Norme Di Progetto

Informazioni sul documento	
Versione	3.0.0
Redazione	Andrea Giacomo Baldan
	Marco Boseggia
	Michael Munaro
Verifica	Francesco Agostini
	Giacomo Vanin
	Andrea Giacomo Baldan
Approvazione	Michael Munaro
Uso	Interno
Lista di Distribuzione	ScalateKids

Diario delle modifiche

Versione	Autore	Ruolo	Data	Descrizione
3.0.0	Michael Munaro	Responsabile	2016-05-08	Documento approvato
2.3.0	Francesco Agostini	Verificatore	2016-05-06	Verifica sezione Analisi Statica 5.4.1.1
2.2.0	Giacomo Vanin	Verificatore	2016-05-06	Verifica sezione Progettazione Architetturale 2.2.3 e sottosezione Definizione di Prodotto 2.2.3.2.2
2.1.2	Andrea Giacomo Baldan	Analista	2016-05-05	Incremento sezione Analisi Statica 5.4.1.1
2.1.1	Andrea Giacomo Baldan	Analista	2016-05-03	Incremento sezione Progettazione Architetturale 2.2.3, sottosezione Definizione di Prodotto 2.2.3.2.2
2.1.0	Andrea Giacomo Baldan	Verificatore	2016-05-01	Verifica sezione Assicurazione Qualità
2.0.3	Marco Boseggia	Analista	2016-04-26	Incremento sezione Assicurazione Qualità
2.0.2	Marco Boseggia	Analista	2016-04-26	Spostamento sezione Assicurazione Qualità
2.0.1	Michael Munaro	Analista	2016-04-26	Integrazione appendice Redmine per calendario attività
2.0.0	Giacomo Vanin	Responsabile	2016-03-20	Approvazione documento
1.6.0	Marco Boseggia	Verificatore	2016-03-13	Verifica sezione test
1.5.1	Francesco Agostini	Analista	2016-03-12	Incremento sezione test
1.5.0	Davide Trevisan	Verificatore	2016-03-12	Verifica sezioni 2.2.3 e sottosezioni
1.4.1	Michael Munaro	Analista	2016-03-11	Incremento sezione 2.2.3 e sottosezioni
1.4.0	Davide Trevisan	Verificatore	2016-03-11	Verifica sezione norme per la progettazione (2.2.3 e sottosezioni)
1.3.1	Michael Munaro	Analista	2016-03-10	Aggiunta sezione norme per la progettazione (2.2.3 e sottosezioni)
1.3.0	Davide Trevisan	Verificatore	2016-03-06	Verifica ultime sezioni modificate (2.2.2.1 e 4.5.1.3)
1.2.2	Andrea Giacomo Baldan	Analista	2016-03-05	Incremento sezione 2.2.2.1 (ScalateTrack)
1.2.1	Andrea Giacomo Baldan	Analista	2016-03-05	Incremento sezione 4.5.1.3 (Script di utilità locale)

1.2.0	Michael Munaro	Verificatore	2016-03-03	Verifica sezioni 3.2, 4 e appendice A
1.1.3	Andrea Giacomo Baldan	Analista	2016-03-01	Incremento Appendice
1.1.2	Andrea Giacomo Baldan	Analista	2016-02-26	Incremento sezione 3.2.1 Norme di Verifica
1.1.1	Andrea Giacomo Baldan	Analista	2016-02-26	Stesura sezione 4 Processi Organizzativi
1.1.0	Marco Boseggia	Verificatore	2016-02-24	Verifica sezioni 1, 2 e 3
1.0.3	Andrea Giacomo Baldan	Analista	2016-02-21	Stesura sezione 3 Processi di Supporto
1.0.2	Andrea Giacomo Baldan	Analista	2016-02-21	Stesura sezione 2 Processi Primari
1.0.1	Andrea Giacomo	Analista	2016-02-19	Riorganizzazione generale dei contenuti
1.0.0	Alberto De Agostini	Responsabile	2015-12-23	Approvazione documento
0.5.0	Davide Trevisan	Verificatore	2015-12-22	Verifica sezione Processi di Sviluppo
0.4.0	Marco Boseggia	Verificatore	2015-12-22	Verifica sezione Processi di Supporto
0.3.2	Andrea Giacomo Baldan	Amministratore	2015-12-22	Correzione sezione Processi di Sviluppo
0.3.1	Francesco Agostini	Amministratore	2015-12-21	Correzione sezione Processi di Supporto
0.3.0	Marco Boseggia	Verificatore	2015-12-20	Verifica sezione Processi di Sviluppo
0.2.0	Michael Munaro	Verificatore	2015-12-20	Verifica sezione Processi di Supporto
0.1.2	Andrea Giacomo Baldan	Amministratore	2015-12-18	Stesura sezione Processi di Sviluppo
0.1.1	Francesco Agostini	Amministratore	2015-12-18	Stesura sezione Processi di Supporto
0.1.0	Giacomo Vanin	Verificatore	2015-12-17	Verifica sezione Processi Organizzativi
0.0.2	Andrea Giacomo Baldan	Amministratore	2015-12-17	Stesura sezione Processi Organizzativi
0.0.1	Andrea Giacomo Baldan	Amministratore	2015-12-16	Creazione scheletro del documento

Indice

1 Sommario

1.1 Scopo del documento

Le Norme di Progetto hanno l'obiettivo di fornire il metodo di lavoro, le convenzioni e formalismi che i membri del gruppo devono adottare nell'attuazione delle strategie scelte durante l'attività$_G$ di progetto.
Tutti i componenti del gruppo sono tenuti a leggere il presente documento e a seguire le regole contenute; esso verrà integrato ogni qualvolta si presenti la necessità di chiarire dubbi o definire comportamenti da seguire in specifici scenari.

1.2 Scopo del Prodotto

Implementazione di un database NoSQL$_G$ di tipo key-value$_G$ orientato alla gestione di grandi moli di dati utilizzando il modello ad attori$_G$ su JVM$_G$, comprensivo di un *Domain Specific Language* (DSL$_G$) da utilizzare da riga di comando per poter interagire con il database.
Il progetto dovrà essere pubblicato su *GitHub* sotto licenza *MIT*.

1.3 Glossario

Tutti i termini di carattere tecnico o fraintendibile e gli acronimi sono raccolti nel file Glossario v3.0.0; ogni occorrenza di parole nel *Glossario* è indicata da una "G" in pedice.

1.4 Riferimenti

1.4.1 Informativi

- **Redmine:** http://www.redmine.org/;
- **Docker:** https://www.docker.io/;
- **Travis-ci:** https://travis-ci.org/;
- **Virtual Box:** https://www.virtualbox.org/.
- **SBT:** Simple Build Tool, ambiente di sviluppo Scala http://www.scala-sbt.org
- **Apache ab:** Uno strumento per benchmarking$_G$ di sistemi basati su comunicazioni HTTP$_G$ (HyperText Transport Protocol)https://httpd.apache.org/docs/2.4/programs/ab.html
- **GNU Parallel:** Uno strumento per eseguire in parallelo più processi http://www.gnu.org/software/parallel/

2 Processi Primari

2.1 Fornitura

2.1.1 Studio di Fattibilità

Il primo compito degli *Analisti* sarà di redigere uno studio comparativo dei lati positivi e negativi dei capitolati pubblicati secondo le opinioni e i confronti dei membri del gruppo, evidenziando in particolare le criticità che hanno portato all'esclusione dei capitolati scartati e, dal lato opposto, i fattori positivi che hanno colpito e influenzato maggiormente la decisione finale sul capitolato.

2.1.2 Pianificazione e preparazione dell'offerta

Al fine dell'aggiudicazione dell'appalto sul capitolato scelto, sarà compito del *Responsabile* stendere un *Piano di Progetto* con sistemi di controllo e rendicontazione ben delineati. Dovrà inoltre includere un'accurata analisi dei rischi corredata di contromisure e un preventivo che all'avanzare del progetto si trasformerà in consuntivo.
Il documento presenterà infine un organigramma, utile a definire la struttura e l'allocazione delle risorse e i ruoli di progetto.
I diagrammi di Gantt$_G$ e PERT$_G$ (Program Evaluation Review Technique) saranno utilizzati per rappresentare graficamente l'allocazione nel tempo e la durata dei compiti, evidenziando le dipendenze tra le attività ed il loro impatto sull'intero progetto.
Vedi 4.4.1.1 per la strumentazione utile allo scopo.

2.2 Sviluppo

Gli *Analisti* analizzeranno i Requisiti di Sistema del prodotto da sviluppare, da questi sarà possibile estrarre i requisiti software e stendere un'analisi dettagliata.

2.2.1 Analisi dei Requisiti

Dal capitolato e dall'attività$_G$ di brainstorming$_G$ effettuata inizialmente, integrata con le riunioni esterne da effettuare con il *Proponente*, gli *Analisti* dovranno stilare una lista di casi d'uso ed estrarre i requisiti emergenti.
I requisiti software dovranno essere:

- precisi;
- non ambigui;
- completi;
- testabili.

Essi saranno composti da una parte testuale che descrive il requisito ed un modello semiformale per esprimerli, nel nostro caso il linguaggio UML$_G$ *versione 1.0*. Ogni diagramma inserito nei documenti dovrà essere etichettato secondo la seguente regola:

`<Tipo><Id>`

Dove `Tipo` può assumere le seguenti sigle:

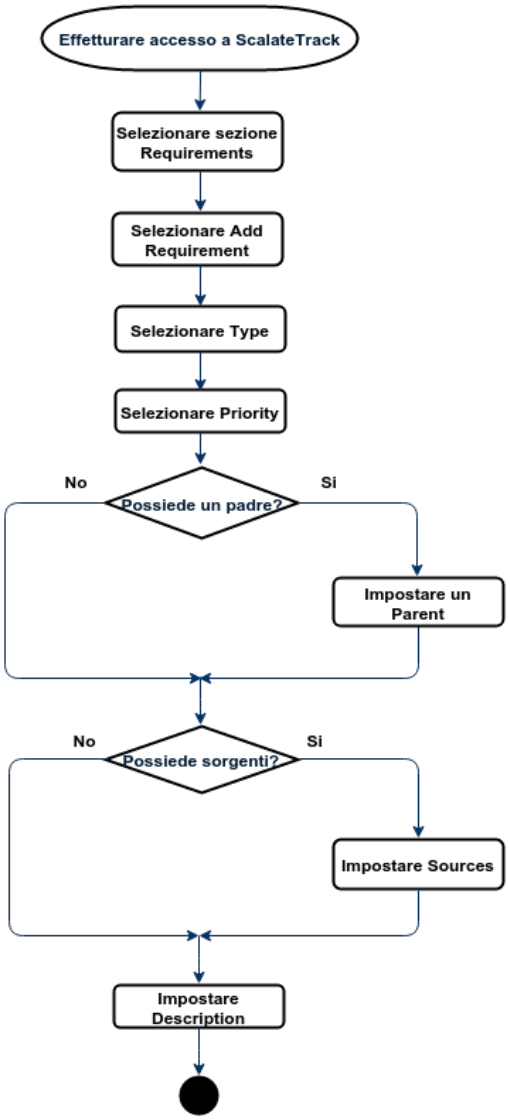
- **UC:** diagramma di caso d'uso;
- **CD:** diagramma di classe;
- **OD:** diagramma degli oggetti;
- **SD:** diagramma di sequenza;
- **AD:** diagramma di attività$_G$;
- **PD:** diagramma dei package$_G$.

`Id` è un numero crescente che identifica univocamente il caso d'uso. Se un caso d'uso è derivato da un altro, il suo id sarà preceduto dall'id del caso d'uso da cui deriva e separato da esso con un punto.

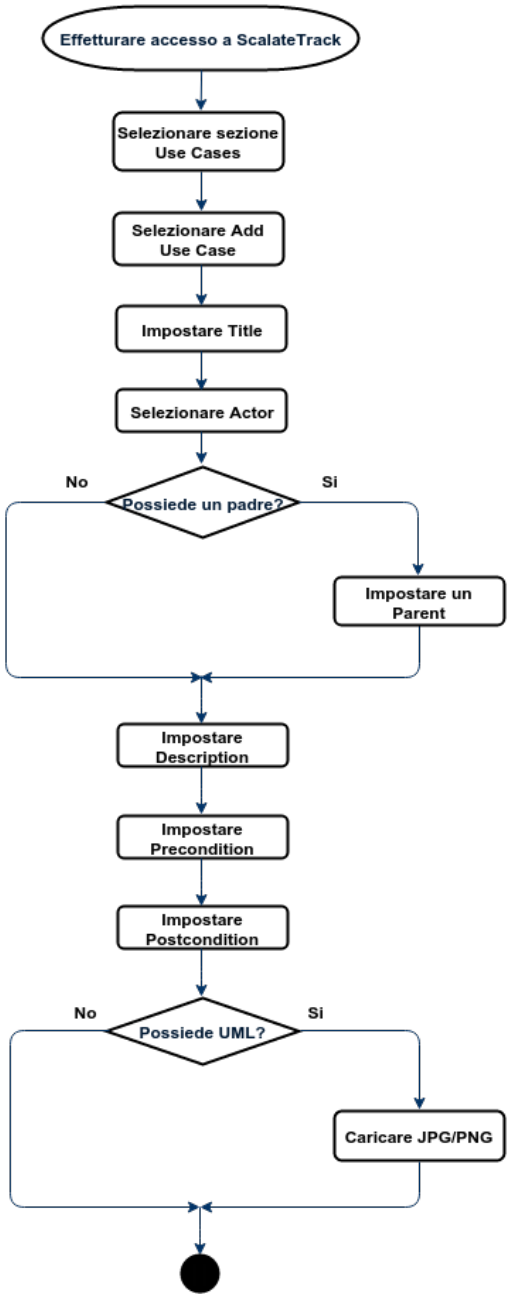
2.2.2 Casi d'uso e tracciamento dei requisiti

2.2.2.1 ScalateTrack

Per facilitare la gestione dei casi d'uso e il tracciamento dei requisiti va utilizzata un'applicazione web sviluppata per lo scopo appositamente dal gruppo.
Essa si interfaccia con un database relazionale residente sul server privato del gruppo (vedi 4.4.2) e permette una visualizzazione chiara dei requisiti, dei casi d'uso e delle componenti. Una volta completata la procedura d'inserimento dei dati, previa verifica, è possibile esportare i requisiti in formato LaTeX ed eseguirne il tracciamento automatico rispetto alle fonti e viceversa. Allo stesso modo è possibile esportare le componenti ed eseguirne il tracciamento automatico rispetto ai requisiti che soddisfano e viceversa.



(a) Creazione nuovo requisito



(b) Creazione nuovo caso d'uso

2.2.2.2 Casi d'uso

Ogni caso d'uso dovrà presentare i seguenti campi:

- Codice identificativo nella forma `<UC><Id>` con `UC` = Use Case mentre `Id` è un numero crescente che identifica univocamente il caso d'uso. Se un caso d'uso è derivato da un altro, il suo `Id` sarà preceduto dall'`Id` del caso d'uso da cui deriva e separato da esso con un punto.
- Titolo;
- Diagramma UML_G;
- Attori primari;
- Attori secondari;
- Scopo e descrizione;
- Precondizione;
- Postcondizione;
- Flusso principale degli eventi;
- Scenari alternativi.

2.2.2.3 Requisiti software

I requisiti all'interno del documento *Analisi dei Requisiti* seguiranno le seguenti regole:

`<Importanza><Tipologia><Id>`

Di seguito sono elencati i possibili valori che ogni termine tra parentesi angolari può assumere:

- **Importanza**:
  - **OB** indica un requisito obbligatorio;
  - **DE** indica un requisito desiderabile;
  - **OP** indica un requisito opzionale.
- **Tipologia**:
  - **F** indica un requisito funzionale;
  - **Q** indica un requisito di qualità;
  - **T** indica un requisito tecnologico;
  - **V** indica un requisito di vincolo.
- **Id**: è un numero crescente che identifica univocamente il requisito. Se un requisito è stato derivato da un altro, il suo id sarà preceduto dall'id del requisito da cui dipende e separato da esso con un punto.

2.2.3 Progettazione architetturale

La *Progettazione* precede la produzione ed è l'attività$_G$ che passa dai requisiti del problema, estratti dagli *Analisti* durante l'**Analisi**, ad una soluzione del problema preposto a carico dei **Progettisti**. Essa dovrà rispettare tutti i requisiti che il gruppo ha concordato con il *Committente*.

2.2.3.1 Specifica Tecnica

Sarà compiti dei *Progettisti* definire la struttura ad alto livello.
Vengono definite:

- Le componenti software, definite dalla struttura ad alto livello;
- Le interfacce verso l'esterno e tra le componenti;
- Classi, tecniche e design pattern per lo sviluppo;
- I Requisiti Software diventano Requisiti Componenti più dettagliati.

Dalla specifica dei Requisiti delle Componenti deriveranno i test da attuare in attività di verifica, come specificato in 5.4;

2.2.3.1.1 Design Pattern

L'attività$_G$ di *Progettazione* dovrà fare utilizzo di design pattern$_G$ globalmente affermati e dovrà giustificarne la scelta.
Dovranno inoltre essere rispettati al massimo i paradigmi di OOP$_G$ (programmazione orientata agli oggetti) del linguaggio Scala$_G$, con eventuale utilizzo di costrutti funzionali$_G$, coerentemente con l'utilizzo delle librerie *Akka*$_G$.

2.2.3.1.2 Componenti software

Le componenti software all'interno del documento *Specifica Tecnica* dovranno essere nominate elencando prima del nome della classe tutti i namespace in cui essa è contenuta separati da una coppia di due punti.
Qui possiamo vedere un esempio:

```
actorbase::cli::views::CommandLoop
```

Inoltre per facilitare la comprensione della lettura dei diagrammi dei package e delle classi si è scelto di:

- usare una colorazione diversa per package di livelli diversi;
- usare il colore verde per indicare classi o package esterni al sistema.

2.2.3.1.3 Tracciamento componenti

Il software ScalateTrack (2.2.2.1) si occupa anche del tracciamento dei componenti. Nello specifico ogni requisito può essere tracciato con uno o più componenti, in questo modo il gruppo potrà controllare e garantire che ogni requisito venga soddisfatto.
ScalateTrack inoltre genererà automaticamente le tabelle dei tracciamenti requisiti-componenti e componenti-requisiti.

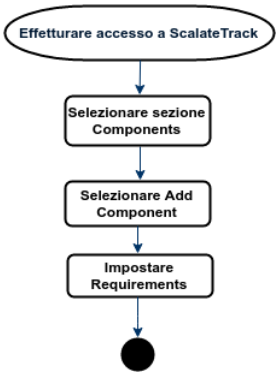


Figura 2: Creazione nuovo componente

2.2.3.1.4 Test di Integrazione

I *Progettisti* devono definire dei test necessari alla verifica del corretto funzionamento delle componenti.

2.2.3.1.5 Diagrammi UML

Nella specifica tecnica dovranno essere presentati diversi tipi di diagrammi, in particolare i *progettisti* avranno il compito di realizzare i seguenti rispettando le regole imposte dallo standard UML$_G$ 1.0:

- diagrammi dei package$_G$;
- diagrammi delle classi$_G$;
- diagrammi di sequenza$_G$;
- diagrammi di attività$_G$.

2.2.3.2 Definizione di prodotto

I *progettisti* oltre a redigere la *Specifica Tecnica* avranno il compito di produrre anche il documento *Definizione di Prodotto* in cui verrà descritta la progettazione di dettaglio del sistema andando ad analizzare più in profondità l'architettura. Il livello di dettaglio sarà maggiore rispetto alla *Specifica Tecnica*, in particolare saranno

definite le interazioni tra le classi e la definizione degli attributi e dei metodi contenuti, in modo da fornire ai programmatori un documento da seguire precisamente per la creazione del prodotto.

2.2.3.2.1 Definizione delle classi

All'interno della *Definizione di prodotto* dovrà essere descritta ogni classe presente all'interno del sistema. Per ognuna di queste classi la descrizione dovrà essere comprensiva di:

- Elenco dei metodi;
- Elenco dei parametri;
- Spiegazione dello scopo della classe;
- Funzionalità che essa modella.

2.2.3.2.2 Tracciamento classi

In questo documento dovrà essere presente il tracciamento dei requisiti abbinato alle classi che lo soddisfano. Come per la *Specifica Tecnica* anche in questo caso verrà utilizzato il software ScalateTrack (2.2.2.1) il quale renderà automatico l'esportazione in formato LaTeX delle tabelle di tracciamento classi-requisiti e requisiti-classi.

Figura 3: Creazione nuova classe

2.2.3.2.3 Diagrammi UML

Dovranno essere aggiornati i seguenti diagrammi rispettando gli stessi standard stabiliti per produrre la *Specifica Tecnica* (2.2.3.1.5):

- diagrammi delle classi$_G$;

- diagrammi di sequenza$_G$;
- diagrammi di attività$_G$.

3 Processi di Supporto

3.1 Documentazione

3.1.1 Struttura dei documenti

Abbiamo standardizzato la scrittura di tutti i documenti attraverso un template$_G$ LaTeX appositamente creato e presente su *GitHub*$_G$ nel repository$_G$ *ActorBase-Documents*.git$_G$ (vedi 4.4.1.1).
L'uso del template$_G$ ci permette di creare una serie di documenti libraryDependenciesstilisticamente uniformi tra loro e ci facilita la modifica delle parti in comune tra essi.
Facilita inoltre la creazione di nuove macro$_G$ e comandi LaTeX. Nella fattispecie abbiamo creato i seguenti:

- `\gloss`: Da utilizzare per inserire le parole da glossario;
- `\glossDef`: Da utilizzare all'interno del documento *Glossario* per inserire nuove definizioni;
- `\glossaryLetter`: Da utilizzare all'interno del documento *Glossario* per inserire le sezioni alfabetiche preformattate;
- `\prodPurpose`: Da utilizzare per creare la sottosezione "Scopo del Prodotto", da inserire nelle parti comuni di tutti i documenti (Sommario);
- `\glossExpl`: Da utilizzare per creare la sottosezione "Glossario", da inserire nelle parti comuni di tutti i documenti per spiegare come vengono identificati i termini da glossario;
- `\multiLineCell`: Da utilizzare per creare celle multilinea nelle tabelle.

Sono stati infine ridefiniti alcuni comportamenti dei comandi standard di LaTeX, per esempio le intestazioni e i piè pagina automatizzati, la profondità delle section$_G$, dell'indice dei contenuti e le colorazioni dei link$_G$:

- **url$_G$:** Blu;
- **citazioni:** Grigio.

Riassumendo il template$_G$ regola:

- Formattazione dei documenti (font);
- Formattazione delle pagine (header e footer pagine);
- Raggruppa tutti i pacchetti utilizzati;
- Aggiunge comandi personalizzati;
- Collegamenti tra indice e categorie o sezioni.

Ogni documento presenta in ordine:

- Logo del gruppo;
- Informazioni del documento;
- Diario delle modifiche: è una tabella in cui sono scritte le diverse modifiche fatte dai membri sul documento;

- Indice: indice con collegamenti alle categorie e alle sezioni del documento;
- Sommario: contiene alcune sottosezioni:
  - Scopo del documento;
  - Scopo del prodotto;
  - Glossario;
  - Riferimenti inerenti al documento in questione;
- Resto del documento: contenuto specifico.

3.1.2 Struttura Pagina

L'intestazione di ogni pagina contiene:

- Logo gruppo;
- Nome gruppo;
- Sezione corrente del documento.

Il piè pagina invece contiene:

- Nome documento;
- Pagina corrente rispetto al numero di pagine totali.

3.1.3 Norme tipografiche

Per uniformare il più possibile la stesura di tutti i documenti queste sono le regole che tutto il gruppo deve seguire:

3.1.3.1 Punteggiatura

- **Punteggiatura**: tutti i segni di punteggiatura devono essere seguiti da uno spazio e non avere spazi precedenti al segno stesso;
- **Maiuscole**: le lettere maiuscole devono essere usate dove previsto dalla lingua italiana. Saranno inoltre scritti con l'iniziale maiuscola i ruoli, le persone inerenti al progetto e i documenti noti (es. *Committente Analisi dei Requisiti*). Gli acronimi saranno interamente scritti in maiuscolo (es. *SQL*).

3.1.3.2 Stile testo

- **Corsivo**: il corsivo dev'essere utilizzato per:
  - Figure inerenti al progetto;

- Abbreviazioni;
- Citazioni;
- Documenti (es. *Analisi dei Requisiti*);
- Porre un'enfasi maggiore alla parola e/o frase;

- **Grassetto**: il grassetto dev'essere usato per:
  - Evidenziare passaggi o concetti importanti;
  - Sezioni e sottosezioni dei documenti;
- **Monospace**: i font monospace saranno usati per riportare parti di codice$_G$.

3.1.3.3 Formati di riferimento

- **Riferimenti**:
  - Percorsi web: per gli indirizzi web e per gli indirizzi e-mail deve essere usato il comando LaTeX:

    ```
    \url{percorso}
    ```

  - Link$_G$ ad altri PDF$_G$:

    ```
    \href{run:<pathToPDF>/<namefile.pdf>}{<name of the link>}
    ```

  - Link$_G$ a sezioni interne al documento: per link$_G$ a sezioni interne al documento corrente deve essere utilizzato il documento LaTeX:

    ```
    \ref{riferimento a sezione}
    ```

- **Date**: Le date devono essere espresse seguendo lo standard *ISO*$_G$ 8601:2004: AAAA-MM-GG AAAA: rappresenta l'anno (es. 2015); MM: rappresenta il mese (es. 12); GG: rappresenta il giorno (es. 21).
- **Abbreviazioni**: per semplicità possono essere abbreviati i nomi dei seguenti documenti: AR: *Analisi dei Requisiti*; RR: *Revisione dei Requisiti*; GL: *Glossario*; NP: *Norme di Progetto*; RP: *Revisione di Progettazione*; PQ: *Piano di Qualifica*; RQ: *Revisione di Qualifica*; PP: *Piano di Progetto*; SF: *Studio di Fattibilità*; RA: *Revisione di Accettazione*.
- **Nomi ricorrenti**:
  - Ruoli: come già scritto in precedenza i ruoli di progetto devono essere scritti con la prima lettera di ogni parola maiuscola ed in corsivo, escludendo le proposizioni (es. *Responsabile di Progetto*);
  - Nomi dei file$_G$: i nomi dei file$_G$ relativi a documenti devono seguire; la notazione *UpperCamelCase*$_G$, seguiti da *_v* e dalla versione del file$_G$ (es. NormeDiProgetto_v1.0.1.pdf);
  - Nome del progetto: sarà indicato come **Actorbase**;
  - Nome del *Committente*: il *Committente* sarà indicato come *prof. Tullio Vardanega*;
  - Nome del *Proponente*: il *Proponente* sarà indicato come *prof. Riccardo Cardin*;

– Nome del gruppo: il gruppo sarà indicato come *ScalateKids*.

libraryDependencies

3.1.3.4 Immagini

Utilizziamo immagini con formato *JPG* o *PNG*, questi ultimi rendono immediata l'inclusione delle suddette immagini nei documenti.

3.1.3.5 Integrazione termini di lingua straniera

Per non incorrere in diverse modalità di integrazione per l'uso di vari termini di lingua straniera (es. file$_G$, repository$_G$ etc.) i termini non italiani non saranno pluralizzati, esempio:

`abbiamo utilizzato dei` *repository* `per... (non` *repositories*`).`

Questa scelta è derivata da una ricerca che ha fornito diverse fonti riguardanti l'uso di questa norma nella lingua italiana, il sottostante sito web ne raccoglie buona parte:

`http://www.darsch.it/?pg=sphere&postid=954`

3.1.4 Versionamento documenti

La documentazione prodotta deve avere un numero di versione avente la seguente struttura:

vX.Y.Z

in cui:

- **X**: aumenta ad ogni approvazione effettuata dal *Responsabile*. Questo valore corrisponderà anche con il numero di uscite formali del documento:
  1. Attività$_G$ di **Analisi** che si conclude con la **RR**;
  2. Attività$_G$ di **Analisi di Dettaglio** che si conclude con l'ingresso alla **RP**;
  3. Attività$_G$ di **Progettazione Architetturale** che si conclude con la **RP**;
  4. Attività$_G$ di **Progettazione di Dettaglio e Codifica** che si conclude con l'ingresso alla **RQ**;
  5. Attività$_G$ di **Verifica e Validazione** che si conclude con la **RA**.
- **Y**: aumenta ad ogni attività$_G$ di verifica applicata sul documento. Quando il documento viene approvato dal *Responsabile* questo numero viene azzerato;
- **Z**: aumenta ad ogni modifica rilevante effettuata sul documento. Questo valore si azzera ad ogni attività$_G$ di verifica o approvazione del documento.

La citazione di una versione specifica di un documento deve comprendere sia il nome che il numero di versione aderente al formato:

NomeDocumento_vX.Y.Z

3.1.5 Ciclo di vita dei documenti

Tutti i documenti nascono nello stato di *in lavorazione*.
Dopo l'avvenuta di tutte le modilibraryDependenciesfiche necessarie per arrivare ad una versione completa del documento, il suddetto si verrà a trovare nello stato *da verificare*.
Alla fine della fase di verifica del documento, se approvato, il documento andrà nello stato di *approvato*. I tre stati sono descritti brevemente come segue:

- **In lavorazione**: il documento nasce in questo stato e ci rimane fino a che non è avvenuta una sua completa stesura;
- **Da verificare**: il documento resta in questo stato finché i *Verificatori* assegnati ad esso non effettueranno un controllo atto a trovare e a correggere errori di ogni tipo;
- **Approvato**: dopo la fase di verifica il *Responsabile* effettuerà una fase di approvazione, se questa fase sarà superata il documento sarà approvato e entrerà in una versione ufficiale.

3.1.6 Norme di codifica dei file

Tutti i file$_G$ contenenti codice$_G$ o documentazione dovranno essere conformi alla codifica UTF-8$_G$.

3.1.6.1 Linee guida stilistiche

La stesura del codice$_G$ dovrà seguire pedissequamente alcune regole al fine di garantire un buon livello di manutenibilità e verificabilità del prodotto, in particolare gli sviluppatori dovranno attenersi alla *Scala Style Guide* (https://docs.scala-lang.org/style/), in particolar modo non possono essere infrante le seguenti regole:

- Lunghezza massima linee 80 caratteri;
- Indentazione di 2 spazi;
- Posizionamento di parentesi graffe in linea con il blocco struttura di riferimento.

Tutti i file$_G$ contenenti codice$_G$ dovranno essere redatti in lingua inglese.

3.1.6.1.1 Nomenclatura

I nomi di variabili, metodi e funzioni seguiranno la convenzione *lowerCamelCase*$_G$, i nomi di classi invece seguiranno la convenzione *UpperCamelCase*$_G$.

3.1.6.1.2 Ricorsione

Il paradigma funzionale incentiva all'uso di ricorsione piuttosto che cicli iterativi, è ammesso dunque l'utilizzo di ricorsione purchè si tratti specificamente di ricorsione in coda, questo perchè in Scala$_G$ tal costrutto è ottimizzato in modo da non rimpeire lo stack$_G$ assegnato alla funzione chiamante. La ricorsione non in coda va invece evitata il più possibile. Per ogni funzione ricorsiva sarà necessario fornire una prova di terminazione. L'utilizzo deve giustificare la spesa in termini di memoria rispetto al risparmio in termini di istruzioni.

3.1.6.1.3 Documentazione

I file$_G$ contenenti codice$_G$ dovranno essere adeguatamente commentati seguendo le linee guida scaladoc$_G$ https://docs.scala-lang.org/style/scaladoc.html ed essere provvisti di un intestazione contenente:

```
/**
* Licenza MIT
* Nome file
* Breve descrizione
*
* @author Autore del file
* @version Versione
* @date Data di creazione
* Descrizione dettagliata del file
*/
```

Prima di ogni metodo dovrà essere presente un commento scaladoc$_G$ contenente:

```
/**
* Breve descrizione del metodo
*
* @param Nome del primo parametro
* @param Nome del secondo parametro
* @return Valore di ritorno del metodo
* @throws Eccezioni lanciate dal metodo
*/
```

La documentazione verrà generata da scaladoc$_G$. Nel caso si verifichi la necessità di documentare del codice$_G$ di difficile comprensione, sarà possibile inserire un commento nelle righe precedenti, dopo aver verificato l'impossibilità di effettuare un refactoring$_G$.

3.1.7 Glossario

Il glossario è un documento unico per tutti i documenti, esso conterrà tutte le definizioni, in ordine lessicografico crescente dei termini inerenti al tema del progetto o che possono essere fraintesi. I termini che dovranno essere inseriti nel glossario saranno contrassegnati da una *G* pedice all'interno dei documenti. Prima di inserire un nuovo termine bisognerà assicurarsi che non sia già presente.

Il comando LaTeX da utilizzare per contrassegnare un termine da glossario all'interno dei documenti è `\gloss`, mentre l'inserimento di una nuova parola all'interno del glossario è `\glossDef` (come specificato in 3.1.1). La scelta di creare un comando apposito per un'operazione "elementare" è scaturita dall'agevolazione che porta alla stesura della documentazione: avendo un modo univoco di riconoscere i termini all'interno del glossario, è possibile automatizzare il controllo delle parole da glossario all'interno dei documenti (vedi sezione 4.5.1.3).

4 Processi Organizzativi

4.1 Ruoli di progetto

Durante l'intera attività$_G$ di sviluppo del progetto, dalla formazione del gruppo alla *Revisione di Accettazione*, vi saranno alcuni ruoli ben precisi da ricoprire. Si tratta di funzioni aziendali assegnate al progetto, specializzate in campi ben specifici all'interno dell'azienda. Tra questi ruoli alcuni avranno maggior presenza in determinate fasi rispetto ad altre, dove addirittura potrebbero non figurare completamente, ma tutti sono essenziali per il buon esito del progetto.
Ciascun componente dovrà ricoprire almeno una volta ogni ruolo. Sarà inoltre possibile ricoprire più ruoli, sia contemporaneamente che in distinte fasi del progetto, purché sia garantita l'assenza di conflitti d'interesse nello svolgimento di attività$_G$ di verifica e approvazione.
Al fine di garantire che tutti i componenti ricoprano tutti i ruoli almeno una volta, il gruppo seguirà le seguenti regole:

- Ogni componente non deve impiegare più del 50% delle ore di lavoro in un unico ruolo;
- Tutti i ruoli eccetto *Responsabile di Progetto* e *Amministratore* dovranno ruotare ogni due settimane;
- I ruoli *Responsabile di Progetto* e *Amministratore* dovranno ruotare ogni venti giorni.

La scelta di ruotare con maggiore frequenza i ruoli al di fuori del *Responsabile* e *Amministratore* è dettata essenzialmente dalla maggiore cardinalità degli altri ruoli all'interno del gruppo, e all'accentramento di responsabilità che *Responsabile* e *Amministratore* comprendono naturalmente da contratto.

4.1.1 Responsabile di Progetto

libraryDependencies Il *Responsabile di Progetto* rappresenta il gruppo presso il *Proponente* e presso il *Committente*. È il ruolo più presente lungo tutto l'arco temporale di sviluppo del prodotto, in quanto deve partecipare e seguire il prodotto dalla crescita fino al rilascio. Deve avere conoscenze e capacità tecniche tali da comprendere e anticipare l'evoluzione del progetto.
Detiene il potere decisionale e ha responsabilità su:

- Pianificazione;
- Gestione delle risorse umane;
- Controllo, coordinamento e relazioni esterne;
- Analisi e gestione dei rischi;
- Approvazione dei documenti;
- Approvazione dell'offerta economica.

Si occupa dunque della distribuzione delle attività$_G$, verifica che esse vengano portate a termine seguendo le *Norme di Progetto* e controlla che non vi siano conflitti d'interesse tra redazione e verifica, ha il compito inoltre di risolvere eventuali situazioni critiche tra i componenti del gruppo qualora sorgessero conflitti.
Redige il *Piano di Progetto* e collabora alla stesura del *Piano di Qualifica*.

4.1.2 Analista

L'*Analista* è una figura molto presente nelle prime fasi del progetto e determina fortemente la buona riuscita del prodotto. Non si occupa della soluzione al problema ma deve offrire una specifica di progetto che comprenda appieno la natura e la complessità del problema, che possa essere in seguito valutata dal *Progettista* al fine di fornire una soluzione.
Redige lo *Studio di Fattibilità* e l'*Analisi dei Requisiti*, partecipa inoltre alla stesura del *Piano di Qualifica*.

4.1.3 Amministratore

L'*Amministratore* si occupa di allestire, seguire e migliorare l'ambiente di lavoro, le responsabilità principali sono:

- Amministrazione delle risorse e delle infrastrutture;
- Risoluzione di problemi legati alla gestione dei processi;
- Gestione della documentazione di progetto;
- Controllo di versioni e configurazioni;
- Ricerca di strumenti che possano automatizzare compiti tediosi;
- Ricerca di strumenti che possano semplificare il lavoro di verifica.

Redige le *Norme di Progetto* e partecipa alla stesura del *Piano di Qualifica*.

4.1.4 Progettista

Il *Progettista* è una figura molto legata all'*Analista*, in quanto è responsabile delle attività$_G$ di progettazione, si occupa dunque di trovare una soluzione efficace al problema, sfruttando le proprie competenze tecniche e tecnologiche sempre aggiornate. Si tratta di un ruolo generalmente presente dalla fase di progettazione fino alla fine del progetto.

4.1.5 Programmatore

È responsabile delle attività$_G$ di codifica e manutenzione del prodotto, gestisce inoltre le componenti di ausilio necessarie all'attività$_G$ di verifica e validazione. Ha competenze tecniche specializzate, ricopre principalmente le seguenti responsabilità:

- Implementazione delle soluzioni descritte dal *Progettista* seguendone i design pattern proposti;
- Documentare e commentare il codice$_G$ in modo da renderlo più facilmente mantenibile;
- Implementare dei test utili in fase di verifica e validazione.

Si occupa della redazione del *Manuale Utente*.

4.1.6 Verificatore

La figura del *Verificatore* partecipa alla realizzazione del prodotto per tutta la durata assieme al *Responsabile*, possiede grandi capacità di giudizio e competenza tecnica, influenza molto fortemente l'aspetto qualitativo del prodotto.
Ricopre le seguenti responsabilità:

- Si assicura che le attività$_G$ seguano le direttive stabilite nelle *Norme di Progetto*;
- Controlla la conformità di ogni stadio del ciclo di vita del prodotto.

Si occupa della redazione del *Piano di Qualifica*.

4.2 Comunicazioni

4.2.1 Comunicazioni esterne

Per le comunicazioni esterne è stata creata una apposita casella di posta elettronica:

`scalatekids@gmail.com`

Questa casella e-mail viene gestita dal *Responsabile di Progetto* che è colui che si occupa di comunicare con il *Committente* del progetto o con qualsiasi altra persona esterna al gruppo di lavoro. È stata impostata per effettuare l'inoltro automatico delle mail in entrata verso gli indirizzi di tutti i componenti del gruppo, in modo tale che ognuno possa ricevere una copia della posta in ingresso. Soltanto il *Responsabile* ha il potere di inviare posta in uscita.

4.2.2 Comunicazioni interne

Per le comunicazioni interne verrà utilizzato un forum$_G$ apposito su *Redmine* (vedi sezione A), suddiviso in quattro sezioni principali corrispondenti alle revisioni di avanzamento del progetto:

- Revisione dei Requisiti;
- Revisione di Progettazione;
- Revisione di Qualifica;
- Revisione di Accettazione.

Ogni sezione è composta da due sottosezioni, riunioni e comunicazioni, le quali conterranno rispettivamente le riunioni indette seguite dalla disponibilità dei membri a partecipare e, a seguito di avvenuta riunione, il riassunto dei punti discussi e le comunicazioni di carattere più o meno generale riguardanti attività$_G$ della revisione di appartenenza.
Ogni topic$_G$ all'interno di queste due sottosezioni dovrà rispettare il seguente formato:

- **Riunioni:**
  Riunione Dominio - Data AAAA-mm-dd - Oggetto
  (es: Riunione Interna - 2015-12-29 - Strategie Ciclo di Vita)

- **Comunicazioni:**
  Oggetto - Data
  (es: Design Pattern - 2016-01-12)

La voce *Oggetto* dovrà essere esplicativa del contenuto, il più possibile stringata e non confondibile con precedenti topic$_G$. È ammesso l'inserimento di allegati purché strettamente pertinenti al topic$_G$, ad esempio il verbale di una riunione o il riassunto di discussioni avvenute al di fuori delle riunioni.
Vengono usati altri strumenti ufficiosi per lo scambio di informazioni:

- applicazioni di instant messaging$_G$ quali *Telegram* e il suo servizio *Telegram Web*;
- applicazioni VoIP$_G$ per effettuare audio conferenze con un qualsiasi numero di componenti, quale *TeamSpeak3*;
- servizi online per la gestione di una whiteboard$_G$ condivisa, quale *Twiddla*.

Qualora membri del gruppo si scambino informazioni attraverso questi strumenti sopra elencati dovranno, se presenti informazioni di interesse per il progetto, mandare un sunto della suddetta conversazione sul forum$_G$ per tenere informati tutti i membri del gruppo sullo stato di avanzamento nella sottosezione comunicazioni.
Solo in caso di necessità derivante dalla mancanza di infrastrutture (esċollegamento a internet) si potrà comunicare attraverso SMS o chiamate telefoniche; anche in questo caso si dovrà verbalizzare quanto detto attraverso il forum$_G$ appena possibile.

4.3 Riunioni

4.3.1 Riunioni Interne

Tutti i componenti del gruppo possono avanzare una richiesta per una riunione interna seguendo le regole apposite (vedi 4.3.3).
Sarà il *Responsabile di Progetto* a decidere se indire o meno la riunione proposta, controllando la disponibilità dei membri attraverso il forum$_G$ e, in caso, avvertendo tutti i componenti tramite un topic$_G$ come spiegato in sezione 4.2.2.
È richiesta la partecipazione di almeno quattro membri del gruppo. In casi particolari, come riunioni che trattano specifici ambiti non di interesse collettivo o per l'avvicinarsi a date di rilevante importanza, è possibile indire una riunione di gruppo con meno componenti presenti, sempre previa approvazione del *Responsabile*.

4.3.2 Riunioni esterne

Le riunioni esterne (incontri col *Committente* o *Proponente*) possono essere proposte da qualunque membro del gruppo. Spetta sempre al *Responsabile di Progetto* la decisione finale.
È necessaria la presenza di almeno il 50% + 1 dei componenti.

4.3.3 Regole per la richiesta

Le richieste vanno effettuate tramite mail al *Responsabile* e devono avere come oggetto:

`Richiesta riunione <tipo riunione>`

`tipo riunione` può essere interna o esterna. Il corpo della mail dovrà avere una struttura del tipo:

`Motivazione: <motivo riunione> Data proposta: <data> Luogo proposto: <luogo>`

4.3.4 Esiti Riunioni

Ad ogni riunione verrà affidato il compito a uno dei presenti di verbalizzare un riassunto sugli argomenti trattati e i chiarimenti emersi durante tale riunione.
Il redattore del verbale avrà poi l'obbligo di condividerlo con tutti sul forum$_G$ del gruppo.

4.4 Infrastruttura

4.4.1 Apparato di collaborazione

Per coordinare il lavoro tra i componenti del gruppo, trovandosi spesso a dover operare in remoto, il gruppo ha scelto di utilizzare quanto più possibile gli strumenti di condivisione e versionamento$_G$ erogati come servizi web, centralizzando gli strumenti organizzativi su server privato.

4.4.1.1 Servizi web

I servizi web utilizzati sono:

- **Google Drive:** è usato per lo scambio di file$_G$ e documenti all'interno del gruppo, principalmente per file$_G$ che non necessitano di versionamento$_G$.
  Questo spazio del gruppo è suddiviso in diverse cartelle per una migliore gestione e per facilitare la ricerca all'interno di esso;
  Il *Responsabile di Progetto* ha il potere amministrativo dell'account$_G$ in questione, quindi sarà quest'ultima persona ad occuparsi di garantire i giusti diritti ai membri del gruppo.
- **GitHub:** per tutti i file$_G$ che necessitano un sistema di versionamento$_G$ viene usato il servizio offerto da *GitHub*, l'indirizzo web del gruppo è:

  https://github.com/scalatekids
- **Twiddla:** è un sistema che offre una lavagna online condivisa, questo servizio è utilizzato dai membri del gruppo per discutere e lavorare assieme da remoto, utile all'attività$_G$ di brainstorming$_G$ e **Analisi**;
- **Gantter for Google Drive:** è uno strumento gratuito per la creazione di diagrammi di Gantt$_G$, utile alla creazione dei Gantt preventivi;
- **Lucidchart:** piattaforma comprensiva di un piano di utilizzo gratuito per studenti, utile per la produzione dei diagrammi UML$_G$ permettendo la collaborazione tra i componenti online.

4.4.2 Server dedicato

Sono state installate alcune applicazioni per il coordinamento del gruppo su server dedicato, gestito dall'*Amministratore* su direttive del *Responsabile*.
Il server è raggiungibile da interfaccia web previo login, i componenti possono usufruire delle seguenti applicazioni:

4.4.2.1 Redmine

Redmine è un'applicazione web atta alla gestione di progetti.
Offre diverse funzionalità che il gruppo dovrà utilizzare, quali:

- **Sezione wiki:** una pagina web in cui il gruppo andrà a condividere una serie di link$_G$ di interesse riguardanti ogni tecnologia o aspetto inerente al progetto.
- **Sistema di segnalazioni:** un sistema di segnalazioni gestite dall'*Responsabile*, ogni segnalazione rappresenta un task$_G$ assegnato ad uno o più membri del gruppo, il suo conseguimento servirà per il soddisfacimento di un requisito.
- **Gantt:** *Redmine* costruisce automaticamente un diagramma di Gantt$_G$ con le segnalazioni create dall'*Responsabile*

Per maggiori esplicazioni sull'utilizzo è stata redatta una sezione apposita A.

4.4.2.2 PHPMyAdmin

PHPMyAdmin è uno strumento scritto in linguaggio *PHP* che offre una facile amministrazione di *database*$_G$.
Questo strumento verrà usato per la creazione di un back-end per la gestione dei requisiti.
Per la gestione e il tracciamento dei requisiti è stata creata un'interfaccia protetta da login (vedi sezione 2.2.2).

4.4.3 Versionamento

Lo strumento scelto è git$_G$, principalmente per la grande diffusione negli ambienti di sviluppo e per l'integrazione offerta dal servizio web *GitHub*. Esso offre, inoltre, un'esaustiva documentazione sull'utilizzo del programma in questione. Infine git$_G$ si è rivelato essere il più conosciuto tra i membri del gruppo. L'alternativa SVN$_G$, nonostante i simili principi di funzionamento, costituiva una scelta più difficoltosa, in quanto nessun membro ha mai avuto esperienze di utilizzo passate.

4.4.3.1 Repository

Sono stati creati due repository$_G$ all'indirizzo `https://github.com/scalatekids`:

- Actorbase.git$_G$: Conterrà i sorgenti del software vero e proprio;
- Actorbase-Documents.git$_G$: Conterrà i sorgenti LaTeX e i PDF$_G$ generati;

- Actorbase-Docker.git$_G$: Conterrà i sorgenti delle immagini Docker$_G$ e i sorgenti dei test di unità da inserire di volta in volta. Si suddivide in:
  - Actorbase-Docker/Latex.git$_G$: Conterrà il Dockerfile$_G$ istruito per la creazione di un'immagine *Ubuntu:Trusty* fornita dei pacchetti latex essenziali alla stesura dei documenti, corredata da script$_G$ di generazione, verifica ortografica e leggibilità dei PDF$_G$ generati.
  - Actorbase-Docker/Scala.git$_G$: Conterrà il Dockerfile$_G$ istruito per la creazione di un'immagine *Ubuntu:Trusty* fornita di *JVM v8* e sbt$_G$, corredata dai test di unità e script$_G$ di analisi statica e qualitativa del codice$_G$.

I branch$_G$ dovranno essere nominati completamente in minuscolo, in lingua inglese ed essere esplicativi del loro utilizzo.

4.5 Integrazione continua

Fin da subito, i repository$_G$ saranno collegati ad un sistema di integrazione continua$_G$, la scelta è ricaduta su *Travis-CI* per la semplicità di utilizzo e modeste esperienze passate di alcuni componenti del gruppo.

4.5.0.1 Travis-CI

È un servizio di integrazione continua$_G$ comprensivo di un piano di utilizzo gratuito. Facilmente configurabile, il collegamento ai repository$_G$ *GitHub* avviene mediante un file$_G$ di configurazione chiamato *.travis.yml* da inserire all'interno del repository$_G$. È inoltre possibile fornire direttive al servizio e automatizzare unità di test su tutti i file$_G$ del repository$_G$, le quali possono essere lanciate sia dopo ogni push$_G$ che dopo ogni pull$_G$ e notificare il *Responsabile* via mail su eventuali commit$_G$ che non superino i test predisposti o violino determinate regole.

4.5.0.2 Docker

Dopo varie ricerche, *Docker$_G$ versione 1.9.1* è stato adottato per uniformare quanto più possibile l'ambiente di test, grazie al sistema di containerizzazione$_G$ e versionamento$_G$ delle immagini virtuali che offre.
Mediante un file$_G$ di configurazione principale, il Dockerfile$_G$, è possibile istruire il programma e generare una o più immagini virtuali fornite dei soli pacchetti o script$_G$ necessari al testing a cui saranno adibite. Utilizzato in coppia con *Travis-CI* permette di effettuare test molto precisi e, allo stesso tempo, di versionare un gran numero di immagini virtuali per diversificare i test e gli ambienti su cui lanciarli. I Dockerfile$_G$ infatti possono essere versionati mediante un repository$_G$ *GitHub* direttamente collegato a *Travis-CI*, il quale, istruito ad hoc, si occuperà di generare l'immagine Docker$_G$ e caricarla mediante un push$_G$ sull'hub$_G$ Docker$_G$ presente nel sito `hub.docker.com`.
In questo modo i test potranno essere versionati di pari passo con l'attività$_G$ di sviluppo, garantendo un accurata attività$_G$ di verifica sui sorgenti e documenti senza gravare in alcun modo sull'ambiente di lavoro di ogni singolo membro.

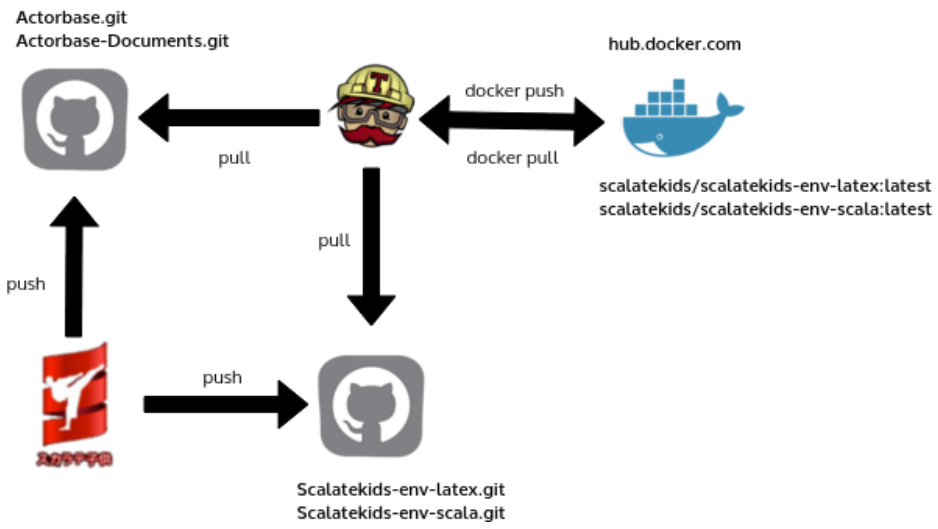


Figura 4: Integrazione continua - Travis-CI + Docker

Attualmente è in uso l'immagine `scalatekids-latex` per la verifica dei documenti.

4.5.1 Ambiente di lavoro individuale

Per il lavoro individuale, è stato pensato ad un modo di uniformare quanto più possibile l'ambiente di sviluppo tra i componenti; il *Responsabile* e l'*Amministratore* hanno stabilito il sistema operativo e l'IDE$_G$ di base ed è stata creato un ambiente virtuale comune utilizzando il software *Virtualbox versione 5.0.12* o superiore. Il sistema scelto è *Ubuntu 14.04*. Tutti i componenti del gruppo utilizzeranno gli strumenti all'interno della virtual machine$_G$ in modo da garantire omogeneità, e riproducibilità di eventuali bug$_G$.
L'*IDE* scelto è *IntelliJIdea*, considerato il più appropriato e compatibile con le tecnologie centrali del progetto. Gli editor *vim*, *emacs* o *Sublime Text 2* potranno essere utilizzati dai componenti del gruppo, purché si tratti di modifiche rapide o stesura di script$_G$ di utilità generale.
Per la stesura dei documenti, l'IDE$_G$ di riferimento è *TeXstudio*.

4.5.1.1 Installazione virtual machine

Una volta ottenuto il file$_G$ *.vdi* presente sul *Drive*$_G$ del gruppo in forma compattata *.7z*, ogni componente dovrà installare il software *Virtualbox* sul proprio computer e procedere con la creazione della macchina virtuale.
I passi da seguire sono:

- Nuovo;
- Tipo: *Ubuntu* Versione: *64-bit*;
- Assegnare la quantità di memoria in base alla propria disponibilità;

- Usa un hard-disk esistente e selezionare *ScalateKids-VMvX.Y.vdi*.

Una volta avviata la macchina, da terminale lanciare il comando:

```
curl http://codep.kissr.com/sk/scripts/init.sh | sh
```

Il comando provvederà al download delle librerie e degli script$_G$ necessari alla corretta configurazione dell'ambiente di lavoro del gruppo. Al termine la macchina virtuale sarà pronta ad essere utilizzata per lo sviluppo.

4.5.1.2 Versionamento virtual machine

Di volta in volta, lungo tutto l'arco del progetto, quando sarà necessario modificare l'ambiente di lavoro o ci sarà bisogno di librerie di supporto ulteriori o modifica di script$_G$ di utilità, il seguente comando si occuperà di scaricare e installare gli aggiornamenti richiesti:

```
curl http://codep.kissr.com/sk/scripts/update.sh | sh
```

Gli script$_G$ di utilità citati comprenderanno strumenti di verifica e leggibilità dei documenti.`Init.sh` e `update.sh`, presenti sul server privato, saranno preparati dall'*Amministratore* e rilasciati di volta in volta previa approvazione del *Responsabile*.

4.5.1.3 Script di utilità locali

L'*Amministratore* si è occupato di scrivere alcuni script$_G$ di utilità generale da utilizzare all'interno della macchina virtuale:

- **build:** lanciato senza argomenti, genera tutti i PDF$_G$ del progetto dai ṫex, eliminando tutti i file$_G$ `.log .aux .out .soc .lof .lot` e `.toc` generati. Può avere un numero indefinito di argomenti, purché siano file$_G$ ṫex, questi verranno allo stesso modo generati eliminando tutti i file$_G$ `.log .aux .out .soc` e `.toc`;
- **extgloss:** estrae tutti i termini da glossario contrassegnati da `\gloss` dai file$_G$ del progetto, anche questo comando può essere lanciato con o senza argomenti;
- **verifygloss:** genera una lista di termini da glossario leggendo il file$_G$ Glossario v3.0.0. In seguito controlla tutti i file$_G$ del progetto (eccetto il glossario), e inserisce il comando `\gloss` sui termini contenuti in lista trovati nei file$_G$. Se il comando è seguito da argomenti esegue le medesime operazioni soltanto sui file$_G$ ṫex scelti;
- **verify:** scansiona tutti i documenti `.tex` del progetto alla ricerca di errori comuni e violazioni delle *Norme di Progetto*, nonché TODO o merge$_G$ irrisolti. Lanciato con l'opzione `-c` esegue le correzioni grammaticali basilari sui documenti che riportano una o più occorrenze di tali sviste.
  Opzionalmente accetta in input liste di errori grammaticali comuni;
- **readability:** Calcola alcune statistiche sul testo, come numero frasi, parole e sillabe, da queste ricava l'indice di Gulpease sulla leggibilità.

4.5.1.4 Script di utilità remoti

Questi script$_G$, **init.sh** e **update.sh**, risiederanno sul server privato, e avranno rispettivamente lo scopo esclusivo di inizializzare e aggiornare l'ambiente di lavoro della virtual machine$_G$.

4.5.1.5 Scala e Java Virtual Machine

La scelta sulla versione di Scala da utilizzare è ricaduta sulla major$_G$ 2.11, principalmente dovuta ai requisiti minimi delle librerie *akka*$_G$, e ad una maggiore sicurezza sul mantenimento del codice$_G$. Infatti le due versioni principali di Scala$_G$ risultano essere non compatibili a livello binario, pertanto è preferibile rimanere sulla versione più recente. La versione 2.11 necessita della JVM$_G$ versione 8.

4.5.1.6 Akka

La versione di riferimento di Akka$_G$ è la `2.4.1`, attualmente mantenuta e stabile, da utilizzare con versioni di Scala$_G$ `2.11` o `2.12.0-M3` e Java 8+.

5 Assicurazione Qualità

5.1 Norme di verifica per i processi

Per avere una misurazione oggettiva e quantitativa della qualità su ogni processo sono state stabilite delle metriche di valutazione descritte nel documento Piano di Qualifica v3.0.0.
Tali misurazioni saranno effettuate periodicamente ogni venti giorni e ad ogni misurazione dovrà essere aggiornata l'apposita appendice del documento Piano di Qualifica v3.0.0.

5.1.1 Procedure di gestione degli errori secondo le metriche definite nel piano di qualifica

Nella tabella che segue vengono riportate le principali procedure di gestione degli errori secondo le metriche definite nel Piano di Qualifica v3.0.0.

Errore trovato	**Gravità**	**Priorità soluzione**	**Modalità operative**
Ritardo maggiore di 3 giorni nelle attività	Alta	Urgente	Ticketing
Errato tracciamento di requisiti e/o casi d'uso	Alta	Urgente	Ticketing
Errore progettazione	Alta	Urgente	Ticketing
Valori gulpease fuori range	Media	Breve	Ticketing
Errore compliazione codice	Alta	Urgente	Ticketing o correzione
Errore compilazione documenti	Alta	Urgente	Ticketing o correzione
Mancato rispetto delle norme di codifica	Media	Breve	Ticketing
Errore formalismo UML	Bassa	entro la milestone	Ticketing
Errori ortografici o di formattazione	Bassa	entro la milestone	Ticketing

Tabella 2: Procedure gestione degli errori

5.2 Norme di verifica per i prodotti

Per avere una misurazione oggettiva e quantitativa della qualità su ogni prodotto sono state stabilite delle metriche di valutazione per documenti e software descritte nel documento Piano di Qualifica. Queste misurazioni dovranno effettuate ogniqualvolta viene prodotto un file, sia esso un documento o un file di codice.

5.3 Norme di verifica documenti

Posto che tutti i componenti del gruppo dovranno effettuare delle attività$_G$ di verifica, il redattore di ogni sezione dei documenti non potrà tassativamente effettuare verifica e approvazione delle parti redatte dallo stesso, coerentemente con le strategie scelte nel **PQ**. I *Verificatori* dovranno attenersi alle seguenti regole per attuare attività$_G$ di verifica:

- **Walkthrough:** per svolgere questa attività$_G$ il *Verificatore* deve passare in rassegna il documento da controllare, rileggere tutto attentamente e stendere una lista degli errori più comuni. Questa lista servirà per attuare solamente verifiche di tipo inspection in futuro;
- **Inspection:** questa attività$_G$ va effettuata solamente quando si ha a disposizione una lista di controllo derivante dalle attività$_G$ di walkthrough precedenti.
  Il *Verificatore* deve controllare secondo la lista di controllo le parti del documento scelto.

Per eseguire un'accurata verifica dei documenti è necessario seguire il seguente protocollo:

- **Controllo ortografico:** Utilizzando *GNU Aspell* vengono evidenziati e corretti gli errori grammaticali più evidenti. Inoltre, come spiegato in sezione 4.5.1.3, sono stati creati alcuni strumenti ad hoc per automatizzare le attività$_G$ di verifica.
  Tuttavia il controllo dei periodi e di parole grammaticalmente corrette ma errate nel contesto richiedono l'intervento di un *Verificatore* umano. Ciò implica che ogni documento dovrà essere sottoposto ad un walkthrough;
- **Uniformità alle norme:** Essendo state redatte alcune regole sulla stesura dei documenti, è compito dei *Verificatori* accertarsi che i documenti siano conformi a tali norme, seguendo il metodo di walkthrough;
- **Verifica termini da glossario:** L'operazione è fortemente automatizzabile, compito del *Verificatore* è utilizzare gli strumenti di supporto forniti nell'ambiente di lavoro descritti nella sezione 4.5.1.3. Tuttavia l'attività$_G$ di verifica finale dovrà essere effettuata secondo il metodo walkthrough;
- **Indice Gulpease:** Su ogni documento redatto il *Verificatore* deve calcolare l'indice di leggibilità utilizzando lo strumento fornito con la macchina virtuale (vedi sezioni 4.5.1.1 e 4.5.1.3);
- **Segnalazione degli errori riscontrati:** Il *Verificatore* dovrà stendere una lista di controllo e inserirla sul forum$_G$ del gruppo.

Successivamente, basandosi sui cambiamenti del diario delle modifiche, i *Verificatori* potranno applicare il metodo di inspection seguendo la lista redatta in prima istanza di verifica.

5.4 Norme di verifica codice

5.4.0.1 Analisi statica

Gli strumenti da utilizzare per le attività$_G$ di analisi statica saranno:

- **ScalaStyle:** plugin$_G$ per compilatori Scala$_G$, fornisce automaticamente controlli qualitativi esaustivi su codice$_G$ durante la compilazione dei sorgenti;
- **loc:** script$_G$ di controllo qualità, fornisce il rapporto tra righe di codice$_G$ e righe di commento.
- **Complessità ciclomatica (McCabe's complexity):** Utilizzata per misurare la complessità di funzioni, moduli, metodi o classi di un programma;
- **Copertura del codice:** Misura la capacità di coprire mediante esecuzione di test tutte le linee di codice di un programma; si suddivide a sua volta in:
  - **Copertura di funzione:** Ogni funzione all'interno del programma è stata chiamata;

- **Copertura di istruzione:** Ogni istruzione all'interno del programma è stata eseguita;
- **Branch$_G$ coverage:** Misura la capacità di coprire mediante esecuzione di test tutti i rami di un modulo;
- **Condition coverage:** Ogni espressione booleana è stata valutata sia in true che in false;

Lo strumento scelto per l'attività in questione è **sbt-coveralls**, si presta perfettamente allo scopo in quanto integrato con lo strumento di integrazione continua scelto$_G$ (Travis-CI);

- **Numero di attributi per classe:** Numero pesato di attributi contenuti in ogni classe di un programma;
- **Parametri per metodo:** Numero pesato di parametri utilizzabili nella chiamata di metodi all'interno di una classe;
- **Livelli di annidamento:** Numero pesato che misura il grado di profondità delle funzioni e dei costrutti all'interno di un programma.

5.4.0.2 Analisi dinamica

Gli strumenti da utilizzare per le attività$_G$ di analisi dinamica saranno:

- **Apache ab:** Uno strumento utilizzato per effettuare benchmark$_G$ su sistemi basati su comunicazioni HTTP$_G$;
- **GNU parallel**: Uno strumento da utilizzare su shell$_G$ per lanciare più processi contemporaneamente, verrà utilzzato in congiunzione ad ab per effettare test di carico sul prodotto.

5.4.1 Test di unità

Sono test destinati al collaudo della più piccola porzione di codice che abbia senso testare, compito dei *progettisti* è la stesura e il tracciamento dei test di unità con i requisiti e viceversa secondo le regole di nomenclatura seguenti:

`TU.NomeTest`

5.4.2 Test di integrazione

Compito dei *progettisti* è la stesura e il tracciamento dei test di integrazione con le componenti secondo le regole di nomenclatura seguenti:

`TI.NomeTest`

5.4.3 Test di sistema

I test di sistema permettono di verificare il comportamento del sistema rispetto ai requisiti. Compito dei *progettisti* è la stesura e il tracciamento dei test di sistema con i requisiti e viceversa secondo le regole di nomenclatura seguenti:

`TS.NomeTest`

5.4.4 Test di validazione

Compito dei *progettisti* è la stesura e il tracciamento dei test di validazione con i requisiti, secondo le seguenti regole di nomenclatura:

`TV.NomeTest`

A Redmine

Questa sezione rappresenta un tutorial su come usare correttamente *Redmine*. Il software in questiona offre una vasta gamma di servizi molto utili per la gestione di un progetto, in particolare il *forum*, la sezione wiki$_G$ ed il sistema di segnalazioni che automaticamente costruisce il diagramma di *Gantt*. Tramite installazioni di plugin$_G$ è possibile aggiungere funzionalità. Qualora un componente del gruppo ritenga necessaria l'installazione di un plugin$_G$, costui può proporla al *Responsabile di Progetto* il quale affiderà il compito ad un *Amministratore* se la riterrà una funzionalità utile.

A.1 Creare un sottoprogetto

La creazione di sottoprogetti è molto utile per mantenere una chiara divisione funzionale tra le attività$_G$. Si tratta di un'operazione molto semplice, come si vede in figura sottostante

Figura 5: come creare un sottoprogetto - passo 1

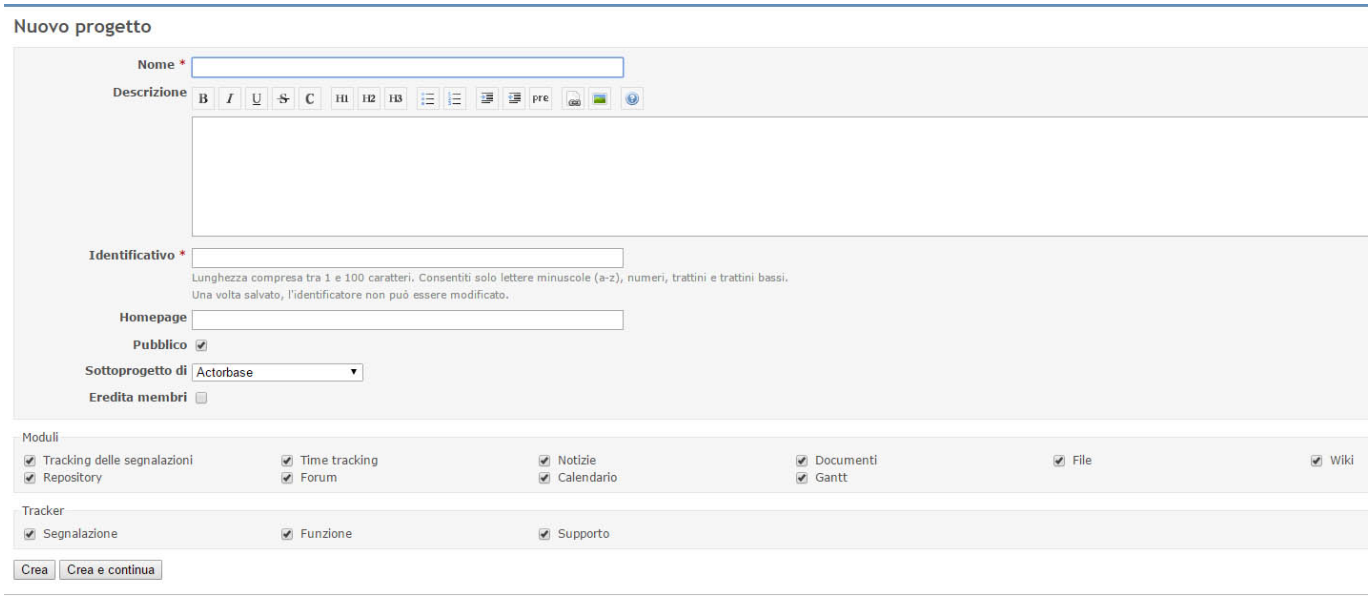


Figura 6: come creare un sottoprogetto - passo 2

Come si può vedere in figura 6 si dovrà completare un form$_G$ con i dati necessari per la creazione del progetto.

A.2 Scrivere sul Forum

Si può partecipare al forum$_G$ rispondendo a messaggi o creandone di nuovi

A.2.1 Creare un messaggio

Per creare un nuovo messaggio basta andare nel topic$_G$ desiderato e seguire la procedura come illustrato nelle seguenti immagini

Figura 7: come creare un messaggio - passo 1

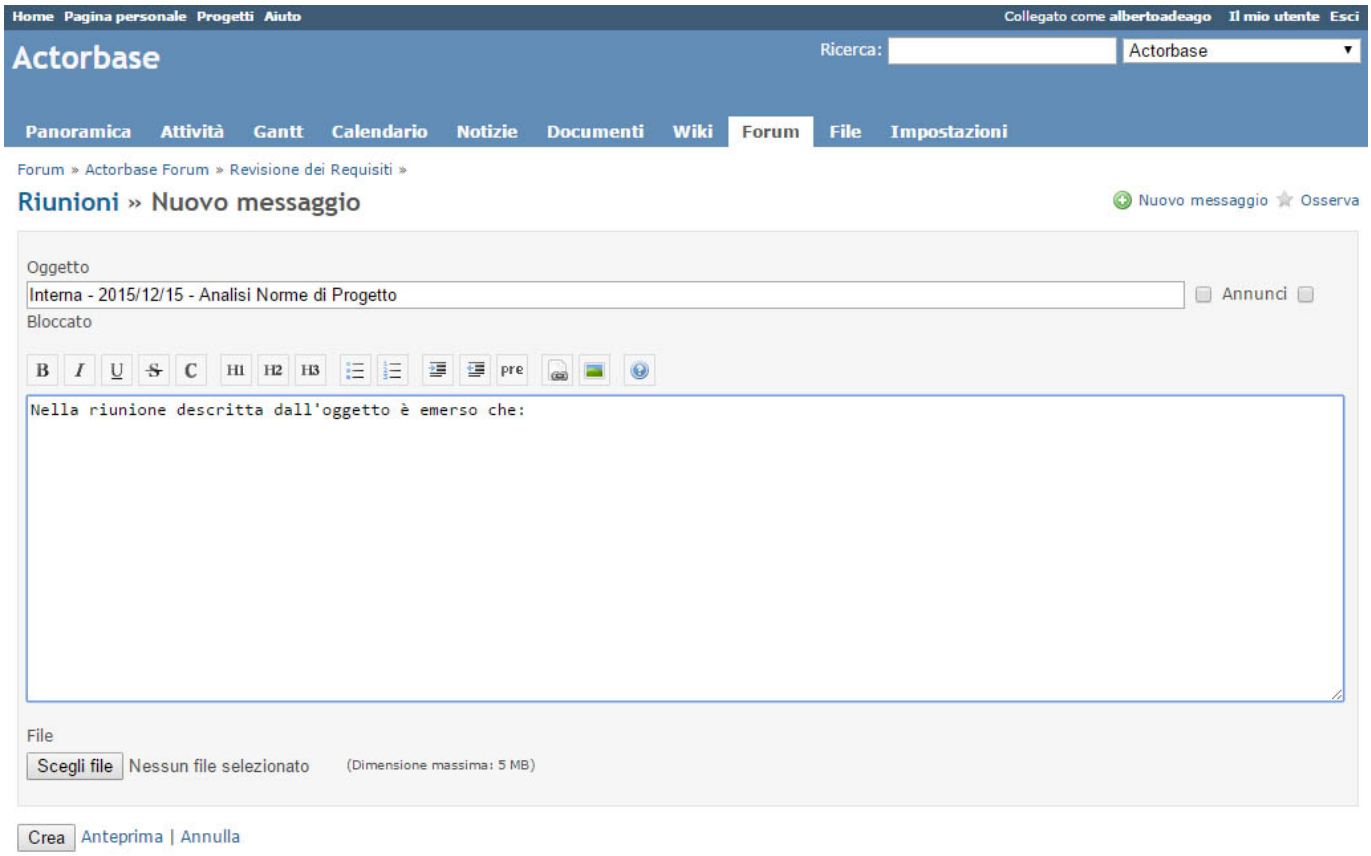


Figura 8: come creare un messaggio - passo 2

Per la corretta creazione di un messaggio è necessario seguire le norme scritte nella sezione 4.2.2

A.2.2 Rispondere ad un messaggio

Per rispondere ad un messaggio presente nel forum$_G$ è necessario aprire il messaggio desiderato e clickare su **Rispondi**. Automaticamente si presenterà un form$_G$ *pre compilato* in cui sarà necessario solamente inserire il corpo della nostra risposta.

A.3 Gestione delle segnalazioni

Una segnalazione rappresenterà una attività$_G$ che il *Responsabile* assegnerà ad uno o più membri del gruppo. Queste attività$_G$ possono essere suddivise in diversi compiti più semplici da assegnare ad un membro; quest'ultimi saranno chiamati task$_G$.

Il sistema di segnalazioni è la cosa più importante di *Redmine,* perciò solamente il *Responsabile di Progetto* ha il potere di creare nuove segnalazioni così da mantenere ordine nel progetto.
Gli altri membri del gruppo potranno invece aggiornare le segnalazioni a loro assegnate.

A.3.1 Creare una segnalazione

Il *Responsabile di Progetto* può creare nuove segnalazioni come illustrato nelle seguenti immagini

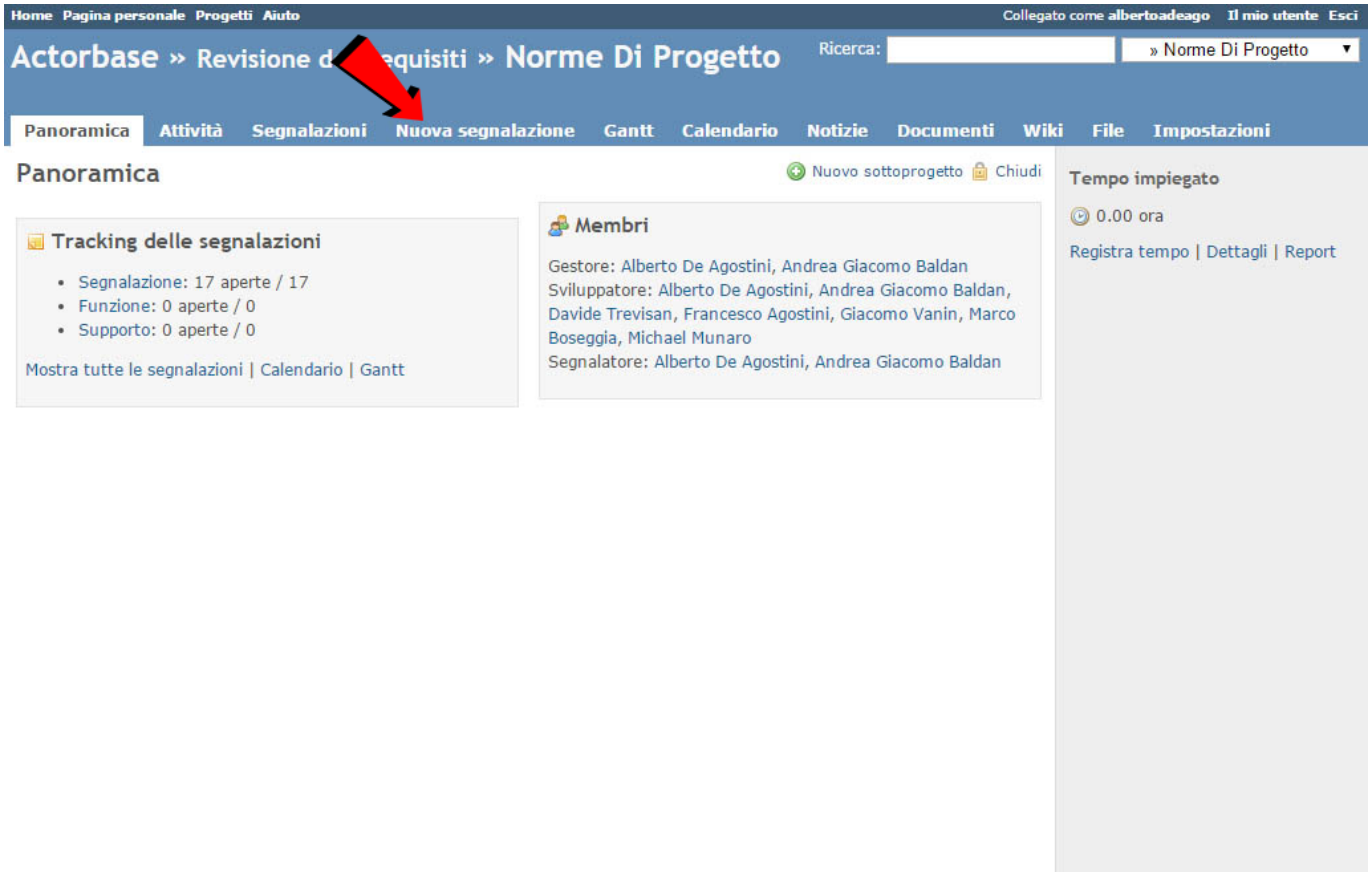


Figura 9: come creare un segnalazione - passo 1

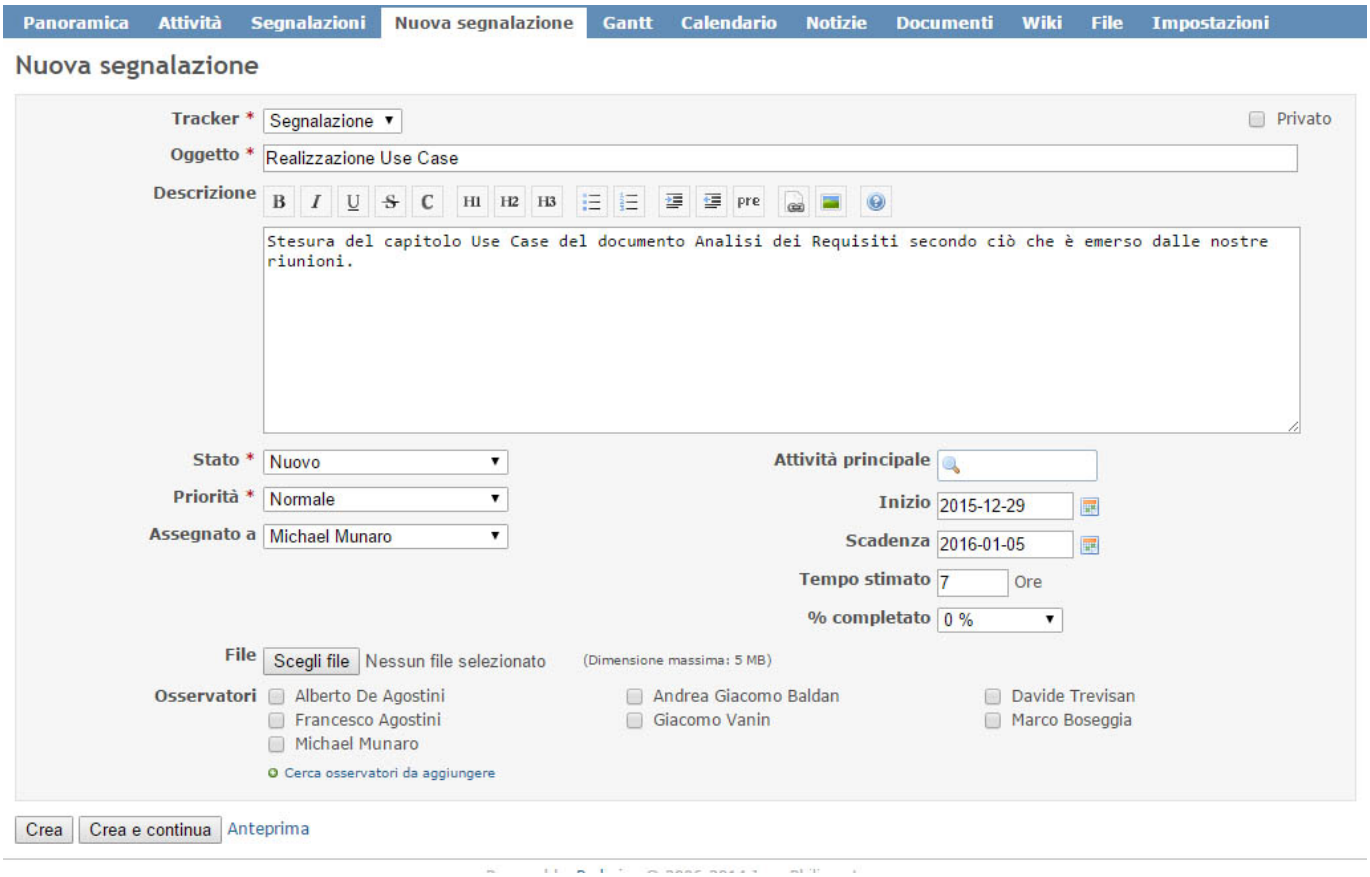


Figura 10: come creare una segnalazione - passo 2

Dalla figura 10 è possibile notare diversi campi da compilare:

- **Tracker:** lasciare selezionato *Segnalazione*;
- **Oggetto:** l'oggetto della segnalazione, questo deve essere breve e non equivocabile;
- **Descrizione:** descrizione dettagliata dell'attività$_G$ da svolgere;
- **Stato:** se è una nuova segnalazione deve rimanere *nuovo*, altrimenti è possibile scegliere altre opzioni; lo stato viene cambiato dall'intestatario della segnalazione stessa;
- **Assegnato a:** la persona che deve svolgere l'attività$_G$;
- **Attività$_G$ principale:** se si tratta di un sotto task$_G$ di un'altra attività, bisogna inserire l'identificatore del task$_G$ padre;
- **Inizio:** data in cui deve iniziare l'attività$_G$;

- **Fine:** data entro cui l'attività$_G$ deve essere completata;
- **Tempo stimato:** il numero di ore stimato entro cui si dovrebbe completare l'attività$_G$;
- **% completato:** la percentuale di completamento dell'attività$_G$; anche questo parametro viene utilizzato dal membro del gruppo che lavora sull'attività$_G$.

A.3.2 Aggiornare una segnalazione

Il lavoro di aggiornamento di una segnalazione deve essere svolto dai vari membri del gruppo che lavorano alla segnalazione stessa.
Ad ogni avanzamento relativo ad una attività$_G$ si deve aggiornare la segnalazione relativa ad essa. Per fare ciò bisogna entrare nella pagina personale di *Redmine* e, nella sezione **Le mie segnalazioni**, selezionare la segnalazione desiderata, quindi premere su **Modifica**. La schermata che si presenterà davanti permetterà diverse operazioni come mostrato in figura 10, si dovrà quindi aggiornare lo **stato** (se cambiato), la **% completata** e aggiornare inoltre il **tempo impiegato**. Vi è inoltre la possibilità di inserire un testo libero; questo può essere utile per spiegare i cambiamenti apportati.

A.4 Modificare la Wiki

La nostra scelta è stata di usarla come raccoglitore di link$_G$ a guide inerenti al nostro progetto. Per contribuire alla Wiki$_G$ basta andare nella suddetta sezione e premere su **modifica**. Si presenterà un editor di testo in cui è possibile modificare o semplicemente aggiungere del testo.

A.5 Aggiungere impegni al calendario

Ogni collaboratore presente su Redmine ha la possibilità di aggiungere i propri impegni sul calendario per facilitare così la suddivisione delle attività da pianificare da parte del *Responsabile*.
Ognuno è invitato a inserire i propri impegni su questo calendario il prima possibile per agevolare il lavoro di pianificazione.

B SBT

Sbt (Simple Build Tool) è un ambiente di build$_G$ simile alla controparte Java Maven$_G$. Fornito di console$_G$ e strumenti di compilazione ed esecuzione, è integrato nella maggior parte degli strumenti di sviluppo utilizzati per progetti Scala$_G$, incluso Intellijidea. Alcuni test e funzionalità del prodotto possono tuttavia richiederne l'utilizzo da riga di comando, in questa sezione vengono brevemente descritti i comandi di uso comune.

B.1 Configurazione

Come prima cosa è necessario assicurarsi di avere installato nel proprio sistema java, aggiornato alla versione 8. Poi sarà necessario installare SBT da repository nel proprio sistema. Per creare un nuovo progetto è necessario avere una struttura di directory che sbt sia in grado di riconoscere. Noi abbiamo preso come modello la struttura di maven. Poi per definire nome del progetto dipendenze e librerie è necessario scrivere un file chiamato "build.sbt" che conterrà: nome del progetto, versione del progetto, versione di scala utilizzata nel progetto, libreire e dipendenze da necessarie ed eventuali plugin di sbt.

B.2 Accesso alla console Scala

Sbt offre la possibilità di accedere mediante la sua cli alla scala console. Per accedervi è sufficiente lanciare il comando `sbt` che ci aprirà una nuova istanza della console di sbt. Dalla quale è possibile manualmente configurare progetti versioni e ogni tipo di impostazione di sbt. Per aprire la console scala è sufficiente dare il comando `consoleProject` su CLI di sbt per avviare la scala console.

B.3 Compilazione di progetti

Per la compilazione di un progetto sbt, una volta configurato a dovere il file build.sbt sarà sufficiente dalla directory del progetto dare il comando `sbt compile` per compilare il progetto e lanciarlo. Alla prima esecuzione tale comando dovrà occuparsi di risolvere tutte le dipendenze del progetto e ottenerle dalla rete. Di conseguenza è necessaria una connessione. Dopo che sarà stata scaricata la corretta versione di scala e le librerie il verrà compilato il progetto e mostrati eventuali messaggi di errore. Dalla seconda esecuzione in poi il progetto verrà solo compilato, usando scala e librerie precedentemente scaricate. Se si desidera fare un clean e riscaricare nuovamente da capo scala e librerie sarà sufficente dare il comando `sbt clean` compile, che eseguirà un clean del progetto per poi ricompilare il tutto da capo.

B.3.1 Esecuzione del codice Scala

Per l'esecuzione del progetto sbt dalla directory home è sufficiente lanciare il comando `sbt run` che automaticamente eseguirà una compilazione se ci sono modifiche dall'ultima compilazione nota, e poi eseguirà il progetto.

B.3.2 Esecuzione dei test di unità previsti

Per l'esecuzione dei test previsti, sbt prevede il comando `sbt test`, fornendo i risultati e il tempo calcolato all'esecuzione delle unità di test.

C GNU Parallel

GNU Parallel permette di eseguire più istanze di un programma contemporaneamente, può essere quindi utilizzato per testare le capacità del sistema sotto sforzo. E' necessario fornire una lista di richieste da effettuare con Apache ab e impostare alcuni parametri, come il numero di thread da lanciare, il numero di richieste per thread e il tipo di connession HTTP$_G$ da utilizzare. Un esempio di utilizzo in congiunzione ad Apache ab e grep$_G$, in modo da stampare solamente le richieste per secondo soddisfatte da un generico servizio HTTP$_G$:

```
➜  Actorbase-Server git:(nomanager) ✗ cat src/main/resources/URLs.txt | parallel -j 5 'ab -ql -n 2000 -c 1 {}' | grep 'Requests per second'
When using programs that use GNU Parallel to process data for publication please cite:

  O. Tange (2011): GNU Parallel - The Command-Line Power Tool,
  ;login: The USENIX Magazine, February 2011:42-47.

This helps funding further development; and it won't cost you a cent.
Or you can get GNU Parallel without this requirement by paying 10000 EUR.

To silence this citation notice run 'parallel --bibtex' once or use '--no-notice'.

Requests per second:    28.04 [#/sec] (mean)
Requests per second:    28.04 [#/sec] (mean)
Requests per second:    28.05 [#/sec] (mean)
Requests per second:    27.81 [#/sec] (mean)
Requests per second:    27.98 [#/sec] (mean)
➜  Actorbase-Server git:(nomanager) ✗
```

Figura 11: GNU Parallel e Apache ab - 2000 richieste da 5 processi paralleli, per un totale di 10000 richieste parallele

Elenco delle figure