Norme di Progetto

Gruppo MINT — Progetto MaaS

Informazioni sul documento

Versione	1.0.0
Redazione	Navid Taha, Tommaso Zagni
Verifica	Enrico Canova, Fabiano Tavallini
Approvazione	Michael Ogbuachi
Uso	Interno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo MINT

Descrizione

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

1 Registro delle modifiche

Versione	**Data**	**Collaboratori**	**Descrizione**
1.0.0	15-03-2016	Michael Ogbuachi	Approvazione del documento.
0.2.0	15-03-2016	Fabiano Tavallini	Verifica del documento.
0.1.0	14-03-2016	Enrico Canova	Verifica del documento.
0.0.4	13-03-2016	Tommaso Zagni	Completata sezione Processi organizzativi.
0.0.3	13-03-2016	Tommaso Zagni	Completata sezione Processi di supporto.
0.0.2	12-03-2016	Navid Taha	Completata sezione Processi primari.
0.0.1	09-03-2016	Navid Taha	Inizio stesura documento.

Indice

Elenco delle figure

2 Introduzione

2.1 Scopo del documento

Questo documento ha come scopo di definire le regole, gli strumenti e le convenzioni adottate dal gruppo MINTdurante l'intero svolgimento del progetto. Tale documento deve essere visionato da tutti i componenti del gruppo, i quali sono obbligati ad applicare quanto scritto, al fine di mantenere omogeneità e coesione in ogni aspetto del progetto.

Qualora vengano apportate modifiche o aggiunte al presente documento è necessario informare ogni membro del gruppo.

2.2 Ambiguità

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti viene fornito il *Glossario v1.0.0*, contenente la definizione dei termini marcati con una G pedice.

2.3 Riferimenti

2.3.1 Normativi

- **ISO/IEC 12207**:
  https://en.wikipedia.org/wiki/ISO/IEC_12207;

- **Capitolato**:
  http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C4.pdf

2.3.2 Informativi

- **Informazioni sui test di verifica del prodotto software**:
  https://msdn.microsoft.com/it-it/library/aa292197(v=vs.71).aspx
  https://msdn.microsoft.com/it-it/library/aa292167(v=vs.71).aspx
  https://msdn.microsoft.com/it-it/library/aa292128(v=vs.71).aspx

3 Processi primari

3.1 Processo di sviluppo

3.1.1 Scopo del processo

Contiene tutte le attività e i compiti svolti dal gruppo al fine di produrre il prodotto software finale.

3.1.2 Aspettative del processo

Le aspettative della corretta implementazione di tale processo sono:

- realizzare un prodotto finale adeguato a soddisfare i test di *validazione*$_G$ e *verifica*$_G$, rispettando le richieste del *proponente*$_G$;
- fissare gli obbiettivi di *sviluppo*$_G$;
- fissare i vincoli tecnologici e di design.

3.1.3 Descrizione

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

- Analisi dei requisiti
- Progettazione
- Codifica

3.1.4 Analisi dei requisiti

3.1.4.1 Scopo dell'attività

Individuare i requisiti del progetto dalle specifiche del *capitolato*$_G$ e tramite incontri con il proponente. Il risultato di tale attività è essere il documento, redatto dagli *Analisti*, contenente una lista dei casi d'uso e dei requisiti, chiamato Analisi dei Requisiti. Tale documento permette di capire le scelte di progettazione effettuate.

3.1.4.2 Aspettative dell'attività

L'attività si pone come obiettivo la creazione di una documentazione formale contenente tutti i requisiti fissati dai proponenti.

3.1.4.3 Descrizione

Tutti i requisiti analizzati, utilizzando le specifiche del capitolato e consultando i proponenti negli incontri effettuati, vanno specificati nell'*Analisi dei Requisiti v1.0.0*. Per analizzare e trovare i requisiti si utilizza la tecnica dei casi d'uso. Il tracciamento dei requisiti avviene tramite il software *Trender*$_G$.

3.1.4.4 Studio di fattibilità

Il *Responsabile* di Progetto deve organizzare delle riunioni preventive, per permettere lo scambio di opinioni tra i membri del gruppo sui capitolati proposti. Il documento derivante da queste riunioni è lo *Studio di Fattibilità v1.0.0*, il quale viene realizzato dagli *Analisti*. Essi devono descrivere i seguenti punti:

- **Dominio tecnologico e applicativo**: si dà una valutazione prendendo in considerazione la conoscenza attuale delle tecnologie richieste dal capitolato in analisi da parte dei membri del gruppo;
- **Interesse strategico**: si valuta l'interesse strategico del gruppo di progetto in relazione al capitolato in analisi;
- **Individuazione dei rischi**: si analizzano i possibili rischi a cui si può incorrere nel capitolato in analisi.

3.1.4.5 Casi d'Uso

Ogni *caso d'uso*$_G$ è descritto dalla seguente struttura:

- Codice identificativo
- Titolo
- Diagramma *UML*$_G$
- Attori primari
- Attori secondari
- Scopo e descrizione
- Precondizione
- Postcondizione
- Scenario principale
- Scenari alternativi

3.1.4.6 Codice identificativo

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

`UC{X} {Gerarchia}`

Dove:

- **X**: corrisponde all'ambito di riferimento e può assumere i seguenti valori:
  - **SA**: Ambito Super Amministratore;
  - **U**: Ambito Utente.
- **Gerarchia**: identifica la relazione gerarchica che c'è tra i casi d'uso di uno stesso ambito. C'è quindi una struttura gerarchica per ogni ambito dei casi d'uso. La numerazione potrebbe non essere continua nel caso in cui vengano rimossi alcuni degli use case numerati in precedenza.

3.1.4.7 Requisiti

Ogni requisito è strutturato come segue:

- Codice identificativo
- Tipologia
- Descrizione
- Fonti

3.1.4.8 Codice identificativo

Ogni requisito è identificato da un codice, che segue il seguente formalismo:

`R{X}{Y}{Z} {Gerarchia}`

Dove:

- **X**: identifica uno dei seguenti ambiti di sviluppo:
  - **A**: requisito appartenente all'applicazione mobile;
  - **S**: requisito appartenente al servizio web.
- **Y**: identifica uno dei seguenti tipi di requisito:
  - **1**: requisito funzionale;
  - **2**: requisito prestazionale;
  - **3**: requisito qualitativo;
  - **4**: vincolo progettuale.
- **Z**: identifica uno dei seguenti gradi di necessità:
  - **O**: requisito obbligatorio;
  - **F**: requisito facoltativo;
  - **D**: requisito desiderabile.
- **Gerarchia** identifica la relazione gerarchica che c'è tra i requisiti di uno stesso tipo. C'è quindi una struttura gerarchica per ogni tipologia di requisito.

3.1.4.9 UML

I diagrammi devono essere realizzati utilizzando il linguaggio *UML versione 2.0*.

3.1.5 Progettazione

3.1.5.1 Scopo dell'attività

L'attività di progettazione definisce le linee essenziali della struttura del prodotto software in funzione dei requisiti individuati dall'analisi. L'obbiettivo del processo consiste nella stesura dei documenti: *Specifica Tecnica* e *Definizione di Prodotto*.

3.1.5.2 Aspettative dell'attività

Il processo porta alla formazione dei documenti sopra citati, i quali garantiscono affidabilità e coerenza.

3.1.5.3 Descrizione

La progettazione deve rispettare tutti i vincoli e i requisiti concordati tra i componenti del gruppo e i proponenti. I documenti derivati da questa attività sono:

- **Specifica tecnica**: descrive la progettazione ad alto livello relativa all'architettura dell'applicazione e dei singoli componenti. Il documento specifica i diagrammi UML ed i *design pattern*$_G$ utilizzati per realizzare l'architettura definendo inoltre i test necessari alla verifica;
- **Definizione di Prodotto**: descrive in dettaglio la progettazione di *sistema*$_G$, integrando quanto scritto nella *Specifica Tecnica*. Il documento specifica i diagrammi UML e le definizioni delle classi definendo inoltre i test necessari alla verifica.

3.1.5.4 Specifica Tecnica

- **Diagrammi UML**:
  - Diagrammi delle classi
  - Diagrammi dei *package*$_G$
  - Diagrammi di attività
  - Diagrammi di sequenza
- **Design pattern**:
  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 essere riferito al componente che lo soddisfa. Nella sezione 3.1.7.1 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, oltre a misurare il progresso dell'attività di progettazione.

- **Test di integrazione**:
  Devono essere definite delle classi di verifica, utili a verificare che ogni componente del sistema funzioni nella maniera appropriata.

3.1.5.5 Definizione di Prodotto

- **Diagrammi UML**:
  - Diagrammi delle classi
  - Diagrammi di attività
  - Diagrammi di sequenza
- **Definizioni delle classi**:
  Ogni classe progettata deve essere descritta in modo da spiegarne lo scopo e definirne le funzionalità ad essa associate.
- **Tracciamento delle classi**:
  Ogni requisito deve essere tracciato, in modo da poter risalire alle classi ad esso associate. Nella sezione 3.1.7.1 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, oltre a misurare il progresso dell'attività di progettazione.
- **Test di unità**:
  Devono essere definiti dei test di $unità_G$ utili a verificare che le componenti del sistema funzionino nel modo previsto.

3.1.6 Codifica

3.1.6.1 Scopo dell'attività

Lo scopo dell'attività è l'implementazione, detta anche sviluppo o codifica del prodotto software. Questa è la fase di realizzazione del software, che concretizza la soluzione attraverso la programmazione, ovvero la stesura di programmi.

3.1.6.2 Aspettative dell'attività

L'aspettativa dell'attività è un prodotto software stabile, affidabile, funzionale, che soddisfa i requisiti accordati con i proponenti.

3.1.6.3 Descrizione

L'attività di codifica deve rispettare i compiti assegnati nel documento *Piano di Progetto v1.0.0* e utilizzare gli strumenti indicati in questo documento.

3.1.6.4 Stile di codifica

E' richiesto che tutti i membri del gruppo utilizzino le stesso stile di codifica, così che tutto il codice del progetto risulti essere uniforme. Sono previste le seguenti norme stilistiche:

- **Indentazione**: è richiesto l'utilizzo di esattamente 4 spazi;
- **Parentesi dei costrutti**: è richiesto di inserire le parentesi di delimitazione dei costrutti al di sotto di essi e non in linea;
- **Nomi**: i nomi di variabili, metodi e funzioni devono avere la prima lettera minuscola e le successive iniziali delle parole, che ne compongono il nome, in maiuscolo. I nomi delle classi devono avere la prima lettera maiuscola. Tutti i nomi devono essere scritti in inglese.

3.1.6.5 Versionamento

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

`X.Y`

- **X**: è l'indice di versione principale, un incremento di tale indice rappresenta un avanzamento della versione stabile, di conseguenza il valore dell'indice Y deve 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.

3.1.6.6 Ricorsione

La ricorsione va evitata ove possibile. Per ogni funzione ricorsiva è richiesta la prova di terminazione e l'analisi del costo in termini di memoria. Nel caso in cui la memoria utilizzata risulti eccessiva, la ricorsione deve essere rimossa.

3.1.7 Strumenti

Di seguito sono elencati gli strumenti utilizzati dal team nel progetto.

3.1.7.1 Trender

Il gruppo si avvale dell'applicativo web Trender per gestire in maniera veloce e automatizzata tutti i dati ricavati dall'analisi dei requisiti.
Trender è accessibile tramite il seguente link:

`http://mintswe.890m.com/`

Ogni componente del gruppo può accedervi, utilizzando l'account comune predisposto dall'Amministratore di Progetto. Il *database*$_G$ viene gestito tramite *MySQL*$_G$, mentre il resto del sito utilizza *JavaScript*$_G$ e *PHP*$_G$.
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
- Tracciamento delle voci del glossario
- Possibilità di stampare direttamente in codice LaTeX$_G$ quanto archiviato nei punti precedenti

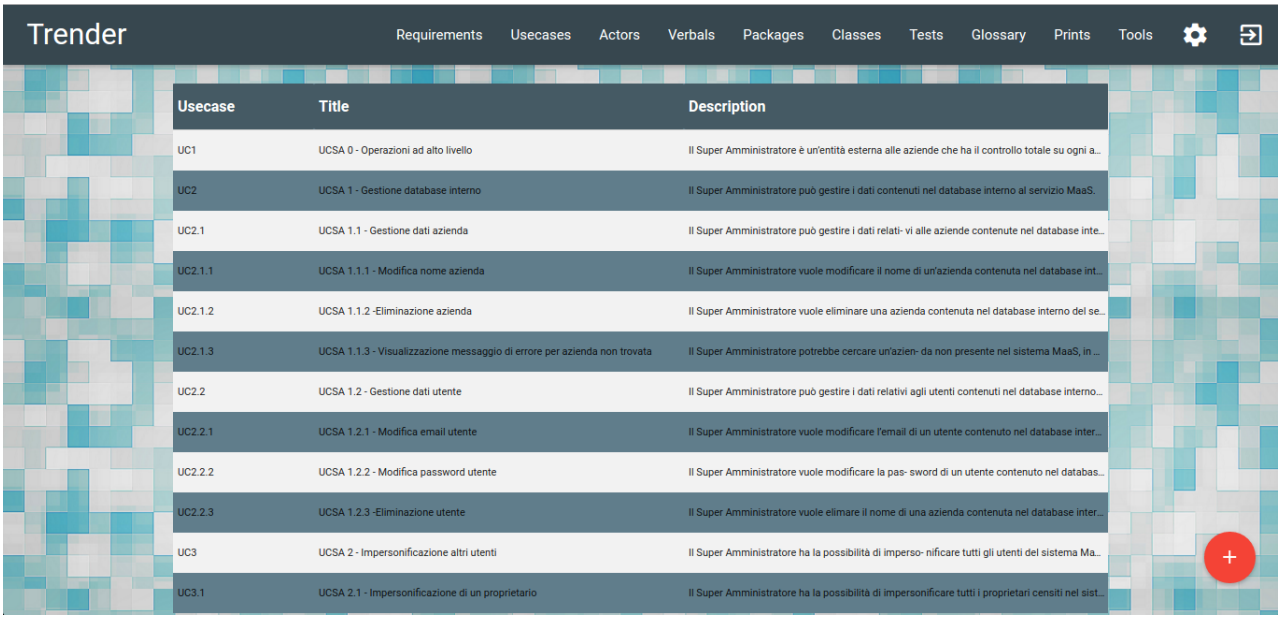

Figura 1: Trender - Casi d'uso

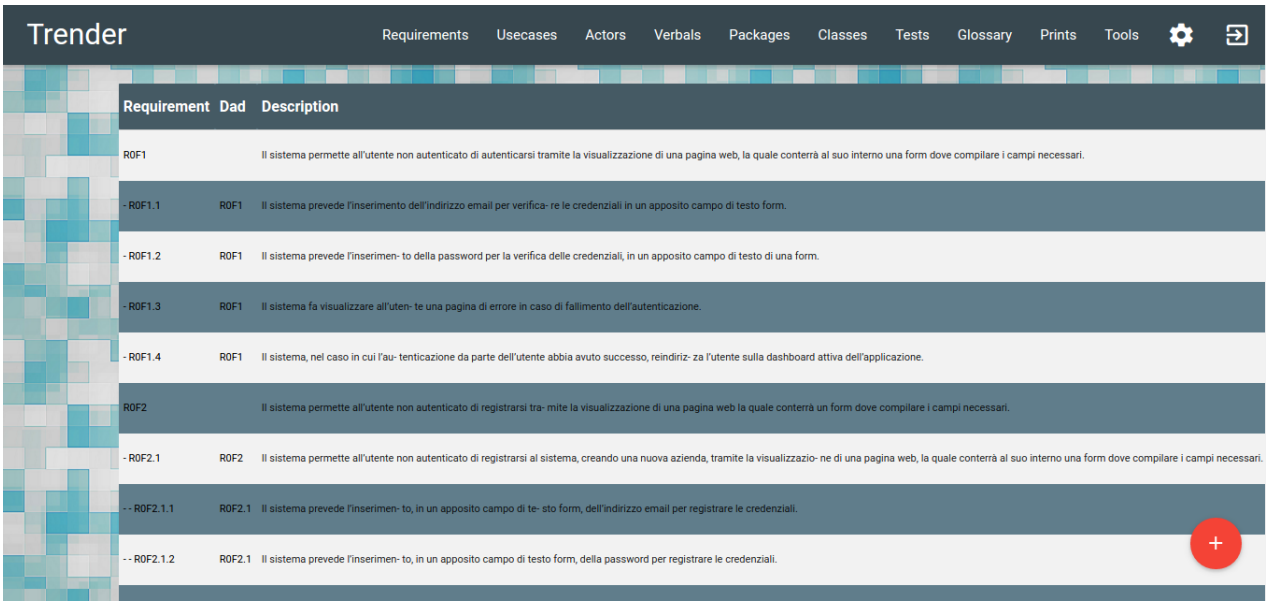

Figura 2: Trender - Requisiti

3.1.7.2 Cloud9

3.1.7.3 Visual Paradigm

Per la produzione dei diagrammi UML viene utilizzato Visual Paradigm Community Edition in versione 13.0, in quanto offre molte agevolazioni per la produzione veloce dei diagrammi e risulta essere semplice da usare.

https://www.visual-paradigm.com/download/community.jsp

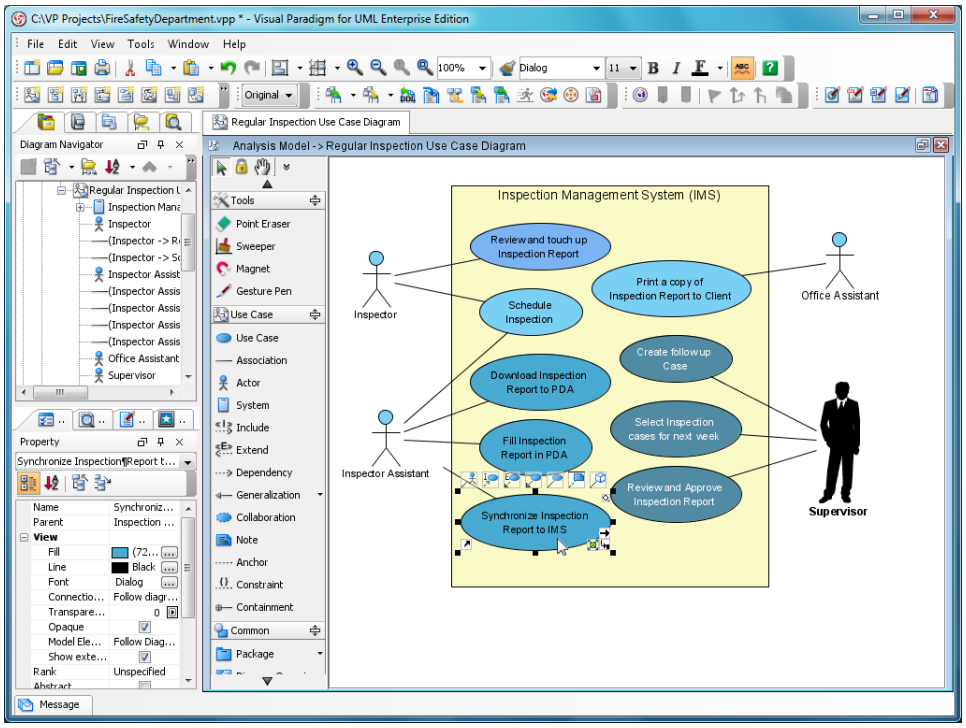

Figura 3: Visual Paradigm

3.1.7.4 Cloud9

Il cloud IDE_G Cloud9 viene utilizzato per la codifica in JavaScript, nello specifico in $Node.js_G$ e $AngularJS_G$. Questo IDE offre un vero e proprio workspace Ubuntu, oltre ad essere un potente editor. Cloud9 è accessibile direttamente dal web, come $SaaS_G$.

https://c9.io/

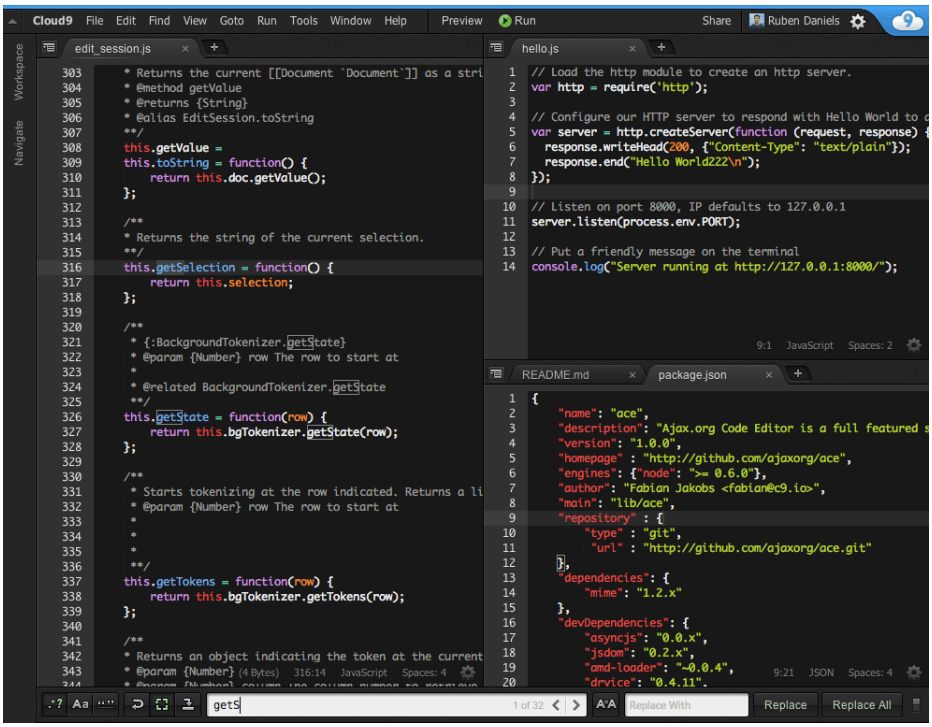

Figura 4: Cloud9

4 Processi di supporto

4.1 Processo di documentazione

4.1.1 Scopo del processo

Lo scopo di questo processo consiste nell'illustrazione di come deve essere redatta e mantenuta la documentazione, durante il ciclo di vita del software.

4.1.2 Aspettative del processo

Le aspettative della corretta implementazione di tale processo sono:

- una chiara visione della documentazione, che deve essere prodotta durante il ciclo di vita del software;
- una serie di norme per la stesura di documenti coerenti e validi;
- una documentazione formale e coerente.

4.1.3 Descrizione

In questo documento devono essere redatte tutte le norme e le convenzioni adottate dal gruppo, in modo da produrre una documentazione valida e coerente.

4.1.4 Procedure

Per la stesura della documentazione si è utilizzato il linguaggio LaTeX, si veda 4.1.11.

4.1.4.1 Approvazione dei documenti

Una volta completata la stesura di un documento non formale, esso viene sottoposto al *Responsabile* di progetto, il quale avrà la responsabilità di incaricare i *Verificatori* a controllare il documento. Nel caso in cui i *Verificatori* trovino degli errori, devono informare il *Responsabile* di progetto, in modo che esso incarichi il redattore del documento a sistemare tali errori. Questo ciclo va eseguito, finché non si ritiene corretto il documento. Nel caso in cui il documento sia considerato corretto dai *Verificatori*, viene sottoposto al *Responsabile* di progetto , che decide la sua approvazione. Se il documento viene approvato, allora è da considerarsi come un documento formale, altrimenti il *Responsabile* di progetto deve comunicare la motivazione e le modifiche, per cui il documento non è stato approvato.

4.1.5 Template

E' stato creato un template LaTeX, che permette di creare con facilità i documenti con la stessa struttura grafica e lo stesso stile di formattazione. Questo permette ai componenti del gruppo di concentrarsi solo nella stesura del contenuto e non sull'aspetto.

4.1.6 Struttura dei documenti

4.1.6.1 Prima pagina

La struttura della prima pagina consiste in:

- **Logo del gruppo**: visibile come primo elemento centrato orizzontalmente in alto;
- **Titolo**: nome del documento visibile subito dopo il logo, centrato orizzontalmente;
- **Tabella descrittiva**: visibile subito dopo il titolo, centrata orizzontalmente e contenente le seguenti informazioni relative al documento:
  - Versione
  - I redattori
  - I verificatori
  - Il responsabile
  - Il tipo di uso
  - La lista di distribuzione

- **Descrizione**: centrata orizzontalmente ed il più possibile sintetica.

4.1.6.2 Registro delle modifiche

Ogni documento formale deve contenere questo registro. Esso è composto da una tabella che contiene le modifiche apportate al documento stesso indicando per ognuna:

- Versione del documento;
- Data della modifica;
- Nome e cognome delle persone coinvolte nella modifica e il ruolo che ricoprono;
- Descrizione concisa della modifica apportata.

4.1.6.3 Indice

Ciascun documento formale deve avere il suo indice, in modo da agevolare la consultazione e permettere una lettura *ipertestuale*$_G$ e non necessariamente sequenziale. Ciascun indice deve essere numerato a partire da 1, per ciascuna sottosezione deve esserci un punto di separazione dalla sezione padre e la numerazione deve ripartire di volta in volta. Per quanto riguarda le appendici, esse non devono essere numerate ma indicate da una lettera maiuscola che verrà incrementata a partire dalla lettera A seguendo l'ordine alfabetico internazionale.

4.1.6.4 Contenuto principale

Ciascuna pagina deve rispettare tutti i margini orizzontali e verticali previsti dal template. Ad eccezione della prima, tutte devono contenere un'intestazione ed un piè di pagina. Ogni intestazione deve essere formattata nel modo seguente:

- Logo di intestazione del gruppo disposto a sinistra;
- Nome del gruppo affiancato al logo;
- Titolo del progetto corrente posizionato subito sotto il nome del gruppo;
- Numero e titolo della sezione corrente del documento disposto a destra dell'intestazione.

Il piè di pagina è strutturato invece nel seguente modo:

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

4.1.6.5 Note a piè di pagina

Per ciascuna pagina interna se dovessero comparire delle note da esplicare esse vanno indicate in basso a sinistra della pagina corrente, riportate con il loro numero e la loro descrizione.

4.1.7 Versionamento

Ciascun documento deve essere versionato, in modo che chiunque lo utilizzi possa avere una visione specifica della sua storia e delle sue modifiche. Ad ogni versione deve corrispondere una riga nel registro delle modifiche.

Verrà applicato il seguente formalismo:

`v{X}.{Y}.{Z}`

dove:

- **X**:
  - Inizia da 0;
  - Viene incrementato quando il Responsabile di Progetto approva il documento;
  - E' limitato superiormente al numero di revisioni.
- **Y**:
  - Inizia da 0;
  - Viene incrementato da parte del Verificatore 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.

4.1.8 Norme tipografiche

4.1.8.1 Stile del testo

- **Glossario**: Ogni parola di glossario deve essere marcata con una G maiuscola a pedice:

  $$repository_G$$

  Per ogni parola ambigua viene segnalata solo la prima occorrenza presente nel documento.
- **Grassetto**: viene applicato ai titoli e agli elementi di un elenco puntato che riassumono il contenuto del paragrafo;
- **Corsivo**: Il corsivo dev'essere utilizzato nei seguenti casi:
  - Citazioni;
  - Abbreviazioni;
  - Parole inserite nel glossario;

  - Riferimenti ad altri documenti;
  - Parole particolari solitamente poco usate o conosciute;
  - Nomi di società o aziende;
  - Ruoli del progetto.

- **Monospace**: i caratteri monospace devono riferirsi a nomi di file e a codice di programmazione:

```
std::cout « "Hello world";
```

- **Maiuscolo**: le parole scritte interamente in maiuscolo dovranno riferirsi soltanto ad acronimi.

4.1.8.2 Elenchi puntati

Tutti gli elenchi puntati sono rappresentati graficamente da un *pallino* nel primo livello, da un *trattino* nel secondo e da un asterisco nel terzo. Ogni elenco puntato permette di esprimere un concetto in modo sintetico preferendolo all'uso di frasi lunghe e discorsive. Ogni elemento di un elenco deve terminare con il punto e virgola, a meno che non sia l'ultimo elemento dell'elenco, in quel caso la frase va terminata con un punto. Fanno eccezione concetti relativamente corti dove questa regola non viene applicata per enfatizzarne il significato.

4.1.8.3 Formati comuni

Per i seguenti concetti vengono espressi i rispettivi formalismi:

- **Date**:

```
GG-MM-AAAA
```

  - **GG**: rappresenta il giorno utilizzando due cifre;
  - **MM**: rappresenta il mese utilizzando due cifre;
  - **AAAA**: rappresenta l'anno 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 con l'iniziale maiuscola e con lo stile corsivo;
  - **Nomi dei documenti**: ogni nome di documento viene scritto con lo stile corsivo e con l'iniziale di ogni parola maiuscola;
  - **Nomi dei file**: ogni nome di file viene scritto con lo stile monospace;

– **Nomi propri**: ogni nome proprio di persone deve essere scritto con il Nome seguito dal Cognome.

4.1.8.4 Sigle

E' 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**: Specifica tecnica
- **MU**: Manuale $utente_G$
- **DP**: Definizione di prodotto
- **RR**: Revisione dei requisiti
- **RP**: Revisione di progettazione
- **RQ**: Revisione di qualifica
- **RA**: Revisione di accettazione

4.1.9 Elementi grafici

4.1.9.1 Tabelle

Ciascuna tabella deve essere centrata orizzontalmente e deve contenere sotto di essa la propria didascalia, per agevolarne il tracciamento, dove deve comparire il numero, incrementale in tutto il documento, della tabella e una breve descrizione del suo contenuto.

4.1.9.2 Immagini

Ogni immagine deve essere centrata orizzontalmente ed avere una larghezza fissa. Inoltre deve essere nettamente separata dai paragrafi che la seguono e la precedono, in modo da definire un netto distacco tra testo e grafica e migliorare conseguentemente la leggibilità. Essa dev'essere accompagnata da una didascalia analoga a quella descritta per le tabelle. Tutti i diagrammi UML vengono inseriti nel documento sotto forma di immagine.

4.1.10 Classificazione dei documenti

4.1.10.1 Documenti informali

Tutti i documenti sono da ritenersi informali fino all'approvazione da parte del *Responsabile* di Progetto, ed in quanto tali sono da considerarsi esclusivamente ad uso interno.

4.1.10.2 Documenti formali

Un documento viene definito formale quando viene 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 passato la verifica e la validazione.

4.1.10.3 Verbali

Il verbale è un documento redatto da un segretario in occasione di incontri con i proponenti o altre entità esterne. Viene redatto una prima volta e non subisce successive modifiche, pertanto non é previsto $versionamento_G$.
Il verbale dovrà essere approvato dal *Responsabile* di Progetto. Ogni verbale dovrà 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.

È presente un primo paragrafo "Informazioni Generali", in cui verranno elencate le informazioni sopra descritte e gli argomenti trattati durante l'incontro. Segue nel secondo paragrafo, "Domande e risposte", la trascrizione delle domande poste al proponente e le relative risposte.

4.1.11 Strumenti

4.1.11.1 LaTeX

Per la stesura della documentazione si è utilizzato il linguaggio LaTeX. Le motivazioni che hanno portato a questa scelta derivano dalle possibilità da esso offre:

- creare documenti formali, divisi in sezioni molto velocemente;
- possibilità di separare contenuto e formattazione, definendo l'aspetto delle pagine in un file template separato e condiviso da tutti i documenti;
- gestione automatica degli indici e dei glossari;
- personalizzare profondamente il documento, grazie ad un elevato numero di librerie.

4.1.11.2 TexMaker

Per la stesura del codice LaTeX si è utilizzato l'editor TeXMaker di versione almeno 4.5. Questo strumento oltre ad integrare un compilatore e visualizzatore PDF, fornisce suggerimenti per il completamento dei comandi LaTeX.

http://www.xm1math.net/texmaker/

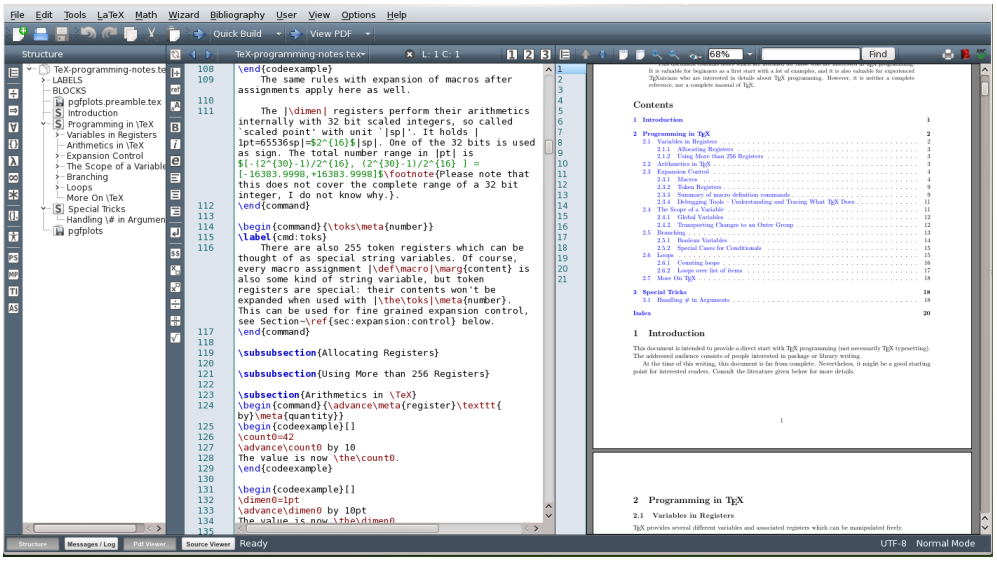

Figura 5: Texmaker

4.1.11.3 Lucidchart

Per la produzione dei diagrammi a scopo illustrativo dei documenti, viene utilizzata la piattaforma Lucidchart. Essendo fruibile direttamente dal web è uno strumento versatile e portabile.

https://www.lucidchart.com/

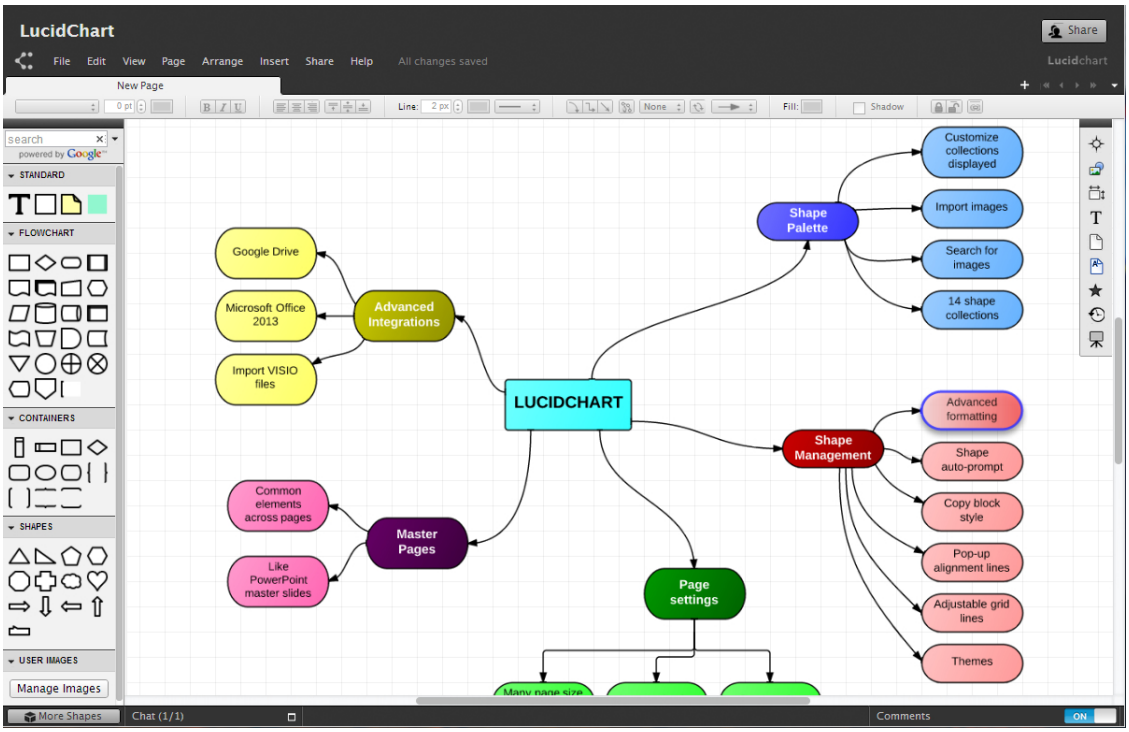

Figura 6: Lucidchart

4.2 Processo di verifica

4.2.1 Scopo del processo

Si occupa di accertare che l'esecuzione delle attività dei processi svolti nella fase in esame non introduca errori nel prodotto.

4.2.2 Aspettative del processo

Una corretta implementazione di tale processo permette di individuare:

- una procedura di verifica;
- i criteri per la verifica del prodotto;
- i difetti, i quali vengono catalogati per essere corretti.

4.2.3 Descrizione

Il processo si suddivide in due attività:

- **Analisi**: viene effettuata tramite due tecniche, l'analisi statica e l'analisi dinamica. Consiste nell'analisi del codice sorgente e la sua successiva esecuzione;

- **Test**: definisce tutti i test che vengono eseguiti sul prodotto software.

4.2.4 Analisi

4.2.4.1 Analisi statica

E' una tecnica di analisi del codice sorgente e della documentazione associata, prevalentemente usata quando il sistema non è ancora disponibile e durante tutto l'arco del suo sviluppo. Non richiede l'esecuzione del prodotto software in alcuna sua parte. Può essere applicata tramite una delle seguenti strategie:

- **Walkthrough**: Si svolge effettuando una lettura a largo spettro. È un'attività onerosa e collaborativa che richiede la cooperazione di più persone. Si tratta di una tecnica non efficiente. Ne è previsto l'utilizzo 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*. L'utilizzo di questa tecnica rende possibile stilare una lista di controllo, dove inserire gli errori più comuni.
- **Inspection**: Si tratta di una lettura mirata e strutturata, volta a localizzare l'errore con il minor costo possibile. Durante tale lettura si cercano gli errori segnalati nella lista di controllo. Progressivamente tramite l'acquisizione di esperienza la lista di controllo viene estesa, rendendo l'inspection sempre più efficacie. Normalmente viene effettuata da una sola persona.

4.2.4.2 Analisi dinamica

E' una tecnica di analisi del prodotto software, che richiede l'esecuzione del programma. Viene effettuata tramite dei test, che verificano il funzionamento del prodotto e in caso di 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. Quindi per ogni test deve essere definito:

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

4.2.5 Test

4.2.5.1 Test di unità

L'obiettivo primario del test di unità consiste nell'isolare la parte più piccola di software testabile nell'applicazione, chiamata unità, dal resto del codice per stabilire se funziona esattamente come

previsto. Ogni singola unità viene sottoposta a test prima di essere integrata in moduli per l'esecuzione del test delle interfacce tra i diversi moduli. Per il più comune approccio di test di unità è necessaria la scrittura di *driver*$_G$ e *stub*$_G$. Il driver simula un'unità chiamante mentre lo stub simula un'unità chiamata.

4.2.5.2 Test di integrazione

Il test di integrazione rappresenta l'estensione logica del test di unità. Il più semplice di questo tipo di test consiste nella combinazione di due unità, già sottoposte a test, in un solo componente e nel test dell'interfaccia presente tra le due. Il concetto che è alla base di questo approccio consiste nell'esecuzione del test delle combinazioni di parti ed, eventualmente, nell'espansione del processo al test dei moduli di un gruppo con quelli di altri gruppi. Alla fine, tutti i moduli che compongono un processo vengono sottoposti al test contemporaneamente.

4.2.5.3 Test di sistema

Il test di sistema rappresenta la validazione del prodotto software finale, ovvero quando lo si ritiene giunto ad una versione definitiva. Viene quindi verificato il completo soddisfacimento dei requisiti da parte del prodotto.

4.2.5.4 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.

4.2.5.5 Test di accettazione

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

4.2.6 Strumenti

4.2.6.1 Verifica ortografia

Viene utilizzata la verifica in tempo reale dell'ortografia, integrata in TexMaker. Essa marca, sottolineando in rosso, le parole errate secondo la lingua italiana.

4.2.6.2 Validazione W3C

Per la validazione delle pagine di markup *HTML*$_G$ 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/

4.2.6.3 Teamwork

Per l'assegnazione e il tracciamento della correzione degli errori viene utilizzato lo strumento Teamwork, descritto in maniera approfondita nella sezione 5.1.10.3.

4.2.6.4 Analisi statica

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

- **JSHint**: è uno strumento open source utile 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**: è uno strumento aggiuntivo a JSHint, che viene utilizzato per compilare il codice JavaScript e analizzare il codice alla ricerca di errori. Closure Compiler è accessibile al seguente indirizzo:

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

4.2.6.5 Analisi dinamica

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

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

  http://mochajs.org/

- **Karma**: è 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. Karma viene utilizzato in concomitanza con Mocha per l'esecuzione dei test di unità. Karma è raggiungibile al seguente indirizzo:

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

- **Protractor**: è un framework che permette l'esecuzione di *test end-to-end*$_G$ per applicazioni scritte in AngularJS. Protractor testa l'esecuzione dell'applicazione in un ambiente reale di utilizzo, quindi su un browser e come se l'esecuzione avvenisse da parte dell'utente. Protractor è accessibile al seguente indirizzo:

  http://angular.github.io/protractor/#/

4.2.6.6 Metriche

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

- **JSMeter**: è uno strumento che permette di calcolare diversi indici di misurazione della complessità del codice JavaScript, descritti nel *Piano di Qualifica v1.0.0*. Viene utilizzato per calcolare volume e potenziale di $Halstead_G$ per funzione, l'indice di manutenibilità. JSMeter è accessibile al seguente indirizzo:

  `http://jsmeter.info/`

- **Istanbul**: è uno strumento che permette di calcolare alcune metriche di complessità del codice JavaScript descritte nel *Piano di Qualifica v1.0.0*:
  - Statement coverage;
  - Branch coverage.

  Instanbul è raggiungibile al seguente indirizzo:

  `http://gotwarlost.github.io/istanbul/`

5 Processi organizzativi

5.1 Processo di gestione

5.1.1 Scopo del processo

Lo scopo di questo processo è la produzione di un documento chiamato Piano di Progetto, utile ai membri del gruppo per pianificare e gestire i ruoli di ognuno di essi.

5.1.2 Aspettative del processo

Le aspettative di tale processo sono:

- Produzione del Piano di Progetto;
- Definire i ruoli dei componenti del gruppo;
- Definire il piano per l'esecuzione dei compiti programmati.

5.1.3 Descrizione

Vengono stabiliti i seguenti orari di lavoro:

- Dalle 9.00 alle 12.00;
- Dalle 13.00 alle 17.00.

Viene trattata la gestione dei seguenti argomenti:

- Ruoli di progetto

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

5.1.4 Ruoli di progetto

Lo sviluppo del progetto prevede diversi ruoli che ogni membro del gruppo è tenuto a ricoprire almeno una volta. Nel *Piano di Progetto v1.0.0* vengono pianificate le attività assegnate ai specifici ruoli previsti nell'attività di progetto, che sono:

5.1.4.1 Amministratore di Progetto

Il compito principale dell'*Amministratore* di Progetto è controllare ed amministrare l'ambiente di lavoro, avendone la diretta responsabilità sull'efficienza e sulla capacità operativa.
Le responsabilità assunte da tale ruolo sono:

- studio e ricerca di strumenti che migliorino il più possibile l'ambiente di lavoro, riducendo il carico di lavoro umano ed automatizzando ove possibile;
- controllare versioni e configurazioni del prodotto software;
- garantire un controllo della *qualità*$_G$ sul prodotto, fornendo procedure e strumenti di monitoraggio o segnalazione;
- risolvere i problemi legati alle difficoltà di gestione dei processi e delle risorse disponibili;
- gestire il versionamento e l'archiviazione della documentazione di progetto.

5.1.4.2 Responsabile di Progetto

Il *Responsabile* di Progetto rappresenta il punto di riferimento dell'intero gruppo di lavoro, sia da parte del *committente*$_G$ che da parte del fornitore. Inoltre, approva le scelte prese dal gruppo e se ne assume la responsabilità.
Le responsabilità assunte da tale ruolo sono:

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

5.1.4.3 Analista

L'*Analista* ha il compito di effettuare studi e ricerche approfondite, in modo da apprendere in maniera esaustiva il dominio del problema. Non è necessaria la sua presenza durante l'intera durata del progetto.
Le responsabilità assunte da tale ruolo sono:

- comprensione della natura del problema e della sua complessità ;
- produrre lo *Studio di Fattibilità* e l'*Analisi dei Requisiti*, delineando specifiche il più possibile comprensibili, sia dal proponente che dal committente.

5.1.4.4 Progettista

Il *Progettista* deve avere profonde conoscenze delle tecnologie utilizzate e competenze tecniche aggiornate, in modo da gestire gli aspetti tecnici e tecnologici del progetto.
Le responsabilità assunte da tale ruolo sono:

- effettuare scelte su aspetti tecnici del progetto, che siano il più possibile efficienti ed ottimizzate;
- effettuare scelte su aspetti tecnici, che rendano il prodotto facilmente mantenibile.

5.1.4.5 Verificatore

Il *Verificatore* deve avere un'ampia conoscenza delle normative del progetto, in modo da garantire una profonda verifica di esso.
Le responsabilità assunte da tale ruolo sono:

- controllare che le attività di progetto siano svolte in conformità alle norme stabilite.

5.1.4.6 Programmatore

Il *Programmatore* è responsabile delle attività di codifica e di creazione delle componenti di supporto, utili a effettuare le prove di verifica e validazione sul prodotto software.
Le responsabilità assunte da tale ruolo sono:

- implementare le soluzioni previste dal *Progettista*;
- scrivere codice pulito e conforme alle norme di progetto, in modo da essere facilmente mantenibile;
- versionare il codice prodotto;
- realizzare gli strumenti utili per poter compiere le prove di verifica e validazione previste.

5.1.5 Gestione delle comunicazioni

5.1.5.1 Comunicazioni interne

Le comunicazioni interne vengono gestite utilizzando un gruppo Telegram, denominato "MINT team", al quale hanno accesso solo i membri del gruppo di progetto. In caso sia necessario effettuare delle videoconferenze, verrà utilizzata l'applicazione Skype.

5.1.5.2 Comunicazioni esterne

E' compito del *Responsabile* di Progetto mantenere i contatti con le componenti esterne al gruppo. Per compiere ciò, si deve avvalere di un'apposita casella di posta elettronica:

mint.swe.unipd@gmail.com

Il *Responsabile* di Progetto, se necessario, dovrà informare gli altri membri del gruppo su quanto discusso con le componenti esterne.

5.1.6 Gestione degli incontri

5.1.6.1 Incontri interni

Il *Responsabile* di Progetto ha il compito di organizzare gli incontri interni, dapprima utilizzando le applicazioni aderenti alle norme (espresse nelle *Comunicazioni interne*, vedi 5.1.5.1), in modo da essere sicuro della presenza di ogni membro, successivamente dovrà fissare l'evento sul calendario della piattaforma di cooperazione e pianificazione dei $task_G$ del gruppo, Teamwork.
Ogni membro del gruppo può richiedere al *Responsabile* di Progetto di organizzare un incontro interno. Il *Responsabile*, una volta accertati i motivi di tale richiesta, dovrà decidere se procedere ed organizzare l'incontro oppure rifiutare.

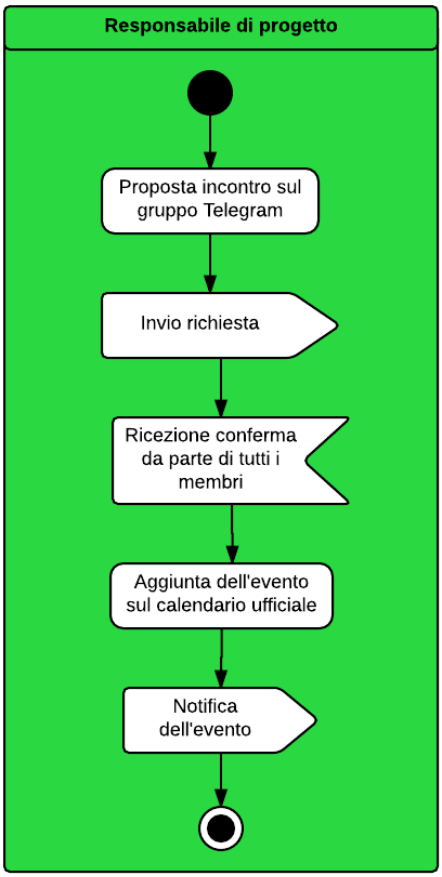

Figura 7: Procedura per l'organizzazione di un incontro interno

5.1.6.2 Incontri esterni

Il *Responsabile* di Progetto ha il compito di comunicare, seguendo le norme previste da questo documento (espresse nelle *Comunicazioni esterne* 5.1.5.2), ed organizzare gli incontri esterni con il proponenti o con i committenti. Ogni membro del gruppo può richiedere al *Responsabile* di Progetto di organizzare un incontro esterno. Egli, una volta accertati i motivi di tale richiesta, dovrà decidere se procedere a richiedere ed organizzare l'incontro oppure rifiutare la richiesta. Nel caso decida di procedere a contattare l'entità esterna, e quest'ultima sia d'accordo, deve comunicare tutte le informazioni riguardanti data, ora e luogo dell'incontro al resto del gruppo. Inoltre, il *Responsabile* dovrà incaricare un membro del gruppo a stendere il verbale dell'incontro avvenuto.

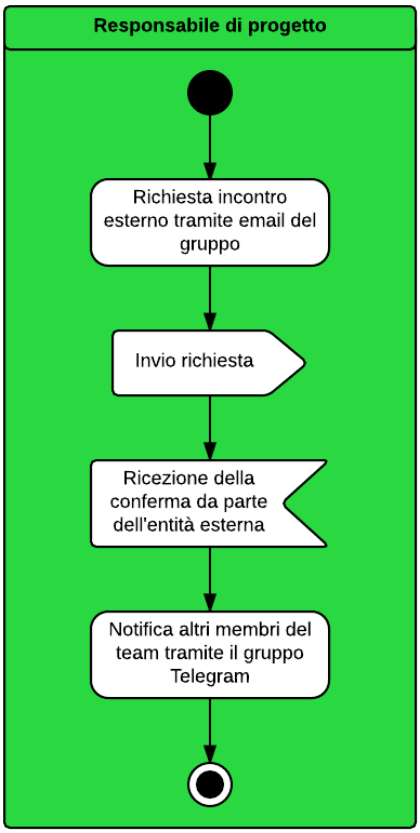

Figura 8: Procedura per l'organizzazione di un incontro esterno

5.1.7 Gestione degli strumenti di coordinamento

5.1.7.1 Ticketing

Il sistema di *ticketing*$_G$ viene utilizzato per suddividere il carico di lavoro in task, i quali verranno suddivisi tra tutti i componenti del gruppo.
Il compito di assegnare i ticket spetta al *Responsabile* di Progetto, che per compierlo dovrà utilizzare la piattaforma Teamwork. Questa strumento permette di avere un quadro completo sullo stato dei lavori e di visionare tutti i compiti completati o ancora da svolgere.
La procedura per l'assegnazione di un ticket segue il seguente schema:

- inserire un titolo riassuntivo del task;
- indicare la/e persona/e incaricate a svolgere tale compito;
- indicare la data di inizio e fine di tale compito;
- inserire una descrizione contenente la specifica del compito e il ruolo assunto in quella fase del progetto dal membro incaricato a svolgere il task;

- impostare l'invio di una notifica tramite e-mail, alle persone coinvolte nel ticket.

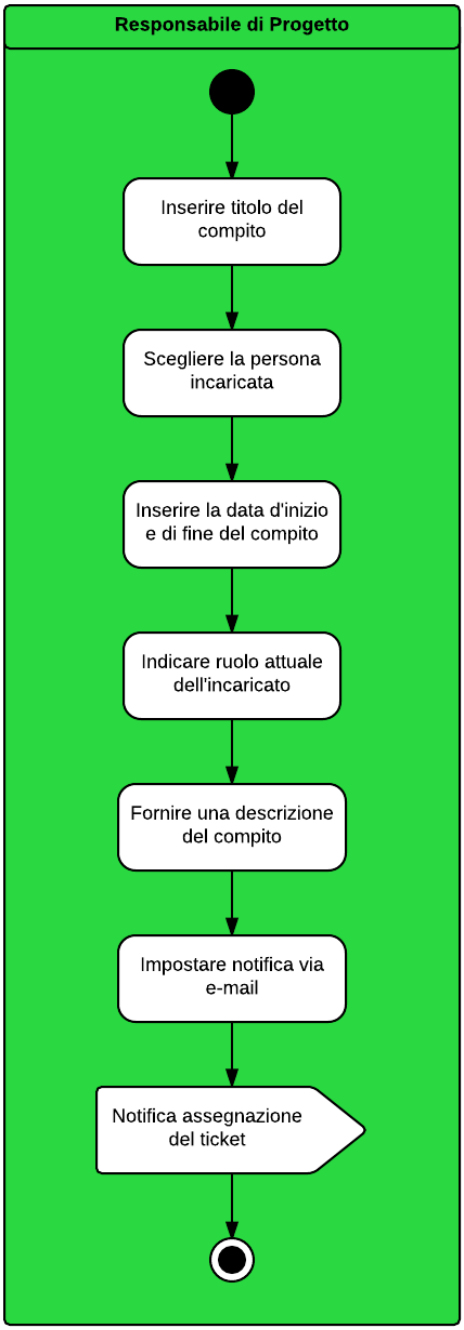

Figura 9: Procedura per l'assegnazione di un ticket

5.1.8 Gestione degli strumenti di versionamento

5.1.8.1 Repository

Per il versionamento e il salvataggio dei file prodotti durante l'attività di progetto è previsto l'utilizzo di diversi repository su GitHub. L'*Amministratore* di Progetto si occupa di creare tali repository con un account generale, rappresentante il gruppo di progetto. Successivamente, tutti gli altri membri del gruppo dovranno creare un account personale, che sarà autorizzato all'accesso dei repository dall'*Amministratore*.
E' previsto l'utilizzo di due repository:

- **docs**: contiene tutta la documentazione dell'attività di progetto;
- **MaaS**: contiene tutti i file rappresentanti l'implementazione del progetto.

5.1.8.2 Struttura del repository

I membri del gruppo sono tenuti a rispettare le seguenti norme sull'organizzazione dei file nei repository.
Struttura del repository docs:

- **modello**: contiene i file $template_G$ dei documenti LaTeX;
- **documenti**: contiene le cartelle dei documenti;
- **cartella documento**: contiene i file LaTeX del documento e il relativo registro delle modifiche, il file pdf e la cartella `images`, nella quale vanno inserite le immagini presenti all'interno del documento.

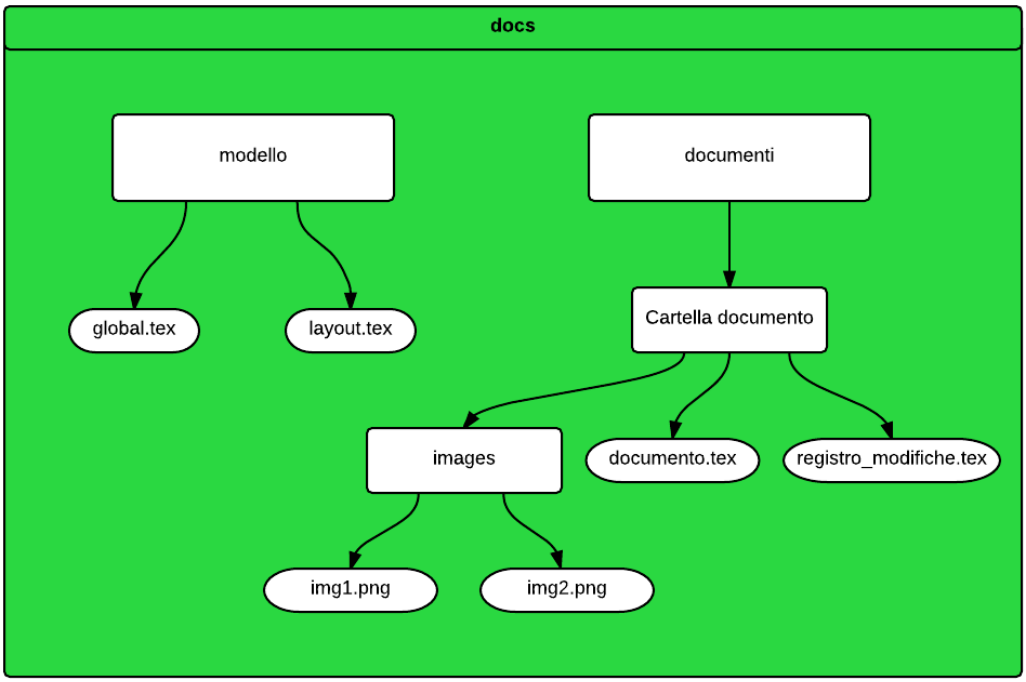

Figura 10: Struttura del repository della documentazione

Struttura del repository MaaS:

-

5.1.8.3 Norme sui commit

Ogni qual 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 liste delle modifiche effettuate, ordinate per ogni singolo file.

Prima di eseguire tale procedura, va aggiornato il diario delle modifiche, secondo le regole viste nella sezione 4.1.7.

5.1.9 Gestione dei rischi

Il *Responsabile* di Progetto ha il compito di individuare i rischi indicati nel *Piano di Progetto v1.0.0*. Nel caso vengano individuati nuovi rischi è suo dovere estendere l'analisi dei rischi con i

nuovi casi rilevati.
La procedura per la gestione dei rischi segue il seguente schema:

- monitorare i rischi previsti ed individuare i problemi non calcolati;
- registrare ogni riscontro effettivo dei rischi nel *Piano di Progetto v1.0.0*;
- aggiornare l'analisi dei rischi nel *Piano di Progetto v1.0.0* con i nuovi rischi individuati;
- ridefinire, se necessario, le strategie di progetto.

5.1.10 Strumenti

5.1.10.1 Sistema operativo

Il gruppo di progetto opera sui seguenti sistemi operativi:

- Ubuntu x64 nella versione 14.04 LTS_G;
- Windows 7 Service Pack 1 Home Edition;
- Windows 10 Pro;
- MacOS 10.9.

5.1.10.2 Telegram

Telegram è un servizio di messaggistica istantanea erogato senza fini di lucro dalla società Telegram LLC. I $client_G$ ufficiali di Telegram sono distribuiti come software libero per diverse piattaforme.

5.1.10.3 Teamwork

Teamwork (https://www.teamwork.com) è una piattaforma di project managment, che permette di pianificare e suddividere i compiti tra i membri del team.
I vantaggi che ci hanno portato a scegliere questo applicativo sono:

- offre una elevata portabilità ed accessibilità, essendo fruibile direttamente dal web;
- offre i servizi essenziali gratuitamente;
- risulta essere molto professionale.

Lo spazio dedicato al gruppo si trova al seguente indirizzo:

https://mintswe.teamwork.com

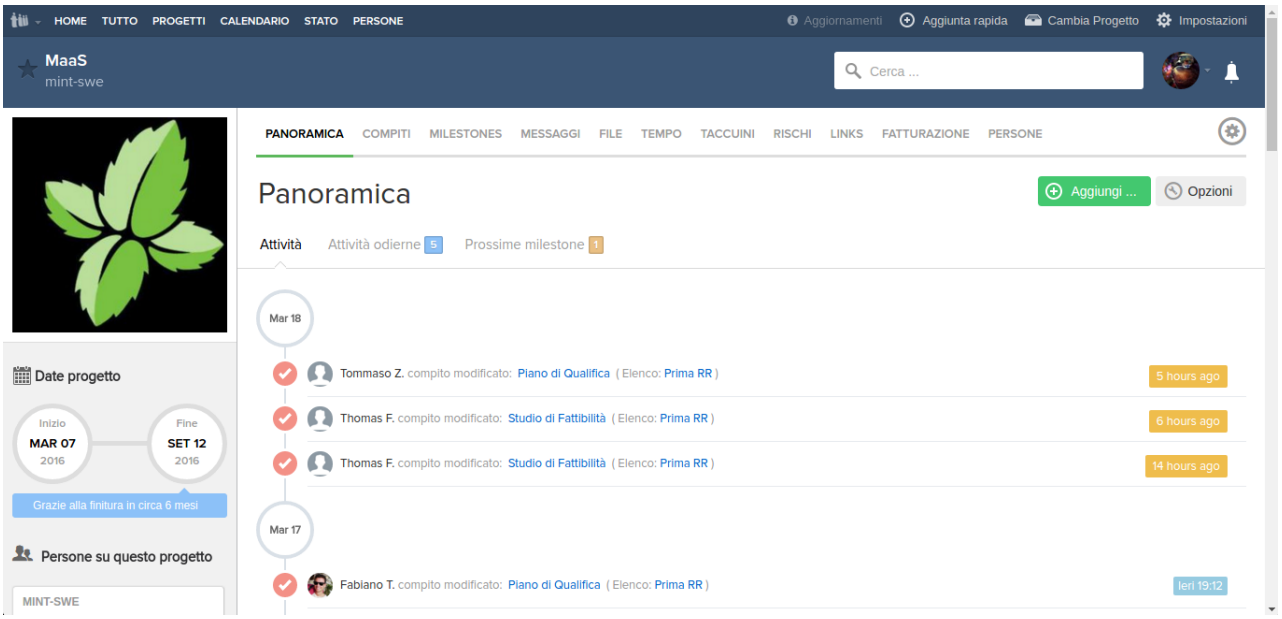

Figura 11: Teamwork

5.1.10.4 ProjectLibre

Si è utilizzato ProjectLibre per il supporto alla pianificazione del progetto e per la gestione delle risorse, tramite la creazione dei diagrammi di Gantt.

http://www.projectlibre.org/

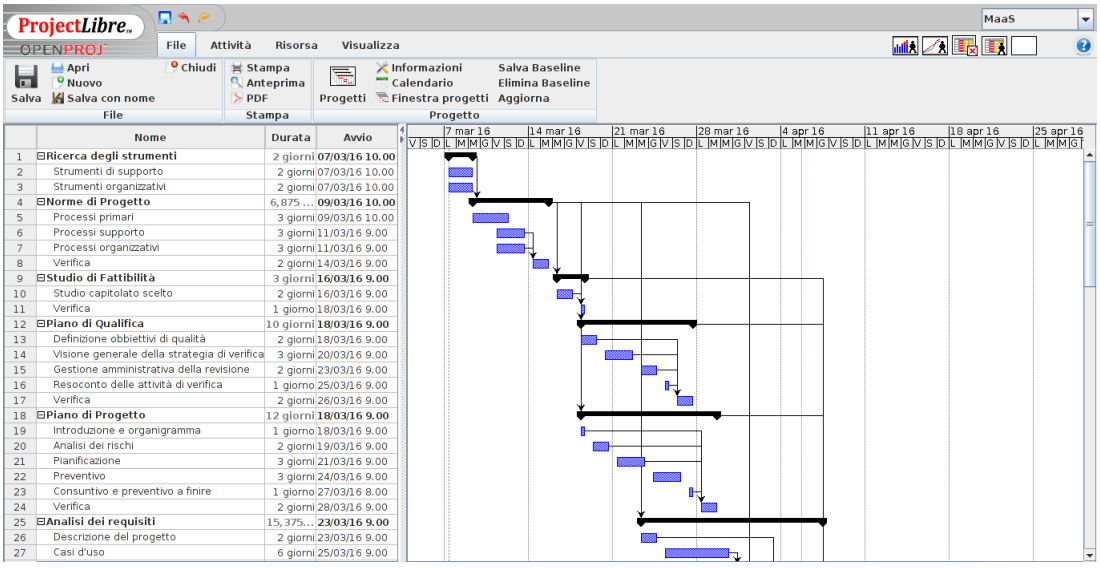

Figura 12: ProjectLibre

5.1.10.5 Git

Git$_G$ 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/

5.1.10.6 GitHub

GitHub$_G$ è un servizio web di hosting 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 *open-source*$_G$.

5.1.10.7 GitHub Desktop

GitHub Desktop è l'applicativo desktop per contribuire e collaborare ai progetti del corrispondente servizio web GitHub.
Vengono utilizzate le versioni per Windows (3.0 o superiori) e per MacOS (215 o superiori).

https://desktop.github.com

Permette fondamentalmente di:

- visualizzare velocemente i repository a cui si ha accesso;
- eseguire nei vari *branch*$_G$ azioni di commit selezionando da una lista le modifiche da apportare;
- visualizzare lo storico delle modifiche apportate al repository;
- sincronizzarsi ed eventualmente effettuare il *merge*$_G$ delle modifiche locali con quelle dello storico.

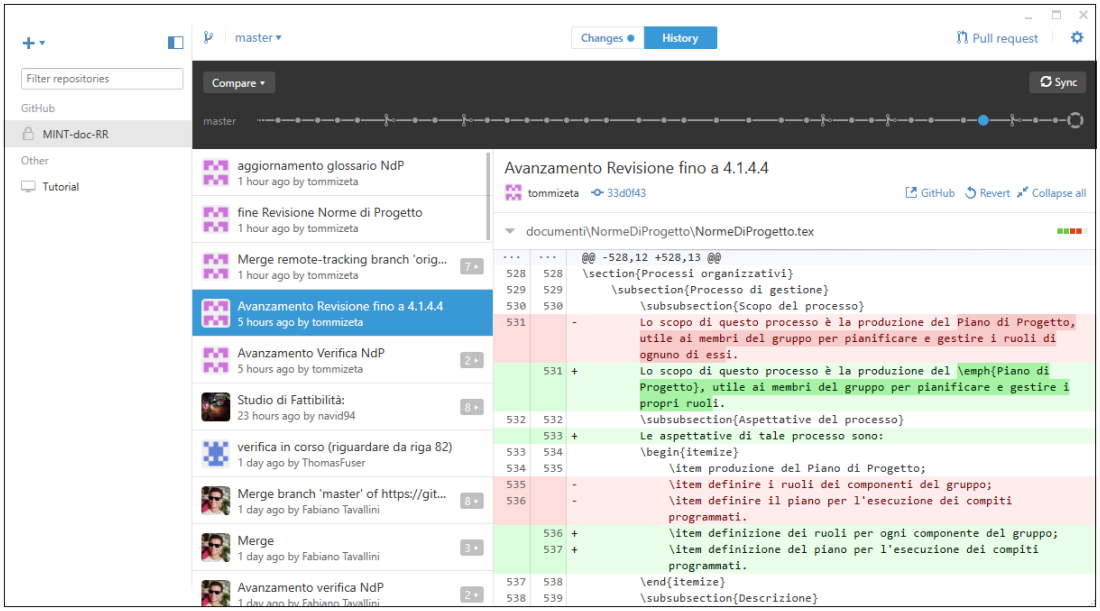

Figura 13: GitHub Desktop per Windows

5.1.10.8 SmartGit

SmartGit è un client desktop multipiattaforma, che permette di interfacciarsi e gestire i repository di GitHub, tramite una interfaccia grafica.
La versione utilizzata è maggiore o uguale alla 7.1.

http://www.syntevo.com/smartgit/

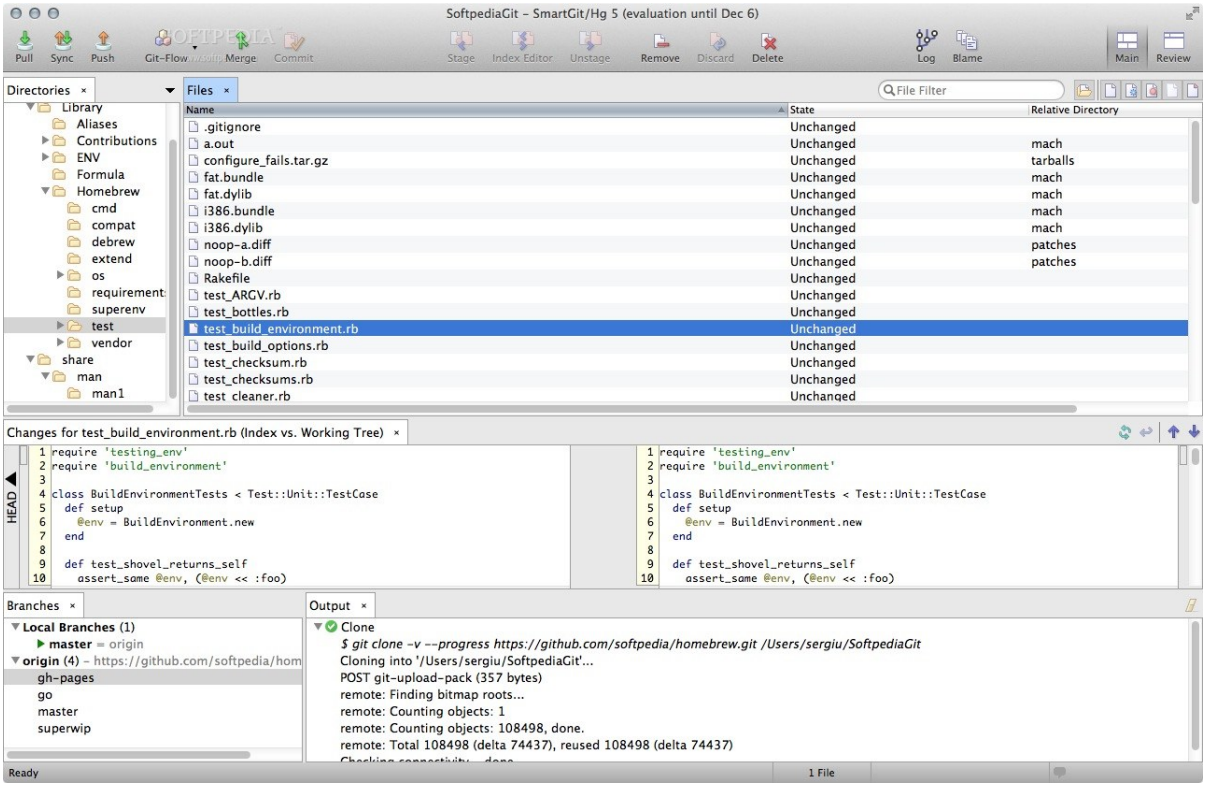

Figura 14: SmartGit