



Norme di Progetto

Gruppo SteakHolders — Progetto MaaP

Informazioni sul documento

Versione	4.0.0
Redazione	Enrico Rotundo Nicolò Tresoldi Federico Poli Giacomo Fornari
Verifica	Serena Girardi Gianluca Donato
Approvazione	Luca De Franceschi
Uso	Interno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo SteakHolders

Descrizione

Questo documento descrive le regole, gli strumenti e le procedure adottate dal gruppo SteakHolders per la realizzazione del progetto MaaP.



Registro delle modifiche

Versione	Data	Persone coinvolte	Descrizione
4.0.0	2014-03-12	Luca De Franceschi (Responsabile)	Approvazione.
3.1.0	2014-02-17	Enrico Rotundo (Verificatore)	Verifica.
3.0.3	2014-02-15	Giacomo Fornari (Amministratore)	Incremento: descrizione Requistest, norme codifica e ambiente sviluppo.
3.0.2	2014-02-13	Gianluca Donato (Amministratore)	Correzione.
3.0.1	2014-02-21	Nicolò Tresoldi (Amministratore)	Ristrutturate le norme secondo normativa ISO/IEC 12207-1995.
3.0.0	2014-02-04	Luca De Franceschi (Responsabile)	Approvazione.
2.2.0	2014-02-02	Giacomo Fornari (Verificatore)	Verifica.
2.1.2	2014-01-26	Serena Girardi (Amministratore)	Inserita descrizione nuovi script e individuato meccanismo di consegna.
2.1.1	2014-01-16	Gianluca Donato (Verificatore)	Correzioni.
2.1.0	2014-01-15	Serena Girardi (Verificatore)	Verifica.
2.0.5	2014-01-13	Enrico Rotundo (Amministratore)	Inserite le "Tecniche di verifica. Inserita norma nella sezione "Struttura del repository.
2.0.4	2014-01-12	Nicolò Tresoldi (Amministratore)	Inserita sezione pianificazione nel capitolo "Procedure a supporto dei processi"
2.0.3	2014-01-12	Giacomo Fornari (Amministratore)	Riorganizzata struttura.
2.0.2	2014-01-10	Nicolò Tresoldi (Amministratore)	Inserito paragrafo 7.1.
2.0.1	2014-01-10	Federico Poli (Amministratore)	Aggiornamento sezione "Procedure di progetto" con le issues.
2.0.0	2014-01-10	Nicolò Tresoldi Amministratore	Aggiornamento sistema di versionamento.
1.3.1	2013-12-5	Gianluca Donato (Responsabile in deroga)	Approvazione.
1.2.1	2013-12-4	Serena Girardi (Verificatore)	Verifica.
1.1.8	2013-12-3	Giacomo Fornari (Amministratore)	Aggiunte le immagini dei diagrammi alla sezione "Procedure".



1.1.7	2013-12-3	Federico Poli (Amministratore)	Stesura sezione “Procedure”.
1.1.6	2013-12-3	Luca De Franceschi (Amministratore)	Stesura sezioni “Analisi”, “Progettazione”.
1.1.5	2013-12-2	Luca De Franceschi (Amministratore)	Stesura sezione “Repository”, “Documenti”.
1.1.4	2013-12-2	Giacomo Fornari (Amministratore)	Stesura sezione “Codifica”.
1.1.3	2013-12-1	Nicolò Tresoldi (Responsabile)	Stesura sezioni “Introduzione”, “Comunicazioni”, “Glossario”.
1.1.2	2013-12-1	Federico Poli (Amministratore)	Stesura sezione “Ambiente di lavoro”.
1.1.1	2013-12-1	Nicolò Tresoldi (Amministratore)	Stesura indice delle sezioni.



Indice

1	Introduzione	7
1.1	Scopo del documento	7
1.2	Ambiguità	7
1.3	Riferimenti normativi	7
2	Processi Primari	8
2.1	Analisi dei requisiti	8
2.1.1	Casi d'Uso	8
2.1.2	Codice identificativo	8
2.1.3	Requisiti	9
2.1.4	Codice identificativo	9
2.1.5	UML	9
2.2	Progettazione	9
2.2.1	Diagrammi UML	10
2.2.2	Stile di progettazione	10
2.3	Codifica	10
2.3.1	Intestazione	10
2.3.2	Formattazione	11
2.3.3	Ricorsione	11
2.4	Strumenti e ambiente di sviluppo	11
2.4.1	Stesura del codice	11
2.4.2	Framework	12
3	Processi di supporto	13
3.1	Gestione ticket	13
3.1.1	Richiesta di modifica e segnalazione bug	13
3.1.2	Progettazione unità di lavoro	14
3.1.3	Valutazione issue	15
3.1.4	Pianificazione issue	16
3.1.5	Creazione task	18
3.1.6	Esecuzione compito	19
3.1.7	Esecuzione verifica	20
3.2	Rotazione dei ruoli	20
3.2.1	Amministratore	21
3.2.2	Analista	21
3.2.3	Progettista	21
3.2.4	Programmatore	22
3.2.5	Responsabile	22
3.2.6	Verificatore	22
3.3	Processo di Coordinamento	22
3.3.1	Comunicazioni interne	22
3.3.1.1	Messaggi	23
3.3.1.2	Procedura per commenti a tasks o sub-tasks	23
3.3.2	Comunicazioni esterne	23
3.3.3	Riunioni	23
3.4	Revisioni di progetto	24



3.5	Procedure per la gestione della Repository	25
3.5.1	Repository dei documenti	25
3.5.1.1	Struttura del repository	25
3.5.1.2	Branch	25
3.5.1.3	Script di pre-commit	25
3.5.2	Repository del codice	26
3.5.2.1	Struttura del repository	26
3.5.2.2	Branch	27
3.5.2.3	Script di pre-commit	27
3.6	Condivisione file	27
3.7	Modalità di consegna	28
3.8	Strumenti per il coordinamento	28
3.8.1	Calendario condiviso	28
3.8.2	Strumenti per la gestione dei ticket	28
3.8.3	Strumenti per la gestione del piano di lavoro	28
3.8.4	Gestione degli eventi	29
4	Processi organizzativi	30
4.1	Processo di Pianificazione	30
4.2	Processo di Documentazione	30
4.2.1	Template	30
4.2.2	Struttura del documento	30
4.2.2.1	Prima pagina	30
4.2.2.2	Registro delle modifiche	31
4.2.2.3	Indice	31
4.2.2.4	Formattazione generale delle pagine	31
4.2.2.5	Note a piè di pagina	32
4.2.3	Versionamento	32
4.2.4	Procedura per la definizione di macro personalizzate	32
4.2.4.1	Costanti	32
4.2.4.2	Funzioni	33
4.2.5	Norme tipografiche e convenzioni	33
4.2.5.1	Punteggiatura	33
4.2.5.2	Stile del testo	33
4.2.5.3	Elenchi puntati	34
4.2.5.4	Formati	34
4.2.5.5	Sigle	34
4.2.5.6	Riferimenti a documenti	35
4.2.5.7	Nomenclatura entità e relazioni	35
4.2.6	Tabelle e immagini	35
4.2.6.1	Tabelle	35
4.2.6.2	Immagini	36
4.2.7	Classificazione di documento	36
4.2.7.1	Documenti preliminari	36
4.2.7.2	Documenti formali	36
4.2.7.3	Verbali	36
4.2.8	Verifica e validazione	36
4.2.8.1	Procedura per la verifica	37
4.2.9	Approvazione	37



4.2.10	Glossario	37
4.3	Processo di Verifica	37
4.3.1	Statiche	37
4.3.1.1	Walkthrough	38
4.3.1.2	Inspection	38
4.3.2	People-intensive	38
4.3.3	Analitiche	38
4.3.4	Dinamiche	39
4.3.4.1	Test di unità	39
4.3.4.2	Test di integrazione	39
4.3.4.3	Test di sistema	39
4.3.4.4	Test di regressione	39
4.3.4.5	Test di accettazione	39
4.4	Strumenti	40
4.4.1	Ambiente e strumenti generali	40
4.4.1.1	Sistema operativo	40
4.4.1.2	Codifica dei caratteri	40
4.4.1.3	Strumenti per il versionamento	40
4.4.2	Strumenti per la produzione dei documenti	40
4.4.2.1	Scrittura	40
4.4.2.2	Strumenti per il controllo ortografico	40
4.4.2.3	Script di Makefile	41
4.4.2.4	UML	41
4.4.2.5	Script download use case e requisiti	41
4.4.2.6	Script di produzione del Gantt e dei grafici	42
4.4.3	Ambiente e strumenti per la verifica e validazione	43
4.4.3.1	Strumenti per l'analisi statica	43
4.4.3.2	Strumenti per l'analisi dinamica	43
4.4.3.3	Strumenti per la validazione	44
4.4.3.4	Script di Makefile	44
4.4.3.5	Strumenti per l'integrazione continua	44
4.4.4	Strumenti per la tracciabilità	45
4.4.4.1	GUI	46
4.4.4.2	Use Cases	47
4.4.4.3	Requirements	48
4.4.4.4	Components	48
4.4.4.5	Classes	48
4.4.4.6	Unit Tests	48
4.4.4.7	Integration Tests	48
4.4.4.8	System Tests	48
4.4.4.9	Validation Tests	49

Elenco delle figure

1	Richiesta di modifica e segnalazione bug	13
2	Creazione compito	14
3	Valutazione issue	15
4	Pianificazione issue	16



5	Creazione task	18
6	Esecuzione compito	19
7	Esecuzione verifica	20
8	Esempio script	43
9	Requisteak GUI: menù e index-page	46
10	Requisteak GUI: show-page	47



1 Introduzione

1.1 Scopo del documento

Questo documento ha come obiettivo quello di definire le regole, gli strumenti e le procedure che tutti i membri del team dovranno adottare per l'intero svolgimento del progetto. Tutti i componenti del gruppo sono obbligati a visionare tale documento e 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 sarà necessario informare tempestivamente ogni membro del gruppo.

1.2 Ambiguità

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti viene fornito il *Glossario v4.0.0*, contenente la definizione dei termini marcati con una G pedice.

In questo documento alcuni termini devono essere interpretati in modo analogo ai termini inglesi descritti in RFC 2119¹:

- I termini **“deve”**, **“è richiesto”** e sinonimi stretti sono da intendersi con lo stesso significato di **“MUST”**;
- I termini **“non deve”**, **“è richiesto che non”** e sinonimi stretti sono da intendersi con lo stesso significato di **“MUST NOT”**;
- I termini **“dovrebbe”**, **“si raccomanda”**, **“è preferibile”** e sinonimi stretti sono da intendersi con lo stesso significato di **“SHOULD”**;
- I termini **“non dovrebbe”**, **“si raccomanda di non”**, **“è preferibile che non”** e sinonimi stretti sono da intendersi con lo stesso significato di **“SHOULD NOT”**;
- I termini **“può”**, **“opzionalmente”** e sinonimi stretti sono da intendersi con lo stesso significato di **“MAY”**.

1.3 Riferimenti normativi

ISO/IEC 12207-1995 http://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf; **IEEE Std 1028** http://en.wikipedia.org/wiki/Software_review#cite_ref-MFaganPapers_3-0;

¹<http://www.ietf.org/rfc/rfc2119.txt>



2 Processi Primari

2.1 Analisi dei requisiti

Dal capitolato e dagli incontri con il proponente gli Analisti dovranno estrarre i requisiti del progetto, producendo l'Analisi dei Requisiti.

Tutti i requisiti che si possono evincere dal capitolato o ad un incontro con il proponente vanno specificati nell'Analisi dei Requisiti.

Per agevolare l'analisi dei requisiti viene utilizzata la tecnica dei casi d'uso.

Il tracciamento dei requisiti avviene tramite il software descritto in 4.4.4.

2.1.1 Casi d'Uso

Ogni caso d'uso dovrà presentare i seguenti campi:

- Codice identificativo
- Titolo
- Diagramma UML
- Attori primari
- Attori secondari
- Scopo e descrizione
- Precondizione
- Postcondizione
- Flusso principale degli eventi
- Scenari alternativi

2.1.2 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:

U = Ambito *Utente*_G;

S = Ambito *Sviluppatore*_G.

- **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.



2.1.3 Requisiti

Ogni requisito dovrà presentare i seguenti campi:

- Codice identificativo
- Descrizione
- Fonti

2.1.4 Codice identificativo

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

$$R\{X\}\{Y\}\{Z\} \{Gerarchia\}$$

Dove:

- **X** corrisponde al sistema di riferimento e può assumere i seguenti valori:
 - A** = Applicazione $Maap_G$;
 - F** = $Framework_G$ di $Maap_G$;
 - S** = $Maas_G$.
- **Y** corrisponde alla tipologia del requisito e può assumere i seguenti valori:
 - 1** = Funzionale;
 - 2** = Di prestazione;
 - 3** = Di qualità;
 - 4** = Vincolo.
- **Z** corrisponde alla priorità del requisito e può assumere i seguenti valori:
 - O** = Obbligatorio
 - D** = Desiderabile
 - F** = Facoltativo o opzionale
- **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.

2.1.5 UML

Per i diagrammi deve essere utilizzato il linguaggio *UML versione 2.0*.

2.2 Progettazione

La progettazione deve dimostrabilmente rispettare tutti i requisiti che il gruppo ha concordato con il committente. In particolare i componenti progettati devono essere tracciabili rispetto al requisito che soddisfano. Di seguito vengono elencate le norme a carico dei *Progettisti*.



2.2.1 Diagrammi UML

Si dovrà usare il linguaggio *UML_G versione 2.0* per i seguenti diagrammi:

- **Diagrammi dei package:** dovranno essere presenti sia per l'architettura generale che di dettaglio, sarà fondamentale per definire i moduli all'interno del *framework_G Node.js_G* richiesto dal capitolato;
- **Diagrammi delle classi:** qualora il progetto utilizzasse delle classi, i diagrammi delle classi dovranno essere presenti sia per l'architettura generale che di dettaglio. Nell'ambiente *Node.js_G* a prima vista sembra che siano poco utilizzate, a favore dei *package_G*;
- **Diagrammi di flusso:** qualora la codifica di un'unità del progetto sia particolarmente complessa, dovrà essere presente il relativo diagramma di flusso che il programmatore dovrà seguire;

2.2.2 Stile di progettazione

- La progettazione dovrà usare quanto più possibile *design pattern_G* globalmente affermati e la loro scelta dovrà essere giustificata;
- Suddividere il progetto in *moduli_G*, in accordo con lo stile di progettazione dell'ambiente *Node.js_G*;
- Non utilizzare codice sincrono per operazioni di *I/O_G*;
- Limitare quanto più possibile le *callback_G* annidate;

2.3 Codifica

2.3.1 Intestazione

```
/*
* Name: {Nome del file}
* Module: {modulo di appartenenza}
* Location: {/path/della/cartella/}
*
* History:
* Version      Date                Programmer
* =====
* 1.1.3        AAAA-MM-GG          {Nome Cognome      }
* -----
* ...
* {Description}
* ...
* =====
* 1.1.2        AAAA-MM-GG          {Nome Cognome      }
* -----
* ...
```



*
*/

- **Name** è il nome del file, estensione compresa.
- **Module** è il nome del modulo di cui il file fa parte.
- **Location** è il percorso del file, a partire dalla cartella principale del progetto “/” fino alla cartella che contiene il file. Deve iniziare e terminare con uno slash “/”.
- **History** è il diario delle modifiche del file. Ogni modifica è composta dai seguenti campi:
 - **Version** è la versione del file.
 - **Date** è la data della modifica.
 - **Programmer** è il nome e cognome dell'autore della modifica. Al massimo può essere lungo 20 caratteri.
 - **Description** è la spiegazione delle modifiche fatte e del motivo per cui sono state fatte.

2.3.2 Formattazione

La formattazione del codice sorgente deve essere definita in modo rigoroso e consistente, così che tutto il codice del progetto sembri che sia stato scritto da un'unica persona.

Per il codice *Javascript* si è scelto di adottare le linee guida utilizzate dal progetto *jQuery*, sia per l'effettiva facilità di lettura sia per la possibilità di automatizzare la formattazione del codice tramite il programma *JSHint*. La pagina di riferimento è <http://contribute.jquery.org/style-guide/js/>.

2.3.3 Ricorsione

La ricorsione va evitata quando possibile. Per ogni funzione ricorsiva sarà necessario fornire una prova di terminazione e sarà necessario valutare il costo in termini di occupazione della memoria. Nel caso l'utilizzo di memoria risulti troppo elevato la ricorsione dovrà essere rimossa.

2.4 Strumenti e ambiente di sviluppo

2.4.1 Stesura del codice

Per la stesura del codice è consigliabile usare i seguenti editor:

- **gedit** $\geq 3.4.1$ (<http://projects.gnome.org/gedit>)
- **Sublime text** $\geq 2.0.2$ (<http://www.sublimetext.com>)
- **Eclipse** (<http://www.eclipse.org>)



È fortemente consigliato l'utilizzo di alcuni plugin per Sublime text che evidenziano le irregolarità nel codice scritto. Tali plugin sono utilizzabili previa installazione di **Package Control** (<https://sublime.wbond.net/installation>), si potrà accedere a **Package Control** digitando `ctrl+shift+p` e selezionando **Install Package**. Procedere con l'installazione di **Sublime Linter**, dopodiché installare:

- **SublimeLinter-annotations** per evidenziare i `TODO` e `README`;
- **SublimeLinter-jshint** per evidenziare errori di sintassi *javascript_G*;
- **SublimeLinter-json** per evidenziare errori nei documenti *json_G*.

2.4.2 Framework

Per lo sviluppo del progetto è previsto l'utilizzo del server **Node.js** $\geq 0.10.22$ e del relativo *package manager_G* **npm** $\geq 1.3.6$.

3 Processi di supporto

3.1 Gestione ticket

Abbiamo ritenuto poco efficiente utilizzare la piattaforma *TeamworkPM_G* per la gestione sia delle attività che delle modifiche, poiché l'utilizzo che volevamo farne non sfruttava appieno le funzionalità di “Diagramma Gantt” e “Task List”. È stata fatta quindi la seguente separazione:

- **gestione delle attività:** viene fatta utilizzando i *task_G* di *TeamworkPM_G*
- **gestione delle modifiche:** viene fatta utilizzando le *issue_G* di *GitHub_G*

Quando non è specificato come modificare il valore di un parametro delle *issue_G* bisogna mantenere il valore già presente, o quello di default se la *issue_G* è appena stata creata.

3.1.1 Richiesta di modifica e segnalazione bug

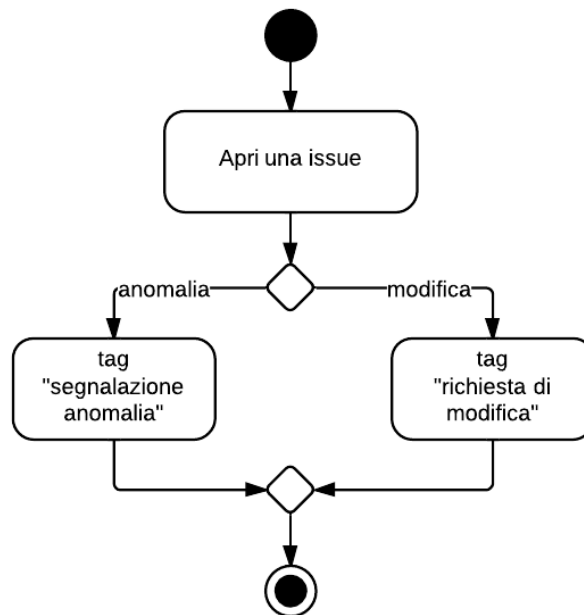


Figura 1: Richiesta di modifica e segnalazione bug

Se un membro del gruppo intende segnalare un bug o richiedere una modifica deve aprire una *issue_G* su *GitHub_G* con i seguenti parametri:

- **Titolo:** un titolo breve ma descrittivo.

- **Descrizione:** una descrizione esaustiva dell'anomalia riscontrata o della modifica richiesta.
- **Tag:** uno tra i seguenti:
 - **segnalazione anomalia:** da usare se si vuole segnalare un bug, un malfunzionamento, un'anomalia in genere che non dovrebbe esserci nei documenti ufficiali o nel prodotto finale;
 - **richiesta di modifica:** da usare se si vuole richiedere al Responsabile una modifica.
- **Assegnato a:** il Responsabile
- **Milestone:** nessuna

3.1.2 Progettazione unità di lavoro

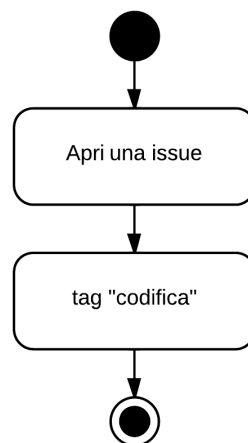


Figura 2: Creazione compito

Il Progettista, dopo aver completata la progettazione di dettaglio, deve creare le $issue_G$ di codifica con i seguenti parametri:

- **Titolo:** un titolo breve ma descrittivo.
- **Descrizione:** una descrizione esaustiva dell'attività da svolgere, con tutti i riferimenti necessari.
- **Tag:** **codifica**
- **Assegnato a:** il Responsabile
- **Milestone:** nessuna

3.1.3 Valutazione issue

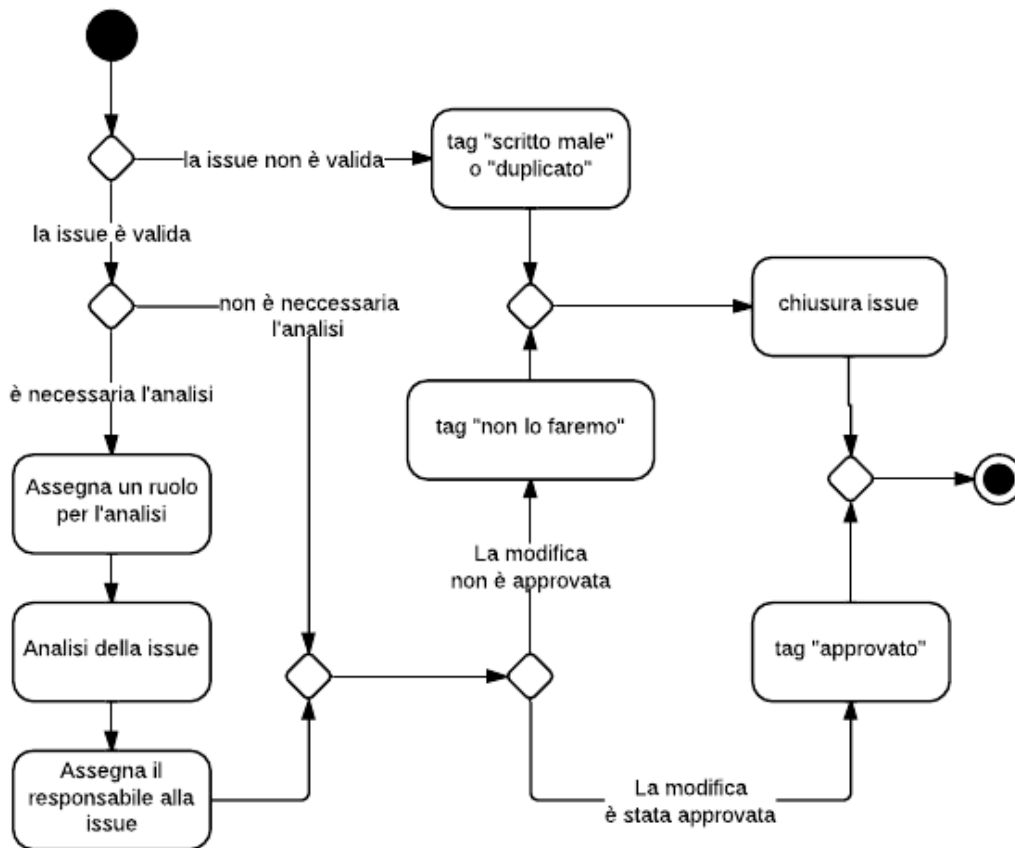


Figura 3: Valutazione issue

Il Responsabile deve valutare ogni nuova *issue_G*. Se ritiene che la formulazione sia corretta e se è necessario approfondire la modalità in cui verrà eseguita l'attività richiesta, allora il Responsabile deve anche assegnare la *issue_G* ad un ruolo competente:

- **Assegnato a:** un ruolo a scelta del Responsabile

Tale ruolo, dopo averla analizzata, deve riassegnarla al Responsabile aggiungendo le informazioni prodotte dall'analisi:

- **Assegnato a:** il Responsabile
- **Descrizione:** la descrizione già presente con in aggiunta i risultati dell'analisi.

A questo punto, se il Responsabile ritiene che l'attività debba essere eseguita, pianifica la *issue* secondo la procedura 3.1.4.

Se il Responsabile non approva o non ritiene valida la *issue_G* deve chiuderla impostando:

- **Tag:** uno tra i seguenti
 - **scritto male:** se la descrizione o il titolo sono poco comprensibili;
 - **non lo faremo:** se ritiene che non valga la pena di svolgere l'attività richiesta;
 - **uplicato:** se l'attività è già stata richiesta da un'altra $issue_G$. In questo caso è raccomandabile aggiungere nella descrizione un riferimento alla $issue_G$ originale.
- **Stato:** "Chiuso".

3.1.4 Pianificazione issue

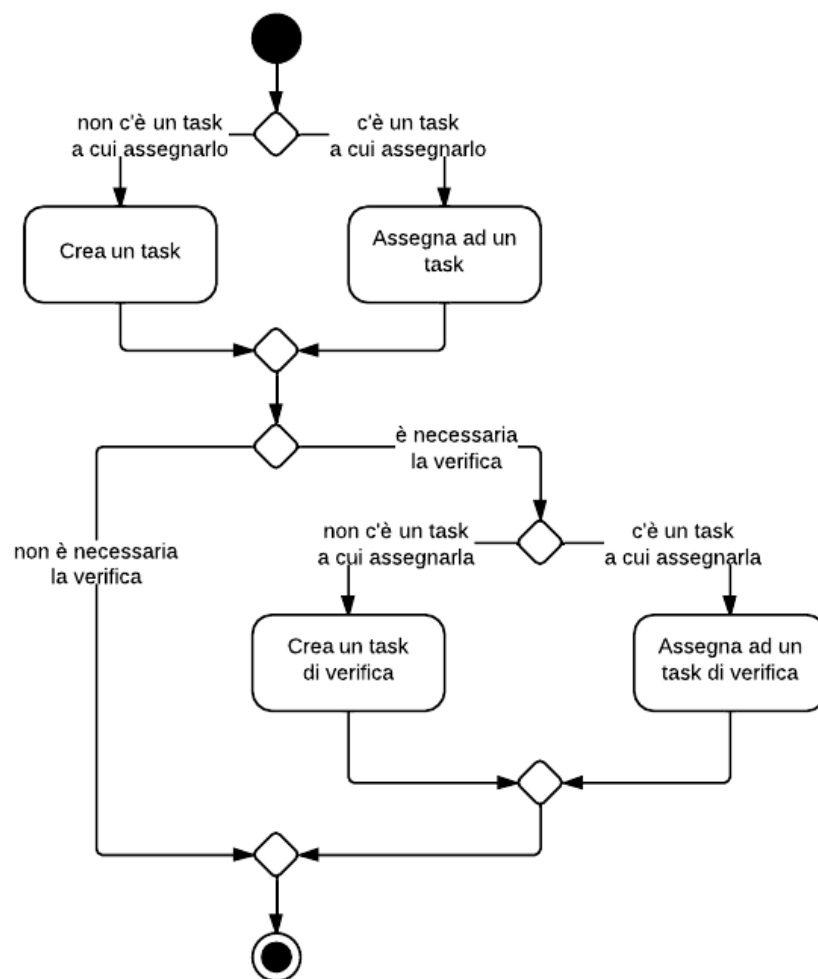


Figura 4: Pianificazione issue



Il Responsabile deve pianificare le $issue_G$ a lui assegnate e marcate “approvato” o “codifica” creando un $task_G$. Deve inoltre pianificare il corrispondente $task_G$ di verifica se necessario. I $task_G$ del compito e della modifica devono essere assegnati a persone diverse.

Modifica i parametri della $issue_G$:

- **Assegnato a:** nessuno.

Se esiste già un $task_G$ a cui associare la $issue_G$ il Responsabile deve scegliere una delle seguenti opzioni:

- modificare tale $task_G$ con questi parametri:
 - **Descrizione:** la descrizione precedente, con in aggiunta un riferimento evidente alla $issue_G$.
- creare un $task_G$ secondario (“subtask” nel gergo di TeamworkPM) con i seguenti parametri:
 - **Titolo:** un titolo breve ma descrittivo, magari simile a quello della $issue_G$.
 - **Descrizione:** una breve descrizione, con un riferimento evidente alla $issue_G$.
 - **Assegnato a:** la stessa persona a cui è associato il $task_G$ principale.

Se il $task_G$ non esiste, allora deve crearne uno nuovo seguendo la procedura 3.1.5.

Analogamente, se esiste già un $task_G$ a cui associare la verifica, il Responsabile deve scegliere una delle seguenti opzioni:

- modificare tale $task_G$ di verifica con questi parametri:
 - **Descrizione:** la descrizione precedente, con in aggiunta un riferimento evidente al $task_G$ da verificare.
- creare un $task_G$ secondario (“subtask” nel gergo di TeamworkPM) con i seguenti parametri:
 - **Titolo:** un titolo breve ma descrittivo, magari simile a quello della $issue_G$.
 - **Descrizione:** una breve descrizione, con un riferimento evidente alla $issue_G$.
 - **Assegnato a:** la stessa persona a cui è associato il $task_G$ principale.

Se il $task_G$ di verifica non esiste, allora deve crearne uno nuovo seguendo la procedura 3.1.5.

3.1.5 Creazione task

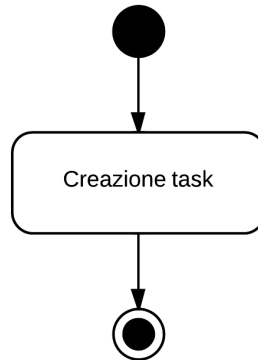


Figura 5: Creazione task

Il Responsabile può creare $task_G$ con i seguenti parametri:

- **Titolo:** il titolo dell'attività nel formato “{identificativo} - {titolo} ({ruolo})”, dove
 - **identificativo:** il codice che identifica l'attività nel *Piano di progetto*, se presente;
 - **titolo:** il titolo, breve e descrittivo, dell'attività.
 - **ruolo:** il ruolo a cui è assegnato il $task_G$
- **Assegnato a:** un membro del gruppo a scelta del Responsabile.
- **Pianificazione:** a scelta del Responsabile.
- **Descrizione:** una descrizione dell'attività (raccomandata).
- **Dipendenze:** i $task_G$ di cui bisogna attendere il termine prima di poter svolgere questo $task_G$. In particolare, i $task_G$ di verifica devono sempre dipendere dai $task_G$ che devono verificare.

Solo il Responsabile può creare nuovi task.

3.1.6 Esecuzione compito

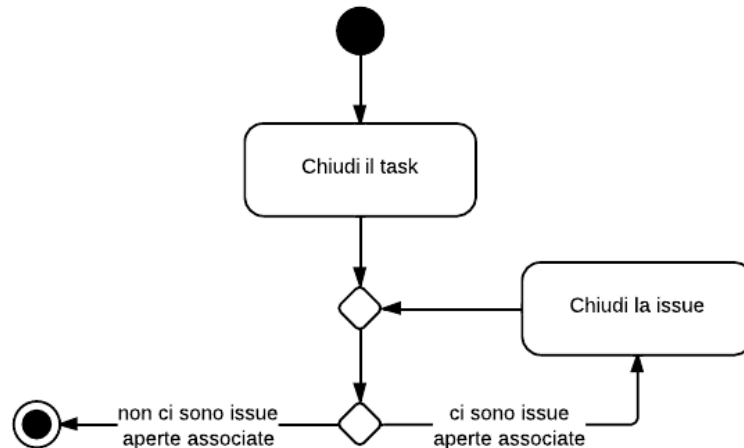


Figura 6: Esecuzione compito

Il ruolo a cui è assegnato un $task_G$ non di verifica, dopo aver svolto l'attività richiesta, deve chiudere il $task_G$:

- **Stato task:** "Chiuso".
- **Funzione "timer":** registrare le ore di lavoro produttivo che sono state necessarie per completare l'attività. (Attenzione: non bisogna modificare, invece, il tempo stimato)

Inoltre deve chiudere ogni eventuale $issue_G$ aperta associata:

- **Stato issue:** "Chiuso".

3.1.7 Esecuzione verifica

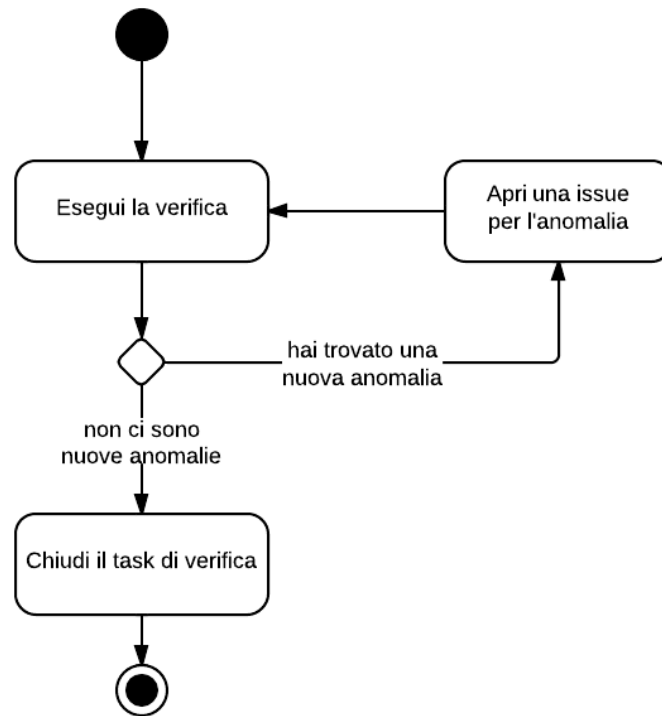


Figura 7: Esecuzione verifica

Il verificatore a cui è assegnato un $task_G$, dopo aver svolto l'attività richiesta, deve chiudere il $task_G$:

- **Stato:** “Chiuso”.
- **Funzione “timer”:** registrare le ore di lavoro produttivo che sono state necessarie per completare l'attività. (Attenzione: non bisogna modificare, invece, il tempo stimato)

Nel caso in cui il verificatore trovi bug o non conformità significative deve creare una $issue_G$, seguendo la procedura descritta nella sezione 3.1.1.

3.2 Rotazione dei ruoli

Durante lo sviluppo del progetto vi sono diversi ruoli che ogni membro del gruppo SteakHolders è tenuto a ricoprire almeno una volta. Per evitare possibili conflitti causati dalla rotazione dei ruoli, le attività principali assegnabili a specifici ruoli sono pianificate nel *Piano di Progetto v4.0.0*. Ogni componente del gruppo è tenuto a svolgere le attività assegnategli e a rispettare il ruolo che ne consegue. Il *Responsabile* di progetto ha il compito



di fare rispettare i ruoli assegnati durante le attività, mentre il *Verificatore* deve individuare le possibili incongruenze tra i ruoli e le modifiche registrate nei diari delle modifiche. Le incongruenze possono essere di due tipi:

- Il compito svolto non fa parte dei compiti propri di un dato ruolo;
- La stessa persona verifica ciò che ha precedentemente prodotto.

Inoltre, si raccomanda che:

- Una persona non impieghi più del 50% delle ore di lavoro in un unico ruolo;
- Sia assegnato un solo task al giorno per persona.
- I ruoli vengano ruotati ogni settimana, ma si lascia libertà al *Responsabile* di ruotare in base alle esigenze.

Si predilige dunque una maggior frequenza di rotazione a discapito dell'efficienza.

Il responsabile quando assegna un ticket, come indicato nella sezione 3.8.2 dovrà indicare il ruolo che il destinatario ricopre nell'adempire tale task. Questo è necessario per il diario delle modifiche dei singoli documenti e per il rendiconto economico. I ruoli sono assegnati basandosi sul *Piano di Progetto v4.0.0*.

Deve inoltre tener conto delle non disponibilità a lavorare come indicato nella sezione 3.8.1. A supporto di tali raccomandazioni si automatizzano le verifiche utilizzando alcune funzionalità predisposte dallo script descritto nella sezione 4.4.2.6.

3.2.1 Amministratore

L'*Amministratore* equipaggia, organizza e gestisce l'ambiente di lavoro e di produzione. Collabora con il *Responsabile* alla stesura del *Piano di Progetto* e redige le *Norme di Progetto*. Le responsabilità assunte dall'*Amministratore* sono:

- Attuare le scelte tecnologiche concordate con il *Responsabile* di progetto;
- Gestire le liste di distribuzione e assicurarne il rispetto;
- Controllare versioni e configurazioni del prodotto;
- Risolvere i problemi legati alla gestione dei processi.

3.2.2 Analista

L'*Analista* è responsabile dell'attività di analisi, pertanto deve comprendere appieno il dominio applicativo. Redige lo *Studio di Fattibilità* e l'*Analisi dei Requisiti* ovvero una specifica di progetto ad alto livello con vincoli e rischi tecnologici, affrontabile dal proponente, dal committente e dal *Progettista*.

3.2.3 Progettista

Il *Progettista* è responsabile dell'attività di progettazione, ha una profonda conoscenza dello stack tecnologico utilizzato e competenze tecniche aggiornate. Grazie a tali caratteristiche,



ha una forte influenza sugli aspetti tecnici e tecnologici del progetto, e spesso ne assume responsabilità di scelta e gestione.

3.2.4 Programmatore

Il *Programmatore* ha responsabilità circoscritte, si occupa dell'attività di codifica, nel rispetto delle *Norme di Progetto*, miranti alla realizzazione del prodotto e delle componenti di ausilio necessarie per l'esecuzione delle prove di verifica e validazione.

3.2.5 Responsabile

Il *Responsabile* ha l'ultima voce in capitolo per quanto concerne le decisioni sul progetto, è il responsabile ultimo dei risultati, infatti approva l'emissione dei documenti. Inoltre, redige il *Piano di Progetto* assieme all'*Amministratore*. Le responsabilità assunte dal *Responsabile* sono:

- Pianificazione e organizzazione dello sviluppo del progetto, stima tempi e costi, e assegnazione delle attività ai componenti del gruppo;
- Riportare lo stato del progetto ai committenti;
- Analizzare i rischi che possono incorrere, monitorarli e prendere provvedimenti a riguardo;
- Stabilire una *ways of working* per ogni componente del gruppo, ai fini di un'influenza positiva delle performance del gruppo.

3.2.6 Verificatore

Il *Verificatore* organizza ed attua le attività di verifica e controlla che le attività siano conformi alle norme. Redige la parte del *Piano di Qualifica* che documenta le attività svolte e i risultati ottenuti, confrontandoli con le metriche espresse nel *Piano di Qualifica v4.0.0*.

Le ore assegnate al verificatore devono corrispondere almeno al 30% delle ore totali suddivise tra ruoli.

3.3 Processo di Coordinamento

3.3.1 Comunicazioni interne

Tutte le comunicazioni ad uso interno del gruppo avvengono tramite il servizio di messaggistica offerto da *TeamworkPM*, altre vie di comunicazione sono sconsigliate.

Nel caso sia necessario comunicare in modo rapido possono essere utilizzati strumenti di messaggistica istantanea come Hangouts o Skype. L'utilizzo di *SMS_G* e di chiamate telefoniche è riservato alle situazioni di urgenza.



3.3.1.1 Messaggi

- L'**oggetto** deve essere sintetico e coerente con il contenuto del messaggio;
- Il **contenuto** deve includere i dettagli necessari per una corretta comprensione del messaggio e non deve essere prolisso. Il mittente può servirsi del supporto al linguaggio di *Markdown_G* disponibile sulla piattaforma per rendere più chiaro il suo messaggio;
- La **categoria** deve essere coerente con l'argomento trattato. Può essere creata una categoria nuova se ritenuto necessario;
- La **notifica** via mail deve includere gli interessati al messaggio;
- Il livello di **privacy** deve sempre essere *Everybody on project* in modo da permettere a tutti i componenti del gruppo di intervenire;
- Data l'esigua dimensione dello *storage_G* che offre la piattaforma nella versione *free* utilizzata dal gruppo e l'assenza di visualizzatori online integrati, il numero e la dimensione degli **allegati** devono essere ridotti quanto più possibile.

3.3.1.2 Procedura per commenti a tasks o sub-tasks Per tali comunicazioni valgono le regole del paragrafo 3.3.1.1. Il contenuto del commento deve riguardare la task che riferisce. Se la discussione si sviluppa in argomenti non più coerenti, i componenti del gruppo SteakHolders devono terminare la discussione, aprire un messaggio secondo le norme descritte al paragrafo 3.3.1.1 e inserire come nuovo commento alla discussione interrotta il link al messaggio creato con l'aggiunta della segnature "[OT]". Questo non vincola il proseguimento della discussione sulla task secondo le norme.

3.3.2 Comunicazioni esterne

Le comunicazioni esterne sono gestite esclusivamente dal Responsabile di Progetto. A tal fine è stato creato l'indirizzo di posta elettronica

steakholders.group@gmail.com

Il Responsabile di Progetto si fa dunque carico di notificare ai restanti membri del gruppo eventuali corrispondenze intrattenute con committenti e proponenti, applicando le norme stabilite al paragrafo 3.3.1.

3.3.3 Riunioni

Qualora fosse necessaria una riunione di tutti o alcuni membri del gruppo sarà compito del Responsabile di Progetto avvisare gli interessati, tramite le procedure stabilite al paragrafo 3.3.1. Il Responsabile di Progetto decide inoltre luogo, date e ora della riunione in base al calendario a sua disposizione. Nel caso in cui qualche membro non risponda entro 24 ore il Responsabile di Progetto dovrà accertarsi con mezzi di comunicazione adeguati che tutti siano stati informati.

Ad ogni riunione verrà prodotto un verbale redatto da un segretario, ruolo svolto a turno da ogni membro del gruppo e deciso di volta in volta dal Responsabile di Progetto. Tale verbale, descritto nel paragrafo 4.2.7.3 dovrà essere reso disponibile per la consultazione a tutti i membri del gruppo.



È inoltre presente un *facilitatore_G*, ruolo svolto a turno da ogni membro del gruppo e deciso di volta in volta dal Responsabile di Progetto, che aiuterà a rispettare l'ordine del giorno senza prolungare eccessivamente la riunione.

3.4 Revisioni di progetto

La vita di questo progetto didattico attraversa le revisioni di progetto specificate in *Piano di Progetto v4.0.0*, esse consistono in *review* (informali) e *audit* (formali) ma le modalità con cui vengono affrontate sono, da parte del gruppo, le medesime, verranno quindi in seguito denominate presentazioni o revisioni. Vengono qui descritte le procedure per le revisioni secondo quanto descritto in IEEE 1028:

0. Entry evaluation: il Responsabile di Progetto prepara una checklist e si assicura che esistano le condizioni che permettano il successo della presentazione;
1. Management preparation: il Responsabile si assicura che tutti i membri del gruppo possano partecipare e l'ambiente di lavoro rispetti i vincoli imposti o suggeriti dal committente. A tal fine è stata elaborata una lista di indicazioni (3.4);
2. Planning the review: il Responsabile deve identificare e confermare gli obiettivi della revisione, inoltre si assicura che tutto il team sia equipaggiato con le risorse necessarie per affrontare la revisione;
3. Overview of review procedures: ci si assicura che tutti i membri del gruppo conoscano gli obiettivi, le procedure e il materiale portato in revisione;
4. Individual preparation: ogni membro del gruppo si prepara individualmente per la presentazione;
5. Group examination: il gruppo si incontra al completo e prova la presentazione confermando il rispetto dei tempi e degli obiettivi;
6. Rework/follow-up: vengono corretti, laddove possibile, gli errori e/o i difetti;
7. Exit evaluation: il Responsabile si assicura che tutti gli output prodotti siano pronti per la revisione.

Le indicazioni su come costruire una presentazione sono le seguenti:

- le diapositive devono avere la stessa densità: il rapporto tra testo, immagini e spazio vuoto deve essere uniforme;
- evitare testi troppo lunghi, immagini piccole o troppo dettagliate;
- non va spiegata la sola teoria: argomentare come è stata implementata nel progetto;
- la lettura delle slide deve avvenire rivolgendosi al pubblico e non rivolti verso la diapositiva; si può usare con parsimonia lo schermo del portatile per avere un riferimento;
- non ripetere quanto scritto nelle diapositive, è inutile;
- il passaggio da una diapositiva all'altra deve avvenire tra un tempo minimo e uno massimo in modo da consentirne la lettura e non annoiare chi ascolta;
- le mani non devono stare nelle tasche;



- ogni slide deve riportare il proprio numero sul totale delle diapositive.

3.5 Procedure per la gestione della Repository

Come repository è stato scelto Git, in particolare il servizio *GitHub_G* come indicato nel capitolo 4.4.1.3.

3.5.1 Repository dei documenti

3.5.1.1 Struttura del repository La struttura del repository, la cui cartella principale chiameremo *root*, è così composta:

- **root/ufficiali/**
Contiene i documenti ufficiali approvati dal Responsabile di progetto.
- **root/modello/**
Contiene i file comuni a due o più documenti.
- **root/documenti/{NomeDelDocumento}/**
Ciascuna cartella descritta da questo percorso contiene i file che vengono utilizzati dal documento {NomeDelDocumento}. In particolare conterrà il file {NomeDelDocumento}.pdf e il file `diario_modifiche.tex`, contenente il diario delle modifiche del documento.

Il nome della cartella {NomeDelDocumento} e il file {NomeDelDocumento}.pdf devono rispettare la convenzione *CamelCase_G*.
- **root/script/**
Contiene gli script di supporto alle operazioni di verifica e compilazione dei documenti.

È raccomandato che tutti i file e le cartelle non contengano spazi nel loro nome. Non devono mai esserci due file o cartelle il cui percorso differisca soltanto per maiuscole/minuscole. Non bisogna inoltre rinominare file o cartelle modificandone soltanto il *case_G* di alcuni caratteri del nome.

3.5.1.2 Branch Il nome utilizzato per le *branch_G* del repository dovrà essere nella forma **nome-branch**, ossia con tutte le lettere minuscole e le parole separate da un trattino.

3.5.1.3 Script di pre-commit Con l'intenzione di ridurre al minimo la presenza di errori nei documenti caricati sul repository verrà utilizzato uno script di **pre-commit**. Lo script, attivato direttamente dal programma `git` nel momento in cui viene eseguito un `commit`, esegue in automatico i comandi `make test` e `make documents` descritti nella sezione 4.4.2.3. Se uno dei comandi fallisce il `commit` viene annullato e all'utente viene mostrato un messaggio di errore.

Per abilitare tale script di **pre-commit** è necessario creare un *link simbolico_G* con il seguente comando:

```
ln -s {percorso assoluto del repository}/script/pre-commit {  
    percorso assoluto del repository}/.git/hooks/pre-commit
```



3.5.2 Repository del codice

3.5.2.1 Struttura del repository La struttura del repository, la cui cartella principale chiameremo `root`, è così composta:

- **root/extra/**
Contiene script e file di supporto utilizzati per le attività di codifica.
- **root/test/**
Contiene i file relativi ai test d'unità della libreria MaaP.
- **root/lib/**
Contiene i file della libreria MaaP usata dalla Backend del progetto, posizionati in una cartella `lib` secondo la convenzione dei moduli di Node.js.
- **root/node_modules/**
Contiene le librerie richieste dalla libreria MaaP.
- **root/lib/model/**
Contiene i file che gestiscono i dati utilizzati dall'applicazione e l'interfacciamento con il database.
- **root/lib/model/dslmodel/**
Contiene i file che gestiscono la gestione dei file di configurazione dsl e l'interfacciamento di questi con il database.
- **root/lib/view/**
Contiene i file che servono da template usati per visualizzare i dati all'utente.
- **root/lib/controller/**
Contiene i file che gestiscono la logica con cui vengono elaborate le richieste inviate all'applicazione.
- **root/lib/controller/middleware/**
Contiene i file che hanno il ruolo di middleware nel framework Express.
- **root/lib/controller/service/**
Contiene i file che hanno il ruolo di service nel framework Express.
- **root/lib/utils/**
Contiene i file di generica utilità, che non rientrano nella classificazione tra model, view e controller.
- **root/scaffold/**
Contiene i file che verranno utilizzati come base di partenza (scaffold) per il progetto dello sviluppatore che utilizzerà il prodotto.
- **root/scaffold/collections/**
Contiene i file di configurazione delle collection utilizzate dall'applicazione scaffold.
- **root/scaffold/app/**
Contiene i file relativi al Frontend dell'applicazione scaffold.
- **root/scaffold/app/view/**
Contiene i file statici html usati dal Frontend dell'applicazione scaffold.



- **root/scaffold/app/style/**
Contiene i file statici di stile usati dal Frontend dell'applicazione scaffold.
- **root/scaffold/app/scripts/**
Contiene i file dell'applicazione AngularJS usata nel Frontend.
- **root/scaffold/app/scripts/services/**
Contiene i file dei service di AngularJS usata nel Frontend.
- **root/scaffold/app/scripts/controllers/**
Contiene i file dei controller di AngularJS usata nel Frontend.
- **root/scaffold/app/bower_components/**
Contiene i file delle librerie utilizzate dal Frontend dell'applicazione scaffold.
- **root/scaffold/node_modules/**
Contiene i file delle librerie utilizzate dal Backend dell'applicazione scaffold.

È raccomandato che tutti i file e le cartelle non contengano spazi nel loro nome. Non devono mai esserci due file o cartelle il cui percorso differisca soltanto per maiuscole/minuscole. Non bisogna inoltre rinominare file o cartelle modificandone soltanto il *case* di alcuni caratteri del nome.

3.5.2.2 Branch Sono previsti due branch ufficiali:

- **stable:** contiene sempre una versione del codice che rispetta i controlli automatici effettuati dallo script 'make test-report', descritto alla sezione 4.4.3.4. Gli sviluppatori non devono effettuare modifiche direttamente su questo branch;
- **master:** è il branch usato dagli sviluppatori per la codifica. Ad ogni commit, se il codice supera determinati test, Jenkins (descritto alla sezione 4.4.3.5) propagherà le modifiche anche sul branch *stable*.

3.5.2.3 Script di pre-commit Con l'intenzione di ridurre al minimo la presenza di errori nel codice caricato sul repository verrà utilizzato uno script di **pre-commit**. Lo script, attivato direttamente dal programma **git** nel momento in cui viene eseguito un commit, esegue in automatico il comando **make jshint** descritto nella sezione 4.4.3.4 e controlla che non si stia facendo un commit alla branch *stable*. Se il comando trova un'anomalia il commit viene annullato e all'utente viene mostrato un messaggio di errore.

Per abilitare tale script di **pre-commit** è necessario creare un *link simbolico* con il seguente comando:

```
ln -s {percorso assoluto del repository}/extra/pre-commit {  
    percorso assoluto del repository}/.git/hooks/pre-commit
```

3.6 Condivisione file

Per condividere internamente al gruppo dei file che non necessitano di versionamento verranno utilizzate le piattaforme:

- **Google Drive** (<http://drive.google.com>), con l'account descritto nella sezione 3.3.2;

- **TeamworkPM**, descritta nella sezione 3.8.

3.7 Modalità di consegna

I documenti verranno consegnati secondo le date prestabilite dal committente e corrispondenti alle *milestone_G*, la modalità di consegna dovrà avvenire utilizzando **Google Drive**, creando una cartella apposita, rendendola pubblica e inviando l'email contenente il link di pubblicazione alle seguenti email: tullio.vardanega@math.unipd.it, rcardin@math.unipd.it.

3.8 Strumenti per il coordinamento

Per coordinare le attività, gli eventi, per le comunicazioni e per il conteggio delle ore viene utilizzata la piattaforma **Teamwork Project Manager** (<http://teamworkpm.net>):

- Offre un'elevata portabilità ed accessibilità essendo una piattaforma online;
- Offre gratuitamente i servizi necessari;
- Mette a disposizione delle *API_G* per comunicare con la piattaforma ed estenderne eventualmente le funzionalità.

L'indirizzo per accedere all'ambiente di lavoro riservato al gruppo è

<https://steakholders.teamworkpm.net>

3.8.1 Calendario condiviso

Il gruppo si avvale del calendario di Google, *Google Calendar_G*, come calendario condiviso. Tramite la propria mail personale ogni membro può accedere al calendario in modalità lettura/scrittura. Ogni membro è tenuto a *segnalare qui i periodi di non disponibilità* nel completare i lavori assegnati, selezionando le ore o i giorni di interesse e marcandoli con la dicitura [ND], si possono aggiungere delle note se si ritiene ve ne sia necessità.

Il responsabile annoterà qui gli impegni comuni al gruppo quali riunioni ed incontri, specificando orario e luogo.

3.8.2 Strumenti per la gestione dei ticket

Per la gestione dei ticket si utilizzerà il sistema di task offerto da *TeamworkPM_G*, utilizzabile unicamente attraverso le procedure descritte nella sezione 3.1.

3.8.3 Strumenti per la gestione del piano di lavoro

Al fine di pianificare le attività da svolgere per lo sviluppo del progetto, il gruppo si è affidato alla piattaforma *TeamworkPM_G* per i seguenti motivi:

- Genera automaticamente un grafico *Gantt_G* che visualizza e permette di modificare i task pianificati, gestendo anche le relative dipendenze;



- Permette di esportare il grafico $Gantt_G$ in un formato compatibile con $GanttProject_G$ e $Microsoft Project_G$;
- Permette di gestire le $Milestone_G$;
- Permette di registrare il tempo di lavoro trascorso su ogni task.

3.8.4 Gestione degli eventi

Gli eventi di interesse collettivo vengono inseriti nel calendario dall'Amministratore sempre e solo nel *Calendario condiviso* descritto in 3.8.1.

Le tipologie di eventi sono:

- **Riunioni:** vanno programmate almeno con 48 ore di anticipo tenendo conto della disponibilità dei membri. Ogni evento riunione avrà un ora di inizio e fine, un luogo e la lista dei membri che hanno confermato la partecipazione. L'*ordine del giorno* dovrà essere compilato nella descrizione dell'evento;
- Le **Revisioni** previste dal committente;
- **Non disponibilità:** un membro del gruppo dichiara di non essere disponibile a svolgere attività legate al progetto. Per creare un evento di *Non disponibilità* bisogna, oltre a compilare i soliti campi (titolo, descrizione, data e ora), impostare la visibilità a *tutti i membri della propria azienda*, selezionare la categoria *Non disponibilità* e segnare la persona interessata. In questo modo tutti i membri del gruppo SteakHolders potranno visualizzare tale evento. Il titolo sarà nel formato **[ND] {nome membro}** ossia le iniziali di *Non disponibilità* racchiuse tra parentesi quadre, seguite da il nome del membro del gruppo. Questo formato permetterà di individuare con immediatezza i giorni nei quali non è possibile fissare delle riunioni.



4 Processi organizzativi

4.1 Processo di Pianificazione

Come convenzione il gruppo ha scelto una soglia oraria giornaliera in cui lavorare al progetto. In particolare l'orario di lavoro giornaliero è da considerarsi dalle 9:00 alle 12:00 e dalle 13:00 alle 17:00, per un massimo di 7 ore. Le ore impiegate oltre queste soglie sono considerate ore di **straordinario**. In particolare al Responsabile è raccomandato pianificare due ore di lavoro al giorno, per un totale di 10 ore settimanali.

Il Responsabile, nell'assegnare, task terrà conto di quanto inserito nel calendario, come indicato nel capitolo 3.8.1, e di quanto indicato nel capitolo 3.2 riguardo la rotazione dei ruoli.

4.2 Processo di Documentazione

In questo capitolo vengono descritte le regole adottate per la stesura di tutti i documenti necessari al corretto svolgimento del progetto. Dopo un'attenta valutazione delle varie alternative si è scelto di utilizzare il linguaggio di markup **L^AT_EX**. La scelta poggia su diverse motivazioni:

- È un linguaggio conosciuto da tutti i membri del gruppo;
- Presenta un numero di librerie molto vaste che permettono di realizzare qualsiasi tipo di documento e di personalizzarlo liberamente in modo dettagliato;
- La stesura viene fatta su file testuali e non binari, il che agevola il suo versionamento e permette una facile gestione su repository. Inoltre permette di utilizzare facilmente gli script che producono parti di documenti;
- È un linguaggio molto diffuso nel mondo informatico e scientifico;
- Si può produrre file in formato PDF_G con estrema facilità grazie al compilatore specifico;
- Permette una facile gestione degli indici e dei glossari.
- Permette di inserire diagrammi di Gantt, rendendo il loro codice tracciabile e facilmente modificabile.

4.2.1 Template

Per la stesura dei documenti è stato creato un apposito template **L^AT_EX** in cui compaiono tutte le convenzioni sullo stile descritte nel documento corrente. Questo è stato fatto per agevolare il più possibile i componenti del gruppo che andranno a produrre i documenti, in modo che chiunque lavori ad un documento debba preoccuparsi solamente della scrittura del testo senza doversi preoccupare della sua formattazione.

4.2.2 Struttura del documento

4.2.2.1 Prima pagina La prima pagina del documento deve essere formattata nel modo seguente:



- **Logo** del gruppo, visibile come primo elemento centrato orizzontalmente in alto;
- Nome del documento, visibile subito dopo il logo, centrato orizzontalmente e marcato come **titolo**;
- Una **tabella** descrittiva, visibile subito dopo il titolo, centrata orizzontalmente e contenente le seguenti informazioni:
 - **Versione** del documento, indicata come da norma;
 - I nomi e cognomi dei **redattori** del documento;
 - I nomi e cognomi dei **verificatori** del documento;
 - Il nome e cognome del **responsabile** di progetto, che ha approvato il documento;
 - Il tipo di **uso** del documento;
 - La **lista di distribuzione** del documento.
- Una **descrizione** testuale sommaria del documento, centrata orizzontalmente ed il più possibile sintetica.

4.2.2.2 Registro delle modifiche È preferibile che la numerazione sia continua. La pagina (o le pagine) che seguono devono contenere uno storico di questo documento, in cui verranno riportate tutte le modifiche apportate ad esso. Il registro delle modifiche deve essere composto da una tabella di tante righe quante sono le modifiche apportate ed un numero di colonne pari agli elementi seguenti:

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

4.2.2.3 Indice Ciascun documento deve contenere un suo indice, in modo da agevolare la consultazione e permettere una lettura *ipertestuale* 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 di volta in volta ripartire. Per quanto riguarda le appendici esse non devono essere numerate ma indicate da una lettera maiuscola che di appendice in appendice verrà incrementata a partire dalla lettera A seguendo l'ordine alfabetico internazionale.

4.2.2.4 Formattazione generale delle pagine Ciascuna pagina deve rispettare tutti i margini orizzontali e verticali previsti dal template. Ciascuna pagina, ad eccezione della prima, deve contenere un'**intestazione** (in alto) ed un **piè di pagina**. Ciascuna intestazione dovrà essere formattata nel modo seguente:

- Logo di intestazione del gruppo disposto a sinistra;
- Nome del gruppo affiancato al logo;
- Titolo del progetto corrente assegnato subito sotto il nome del gruppo;



- Numero e titolo della sezione corrente del documento disposto a destra dell'intestazione.

Il piè di pagina sarà strutturato invece nel seguente modo:

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

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

4.2.3 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 adottata una numerazione della forma

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

Riguardo a $\{X\}$:

- Parte da 1;
- Viene incrementato dal Responsabile di Progetto quando approva il documento;
- Quando viene incrementato y e z sono riportati a zero.
- Limitato superiormente dal numero di revisioni.

Riguardo a $\{Y\}$:

- Parte da 0;
- Viene incrementato dal verificatore del documento ad ogni verifica;
- Quando viene incrementato z è riportato a zero;
- Non è limitato superiormente.

Riguardo a $\{Z\}$:

- Parte da 0;
- Viene incrementato ad ogni modifica dalla persona che modifica il documento;
- Non è limitato superiormente.

4.2.4 Procedura per la definizione di macro personalizzate

4.2.4.1 Costanti Tutte le macro \LaTeX utilizzate come costanti devono iniziare con la lettera maiuscola. Se il nome della macro è composto da più parole unite allora ciascuna deve avere l'iniziale maiuscola e il resto delle lettere minuscolo. Per esempio, `\NomeDellaCostante` è un nome di costante valido.



- Le costanti **utilizzate da due o più documenti** (es. il nome del gruppo) devono essere definite all'inizio del file `root/modello/global.tex`;
- Le costanti **utilizzate da un solo documento** (es. titolo) devono essere definite nella cartella del documento, all'inizio del file `local.tex`;

4.2.4.2 Funzioni Tutte le macro L^AT_EX utilizzate come funzioni devono iniziare con la lettera minuscola. Se il nome della macro è composto da più parole unite allora ciascuna deve avere l'iniziale maiuscola e il resto delle lettere minuscolo. Per esempio, `\nomeDellaFunzione` è un nome di funzione valido.

- Le funzioni **utilizzate da due o più documenti** (es. il comando che genera la pagina di copertina) devono essere definite nella seconda parte del file `root/modello/global.tex`;
- Le funzioni **utilizzate da un solo documento** devono essere definiti nella cartella del documento che lo utilizza, nella seconda parte di `local.tex`.

4.2.5 Norme tipografiche e convenzioni

Ciascun documento deve rispettare le norme tipografiche definite nella seguente sezione, in modo da rendere il tutto uniforme. Per le situazioni che non sono coperti dalle seguenti norme è preferibile riferirsi a quanto stabilito dall'Accademia della Crusca (<http://www.accademiadellacrusca.it>).

4.2.5.1 Punteggiatura

- Ciascuna frase deve essere separata da un **punto** o da un **punto virgola**;
- Dopo ogni punto vi dev'essere uno spazio prima dell'inizio della frase successiva;
- Ogni frase (ad esclusione di quelle dopo il punto e virgola) deve iniziare con una **lettera maiuscola**;
- Per i **punti di domanda** e i **punti esclamativi** valgono le stesse regole per i punti;
- Dopo ogni **virgola** deve esserci uno spazio che separa la parte restante della frase;
- Dopo ogni **apostrofo** non deve esserci un carattere di spaziatura;
- Ogni elemento di un elenco puntato deve terminare con un punto e virgola, ad eccezione dell'ultimo che deve terminare con un punto;
- Ogni elemento di un elenco puntato deve iniziare con una lettera maiuscola.

4.2.5.2 Stile del testo Ogni parola di **glossario** deve essere marcata con una *G* maiuscola a pedice:

repository_G

Ciascuna parola chiave deve essere evidenziata in grassetto. Il **corsivo** dev'essere utilizzato nei seguenti casi:

- **Citazioni**;

- **Abbreviazioni;**
- **Riferimenti** ad altri documenti;
- **Parole particolari** solitamente poco usate o conosciute;
- **Nomi di società o aziende;**
- **Nome di programmi o *framework_G*;**
- **Ruoli di progetto;**
- **Nome del progetto;**
- **Nome del gruppo.**

4.2.5.3 Elenchi puntati È raccomandato che ciascun elenco puntato sia graficamente rappresentato da un *pallino* nella gerarchia principale e da un trattino nella gerarchia secondaria. Ogni elenco puntato corrisponde ad un concetto che va espresso in modo sintetico. È normalmente preferibile usare elenchi puntati piuttosto che frasi lunghe e discorsive.

4.2.5.4 Formati

- Quando ci si riferisce a una **data**, se non diversamente specificato, bisogna applicare il formato descritto dallo standard ISO 8601:

AAAA-MM-GG

dove:

- *AAAA* si riferisce all'anno utilizzando quattro cifre;
 - *MM* si riferisce al mese utilizzando due cifre;
 - *GG* si riferisce al giorno utilizzando due cifre.
- Per indicare un **orario** si usa il formato ventiquattrore nel modo seguente:

hh:mm

dove:

- *hh* si riferisce all'ora e può assumere valori da 0 a 23;
 - *mm* si riferisce al minuto e può assumere valori da 0 a 59.
- Ogni **nome** di un membro del gruppo va indicato con il *Nome* seguito dal *Cognome*, a meno che il contesto non richieda diversamente (es. un elenco ordinato per cognome);
 - Quando ci si riferisce a **nomi dei file** si deve usare il carattere **monospace**.

4.2.5.5 Sigle

Vengono previste le 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;
- **DP** = Definizione di prodotto;
- **RR** = Revisione dei requisiti;
- **RP** = Revisione di progettazione;
- **RQ** = Revisione di qualifica;
- **RA** = Revisione di accettazione.

4.2.5.6 Riferimenti a documenti Quando è necessario riferirsi ai contenuti di un altro documento bisogna specificarne il nome completo e la versione.

4.2.5.7 Nomenclatura entità e relazioni I nomi dei package seguono la specifica *upper camel case*. Nel caso un package appartenga ad un altro package, il nome sarà formato da il nome del padre posto in prefisso secondo lo schema `NomePackagePadre::NomePackageFiglio`.

I nomi delle classi devono riportare come prefisso il nome del package di appartenenza separati da una coppia di due punti, seguendo lo schema `NomePackage::NomeClasse`. I nomi delle classi seguono la specifica *upper camel case*.

I nomi di:

- metodi;
- campi dati;
- parametri;
- variabili

seguono la specifica *lower camel case*.

I nomi delle costanti vanno riportati con tutte le lettere maiuscole.

4.2.6 Tabelle e immagini

4.2.6.1 Tabelle Ciascuna tabella deve essere allineata al centro orizzontalmente e deve contenere sotto di essa la propria didascalia, per agevolarne il tracciamento. In questa didascalia deve comparire il numero della tabella, che dev'essere incrementale in tutto il documento, e una breve descrizione del suo contenuto.



4.2.6.2 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 stacco 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.

4.2.7 Classificazione di documento

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

4.2.7.2 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 *verifica_G* e *validazione_G*.

4.2.7.3 Verbali Con verbale ci si riferisce ad un documento redatto da un segretario in occasione di riunioni interne al gruppo e di incontri con i proponenti. Un verbale viene redatto una prima volta e non subisce successive modifiche, pertanto non é previsto versionamento. 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.

Nel primo paragrafo Informazioni Generali vanno 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.

4.2.8 Verifica e validazione

La *verifica_G* del documento deve essere eseguita manualmente da parte di un *Verificatore*, scelto dal *Responsabile* tra i membri del gruppo secondo il principio di assenza di conflitti d'interesse, ossia colui che viene chiamato a verificare un determinato componente non può aver in alcun modo partecipato alla creazione. Le verifiche automatizzate sui documenti, dove previste, difficilmente sono esaustive e possono tralasciare delle anomalie. Per eseguire la verifica è necessario controllare che il documento rispetti tutte le norme descritte nelle *Norme di Progetto*.

La *validazione_G* del documento consiste nel controllare che il documento abbia *il giusto contenuto*, è un compito che va oltre la semplice verifica delle norme. Richiede una conoscenza a priori del contenuto e degli scopi del documento.



4.2.8.1 Procedura per la verifica La verifica di un documento è composta come minimo dai seguenti passi:

1. **Controllo ortografico e del periodo:** con un controllo walktrough bisognerà analizzare i periodi ed eventualmente correggerne la forma, oltre a controllare la presenza di errori ortografici con l'aiuto dello script `make test` (vedi sezione 4.4.2.3);
2. **Verifica delle proprietà di glossario:** con l'aiuto dell'apposito script `make test-glossary` (vedi sezione 4.4.2.3) ci si deve assicurare che ogni termine marcato come glossario sia effettivamente definito nel glossario;
3. **Riportare gli errori frequenti:** per migliorare ciclicamente il processo di verifica, verranno riportati gli errori frequenti. In questo modo sarà più facile eseguire, negli incrementi successivi, controlli di tipo inspection.
4. **Segnalazione delle anomalie riscontrate:** il *Verificatore* deve aprire un ticket secondo le modalità descritte nella sezione 3.1.1.

4.2.9 Approvazione

L'**approvazione** di un documento deve essere eseguita dal *Responsabile di Progetto*. Consiste in un accertamento del percorso di vita del documento e serve a fare in modo che soltanto documenti verificati e validati vengano distribuiti ufficialmente all'esterno del gruppo.

4.2.10 Glossario

Il *Glossario* contiene le definizioni di tutte le parole che possono creare ambiguità o essere fraintese. Nei documenti tali parole saranno marcate con una G pedice.

Di pari passo con la stesura dei documenti sarà compito degli autori mantenere aggiornato il *Glossario*, inserendo di volta in volta i termini e le relative definizioni. È preferibile inserire nel glossario perlomeno la voce priva di descrizione, per poterla successivamente completare più facilmente.

4.3 Processo di Verifica

Le *Software Quality Management Techniques_G* adottate dal gruppo sono suddivise in quattro categorie. Di seguito ne vengono date le definizioni. Tuttavia non sempre si tratta di classi nettamente distinte e la loro applicazione molto spesso prevede sovrapposizioni. Nella pianificazione si cerca di assegnare alla verifica almeno una fetta pari al 30% delle ore totali.

4.3.1 Statiche

Consiste nello studiare la documentazione ed il software senza istanze di esecuzione del sorgente. Queste tecniche possono includere attività *people-intensive* o di *analisi* condotte da singoli individui con o senza l'assistenza di *tools_G* automatizzati.

4.3.1.1 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 pertanto se ne sconsiglia l'attuazione. 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 Qualità*.

4.3.1.2 Inspection

Si tratta di una lettura mirata e strutturata, volta a localizzare l'errore con il minor costo possibile. Richiede una buona padronanza di tutti gli strumenti utilizzati e conoscenza di tutti i documenti di progetto. Solitamente viene effettuata da un singolo individuo. La ricerca mirata si basa sugli errori ricorrenti, gli script da utilizzare sono descritti in 4.4.2.3. Le anomalie individuate vanno segnalate secondo le modalità descritte in 3.1.1. Le anomalie più frequenti sono:

- Utilizzo di comandi locali L^AT_EX deprecati, ossia quei comandi che erano stati definiti alla prima creazione del *repository_G* ma che poi sono stati ridefiniti;
- Utilizzo di caratteri di underscore senza prima anteporre una backslash;
- Errori ortografici come rappresentare la *È* con una *e* maiuscola apostrofata;
- Inserimento di parole comuni come placeholder, e.g. ciao, blabla, asd . . . ;
- Formato delle date errato o non conforme alle *Norme di Progetto v4.0.0*;
- Formato dei nomi errato, le *Norme di Progetto v4.0.0* indicano prima il nome e poi il cognome;
- Elenchi puntati non conformi alle norme, in particolare i punti spesso non terminano con il punto e virgola.

4.3.2 People-intensive

Sono attività che coinvolgono almeno due persone impegnate a svolgere compiti anche parzialmente non automatizzabili. Solitamente l'efficienza non è massima in quanto si paga il coordinamento tra le persone. Le risorse necessarie sono checklists e i risultati dei tests e delle altre tecniche di analisi.

4.3.3 Analitiche

Le tecniche analitiche comprendono tutte quelle azioni volte a valutare criticamente un componente del prodotto. Tali tecniche si servono di metodi come l'analisi della complessità, degli algoritmi e del controllo di flusso. L'analisi di complessità è utile per valutare quanto un componente è articolato da progettare, implementare, testare o mantenere. Il controllo di flusso è volto al riconoscimento delle anomalie e può essere usato a supporto di altre attività. Infine, l'analisi degli algoritmi è fondamentale in quei software in cui vi è una componente algoritmica importante e vi è la necessità di assicurare output consistenti e corretti.

4.3.4 Dinamiche

Le tecniche dinamiche vengono generalmente utilizzate durante lo sviluppo e la manutenzione. Si eseguono dei test ripetibili basati sull'effettiva esecuzione del codice. È quindi opportuno costruire un set di strumenti per riprodurre insiemi di *input* e portare il software in uno stato iniziale, al fine di verificare velocemente che gli output siano quelli attesi. Un file di *log_G* permetterà, in base al contenuto, di ricostruire le sequenze di operazioni effettuate. Di seguito vengono elencati i test che si andranno ad effettuare in ordine di granularità.

Il *proponente_G*, nel corso del *Verbale* del 2013-12-05, ha indicato che dovrà essere testato almeno il 70% del codice prodotto.

4.3.4.1 Test di unità

Si tratta di testare il funzionamento delle *unità_G* tramite l'utilizzo di *stub_G*, *test driver_G*, e *logger_G*. Un'*unità_G* consiste nella più piccola quantità di software che è utile verificare singolarmente. Questi test dovrebbero procedere quanto più in parallelo, assegnando priorità alle *unità_G* che mostrano dei risultati utilizzabili per definire dei *prototipi_G*. Il corretto funzionamento di tutte le unità riduce al minimo la presenza di errori di programmazione e permette di integrare queste componenti secondo le specifiche di progetto con la garanzia che esse funzionino.

4.3.4.2 Test di integrazione

Consiste nel test di una parte di due o più *unità_G* e ne valuta globalmente i risultati. Le componenti non ancora sviluppate vanno simulate con dei sostituti fittizi.

4.3.4.3 Test di sistema

Consiste nella validazione del sistema per accertare la copertura dei requisiti software e il suo collaudo viene supervisionato dal committente per mostrare la conformità del prodotto.

4.3.4.4 Test di regressione

Consiste nell'eseguire nuovamente i test riguardanti le componenti software che hanno subito modifiche. Tale operazione è aiutata dal tracciamento, il quale permette di individuare e ripetere facilmente i test di unità, di integrazione e di sistema che sono stati potenzialmente influenzati dalla modifica.

4.3.4.5 Test di accettazione

Si tratta del collaudo del prodotto in presenza del proponente. Al superamento di tale collaudo segue il rilascio ufficiale del prodotto sviluppato.



4.4 Strumenti

4.4.1 Ambiente e strumenti generali

4.4.1.1 Sistema operativo Il progetto verrà sviluppato su sistemi **Linux**, per la facilità con cui è possibile preparare l'ambiente di lavoro adatto allo sviluppo dell'applicazione descritta dal capitolato. In particolare è raccomandato il sistema operativo **Ubuntu** ≥ 12.04 .

4.4.1.2 Codifica dei caratteri Per assicurarsi la corretta visualizzazione dei caratteri accentati tutti i file testuali presenti nel *repository_G* devono essere memorizzati con la codifica **UTF-8_G**.

4.4.1.3 Strumenti per il versionamento Per il versionamento dei documenti e del codice viene usato **Git** $\geq v1.7.9.5$ (<http://git-scm.com/>). Si utilizzerà il servizio **GitHub** per creare e gestire i repository privati utilizzati dal gruppo.

4.4.2 Strumenti per la produzione dei documenti

4.4.2.1 Scrittura Per la stesura dei documenti verrà utilizzato il linguaggio **L^AT_EX** (<http://www.latex-project.org>). Per la stesura sono raccomandati i seguenti editor:

- **TexMaker** ≥ 3.2 (<http://www.xmlmath.net/texmaker>)
- **Kile** $\geq 2.1.3$ (<http://kile.sourceforge.net>)

L'output dei documenti sarà in formato PDF_G e verrà prodotto attraverso il comando **pdflatex** (versione $\geq 3.1415926-1.40.10-2.2$). Per velocizzare questa operazione è stato predisposto il comando **make documents**, il cui utilizzo è descritto nella sezione 4.4.2.3.

4.4.2.2 Strumenti per il controllo ortografico Per aiutare il controllo ortografico verrà utilizzato il software *Aspell* (<http://aspell.net>, versione $\geq 0.60.7$) con il dizionario italiano.

Per controllare un file **L^AT_EX** con *Aspell* l'utilizzo da terminale è il seguente:

```
aspell --lang it --mode tex --encoding utf-8 --personal root/  
script/aspell_personal.txt --repl root/script/  
aspell_replacements.txt check {nome del file da controllare}
```

Per comodità è stato predisposto il comando **make check**, come descritto nella sezione 4.4.2.3.

Quando *Aspell* segnala un errore su una parola:

- Se la parola ha un errore ortografico, correggerla scegliendo una delle parole proposte da *Aspell* oppure usando il comando **“rimpiazza”**;
- Se la parola è scorretta e deve essere modificata tutta la frase, selezionare il comando **“abbandona”** e fare la modifica utilizzando una IDE_G ;
- Se l'errore segnalato da *Aspell* è un falso positivo, selezionare il comando **“aggiungi”**.



4.4.2.3 Script di Makefile Per agevolare molte operazioni è stato predisposto uno script *Makefile_G*. Al momento sono previsti i seguenti comandi:

- **make test**

Per ogni file *.tex contenuto nella cartella e nelle sue sotto-cartelle:

- Verifica che i file siano memorizzati con la codifica *UTF-8_G*;
- Verifica con *Aspell* che le parole utilizzate nei documenti siano comprese nel dizionario italiano di *Aspell* o nel dizionario personalizzato *root/script/aspell_personal.txt*.

- **make documents**

Compila tutti i documenti presenti nelle sotto-cartelle.

- **make test-glossary**

Controlla che tutti i termini marcati con il comando *glossario{...}* siano definiti da un corrispondente comando *definizione{...}*.

- **make test-regex**

Per ogni regola della forma *grep_test {RegExp} {Messaggio} {File}* definita internamente allo script controlla se ci sono occorrenze dell'espressione regolare *{RegExp}* nei file *{File}*. Per ciascuna occorrenza visualizza il messaggio *{Messaggio}* seguito dal nome del file e la linea in cui è stata trovata l'occorrenza. È raccomandato usarlo per la verifica dei documenti.

- **make gulpease**

Per ogni documento *PDF_G* calcola l'indice di leggibilità *Gulpease_G* e lo visualizza. Richiede di avere installato il framework *python-nltk* e il programma *pdftotext*.

È possibile eseguire i primi due comandi dalle cartelle dei documenti e dalle loro cartelle superiori, fino alla cartella principale del repository. I restanti comandi si possono eseguire soltanto dalla cartella principale del repository.

4.4.2.4 UML Per la produzione dei diagrammi verrà utilizzata la piattaforma *Lucidchart* (<https://www.lucidchart.com>).

È stato valutato anche il software *Astah Professional* (<http://astah.net/editions/professional>), utilizzabile con una licenza accademica gratuita, ma è stato preferito *Lucidchart* per la facilità con cui è possibile collaborare online.

Per automatizzare il reperimento delle immagini relative ai diagrammi si è creato uno script apposito il quale scaricherà nella cartella */AnalisiDeiRequisiti/UML* i diagrammi presenti in *Lucidchart*.

Per utilizzarlo bisogna installare *ngrok_G* ≥ 1.3 e per la prima esecuzione, eseguire le istruzioni nel file *download_UML.py* nella cartella */AnalisiDeiRequisiti*.

Per le esecuzioni successive basterà utilizzare dal terminale il comando: *python download_UML.py*.

4.4.2.5 Script download use case e requisiti In *Requisteak* sono presenti le descrizioni degli use case e i requisiti, per effettuarne il download in formato *.tex* è stato predisposto

uno script che andrà a prendere tali informazioni e creerà i corrispettivi capitoli.
Dopo essersi posizionati all'interno della cartella `/AnalisiDeiRequisiti`, eseguire lo script con il comando da terminale: `python download_requisiti.py`

4.4.2.6 Script di produzione del Gantt e dei grafici Per agevolare la produzione dinamica del *Piano di progetto* è stato scritto un apposito script che genera automaticamente i diagrammi di $Gantt_G$ e i grafici relativi alla suddivisione del lavoro e al prospetto economico. Tale script sfrutta le API_G messe a disposizione da $TeamworkPM_G$ che permettono di scaricare tutto ciò che riguarda il progetto aperto sulla piattaforma, includendo $milestone_G$, persone, TaskList e ore. Lo script si preoccupa, attraverso le informazioni prelevate, di :

- Generare in output le tabelle delle ore per ruolo divise per ogni $milestone_G$ pianificata in $TeamworkPM_G$ in un corrispondente file `.tex`,
- Generare per ogni $milestone_G$ pianificata, un file `.tex` con il corrispondente diagramma di $Gantt_G$;
- Generare i grafici a torta riguardanti le ore per ruolo in ogni $milestone_G$;
- Generare i grafici a torta riguardanti i costi per ruolo in ogni $milestone_G$ e in totale;
- Generare i grafici a barre relativi a ore per ruolo per componente corrispondenti ad ogni $milestone_G$ e in totale;
- Effettuare la verifica, con segnalazione di errore se l'esito è negativo e descrizione di tali segnalazioni in un file `.txt`, del rispetto delle norme nella pianificazione :
 - Verifica che non siano presenti per ogni persona dei task sovrapposti;
 - Controlla i vincoli sulle ore per persona;
 - Controlla i vincoli sul preventivo;
 - Verifica che la rotazione dei ruoli sia corretta;
 - Controlla che le ore di verifica siano maggiori del 30%.

Un esempio di parte di output relativo alla funzionalità di verifica delle norme nella pianificazione è dato dalla figura:

```
Controllo vincoli sul preventivo
-----
- amministratore: 106 ore, per un costo di 2120 €
- programmatore : 126 ore, per un costo di 1890 €
- responsabile : 35 ore, per un costo di 1050 €
- analista : 30 ore, per un costo di 750 €
- verificatore : 220 ore, per un costo di 3300 €
- progettista : 182 ore, per un costo di 4004 €
Totale : 13114 €
Ore di verifica : 31.5 %

Controllo vincoli sulle persone
-----
- Serena Girardi: 102 ore di lavoro
- Enrico Rotundo: 96 ore di lavoro
- Nicolò Tresoldi: 98 ore di lavoro
- Federico Poli: 99 ore di lavoro
- Giacomo Fornari: 104 ore di lavoro
- Gianluca Donato: 102 ore di lavoro
- Luca De Franceschi: 102 ore di lavoro

ATTENZIONE: Enrico Rotundo ha troppi pochi ruoli (5). Dovrebbe fare più responsa
bile.
ATTENZIONE: Nicolò Tresoldi ha troppi pochi ruoli (5). Dovrebbe fare più analisi
```

Figura 8: Esempio script

Per poterlo utilizzare, abilitare le API nel proprio account *TeamworkPM_G* per ottenere il token da utilizzare alla richiesta dello script.

Eseguirlo da terminale posizionandosi nella cartella */PianoDiProgetto* con il comando `python download_gantt.py`.

4.4.3 Ambiente e strumenti per la verifica e validazione

4.4.3.1 Strumenti per l'analisi statica Gli strumenti utilizzati per effettuare l'analisi statica del prodotto sono stati integrati in uno script *Makefile_G*, descritto alla sezione 4.4.3.4.

Sono stati utilizzati i seguenti software:

- **JSHint** (<http://www.jshint.com>), *tool_G* che aiuta a controllare che il codice sorgente rispetti le regole descritte nella sezione 2.3.2;
- **Plato** (<https://github.com/es-analysis/plato>), *tool_G* usato per misurare e visualizzare le metriche descritte nel *Piano di Qualifica v4.0.0*;
- uno script che si appoggia a **complexity-report** (<https://github.com/philbooth/complexity-report>), *tool_G* usato per misurare le metriche descritte nel *Piano di Qualifica v4.0.0*.

4.4.3.2 Strumenti per l'analisi dinamica Gli strumenti utilizzati per effettuare l'analisi dinamica del prodotto sono stati integrati in uno script *Makefile_G*, descritto alla sezione 4.4.3.4.

Sono stati utilizzati i seguenti software:

- **Mocha** (<https://github.com/visionmedia/mocha>) Mocha è una libreria ricca di funzionalità per l'esecuzione di test Javascript, con esecuzione di quest'ultimi asincrona ed in serie, permettendo segnalazioni flessibili ed accurate.



- **Karma** (<https://github.com/visionmedia/mocha>) Karma è un ambiente di testing utile per effettuare test lato-browser. Richiede una libreria di testing come Mocha per l'esecuzione dei test.
- **Istanbul** (<http://gotwarlost.github.io/istanbul/>) Istanbul è uno strumento che effettua statement, branch, function e line coverage.

4.4.3.3 Strumenti per la validazione Per la validazione delle pagine web si utilizzeranno gli strumenti messi a disposizione da $W3C_G$ alle pagine (<http://validator.w3.org>) e (<http://jigsaw.w3.org/css-validator>).

4.4.3.4 Script di Makefile Per agevolare molte operazioni è stato predisposto uno script $Makefile_G$ per le attività di codifica. Al momento sono previsti i seguenti comandi:

- **make prepare**
Prepara l'ambiente di lavoro di Node.js, principalmente scaricando e installando le librerie richieste dal progetto.
- **make clean**
Annulla le operazioni effettuate da **make prepare**.
- **make test**
Esegue i principali script di analisi statica e dinamica. In particolare:
 - Controlla la formattazione del codice con JSHint;
 - Esegue i test d'unità con Mocha e con Karma.
- **make test-report**
Esegue gli script di analisi statica e dinamica, inoltre memorizza l'esito in dei file di report interpretabile automaticamente da Jenkins. In particolare:
 - Controlla la formattazione del codice con JSHint;
 - Effettua le misurazioni sul software con Plato;
 - Esegue i test d'unità con Mocha e Karma;
 - Misura la copertura del software con Istanbul.
- **make jshint**
Controlla la formattazione del codice con JSHint.

È possibile eseguire i comandi soltanto dalla cartella principale del repository del codice.

4.4.3.5 Strumenti per l'integrazione continua Grazie all'utilizzo del software $Jenkins_G$ (v1.548) è stata predisposta un'esecuzione automatica dello script **make test-report** ad ogni $push_G$ effettuato sul $repository_G$, in modo da verificare costantemente che le modifiche non introducano errori e che rispettino le metriche software riportate nel *Piano di Qualifica v4.0.0*.

Ogni build fallita viene notificata all'autore del commit e al Responsabile di progetto via email. Le build che superano la verifica automatica, invece, vengono integrate automaticamente nel branch "stable" del repository.



4.4.4 Strumenti per la tracciabilità

Di seguito viene descritto lo strumento principale che è stato sviluppato dal gruppo per gestire la tracciabilità di molti degli aspetti affrontati durante il progetto. Il software in questione, denominato *Requisteak*, è strutturato secondo il V-Model descritto nel *Piano di Qualifica v4.0.0*. Tale strumento è raggiungibile all'indirizzo:

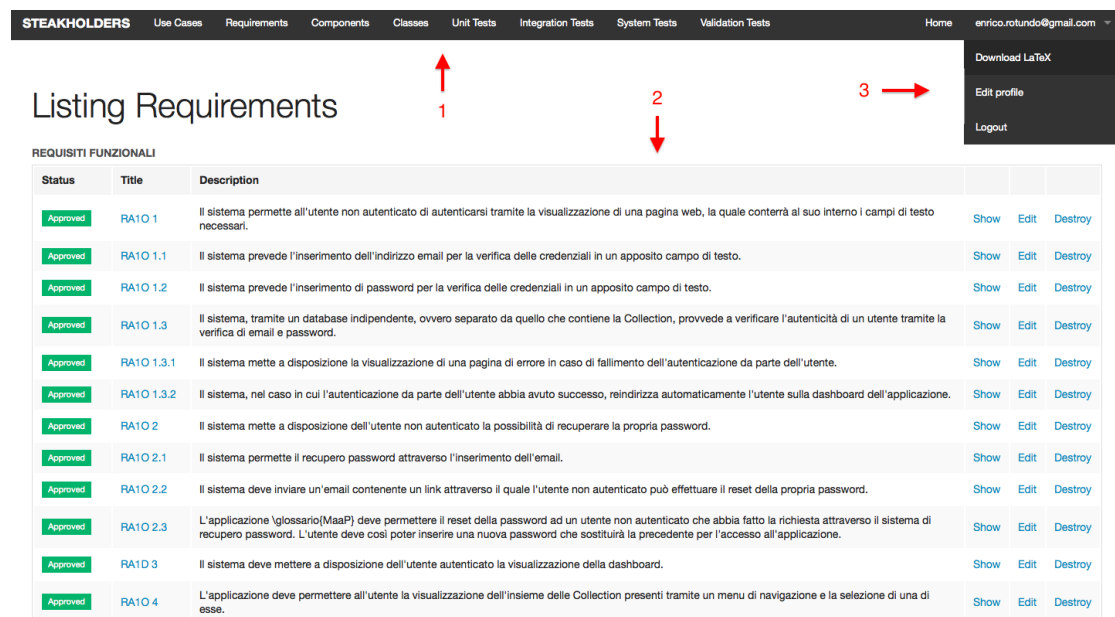
<http://steakholders.herokuapp.com>

Le credenziali di accesso sono le seguenti:

- **Username:** `committente@steakholders.com`
- **Password:** `unipd2013`

4.4.4.1 GUI Viene di seguito illustrata l'interfaccia grafica dello strumento in questione. I punti 1..3 sono relativi alla Figura 9, i punti 4..6 alla Figura 10.

1. Barra del menù, situata nella parte superiore della finestra, permette la navigazione tra le principali sezioni di Requistest. Da sinistra verso destra è possibile apprezzare come il V-Model sia stato implementato;
2. Index page della collection selezionata nella barra del menù. Cliccando sul codice di un record si accede alla relativa show-page;
3. Menù utility, oltre alla possibilità di modificare i propri dati profilo permette di scaricare i file LaTeX relativi alle collection.



Listing Requirements

REQUISITI FUNZIONALI

Status	Title	Description			
Approved	RA10 1	Il sistema permette all'utente non autenticato di autenticarsi tramite la visualizzazione di una pagina web, la quale conterrà al suo interno i campi di testo necessari.	Show	Edit	Destroy
Approved	RA10 1.1	Il sistema prevede l'inserimento dell'indirizzo email per la verifica delle credenziali in un apposito campo di testo.	Show	Edit	Destroy
Approved	RA10 1.2	Il sistema prevede l'inserimento di password per la verifica delle credenziali in un apposito campo di testo.	Show	Edit	Destroy
Approved	RA10 1.3	Il sistema, tramite un database indipendente, ovvero separato da quello che contiene la Collection, provvede a verificare l'autenticità di un utente tramite la verifica di email e password.	Show	Edit	Destroy
Approved	RA10 1.3.1	Il sistema mette a disposizione la visualizzazione di una pagina di errore in caso di fallimento dell'autenticazione da parte dell'utente.	Show	Edit	Destroy
Approved	RA10 1.3.2	Il sistema, nel caso in cui l'autenticazione da parte dell'utente abbia avuto successo, reindirizza automaticamente l'utente sulla dashboard dell'applicazione.	Show	Edit	Destroy
Approved	RA10 2	Il sistema mette a disposizione dell'utente non autenticato la possibilità di recuperare la propria password.	Show	Edit	Destroy
Approved	RA10 2.1	Il sistema permette il recupero password attraverso l'inserimento dell'email.	Show	Edit	Destroy
Approved	RA10 2.2	Il sistema deve inviare un'email contenente un link attraverso il quale l'utente non autenticato può effettuare il reset della propria password.	Show	Edit	Destroy
Approved	RA10 2.3	L'applicazione 'glossario[MaaP]' deve permettere il reset della password ad un utente non autenticato che abbia fatto la richiesta attraverso il sistema di recupero password. L'utente deve così poter inserire una nuova password che sostituirà la precedente per l'accesso all'applicazione.	Show	Edit	Destroy
Approved	RA1D 3	Il sistema deve mettere a disposizione dell'utente autenticato la visualizzazione della dashboard.	Show	Edit	Destroy
Approved	RA10 4	L'applicazione deve permettere all'utente la visualizzazione dell'insieme delle Collection presenti tramite un menu di navigazione e la selezione di una di esse.	Show	Edit	Destroy

Figura 9: Requistest GUI: menù e index-page

4. Dettaglio del document visualizzato;
5. Entità collegate con il document selezionato, possono essere Use Case, test vari e per le componenti software classi e metodi;
6. Schema gerarchico del document, è possibile visualizzare velocemente l'intera gerarchia del documento e per ogni nodo, se esso è associato ad un test di validazione o sistema.

The screenshot displays the SteakHolders GUI. At the top, there is a navigation bar with tabs for 'Use Cases', 'Requirements', 'Components', 'Classes', 'Unit Tests', 'Integration Tests', 'System Tests', and 'Validation Tests'. Below the navigation bar is a toolbar with buttons for 'Edit', 'Delete', 'Add Child', 'Add Use Case', 'Add System Test', and 'Add Validation Test'. The main content area is divided into three sections:

- 4. REQUIREMENT**: A table with columns 'Status', 'Title', 'Description', and 'Fonti'. It shows a requirement titled 'RA10 1' with a status of 'Approved' and a description: 'Il sistema permette all'utente non autenticato di autenticarsi tramite la visualizzazione di una pagina web, la quale conterrà al suo interno i campi di testo necessari.' The font is 'Capitolato'.
- 5. USE CASE, SYSTEM TEST, and VALIDATION TEST**: Three tables side-by-side. The 'USE CASE' table has columns 'Title' and 'Description' and shows 'UCU 1 - Login' with a description: 'L'utente non autenticato intende accedere all'applicazione, per farlo deve inserire le proprie credenziali composte da una email ed una password.' The 'SYSTEM TEST' table has columns 'Title' and 'Description' and is empty. The 'VALIDATION TEST' table has columns 'Title' and 'Description' and shows 'TV-RA10 1' with a description: 'L'utente non autenticato intende accedere all'applicazione, per farlo deve inserire le proprie credenziali composte da una email ed una password. All'utente è richiesto di: \begin{itemize} \item Raggiungere la pagina di autenticazione; \item Inserire la mail nel campo apposito; \item Inserire la password; \item Procedere con l'autenticazione. \end{itemize}'.
- 6. PARENTS and CHILDS**: Two sections. The 'PARENTS' section shows 'No parents :('. The 'CHILDS' section shows a list of children with a table containing 'RA10 1.1' and 'UCU 1'.

Figura 10: Requistek GUI: show-page

4.4.4.2 Use Cases Permette di visualizzare una index-page della collection dei casi d'uso da dove è possibile creare, eliminare, modificare un caso d'uso. Sarà possibile creare dei casi d'uso figli dalla show-page del padre utilizzando l'apposito pulsante in alto a destra. In dettaglio, i casi d'uso sono composti dai seguenti attributi:

- Ambito (Utenente, Sviluppatore, Utente MaaS);
- Subtitle;
- Codice Gerarchia;
- Primary Actors;
- Secondary Actors;
- Description;
- Precondition;
- Postcondition;
- Main flow;
- Alternative flow;
- Inclusion;
- Extension;
- Graph: fa riferimento al relativo diagramma UML in lucidchart.com.



4.4.4.3 Requirements Permette di creare, visualizzare, modificare ed eliminare i requisiti del progetto. Ad un requisito possono essere associati requisiti figlio, casi d'uso, test di sistema e test di validazione; la loro gestione avviene tramite la show-page del requisito. In dettaglio, i requisiti sono composti dai seguenti attributi:

- Sistema (Application, Eramework, MaaS);
- Typology (Funzionale, Prestazionale, Di Qualità, Di Vincolo);
- Priority;
- Numero gerarchia;
- Descrizione;
- Status (Pending, Approved);

4.4.4.4 Components Corrispondono ai package e sono divisi in *front end_G* e *back end_G*, hanno struttura gerarchica e sono associati a classi, test di integrazione e altri package. In dettaglio, i componenti sono composti dai seguenti attributi:

- Title;
- Parent;
- Description;
- Use;
- Graph: fa riferimento al relativo diagramma UML in lucidchart.com.

4.4.4.5 Classes Le classi sono interne ai package, hanno un gerarchia, possono essere relazionate tra loro e con i test di unità. In dettaglio, le classi sono composti dai seguenti attributi:

- Title;
- Parent;
- Description;
- Typology;
- Use.

4.4.4.6 Unit Tests Hanno un titolo e una descrizione, il loro inserimento avviene a partire dalla classe che si vuole testare.

4.4.4.7 Integration Tests Hanno un titolo e una descrizione, il loro inserimento avviene a partire dal componente che si vuole testare.

4.4.4.8 System Tests Hanno un titolo e una descrizione, il loro inserimento avviene a partire dal requisito aggregato che si vuole testare.



4.4.4.9 Validation Tests Hanno un titolo e una descrizione, il loro inserimento avviene a partire dal requisito radice che si vuole testare.