **Nomi di variabili globali:** le variabili globali vanno scritte con lo stile *UPPERCASE<sub>G</sub>*.
- **Nomenclatura:** gli elementi devono seguire le seguenti convenzioni:
  - ogni elemento deve avere un nome univoco, chiaro e privo di ambiguità rispetto alla funzione svolta;
  - ogni metodo che esegue operazioni su uno specifico oggetto deve utilizzare la struttura *verboNome*;
  - nomi ortograficamente errati e abbreviazioni poco chiare devono essere evitati.

#### 2.2.4.3.6 Intestazione e commenti

Ogni *file<sub>G</sub>* contenente codice deve avere la seguente intestazione:

```
/*
* File: nome file
* Version: versione file
* Type: tipo file
* Date: data di creazione
* Author: nome cognome autore/i
* E-mail: email gruppo
*
* License: tipo licenza
*
* Registro modifiche:
```

```
* Autore || Data || breve descrizione modifiche  
*  
*/
```

Ogni metodo JavaScript deve avere un commento la cui struttura segue la seguente sintassi:

```
/**  
 * Descrizione del metodo  
 * @param {TipoParametro} NomeParametro - Descrizione  
 *   parametro  
 */
```

Nella stesura del commento il *Programmatore<sub>G</sub>* è tenuto a rispettare le seguenti convenzioni:

- quando possibile, è bene documentare con un commento in linea parti di codice non immediatamente comprensibili;
- è opportuno usare frasi di senso compiuto in modo da essere più chiari possibili;
- è preferibile porre il commento sopra una riga di codice anziché alla fine di essa (in particolare all'inizio della porzione di codice interessata), per una miglior facilità di lettura;
- ogniquale volta viene aggiornato un metodo, deve essere mantenuto aggiornato il relativo commento.

#### 2.2.4.3.7 Versionamento

La versione del codice viene inserita all'interno dell'intestazione del file e rispetta il seguente formalismo:

**X.Y**

- **X**: è l'indice di versione principale, un incremento di tale indice rappresenta un avanzamento della versione stabile, che porta il valore dell'indice Y ad essere azzerato;
- **Y**: è l'indice di modifica parziale, un incremento di tale indice rappresenta una verifica o una modifica rilevante, come per esempio la rimozione o l'aggiunta di una istruzione.

La versione *1.0* deve rappresentare la prima versione del file completo e stabile, cioè quando le sue funzionalità obbligatorie sono state definite e si considerano funzionanti.

Solo dalla versione *1.0* è possibile testare il file, con degli appositi test definiti, per verificarne l'effettivo funzionamento.

#### **2.2.4.3.8 Ricorsione**

Il *Programmatore* deve produrre codice sorgente efficiente. In particolare, deve evitare la ricorsione; nel caso fosse molto difficile trovare un metodo iterativo equivalente, egli è tenuto a fornire una prova di terminazione dell'algoritmo ricorsivo.

#### **2.2.5 Strumenti**

Di seguito sono elencati gli strumenti utilizzati dal gruppo durante il progetto.

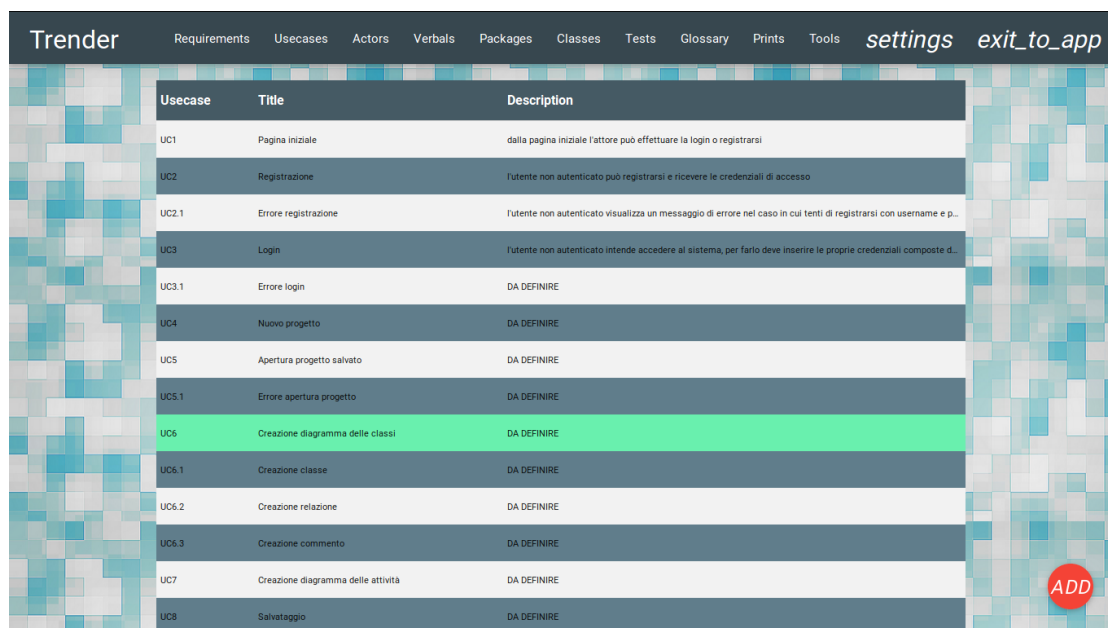
##### **2.2.5.1 Trender**

Il gruppo utilizza l'applicativo web Trender per gestire in maniera veloce e automatizzata tutti i dati ricavati dall'analisi dei requisiti.

Ogni componente del gruppo può accedervi, utilizzando l'account comune predisposto dall'*Amministratore<sub>G</sub>* di Progetto. Il *database<sub>G</sub>* viene gestito tramite *MySQL<sub>G</sub>*, mentre il resto del sito utilizza JavaScript e *PHP<sub>G</sub>*.

Le funzioni offerte da tale applicativo sono:

- Tracciamento dei requisiti
- Tracciamento dei casi d'uso
- Tracciamento dei verbali
- Tracciamento degli attori presenti nel sistema
- Tracciamento dei packages
- Tracciamento delle classi
- Tracciamento dei test
- Possibilità di stampare direttamente in codice  $\text{\LaTeX}$  quanto archiviato



The screenshot shows the Trender application interface. At the top, there is a navigation bar with the following items: Trender, Requirements, Usecases, Actors, Verbals, Packages, Classes, Tests, Glossary, Prints, Tools, settings, and exit\_to\_app. Below the navigation bar is a table with three columns: Usecase, Title, and Description. The table contains 15 rows of use cases. The row for UC6, 'Creazione diagramma delle classi', is highlighted in green. A red circular button with the text 'ADD' is located at the bottom right of the table.

Usecase	Title	Description
UC1	Pagina iniziale	dalla pagina iniziale l'attore può effettuare la login o registrarsi
UC2	Registrazione	l'utente non autenticato può registrarsi e ricevere le credenziali di accesso
UC2.1	Errore registrazione	l'utente non autenticato visualizza un messaggio di errore nel caso in cui tenti di registrarsi con username e p...
UC3	Login	l'utente non autenticato intende accedere al sistema, per farlo deve inserire le proprie credenziali composte d...
UC3.1	Errore login	DA DEFINIRE
UC4	Nuovo progetto	DA DEFINIRE
UC5	Apertura progetto salvato	DA DEFINIRE
UC5.1	Errore apertura progetto	DA DEFINIRE
UC6	Creazione diagramma delle classi	DA DEFINIRE
UC6.1	Creazione classe	DA DEFINIRE
UC6.2	Creazione relazione	DA DEFINIRE
UC6.3	Creazione commento	DA DEFINIRE
UC7	Creazione diagramma delle attività	DA DEFINIRE
UC8	Salvataggio	DA DEFINIRE

Figura 1: Trender

### 2.2.5.2 Astah

Per la produzione dei diagrammi UML viene utilizzato *Astah Professional Edition<sub>G</sub>* versione 7.2, in quanto offre molte agevolazioni per la produzione veloce dei diagrammi e risulta semplice da usare.

<http://astah.net/>

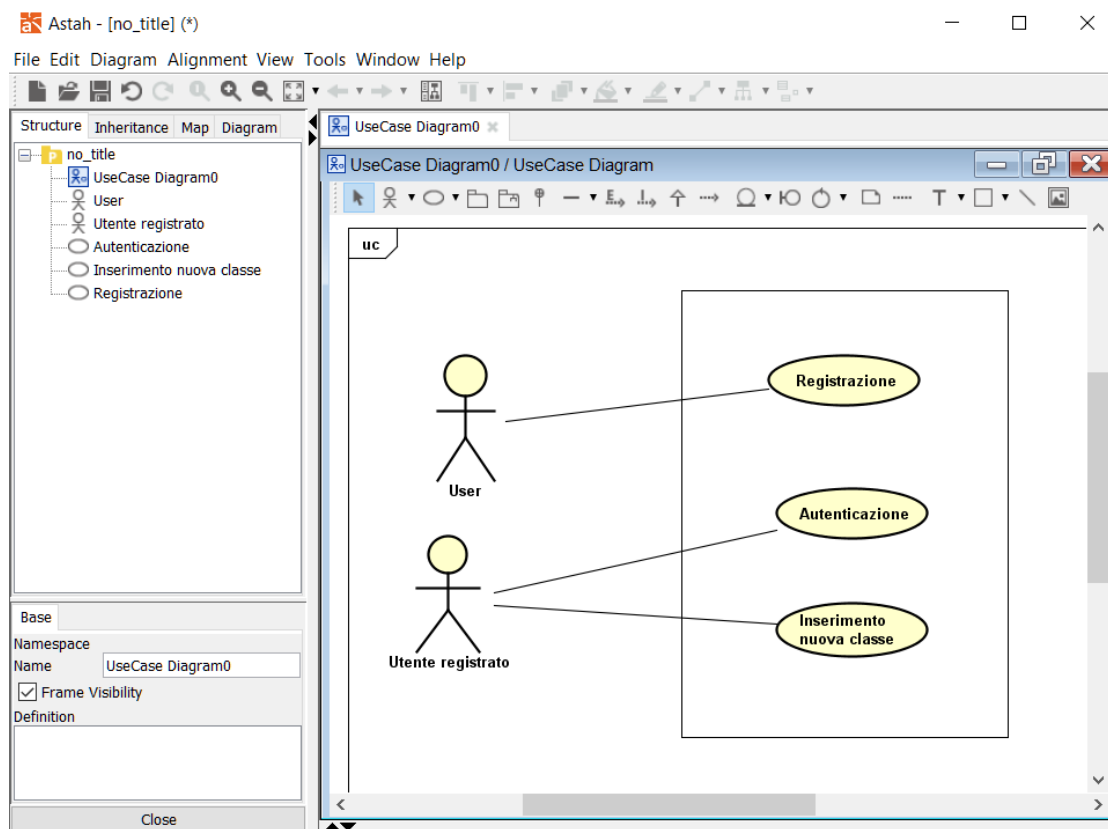


Figura 2: Astah Desktop per Windows

### 2.2.5.3 IntelliJ IDEA

IntelliJ IDEA viene utilizzato per la codifica in JavaScript e TypeScript. Questo  $IDE_G$  offre piena compatibilità con  $Linux_G$ ,  $Windows_G$  e  $macOS_G$ , oltre ad essere un potente editor con molte funzionalità integrate.

<https://www.jetbrains.com/idea/>



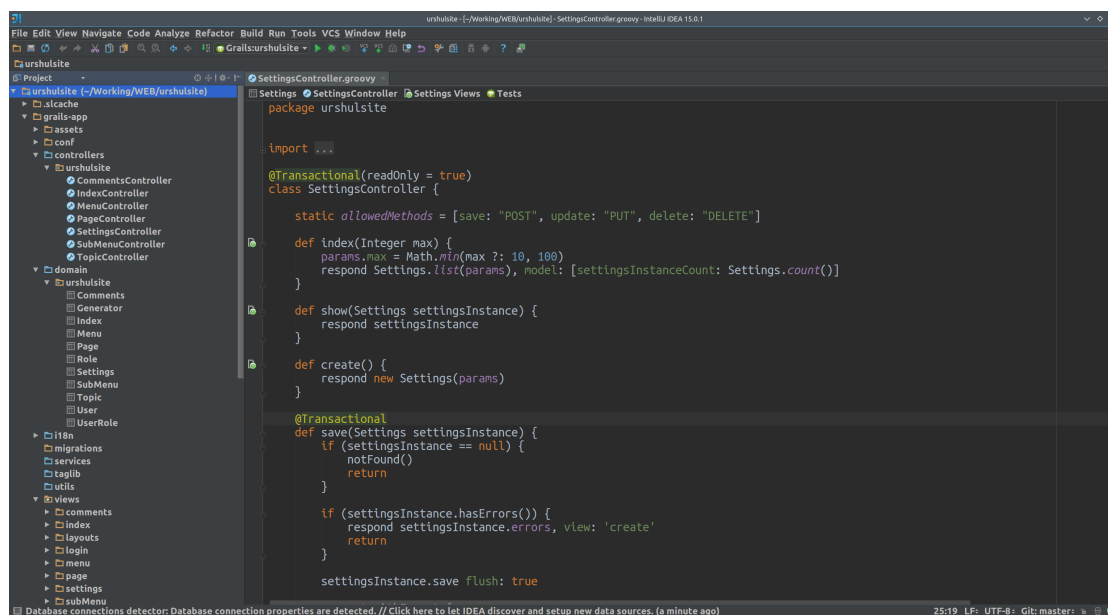


Figura 3: IntelliJ IDEA Desktop per Windows

## 3 Processi di supporto

### 3.1 Documentazione

#### 3.1.1 Scopo

Questo processo include i dettagli su come deve essere redatta e mantenuta la documentazione durante il ciclo di vita del software.

#### 3.1.2 Aspettative

Le aspettative della corretta implementazione di tale processo sono:

- una visione precisa della documentazione che va prodotta durante il ciclo di vita del software;
- l'individuazione di una serie di norme per la stesura di documenti coerenti e validi;
- la stesura di una documentazione formale e coerente.

#### 3.1.3 Descrizione

In questa sezione devono essere indicate tutte le norme e le convenzioni adottate dal gruppo, per consentire la stesura di una documentazione valida e coerente.

#### 3.1.4 Attività

##### 3.1.4.1 Analisi dei Requisiti

Questo documento sarà scritto dall'*Analista<sub>G</sub>* e dovrà includere:

- **Casi d'uso<sub>G</sub>**: devono essere descritti i casi d'uso con una descrizione testuale e un diagramma rappresentativo dove utile per una più immediata comprensione;
- **Requisiti**: devono essere descritti i requisiti individuati in accordo col proponente, specificandone in particolare il tipo e il livello di necessità;
- **Tracciamento dei requisiti**: ogni requisito deve essere tracciato alle fonti da cui è stato generato. Nella sezione 2.2.5.1 viene descritto l'applicativo web *Trender*, utile anche a generare automaticamente le tabelle di tracciamento.

#### 3.1.4.1.1 Casi d'uso

Ogni caso d'uso è descritto dalla seguente struttura:

- Codice identificativo
- Titolo
- Diagramma UML
- Attori primari
- Attori secondari
- Scopo e descrizione
- Precondizione
- Postcondizione
- Flusso base degli eventi
- Inclusioni (se presenti)
- Estensioni (se presenti)

Ogni caso d'uso è identificato da un codice, che segue il seguente formalismo:

$$\text{UC}\{\text{codice\_padre}\}.\{\text{codice\_livello}\}$$

Dove:

- **codice\_padre**: numero che identifica univocamente i casi d'uso;
- **codice\_livello**: numero progressivo che identifica i sottocasi. Può a sua volta includere altri livelli.

#### 3.1.4.1.2 Requisiti

Ogni requisito è strutturato come segue:

- Codice identificativo
- Descrizione
- Fonti

Ogni requisito ha il seguente formalismo:

$$\text{R}\{\text{X}\}\{\text{Y}\}\{\text{codice\_padre}\}.\{\text{codice\_livello}\}$$

Dove:

- **X**: identifica uno dei seguenti tipi di requisito:

- 1: requisito funzionale;
- 2: requisito prestazionale;
- 3: requisito qualitativo;
- 4: vincolo progettuale.
- **Y**: identifica uno dei seguenti gradi di necessità:
  - *O*: requisito obbligatorio;
  - *F*: requisito facoltativo;
  - *D*: requisito desiderabile.
- **codice\_padre**: numero che identifica univocamente i requisiti;
- **codice\_livello**: numero progressivo che identifica i sottorequisiti. Può a sua volta includere altri livelli.

#### 3.1.4.2 Specifica Tecnica

Questo documento sarà scritto dal Progettista e dovrà includere:

- **Diagrammi UML**:
  - Diagrammi delle classi
  - Diagrammi dei *package<sub>G</sub>*
  - Diagrammi di attività
  - Diagrammi di sequenza
- **Tecnologie utilizzate**: devono essere descritte le tecnologie adottate specificandone l'utilizzo nel progetto, i vantaggi e gli svantaggi;
- **Design pattern<sub>G</sub>**: devono essere descritti i design pattern utilizzati per realizzare l'architettura. Ogni design pattern deve essere accompagnato da una descrizione ed un diagramma, che ne esponga il significato e la struttura;
- **Tracciamento delle componenti**: ogni requisito deve riferirsi al componente che lo soddisfa. Nella sezione 2.2.5.1 viene descritto l'applicativo web *Trender*, utile a generare automaticamente le tabelle di tracciamento. Tramite questa operazione è possibile garantire che ogni requisito venga soddisfatto;

##### 3.1.4.2.1 Definizione dei Package

I package, individuati all'interno dell'architettura durante il periodo di progettazione, devono avere un codice univoco che segue il seguente formalismo:

### **Parte\_\_Software::Package::SottoPackage::[...]**

Dove:

- **Parte\_\_Software:** assume i significati *Front-end<sub>G</sub>* o *Back-end<sub>G</sub>* del prodotto SWE-Designer;
- **Package:** rappresenta il nome di un package padre che racchiuderà all'interno altri package oppure classi;
- **SottoPackage:** rappresenta il figlio del package padre. Non è detto che ogni package padre contenga dei sottopackage, quindi è un campo opzionale. Un sottopackage potrebbe a sua volta contenere altri package come figli e così via.

Per ogni package vengono inoltre definite:

- **Descrizione:** una breve descrizione del package in questione;
- **Package contenuti:** un elenco delle altre componenti contenute all'interno del proprio package;
- **Framework<sub>G</sub> esterni:** le interazioni del package in questione con altre componenti esterne (framework).

#### **3.1.4.2.2 Definizione delle Classi**

Le classi rappresentano il dettaglio massimo raggiungibile all'interno dell'architettura. La definizione di una classe si divide in una descrizione ad alto livello presente all'interno del documento *Specifica Tecnica v2.0.0* e in una descrizione più dettagliata presente all'interno del documento Definizione di Prodotto.

Le classi devono avere un codice identificativo univoco che segue il seguente formalismo:

### **Parte\_\_Software::Package::SottoPackage::[...]:Classe**

Dove:

- **Parte\_\_Software, Package, SottoPackage** sono descritti nella sezione precedente;
- **Classe:** rappresenta la componente più piccola, contenuta nei package, che contiene attributi e metodi.

Per ogni classe vengono inoltre definite:

- **Descrizione:** una descrizione della classe (o interfaccia) in questione;
- **Utilizzo:** una descrizione dell'utilizzo che deve essere fatto di questa classe all'interno del prodotto;
- **Relazione con altre classi:** le relazioni che la classe ha con altre classi della stessa parte software, suddivise in:

- **IN**: relazioni entranti;

- **OUT**: relazioni uscenti;

(può essere omesso se non necessario)

- **Interfacce implementate**: la lista delle eventuali interfacce implementate dalla classe (può essere omesso se non necessario)
- **Classi ereditate**: la lista delle eventuali classi ereditate dalla classe (può essere omesso se non necessario)
- **Sottoclassi**: la lista delle eventuali classi che ereditano dalla classe o interfaccia (può essere omesso se necessario)

### 3.1.4.3 Definizione di Prodotto

Questo documento sarà scritto dal *Progettista* e dovrà includere:

- **Diagrammi UML**:
  - Diagrammi delle classi
  - Diagrammi di attività
  - Diagrammi di sequenza
- **Definizioni delle classi**: ogni  $classe_G$  e  $interfaccia_G$  deve essere descritta in modo da spiegarne in maniera esaustiva lo scopo e le funzionalità, evitando ridondanze;
- **Definizione dei metodi**: ogni metodo deve essere descritto specificandone in dettaglio il suo funzionamento;
- **Tracciamento delle classi**: ogni requisito deve essere tracciato in modo da garantire che ogni classe ne soddisfi almeno uno e poter risalire alle classi a esso associate. Nella sezione 2.2.5.1 viene descritto l'applicativo web *Trender*, utile anche a generare automaticamente le tabelle di tracciamento. Tramite questa operazione è possibile garantire che ogni classe soddisfi almeno un requisito.

#### 3.1.4.3.1 Definizione dei metodi

All'interno delle classi vengono definiti gli attributi e i metodi che gli appartengono. La sintassi per descrivere gli attributi segue il seguente formalismo:

**visibilità [nome: tipo] descrizione**

Dove:

- **visibilità**: assume i significati:

- +: public
- -: private
- ~: package
- #: protected
- **nome**: rappresenta il nome dell'attributo;
- **tipo**: rappresenta il tipo dell'attributo.
- **descrizione**: rappresenta una breve descrizione dell'attributo.

La sintassi per descrivere i metodi segue il seguente formalismo:

**visibilità nome(nomeAttr: tipoAttr): tipoRitorno descrizione**

Dove:

- **visibilità**: assume i significati:
  - +: public
  - -: private
  - ~: package
  - #: protected
- **nome**: rappresenta il nome del metodo;
- **tipoRitorno**: rappresenta il tipo di ritorno del metodo.
- **nomeAttr: tipoAttr**: rappresentano il nome e il tipo del parametro del metodo.
- **descrizione**: rappresenta una breve descrizione del metodo.

Per ogni metodo viene inoltre descritto ciascun parametro seguendo la sintassi utilizzata per gli attributi.

**3.1.4.4 Manuale Utente** I *Programmatori* dovranno redigere il *Manuale Utente* nel quale vengono fornite, a chi utilizzerà il prodotto, le informazioni principali per aiutarlo nel suo uso e fornirgli una spiegazione delle funzionalità presenti nell'applicazione.

**3.1.4.5 Manuale Manutentore** I *Programmatori* dovranno redigere il *Manuale Manutentore*. Il documento ha come destinatari coloro che hanno il compito di installare e configurare il prodotto prima del suo utilizzo. Questo documento permetterà inoltre di favorire la *manutenibilità<sub>G</sub>* del prodotto e l'eventuale estensione di nuove funzionalità.

### 3.1.5 Procedure

Per la stesura della documentazione il gruppo ha utilizzato il linguaggio  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ .

#### 3.1.5.1 Approvazione dei documenti

Ogni documento non formale di cui sia stata completata la stesura dovrà essere sottoposto al *Responsabile* di Progetto, che a sua volta si occuperà di incaricare i *Verificatori* di controllarne il contenuto e la forma. Nel caso tali *Verificatori* trovino degli errori, sarà loro compito riportarli al *Responsabile* di Progetto, che a sua volta incaricherà il redattore del documento di correggerli.

Questo ciclo va ripetuto fino a che il documento non è considerato completamente corretto dai *Verificatori*. A tal punto sarà sottoposto al *Responsabile* di Progetto, che potrà approvarlo o meno. Se approvato, il documento è da considerarsi come un documento formale. In caso contrario il *Responsabile* di Progetto dovrà comunicare le motivazioni per cui il documento non è stato approvato, esplicitando le modifiche da apportare.

### 3.1.6 Template

Il gruppo ha creato un template  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  per uniformare velocemente la struttura grafica e lo stile di formattazione dei documenti, in modo che i membri del gruppo debbano concentrarsi solo sulla stesura del contenuto e non sul suo aspetto.

### 3.1.7 Struttura dei documenti

#### 3.1.7.1 Prima pagina

La prima pagina di ogni documento è così strutturata:

- **Logo del gruppo:** visibile come primo elemento centrato orizzontalmente in alto;
- **Titolo:** nome del documento, visibile subito dopo il logo e centrato orizzontalmente;
- **Gruppo e progetto:** nome del gruppo e del progetto, appena sotto il titolo del documento e centrato orizzontalmente;
- **Recapito:** indirizzo di posta elettronica del gruppo, centrato orizzontalmente sotto il nome del gruppo e del progetto;
- **Tabella descrittiva:** visibile subito dopo il titolo, centrata orizzontalmente e contenente le seguenti informazioni:
  - versione;



- nome e cognome dei membri del gruppo incaricati della redazione del documento;
  - nome e cognome dei membri del gruppo incaricati della verifica del documento;
  - nome e cognome dei membri del gruppo incaricati dell’approvazione del documento;
  - tipo di uso;
  - destinatari del documento.
- **Descrizione:** centrata orizzontalmente ed il più possibile sintetica.

#### 3.1.7.2 Registro delle modifiche

Ogni documento, eccezion fatta per i verbali, deve contenere questo registro a seguito della prima pagina. Tale registro deve contenere le modifiche apportate al documento stesso, indicando per ognuna:

- versione del documento dopo la modifica;
- data della modifica apportata;
- nome e cognome della persona coinvolta nella modifica;
- ruolo ricoperto dalla persona coinvolta nella modifica;
- descrizione concisa della modifica apportata.

#### 3.1.7.3 Indice

Ogni documento, eccezion fatta per i verbali, deve avere un indice che ne agevoli la consultazione e permetta una lettura *ipertestuale*<sub>G</sub> e non necessariamente sequenziale. La numerazione di ogni indice deve partire da 1; ciascuna sottosezione deve essere separata dalla sezione padre tramite un punto, e la numerazione deve ripartire di volta in volta da 1.

#### 3.1.7.4 Contenuto principale

I margini orizzontali e verticali previsti dal template devono essere rispettati in ogni pagina. Ad eccezione della prima, tutte le pagine devono contenere un’intestazione ed un piè di pagina. L’intestazione è così strutturata:

- Logo del gruppo posto a sinistra;

- Indirizzo di posta elettronica del gruppo posto a destra;

Il piè di pagina è così strutturato:

- Nome e versione del documento corrente, posti a sinistra;
- Numerazione progressiva della pagina rispetto al totale posta a destra.

### 3.1.7.5 Note a piè di pagina

In caso di presenza in una pagina interna di note da esplicitare, esse vanno indicate nella pagina corrente, in basso a sinistra. Ogni nota deve riportare un numero e una descrizione.

### 3.1.8 Versionamento

Ogni documento, eccezion fatta per i verbali, deve essere versionato, in modo che sia possibile avere una visione specifica della sua storia e delle sue modifiche. Ad ogni versione deve corrispondere una riga nel registro delle modifiche.

Il formalismo da applicare è il seguente:

$$v\{X\}.\{Y\}.\{Z\}$$

dove:

- **X**:
  - Inizia da 0;
  - Viene incrementato da parte del *Responsabile* di Progetto all'approvazione del documento;
  - È limitato superiormente dal numero di revisioni.
- **Y**
  - Inizia da 0;
  - Viene incrementato da parte del *Verificatore*<sub>G</sub> ad ogni verifica;
  - Non è limitato superiormente;
  - Quando viene incrementato X, viene riportato a 0.
- **Z**
  - Inizia da 0;
  - Viene incrementato da parte del redattore del documento ad ogni modifica;
  - Non è limitato superiormente;

- Quando viene incrementato  $Y$ , viene riportato a 0.

### 3.1.9 Norme tipografiche

#### 3.1.9.1 Stile del testo

- **Glossario:** ogni parola contenuta nel glossario deve essere marcata, alla sua prima occorrenza in ogni documento, in carattere corsivo e con una  $G$  maiuscola a pedice:

*repository<sub>G</sub>*;

- **Grassetto:** viene applicato ai titoli e agli elementi di un elenco puntato che riassumono il contenuto di tale voce;
- **Corsivo:** Il corsivo dev'essere utilizzato per le seguenti occorrenze:
  - citazioni;
  - parole inserite nel glossario;
  - attività del progetto;
  - ruoli del progetto;
  - riferimenti ad altri documenti;
  - parole particolari solitamente poco usate o conosciute.
- **Maiuscolo:** le uniche parole che è consentito scrivere interamente in maiuscolo sono gli acronimi.

#### 3.1.9.2 Elenchi puntati

Gli elenchi puntati vengono rappresentati graficamente da un *pallino* nel primo livello, da un *trattino* nel secondo e da un *asterisco* nel terzo.

Gli elenchi puntati servono ad esprimere in modo sintetico un concetto, evitando frasi lunghe e discorsive. Ogni voce di un elenco puntato deve terminare con un punto e virgola, ad eccezione dell'ultima, che va terminata con un punto. Questa regola può non venire applicata per enfatizzare concetti relativamente corti.

#### 3.1.9.3 Formati comuni

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

- **Date:**

**GG-MM-AAAA**

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

- **Orari:**

**HH:MM**

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

- **Nomi ricorrenti:**

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

#### 3.1.9.4 Sigle

È previsto l'utilizzo delle seguenti sigle:

- **AR**: *Analisi dei Requisiti*;
- **PP**: *Piano di Progetto*;
- **NP**: *Norme di Progetto*;
- **SF**: *Studio di Fattibilità*;
- **PQ**: *Piano di Qualifica*;
- **ST**: *Specifiche Tecnica*;
- **MU**: *Manuale utente*;
- **DP**: *Definizione di Prodotto*;
- **RR**: Revisione dei requisiti;
- **RP**: Revisione di progettazione;
- **RQ**: Revisione di qualifica;
- **RA**: Revisione di accettazione;
- **Re**: *Responsabile di Progetto*;

- **Am:** *Amministratore* di Progetto;
- **An:** *Analista*;
- **Pt:** *Progettista*;
- **Pr:** *Programmatore*;
- **Ve:** *Verificatore*.

### 3.1.10 Elementi grafici

#### 3.1.10.1 Tabelle

Ogni tabella deve essere centrata orizzontalmente nella pagina e deve presentare sotto di essa la propria didascalia; in tale didascalia deve comparire il numero della tabella, incrementale in tutto il documento per agevolarne il tracciamento, oltre che una breve descrizione del suo contenuto.

Fanno eccezione le tabelle del registro delle modifiche che non hanno nessuna descrizione e le tabelle dei casi d'uso presenti nell'*Analisi dei requisiti v3.0.0* che hanno anche un diverso layout.

Al fine di aumentare la leggibilità delle tabelle, nelle celle contenenti uno 0 (zero) che non sia significativo per la comprensione della tabella stessa verrà inserito un trattino (-). Qualora il redattore lo ritenesse necessario per aumentare la leggibilità è permesso anche l'uso di colori di sfondo.

#### 3.1.10.2 Immagini

Ogni immagine deve essere centrata orizzontalmente ed essere nettamente separata dai paragrafi che la seguono e la precedono, per marcare un netto distacco tra testo e grafica e migliorare conseguentemente la leggibilità. Le immagini devono essere accompagnate da una didascalia analoga a quella descritta per le tabelle. Tutti i diagrammi UML vengono inseriti nei documenti sotto forma di immagine.

### 3.1.11 Classificazione dei documenti

#### 3.1.11.1 Documenti informali

Tutte le versioni dei documenti che non siano state direttamente approvate da parte del *Responsabile* di Progetto sono ritenute informali e in quanto tali sono considerate esclusivamente ad uso interno.

### 3.1.11.2 Documenti formali

Una versione di un documento viene definita formale quando è stata validato dal *Responsabile* di Progetto. Solo i documenti formali possono essere distribuiti all'esterno del gruppo. Per arrivare a tale stato il documento deve aver già superato la verifica e la validazione.

### 3.1.11.3 Verbali

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

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

Tali informazioni devono essere contenute nel primo paragrafo *Informazioni Generali*, che deve specificare anche gli argomenti trattati durante l'incontro.

Il secondo paragrafo si distingue invece a seconda del tipo di verbale:

- **Interno:** deve contenere la sezione *Riassunto Incontro*, dove vengono esplicitati più in dettaglio gli argomenti trattati;
- **Esterno:** deve contenere la sezione *Domande e Risposte*, dove vengono riportate le domande poste ai partecipanti esterni, seguite dalle loro risposte.

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

$$V\{X\}_{Y}$$

Dove:

- **X:** identifica uno dei seguenti tipi di verbale:
  - *E*: verbale esterno
  - *I*: verbale interno
- **Y:** identifica la data nel formato YYYYMMDD.

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

**CodiceVerbale.{X}**

Dove:

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

### 3.1.12 Strumenti

#### 3.1.12.1 $\text{\LaTeX}$

Per la stesura della documentazione il gruppo ha utilizzato il linguaggio  $\text{\LaTeX}$  per via delle possibilità che esso offre:

- creazione di documenti formali e divisi in sezioni molto velocemente;
- separazione del contenuto dalla formattazione tramite un file template separato e condiviso da tutti i documenti;
- personalizzazione del documento grazie all'elevato numero di librerie.

Inoltre è stata redatta una guida  $\text{\LaTeX}$  nella quale vengono descritti dei comandi ad uso interno per facilitare la stesura dei documenti e di conseguenza standardizzarne la forma. Per maggiori dettagli sul loro funzionamento si rimanda a *Guida ai comandi  $\text{\LaTeX}$  v1.0.0*.

#### 3.1.12.2 TexStudio

Per la stesura del codice  $\text{\LaTeX}$  è stato utilizzato l'editor *TexStudio<sub>G</sub>* nella versione 2.12.2. Questo strumento oltre ad integrare un compilatore e visualizzatore PDF, fornisce suggerimenti per completare i comandi di  $\text{\LaTeX}$ .

<http://www.texstudio.org/>

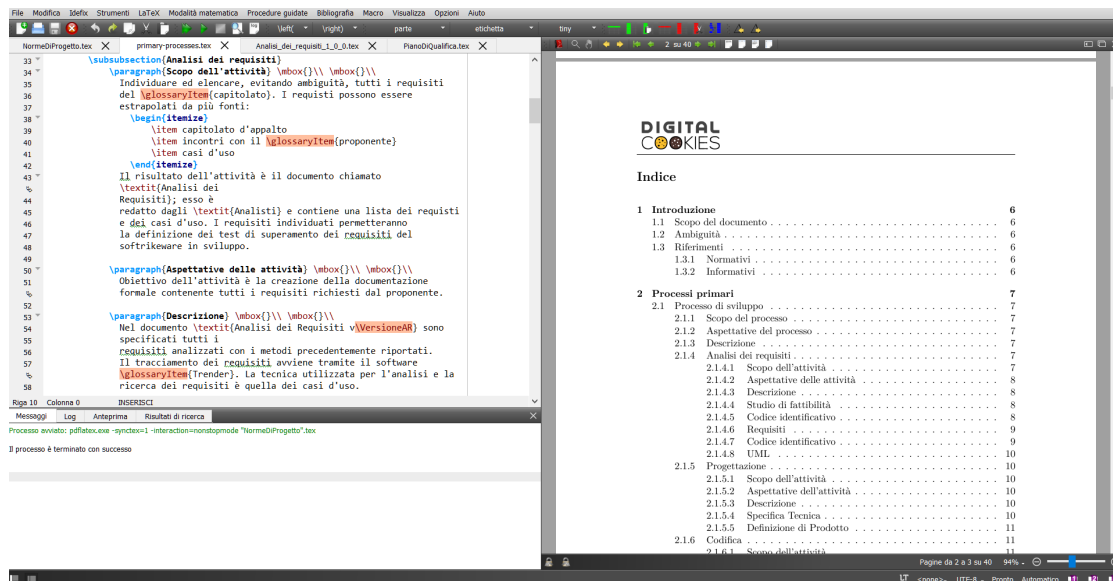


Figura 4: TexStudio Desktop per Windows

### 3.1.12.3 Lucidchart

Per la realizzazione di diagrammi illustrativi per i documenti viene utilizzata la piattaforma web *Lucidchart*<sub>G</sub>. Questo strumento è versatile e portatile, essendo utilizzabile direttamente via web.

<https://www.lucidchart.com/>

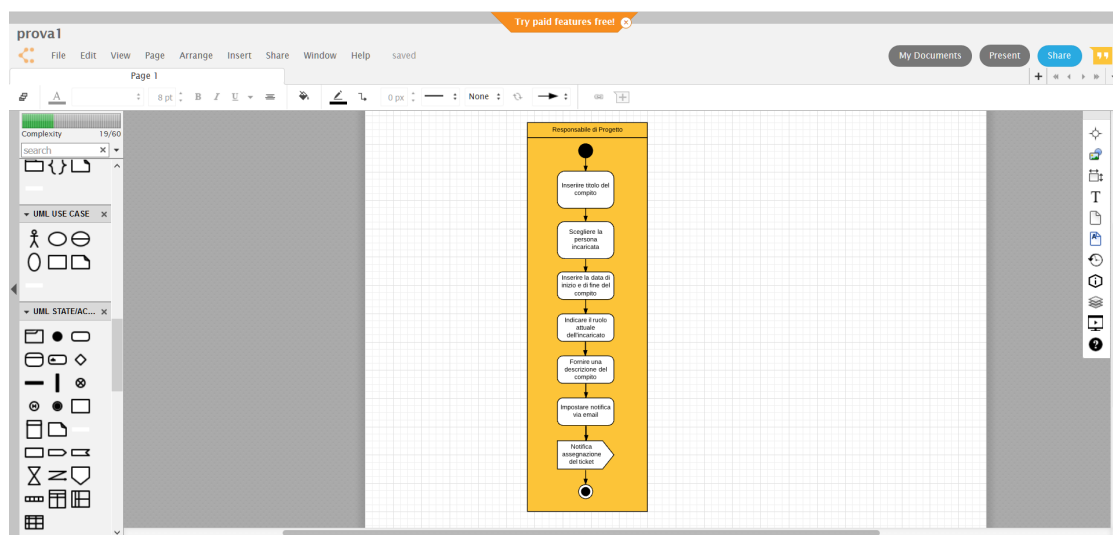


Figura 5: Lucidchart



## 3.2 Verifica

### 3.2.1 Scopo

Si occupa di accertare che non vengano introdotti errori nel prodotto a seguito dell'esecuzione delle attività dei processi svolti nella fase in esame.

### 3.2.2 Aspettative

Una corretta implementazione di tale processo permette di individuare:

- una procedura di verifica;
- i criteri per la verifica del prodotto;
- i difetti, da catalogare per essere corretti.

### 3.2.3 Descrizione

Il processo è suddiviso in due attività:

- **Analisi:** consiste nell'analisi del codice sorgente e la sua successiva esecuzione. Viene effettuata tramite due tecniche, l'analisi statica e l'analisi dinamica;
- **Test:** definisce tutti i test che vengono eseguiti sul prodotto software.

### 3.2.4 Attività

#### 3.2.4.1 Metriche

Per garantire la qualità del lavoro del gruppo, i *Verificatori* hanno definito delle metriche, riportandone gli obiettivi di qualità nel *Piano di Qualifica v3.0.0*. Tali metriche devono rispettare la seguente notazione:

$$M\{\mathbf{X}\}\{\mathbf{Y}\}$$

Dove:

- **X:** indica se la metrica si riferisce a processi, prodotto documento o prodotto software e può assumere i seguenti valori:
  - **PC:** per indicare una metrica per il processo;
  - **PD:** per indicare una metrica per il documento;
  - **PS:** per indicare una metrica per il software.

- **Y**: indica il codice univoco della metrica (numero intero incrementale a partire da 1).

#### 3.2.4.1.1 Metriche per i processi

Le seguenti metriche rappresentano un indicatore utilizzato per monitorare e prevedere l'andamento delle principali variabili critiche del progetto: tempi e costi. Sono utilizzate metriche di tipo consuntivo perché danno un riscontro immediato sullo stato attuale del progetto. Ad ogni incremento verranno valutati tali indici e, se necessario, verranno stabiliti opportuni provvedimenti da parte del *Responsabile* di Progetto.

Nome	Codice	Descrizione
Requirement Stability Index	MPC1	Indica la percentuale dei requisiti rimasti invariati nel tempo. Un valore elevato di tale metrica indica un'attività di analisi attenta e corretta. Viene calcolata tramite la seguente formula, dove ogni voce fa implicitamente riferimento ai requisiti: $RSI[\%] = 1 - \frac{\#aggiunti + \#tolti + \#modificati}{\#totali\ iniziali}$
Instability	MPC2	Calcola l'instabilità di un package, definita dalla seguente formula: $Instability[\%] = \frac{accoppiamento\ afferente}{accoppiamento\ afferente + accoppiamento\ efferente}$ Dove accoppiamento afferente indica il numero di riferimenti entranti dall'esterno del package, mentre accoppiamento efferente indica il numero di riferimenti uscenti dal package. Tale metrica indica la possibilità di effettuare modifiche ad una componente del package senza influenzarne altri all'interno dell'applicazione.
Violazioni dello stile di codifica	MPC3	Calcola il numero di occorrenze di violazioni dello stile di codifica, definito in 2.2.4.3.5, all'interno del codice. Un riscontro elevato indica poca cura nel processo di codifica e, di conseguenza, codice poco affidabile.
Errori frequenti nella documentazione	MPC4	Calcola il numero di errori frequenti, descritti nella tabella 3.2.4.2.1, individuati durante una verifica tramite Inspection. Un riscontro elevato indica poca cura nella stesura dei documenti.

Test di unità eseguiti	MPC5	<p>Indica la percentuale di test di unità effettivamente eseguiti. Un valore elevato di tale metrica indica un livello soddisfacente di test sulle componenti base del sistema. Viene calcolata tramite la seguente formula:</p> $TUI[\%] = \frac{\#test\ unità\ eseguiti}{\#test\ unità\ totali}$
Test di integrazione eseguiti	MPC6	<p>Indica la percentuale di test di integrazione effettivamente eseguiti. Un valore elevato di tale metrica indica un livello soddisfacente di test per le interazioni tra le varie componenti del sistema. Viene calcolata tramite la seguente formula:</p> $TII[\%] = \frac{\#test\ integrazione\ eseguiti}{\#test\ integrazione\ totali}$
Test di sistema eseguiti	MPC7	<p>Indica la percentuale di test di sistema effettivamente eseguiti. Un valore elevato di tale metrica indica un buon funzionamento delle funzionalità di sistema. Viene calcolata tramite la seguente formula:</p> $TSI[\%] = \frac{\#test\ sistema\ eseguiti}{\#test\ sistema\ totali}$
Test di validazione eseguiti	MPC8	<p>Indica la percentuale di test di validazione effettivamente eseguiti. Un valore elevato di tale metrica indica il soddisfacimento delle aspettative del proponente. Viene calcolata tramite la seguente formula:</p> $TVI[\%] = \frac{\#test\ validazione\ eseguiti}{\#test\ validazione\ totali}$

Schedule Variance	MPC9	<p>Permette di calcolare le tempistiche raggiunte alla data corrente rispetto alla schedulazione delle attività pianificate. È un indicatore di efficacia importante per il cliente.</p> $SV[\%] = \frac{BCWP - BCWS}{BCWP}$ <p>Dove:</p> <ul style="list-style-type: none"> <li>• <b>BCWP</b>: indica il valore delle attività realizzate alla data corrente;</li> <li>• <b>BCWS</b>: indica il costo pianificato per realizzare le attività di progetto alla data corrente.</li> </ul> <p>Se positivo lo schedule variance indica che il lavoro viene prodotto in anticipo rispetto quanto pianificato mentre se negativo il lavoro è in ritardo.</p>
Budget Variance	MPC10	<p>Permette di raffrontare i costi sostenuti alla data corrente rispetto al budget preventivato. È un indicatore finanziario.</p> $BV[\%] = \frac{BCWS - ACWP}{ACWP}$ <p>Dove:</p> <ul style="list-style-type: none"> <li>• <b>BCWS</b>: indica il costo pianificato per realizzare le attività di progetto alla data corrente;</li> <li>• <b>ACWP</b>: indica il costo effettivamente sostenuto per realizzare le attività di progetto alla data corrente.</li> </ul> <p>Se positivo il budget variance indica che il budget sta venendo speso più lentamente di quanto pianificato mentre se negativo indica che il budget sta venendo speso più velocemente di quanto pianificato.</p>

Tabella 2: Metriche per i processi

### 3.2.4.1.2 Metriche per i prodotti

#### • Documenti

La leggibilità dei documenti è indispensabile per garantirne la qualità. Il gruppo ha scelto di utilizzare un indice per misurare la leggibilità dei testi in lingua italiana:

Nome	Codice	Descrizione
Indice Gulpease	MPD1	<p>L'indice <math>Gulpease_G</math> è un indice di leggibilità di un testo tarato sulla lingua italiana e basato su due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere. La formula per il suo calcolo è:</p> $89 + \frac{300 \times (\# \text{ frasi}) - 10 \times (\# \text{ lettere})}{\# \text{ parole}}$ <p>Il risultato è un valore compreso nell'intervallo tra 0 e 100, dove il valore 100 indica la più alta leggibilità. Un indice inferiore a 80 indica documenti di difficile leggibilità per chi ha la licenza elementare, inferiore a 60 per chi ha la licenza media, inferiore a 40 per chi ha un diploma superiore.</p> <p>Inoltre non devono essere sottovalutate le seguenti considerazioni su tale indice:</p> <ul style="list-style-type: none"><li>• non indica la comprensibilità del testo, il contenuto delle cui frasi potrebbe essere totalmente non comprensibile, ma avere lo stesso un ottimo indice;</li><li>• nei documenti saranno spesso usati termini tecnici che non si possono sostituire;</li><li>• interrompere frasi a favore di un indice più alto potrebbe storpiarne il contenuto;</li><li>• usare frasi troppo dirette potrebbe risultare poco professionale.</li></ul> <p>Per queste ragioni i documenti saranno sempre valutati da un membro del gruppo, per stabilire se e come il testo possa essere semplificato.</p>

Tabella 3: Metriche per i documenti

- Software

Nome	Codice	Descrizione
Copertura requisiti obbligatori	MPS1	<p>Questa metrica permette di tracciare il numero di requisiti obbligatori coperti dal prodotto software. Viene calcolata in forma percentuale utilizzando la seguente formula:</p> $\text{Copertura req. obbligatori}[\%] = \frac{\# \text{req. obbligatori soddisfatti}}{\# \text{req. obbligatori totali}}$
Copertura requisiti desiderabili	MPS2	<p>Questa metrica permette di tracciare il numero di requisiti desiderabili coperti dal prodotto software. Viene calcolata in forma percentuale utilizzando la seguente formula:</p> $\text{Copertura req. desiderabili}[\%] = \frac{\# \text{req. desiderabili soddisfatti}}{\# \text{req. desiderabili totali}}$
Average Cyclomatic Complexity	MPS3	<p>La complessità ciclomatica è una metrica software che indica la complessità di un programma misurando il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. Valori troppo elevati sono indice di una scarsa manutenibilità del codice, mentre valori troppo bassi potrebbero indicare scarsa efficienza. Viene calcolata da <i>IntelliJ IDEA</i>.</p>
Coupling Between Object Classes	MPS4	<p>Calcola il numero di classi con cui ogni classe è accoppiata. Una classe si dice accoppiata con un'altra se è presente una dipendenza tra le due, indipendentemente dal verso della dipendenza. Un valore elevato di questa metrica non è desiderabile poiché evidenzia una bassa modularità del codice. Viene calcolata da <i>IntelliJ IDEA</i>.</p>
Weighted Method Complexity	MPS5	<p>Rappresenta la somma pesata dei metodi di una classe, dove il peso di un metodo è dato dalla sua complessità ciclomatica. Tale metrica se si attesta su valori bassi garantisce una buona riusabilità dei metodi, mentre valori elevati fanno trasparire una dimensione eccessiva della classe. Viene calcolata da <i>IntelliJ IDEA</i>.</p>
Class Size Attributes	MPS6	<p>Calcola il numero totale di campi dati per ogni classe, escludendo i campi dati statici ereditati. Tale metrica è utile per valutare il grado di manutenibilità e comprensibilità del codice. Viene calcolata da <i>IntelliJ IDEA</i>.</p>

Number of Parameters	MPS7	Calcola il numero di parametri associati ad ogni metodo. Un valore elevato è indice di scarsa manutenibilità del codice. Viene calcolata da <i>IntelliJ IDEA</i> .
Non-comment Lines of Code	MPS8	Calcola il numero di linee di codice per ogni metodo, non conteggiando i commenti e le linee vuote. Un valore elevato di tale metrica può indicare un'eccessiva complessità del metodo. Viene calcolata da <i>IntelliJ IDEA</i> .
Nested Block Depth	MPS9	Rappresenta il massimo numero di livelli di annidamento delle strutture di controllo nei metodi. Un valore elevato di tale indice implica un'alta complessità ed un basso livello di astrazione del codice. Viene calcolata da <i>IntelliJ IDEA</i> .
Copertura del codice	MPS10	Indica in percentuale il codice che è stato testato con esito positivo tramite i test eseguiti. Viene calcolata tramite la seguente formula: $Copertura_{del\ codice}[\%] = \frac{\#test\ passati}{\#totale\ test\ richiesti}$ Un valore elevato di tale metrica diminuisce la probabilità di errori, andando a garantire un livello maggiore di affidabilità del software.

Tabella 4: Metriche per il software

### 3.2.4.1.3 Strumenti

Per il calcolo delle varie metriche vengono utilizzati i seguenti strumenti:

- È stato utilizzato il software IntelliJ IDEA per il calcolo di alcune delle metriche descritte nella sezione 3.2.4.1.2;
- È stato inoltre utilizzato per calcolare l'indice Gulpease uno strumento disponibile al seguente indirizzo:

[http://farfalla-project.org/readability\\_static/](http://farfalla-project.org/readability_static/)  
(ultima consultazione effettuata in data 18-04-2017)

### 3.2.4.2 Analisi

#### 3.2.4.2.1 Analisi statica

L'analisi statica è una tecnica che permette di individuare anomalie all'interno di documenti e codice sorgente durante tutto il loro ciclo di vita. È applicabile tramite due tecniche diverse:

- **Walkthrough:** viene svolta effettuando una lettura a largo spettro. Si tratta di un'attività onerosa e collaborativa che richiede la cooperazione di più persone, essendo una tecnica non efficiente. Verrà utilizzata principalmente durante la prima parte del progetto, quando non tutti i membri del gruppo hanno piena padronanza e conoscenza delle *Norme di Progetto* e del *Piano di Qualifica*. Utilizzando questa tecnica è possibile stilare una lista di controllo contenente gli errori più comuni.
- **Inspection:** viene svolta una lettura mirata e strutturata, volta a localizzare gli errori segnalati nella lista di controllo, con il minor costo possibile. Tramite l'acquisizione di esperienza la lista di controllo viene progressivamente estesa, rendendo l'inspection via via più efficace. Normalmente è effettuata da una persona sola.



- **Lista di controllo per la documentazione**

Anomalia	Descrizione
Comandi $\text{\LaTeX}$ deprecati	Utilizzo di comandi $\text{\LaTeX}$ che sono stati ridefiniti successivamente
Caratteri underscore	Utilizzo di caratteri underscore senza prima anteporre una backslash
Date	Formato delle date errato o non conforme alle <i>Norme di Progetto v3.0.0</i>
Elenchi puntati	Elenchi puntati non conformi alle norme, in particolare i punti spesso non terminano con il punto e virgola
Accenti errati	Avverbi che necessitano di accenti acuti scritti invece con accenti gravi

Tabella 5: Lista anomalie comuni nei documenti

- **Lista di controllo per il codice**

Anomalia	Descrizione
Funzioni non delimitate	Omissione delle parentesi graffe dopo dichiarazioni di funzioni in JavaScript
Statement non delimitati	Omissione di punti e virgola alla fine degli statement in JavaScript
Nomenclatura non rispettata	Le variabili e le funzioni non rispettano lo stile camelCase
Richiamo sbagliato variabili	Utilizzo di variabili già definite ma richiamate con errori di battitura

Tabella 6: Lista anomalie comuni nel codice

#### 3.2.4.2.2 Analisi dinamica

L'analisi dinamica è una tecnica di analisi del prodotto software che richiede la sua esecuzione.

Viene effettuata mediante dei test volti a verificare il funzionamento del prodotto e nel caso in cui vengano riscontrate anomalie ne permette l'identificazione.

I test devono essere ripetibili, cioè deve essere possibile, dato lo stesso input e nello stesso ambiente, risalire allo stesso output. Per ogni test devono dunque essere definiti i seguenti parametri:

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

### 3.2.5 Procedure

#### 3.2.5.1 Controllo qualità di prodotto

Il controllo della qualità del prodotto viene garantito da due tipi di processi, quello di verifica e quello di validazione e dal rispetto delle norme.

- **Verifica:** questo processo si occupa di accertare che l'esecuzione delle attività di processo siano corrette. È un'azione che deve essere svolta su ogni attività o risultato che fa progredire il progetto da una *baseline<sub>G</sub>* a quella successiva, per garantire l'assenza di errori.
- **Validazione:** questo processo viene svolto come azione conclusiva per accertare che il prodotto rispecchi le aspettative utilizzando un metodo sistematico, disciplinato e quantificabile. Tale processo è descritto in maggior dettaglio nella sezione 3.3.
- **Norme:** il prodotto dovrà rispettare le norme e gli strumenti descritti all'interno di questo documento.

#### 3.2.5.2 Controllo qualità di processo

La qualità di un processo viene raffinata attraverso un continuo miglioramento grazie al modello *PDCA<sub>G</sub>* (Plan-Do-Check-Act) che apporta in modo incrementale ulteriore qualità. Ogni attività del modello deve avere una scrupolosa pianificazione ed un'ottima strategia di verifica che, svolta fin dall'inizio, deve permettere una serie di iterazioni all'interno di una determinata attività fino al raggiungimento di un grado di qualità accettabile per poter procedere ad un ulteriore incremento.

Per poter parlare di un possibile miglioramento della qualità di un processo, bisogna quantificare la valutazione della qualità di un processo studiando le sue caratteristiche di interesse. Questo è possibile basandosi sullo standard  $SPICE_G$ .

### 3.2.5.3 Gestione anomalie

Il *Verificatore* deve procedere alla verifica dell'operato, alla risoluzione di ogni  $ticket_G$ . Ciascuna anomalia individuata viene inserita all'interno di un elenco. L'elenco delle anomalie è comunicato al *Responsabile* che si occupa di assegnare un *ticket* per la correzione di esse.

### 3.2.5.4 Gestione modifiche

Qualora durante l'attività di verifica si presenti un problema relativo alla documentazione o al codice prodotto, che necessita di una correzione, verrà inoltrata una richiesta al *Responsabile* in modo tale che possa essere controllata l'effettiva esistenza del problema o meno. Nel caso si verifichi il problema, il *Responsabile* deve assegnare la modifica a chi ritiene più opportuno. Il *Verificatore* constata che la modifica sia avvenuta con successo. La richiesta da presentare al *Responsabile* deve avere la seguente struttura:

- **autore:** il nome e cognome di colui che invia la richiesta di modifica;
- **documento/file:** indica quale documento/file richiede una modifica;
- **motivazione:** spiegare il motivo per cui è richiesta una modifica.

Per tenere traccia dell'intero processo di modifica alla richiesta viene aggiunto se la stessa è stata approvata oppure rifiutata.

### 3.2.5.5 Test

#### 3.2.5.5.1 Test di unità

Il test di unità si pone come obiettivo primario l'isolare dal resto del codice la parte più piccola di software testabile nell'applicazione, chiamata unità, per stabilire se essa funziona esattamente come previsto. Ogni unità deve essere sottoposta a test prima di integrarla in modulo per l'esecuzione del test delle interfacce tra i diversi moduli.

L'approccio più comune al test di unità necessita della scrittura di  $driver_G$  e  $stub_G$ , dove il driver simula un'unità chiamante mentre lo stub simula un'unità chiamata.

#### 3.2.5.5.2 Test di integrazione

Il test di integrazione rappresenta l'estensione logica del test di unità. La forma più semplice di questo tipo di test prevede la combinazione di due unità già sottoposte a test in un unico componente e il test dell'interfaccia presente tra le due. Il concetto alla base in questo approccio consiste nell'esecuzione di test per la combinazione di più parti, espandendo progressivamente il processo al test di moduli di un gruppo con quelli di altri gruppi. L'obiettivo finale è di sottoporre al test tutti i moduli che compongono un processo contemporaneamente.

#### **3.2.5.5.3 Test di sistema**

Il test di sistema sancisce la validazione del prodotto software finale, giunto ad una versione definitiva, e verifica dunque che esso soddisfi in modo completo i requisiti.

#### **3.2.5.5.4 Test di regressione**

Il test di regressione va eseguito ogni volta che viene modificata un'implementazione del sistema. A tale scopo, è necessario eseguire nuovamente i test esistenti sul codice modificato per stabilire se le modifiche apportate hanno alterato elementi precedentemente funzionanti.

#### **3.2.5.6 Codice identificativo**

Ogni Test è strutturato come segue:

- Codice identificativo
- Descrizione
- Stato

Per ciascun test è assegnato un codice identificativo la cui sintassi segue il seguente formalismo:

$$\mathbf{T\{tipo\}\{codice\_identificativo\}}$$

Dove:

- **lettera**: identifica uno dei seguenti tipi di test:
  - **U**: test di unità;
  - **I**: test di integrazione;
  - **S**: test di sistema.
- **codice\_identificativo**: assume i seguenti valori in base al tipo di test:

- **codice\_numerico**: associato ai test di unità e di integrazione, è un codice progressivo che parte da 1;
- **codice\_requisito**: associato ai test di sistema. Identifica il codice univoco associato ad ogni requisito descritto nel documento *Analisi dei Requisiti v3.0.0*.

Inoltre lo Stato del test può assumere i seguenti valori:

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

### 3.3 Validazione

#### 3.3.1 Scopo

Il processo di validazione serve per determinare se il prodotto finale è conforme a quanto è stato pianificato. Tale processo viene effettuato solo dopo che sono state eseguite molteplici verifiche sul prodotto, poiché non è desiderabile ottenere un risultato negativo.

#### 3.3.2 Aspettative

Una corretta implementazione di tale processo permette di individuare:

- una procedura di validazione;
- i criteri per la validazione del prodotto;
- la conformità del prodotto finito.

#### 3.3.3 Descrizione

L'attività di validazione consiste nell'eseguire i test di validazione per poi analizzare i risultati e assicurarsi che il prodotto soddisfi le caratteristiche per cui è stato sviluppato. Per le norme sulla sintassi di tali test si veda la sezione 3.3.5.1.

### 3.3.4 Attività

#### 3.3.4.1 Analisi dinamica

Il *Verificatore* ha il compito di rieseguire tutti i test ponendo attenzione ai risultati, in particolare a quelli ottenuti dai test di validazione. Il *Responsabile* dovrà poi analizzare i risultati ottenuti per decidere se rieseguire i test ulteriormente.

### 3.3.5 Procedure

#### 3.3.5.1 Test

##### 3.3.5.1.1 Test di validazione

Il test di validazione rappresenta il collaudo del prodotto in presenza del proponente. Al superamento di tale collaudo segue il rilascio ufficiale dl prodotto sviluppato.

#### 3.3.5.2 Codice identificativo

Ogni Test è strutturato come segue:

- Codice identificativo
- Descrizione
- Stato

Per ciascun test di validazione è assegnato un codice identificativo la cui sintassi segue il seguente formalismo:

$$\mathbf{TV}\{\mathbf{codice\_requisito}\}$$

Dove:

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

Inoltre lo Stato del test può assumere i seguenti valori:

- implementato;
- non implementato;
- non eseguito;

- superato;
- non superato.

### 3.4 Strumenti

#### 3.4.1 Verifica ortografica

Viene utilizzata la verifica dell'ortografia in tempo reale, strumento integrato in TexStudio che sottolinea in rosso le parole errate secondo la lingua italiana.

#### 3.4.2 Validazione W3C

Per la validazione delle pagine di markup HTML viene utilizzato lo strumento offerto dal  $W3C_G$ , raggiungibile al seguente indirizzo:

<https://validator.w3.org/>

Per la validazione dei fogli di stile  $CSS_G$  viene utilizzato lo strumento offerto dal W3C, raggiungibile al seguente indirizzo:

<https://jigsaw.w3.org/css-validator/>

#### 3.4.3 Analisi statica

Per l'analisi statica del codice JavaScript vengono utilizzati i seguenti strumenti:

- **JSHint**: uno strumento *OpenSource\_G* atto a rilevare gli errori e i potenziali problemi nel codice JavaScript, oltre che ad imporre delle convenzioni di codifica al team. JSHint è accessibile al seguente indirizzo:

<http://www.jshint.com/>

- **Closure Compiler**: uno strumento aggiuntivo a JSHint, che permette di compilare il codice JavaScript e analizzarlo alla ricerca di errori. Closure Compiler è accessibile al seguente indirizzo:

<https://closure-compiler.appspot.com/home>

#### 3.4.4 Analisi dinamica

Per l'esecuzione dei test di analisi dinamica vengono utilizzati i seguenti strumenti:

- **Mocha:** un framework ricco di funzionalità per l'esecuzione di test JavaScript, scritto in *Node.js*; permette l'esecuzione di test asincroni e in serie, consentendo segnalazioni flessibili e accurate. Mocha è raggiungibile al seguente indirizzo:

<http://mochajs.org/>

- **Karma:** un ambiente di testing, utile ad effettuare test su browser e dispositivi. Per descrivere i test vengono utilizzati dei framework esterni, come per esempio Mocha, assieme a cui viene utilizzato per l'esecuzione dei test di unità. Karma è raggiungibile al seguente indirizzo:

<http://karma-runner.github.io/0.13/index.html>



## 4 Processi organizzativi

### 4.1 Gestione

#### 4.1.1 Scopo

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

#### 4.1.2 Aspettative

Le aspettative del processo sono:

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

#### 4.1.3 Descrizione

Orari di lavoro:

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

Viene trattata la gestione dei seguenti argomenti:

- Ruoli di progetto
- Comunicazioni
- Incontri
- Strumenti di coordinamento
- Strumenti di versionamento
- Rischi

#### 4.1.4 Ruoli di progetto

Tutti i ruoli saranno ricoperti da ciascun componente del gruppo in rotazione, in modo che ogni membro possa assumere almeno una volta ciascuno di essi. Nel documento *Piano di Progetto v3.0.0* vengono organizzate e pianificate le attività assegnate ai specifici ruoli previsti nell'attività di progetto, che sono:

#### 4.1.4.1 Amministratore di Progetto

L'*Amministratore* di Progetto controlla e amministra tutto l'ambiente di lavoro con piena responsabilità sulla capacità operativa e sull'efficienza.

Le sue responsabilità sono:

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

#### 4.1.4.2 Responsabile di Progetto

Il *Responsabile* di Progetto è il punto di riferimento sia per il *committente<sub>G</sub>* che per il fornitore. Inoltre, approva le scelte prese dal gruppo e se ne assume la responsabilità.

Le responsabilità di tale ruolo sono:

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

#### 4.1.4.3 Analista

L'*Analista* si occupa dell'analisi dei problemi e del dominio applicativo. La sua presenza non è sempre necessaria durante il progetto.

Le sue responsabilità sono:

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

#### 4.1.4.4 Progettista

Il *Progettista* gestisce gli aspetti tecnologici e tecnici del progetto.

Le sue responsabilità sono:

- effettuare scelte efficienti ed ottimizzate su aspetti tecnici del progetto;
- rendere facilmente mantenibile il progetto.

#### 4.1.4.5 Verificatore

Il *Verificatore* avendo delle solide conoscenze delle normative di progetto garantirà una esaustiva verifica dello esso.

Le responsabilità assunte da tale ruolo sono:

- controllo delle attività di progetto secondo le normative stabilite.

#### 4.1.4.6 Programmatore

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

Le sue responsabilità sono:

- implementare le decisioni del *Progettista*;
- scrittura di un codice pulito e facile da mantenere, che rispetti le *Norme di Progetto*;
- versionamento del codice prodotto;
- realizzazione degli strumenti per la verifica e la validazione del software.

#### 4.1.5 Procedure

##### 4.1.5.1 Gestione delle comunicazioni

###### 4.1.5.1.1 Comunicazioni interne

Le comunicazioni interne sono gestite utilizzando un tool denominato  $Slack_G$ . Questo servizio offre al suo interno una suddivisione dei canali per lo scambio di informazioni tra i componenti del gruppo. Ogni canale è specifico per una determinata attività. All'interno di ogni canale possono essere creati dei  $thread_G$  privati tra due membri del gruppo. Molto più comodo rispetto ad un servizio di semplice messaggistica istantanea come Telegram, Slack offre un ambiente puramente creato per gestire al meglio un gruppo di lavoro.

###### 4.1.5.1.2 Comunicazioni esterne

Le comunicazioni esterne sono affidate al *Responsabile* di Progetto, il quale si avvale di



una apposita casella di posta elettronica:

digitalcookies.group@gmail.com;

Il *Responsabile* di Progetto, se necessario, dovrà tenere informati tutti i componenti del gruppo sulle discussioni con le componenti esterne tramite i canali di comunicazione interna.

#### **4.1.5.2 Gestione degli incontri**

##### **4.1.5.2.1 Incontri Interni**

Il *Responsabile* di progetto ha il compito di organizzare gli incontri interni utilizzando i canali di comunicazione. Il *Responsabile* dovrà essere certo della partecipazione di ogni membro del gruppo, al fine di fissare definitivamente l'evento sulla piattaforma Slack. Ogni componente ha diritto di presentare una richiesta al *Responsabile* di Progetto di organizzazione di un incontro, il quale dovrà decidere se accettare o meno la proposta. Inoltre, il *Responsabile* dovrà incaricare un membro del gruppo a stendere il verbale dell'incontro avvenuto secondo le norme previste.



Figura 6: Organizzazione di un incontro interno

#### 4.1.5.2.2 Incontri Esterni

Il *Responsabile* di Progetto ha il compito di comunicare, ed organizzare gli incontri esterni con il proponente o committente. Come per gli incontri interni, ogni componente del gruppo ha diritto ad una richiesta di organizzazione di un incontro esterno. Il *Responsabile* di Progetto deciderà se organizzare o meno l'incontro una volta accertati i motivi della richiesta. Nel caso decida di procedere a contattare l'entità esterna (proponente o committente), e quest'ultima sia d'accordo, deve comunicare tutte le informazioni riguardanti data, ora e luogo dell'incontro a tutti i componenti del gruppo per mezzo dei canali di comunicazione interni. Inoltre, il *Responsabile* dovrà incaricare un membro del gruppo a stendere il verbale dell'incontro avvenuto secondo le norme previste.



Figura 7: Organizzazione di un incontro esterno

#### 4.1.5.3 Gestione degli strumenti di coordinamento

##### 4.1.5.3.1 Ticketing

Per suddividere il carico di lavoro in  $Task_G$  che saranno divisi tra tutti i componenti del gruppo viene utilizzata la piattaforma  $Wrike_G$ . Questo compito spetta al *Responsabile di Progetto*. Wrike permette di avere un quadro completo delle attività, delle relative scadenze prefissate e dello stato di ogni singolo Task (completo o ancora da svolgere). La procedura per l'assegnazione di un Task segue il seguente schema:

- inserire un titolo al task;
- indicare la/e persona/e a cui è stato assegnato tale compito;
- inserire la data di inizio e di fine di tale compito;
- inserire una descrizione che contiene un breve riassunto del compito assegnato e il ruolo assunto in quella fase del progetto;

- inviare una notifica via email alle persone a cui è stato assegnato il task.

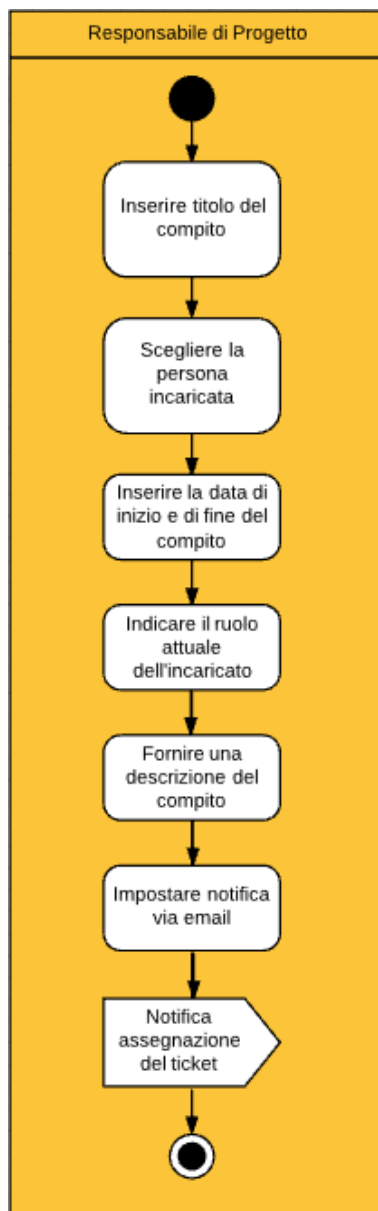


Figura 8: Assegnazione di un ticket

#### 4.1.5.4 Gestione degli strumenti di versionamento

##### 4.1.5.4.1 Repository

Per il versionamento e il salvataggio dei file è previsto l'utilizzo di diversi repository su *GitHub<sub>G</sub>*. L'*Amministratore* di Progetto si deve occupare della creazione dei repository, che rappresentano il gruppo del progetto. Successivamente l'*Amministratore* inserirà tutti i componenti del gruppo, i quali dovranno essere in possesso di un account personale, come collaboratori.

E' previsto l'utilizzo di due repository:

- **Documents:** contiene tutta la documentazione dell'attività di progetto;
- **SWEDesigner:** contiene tutti i file implementativi del progetto.

#### 4.1.5.4.2 Struttura del repository

I membri del gruppo sono tenuti a rispettare le seguenti norme sull'organizzazione dei file nel repository.

Struttura del repository Documents:

- **StyleLatex:** contiene i file del template e il layout dei documenti  $\text{\LaTeX}$ ;
- **Cartella documenti:** contiene due sottocartelle che dividono i documenti interni ed esterni al gruppo. All'interno è presente una suddivisione dei singoli documenti in cartelle dove in ciascuna di esse sono contenuti tutti i relativi file  $\text{\LaTeX}$  i file pdf e una cartella images che contiene le immagini presenti nei documenti;
- **Script:** contiene tutti gli script utilizzati.

La struttura del repository SWEDesigner sarà definita in modo dettagliato nelle successive revisioni.

#### 4.1.5.4.3 Tipi di file e .gitignore

Nelle cartelle contenenti tutti i documenti saranno presenti solamente i file .tex, .pdf, .jpg, .png. Le estensioni dei file generati automaticamente dalla compilazione sono stati aggiunti a .gitignore, e quindi vengono ignorati e resi invisibili a Git.

#### 4.1.5.4.4 Norme sui commit

Ogni volta che vengono effettuate delle modifiche ai file del repository, le quali poi vengono caricate su di esso, bisogna specificarne le motivazioni. Questo avviene utilizzando il comando `commit` accompagnato da un messaggio riassuntivo e una descrizione in cui va specificato:

- la lista dei file coinvolti;
- la lista delle modifiche effettuate, ordinate per ogni singolo file.



Prima di eseguire tale procedura, va aggiornato il diario delle modifiche, secondo le regole viste in 3.1.7.2.

#### 4.1.5.5 Gestione dei rischi

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

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

##### 4.1.5.5.1 Codice identificativo

Ogni rischio è strutturato come segue:

- Codice
- Descrizione
- Probabilità
- Gravità

Per ciascun rischio è assegnato un codice identificativo la cui sintassi segue il seguente formalismo:

$$\mathbf{R\{tipologia\}\{numero\}}$$

Dove:

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

#### 4.1.6 Strumenti

##### 4.1.6.1 Sistema operativo

Il gruppo di progetto lavora sui seguenti sistemi operativi:

- *Ubuntu<sub>G</sub>* 16.04 *LTS<sub>G</sub>* x64
- Windows 10 Pro x64
- Windows 10 Home x64
- MAC OS Sierra 10.12.3 x64
- Manjaro 17.0 x64

##### 4.1.6.2 Slack

Slack è uno strumento di collaborazione utile per le comunicazioni interne ad un gruppo di lavoro. L'applicazione appare come una comune sistema di messaggistica, con in più la possibilità di integrare numerosi servizi tra cui GitHub e Wrike. Gli utenti del team possono suddividere gli argomenti di discussione utilizzando degli *hashtag<sub>G</sub>* per organizzare al meglio la comunicazione. L'applicazione è gratuita ed è necessaria la registrazione di un dominio per il proprio team. Una volta registrati gli utenti possono essere invitati all'interno del gruppo.

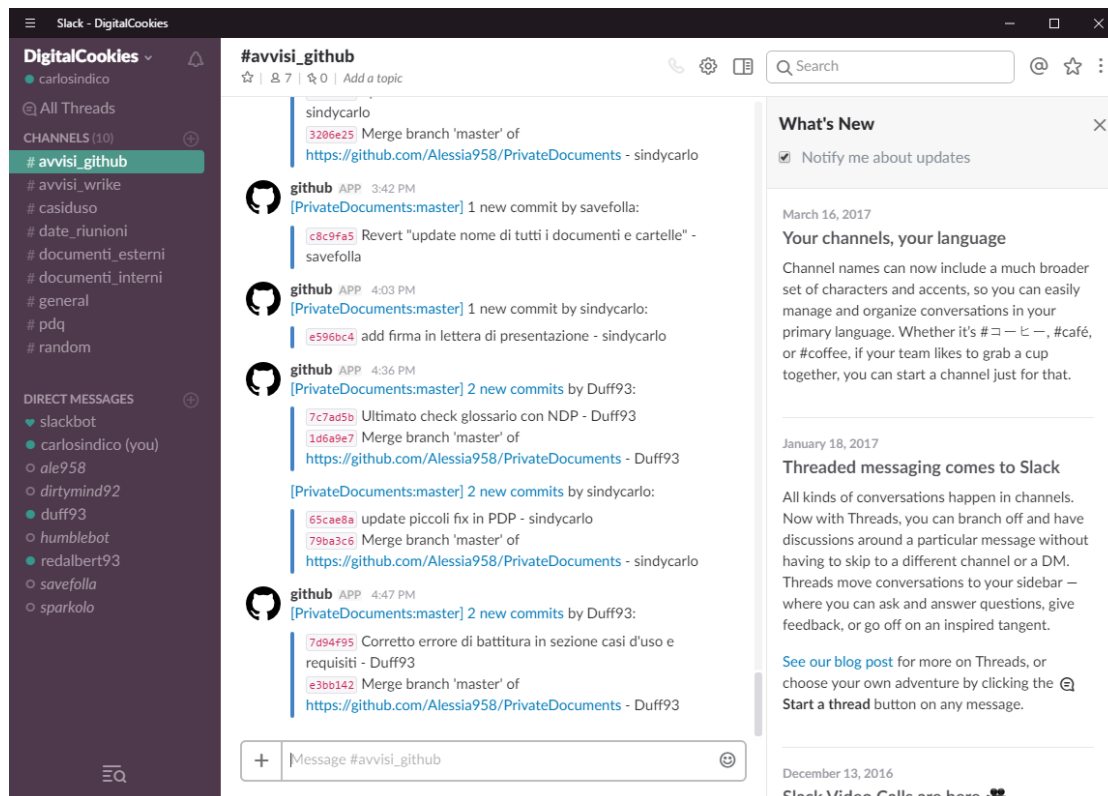


Figura 9: Slack Desktop per Windows

#### 4.1.6.3 Wrike

Wrike è uno strumento online per la collaborazione e il *project management*<sub>G</sub>. Permette ai suoi utenti di modificare progetti, classificare le attività per importanza, tenere traccia dei programmi e collaborare online con altri utenti dello stesso gruppo. Wrike ha diverse funzionalità, tra le più importanti ci sono:

- flusso dei movimenti in tempo reale;
- diagrammi di *Gantt*<sub>G</sub>;
- condivisione di file e modifiche online;
- gestione del carico di lavoro e rilevamento temporale;
- discussioni;
- classificazione delle attività per importanza.

Il programma Student di Wrike consente agli studenti universitari di sfruttare gratuitamente un abbonamento Wrike Professional per 15 utenti.

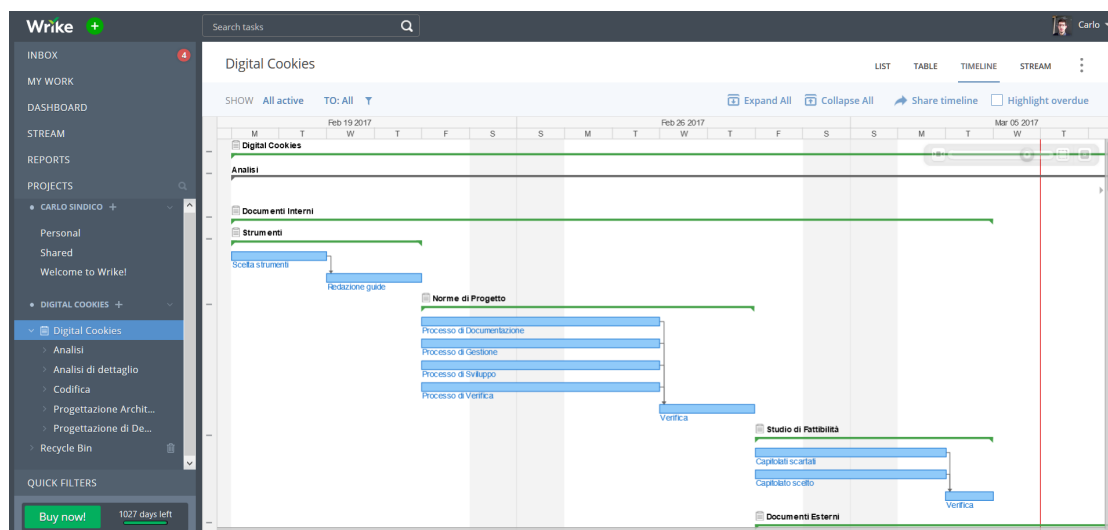


Figura 10: Wrike Web application per Windows

#### 4.1.6.4 Git

Git è un sistema software di controllo di versione distribuito, creato da Linus Torvalds nel 2005. La versione utilizzata è maggiore o uguale alla 2.7.4.

<https://git-scm.com/>

Inoltre è stata redatta una guida Git volta a formare in maniera uniforme i componenti del gruppo sull'utilizzo dello strumento. Per maggiori dettagli sul funzionamento dei comandi si rimanda a *Guida ai comandi Git v1.0.0*.

#### 4.1.6.5 GitHub

GitHub è un servizio web di *hosting<sub>G</sub>* per lo sviluppo di progetti software, che usa il sistema di controllo di versione Git. Può essere utilizzato anche per la condivisione e la modifica di file di testo e documenti revisionabili (sfruttando il sistema di versionamento dei file di Git). GitHub offre diversi piani per repository privati sia a pagamento, sia gratuiti, molto utilizzati per lo sviluppo di progetti OpenSource.