24 giugno 2013

Norme di Progetto

Informazioni sul documento

Nome Documento	Norme di Progetto
Versione	4.0
Stato	*Formale*
Uso	*Interno*
Data Creazione	30 novembre 2012
Data Ultima Modifica	24 giugno 2013
Redazione	Elena Zerbato
Approvazione	Matteo Belletti
Verifica	Sara Lazzaretto
Lista distribuzione	*GoGo Team*
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Proponente Zucchetti S.p.A.

Sommario

Il presente documento contiene le norme e le convenzioni che il gruppo *GoGo Team* intende adottare durante l'intero ciclo di vita del prodotto software **MyTalk**.

Registro delle modifiche

Versione	Autore	Data	Descrizione
4.0	Matteo Belletti	2013-06-24	Approvazione. Approvazione del documento, cambio di stato in Formale a uso interno e avanzamento di versione.
3.2	Sara Lazzaretto	2013-06-24	Verifica e correzioni. Verifica del documento e correzione degli errori lessicali ed ortografici.
3.1	Elena Zerbato	2013-06-24	Modifica Contenuti. Aggiunti particolari nei capitoli descrittivi delle attività.
3.0	Francesco Zattarin	2013-06-10	Approvazione. Approvazione del documento, cambio di stato in Formale a uso interno e avanzamento di versione.
2.3	Sara Lazzaretto	2013-05-30	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Elena Zerbato in data 2013-05-29.
2.2	Sara Lazzaretto	2013-05-28	Aggiunte sezioni di *Progettazione di dettaglio* e *Verifica e validazione*; estensione della sezione di *Codifica*.
2.1	Sara Lazzaretto	2013-02-11	Modifica struttura interna del documento. Aggiunte le sezioni di *Analisi* e *Progettazione architetturale*.
2.0	Davide Ceccon	2013-01-28	Approvazione. Approvazione del documento, cambio di stato in Formale a uso interno e avanzamento di versione.

1.5	Matteo Belletti	2013-01-26	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Sara Lazzaretto in data 2013-01-25.
1.4	Francesco Zattarin	2013-01-24	Aggiunta Contenuti. Stesura della sessione *Ambiente di verifica e validazione* e integrazione di contenuti.
1.3	Alessandro Bonaldo	2013-01-21	Correzione. Correzione degli errori lessicali ed ortografici.
1.2	Alessandro Bonaldo	2013-01-20	Aggiunta Contenuti. Integrazione nomenclatura diagrammi UML.
1.1	Francesco Zattarin	2013-01-18	Correzione. Correzione a seguito delle segnalazioni effettuate in sede di RR dal *Committente* prof. Tullio Vardanega in data 2013-01-11.
1.0	Davide Ceccon	2012-12-07	Approvazione. Approvazione del documento, cambio di stato in Formale a uso interno e avanzamento di versione
0.5	Francesco Zattarin	2012-12-06	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Francesco Zattarin in data 2010-12-05. Correzione. Correzione errori lessicali ed ortografici.
0.4	Valentina Pasqualotto	2012-12-05	Completamento. Termine stesura di tutte le sezioni.
0.3	Valentina Pasqualotto	2012-12-05	Aggiunta Contenuti. Stesura della sezione *Procedura per lo sviluppo delle applicazioni.* Correzione. Correzione degli errori lessicali ed ortografici.

0.2	Elena Zerbato	2012-12-04	Aggiunta Contenuti. Completata la normativa *documenti*
0.1	Valentina Pasqualotto	2012-11-30	Prima Stesura. Contenuti documento: scheletro di base dell'intero documento, scelta versionamento, ambiente di sviluppo, norme di scrittura, procedura di sviluppo applicazione e contenuto dei documenti.

Tabella 1: Versionamento del documento

Storico

pre-RR

Versione 1.0	Nominativo
Redazione	Valentina Pasqualotto, Elena Zerbato
Verifica	Francesco Zattarin
Approvazione	Davide Ceccon

Tabella 2: Storico ruoli pre-RR

RR ->RP

Versione 2.0	Nominativo
Redazione	Alessandro Bonaldo, Francesco Zattarin
Verifica	Matteo Belletti
Approvazione	Davide Ceccon

Tabella 3: Storico ruoli RR ->RP

RP ->RQ

Versione 3.0	Nominativo
Redazione	Sara Lazzaretto
Verifica	Elena Zerbato
Approvazione	Francesco Zattarin

Tabella 4: Storico ruoli RP ->RQ

RQ ->RA

Versione 3.0	Nominativo
Redazione	Elena Zerbato
Verifica	Sara Lazzaretto
Approvazione	Matteo Belletti

Tabella 5: Storico ruoli RQ ->RA

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Le Norme di Progetto hanno l'obiettivo di fornire una serie di norme, convenzioni e formalismi che ogni membro del gruppo deve adottare durante l'attività di progetto per uniformarsi al resto del team. Il presente documento viene integrato ogni qualvolta si presenti la necessità di chiarire dubbi o definire i comportamenti da seguire in specifiche situazioni. In alcuni casi si consiglia una prassi da seguire piuttosto che una norma rigida.

1.2 Glossario

Ai documenti formali viene allegato il Glossario nel file *Glossario_v3.0.pdf* nel quale vengono definiti tutti i termini indicati, seguendo specifica convenzione, all'interno del documento.

2 Comunicazioni

2.1 Interne

Per gli avvisi generici rivolti all'intero gruppo, è stato aperto uno spazio privato ai componenti del gruppo basato su piattaforma Google Groups.

https://groups.google.com/d/forum/gogoteam-swe

Per avere un'interazione più rapida tra singoli membri del gruppo, si è scelto di utilizzare il software$_{|g|}$ TeamSpeak (http://www.teamspeak.com).
Quest'ultimo permette di interagire con i componenti del gruppo qualora non sia possibile la riunione in una sede comune.

2.2 Esterne

Le comunicazioni verso il *Committente* e *Proponente* sono gestite direttamente dal Responsabile di Progetto attraverso l'indirizzo di posta elettronica:

gogoteam.info@gmail.com

Il Responsabile deve tenere informato il gruppo rispetto ad eventuali comunicazioni da e per gli esterni secondo le modalità indicate nella sezione Comunicazioni Interne (cap. 2.1).

3 Riunioni

3.1 Interne

1. Ciascun componente del gruppo può avanzare una richiesta di riunione interna. Tale richiesta deve pervenire al Responsabile via e-mail ufficiale del gruppo.

2. Una volta valutata la motivazione, allegata alla richiesta della nuova riunione, il Responsabile verifica la disponibilità nel calendario di tutti i membri del gruppo.

3. Il Responsabile, entro due giorni lavorativi, pubblica un nuova discussione sulla pagina dedicata al gruppo, sulla piattaforma Google Groups (cap. 2.1), indicando la data e il luogo dell'incontro con prefissato il tag `[Riunione Interna]`.

4. La riunione può essere svolta con l'accettazione da parte di almeno altri tre membri del gruppo (Responsabile a parte). Qualora non si raggiunga il numero minimo di componenti si ritorna al punto due.

3.1.1 Casi particolari

La richiesta di riunione interna è stata indetta nell'avvicinamento (cinque giorni lavorativi) di una milestone$_{|g|}$. Se approvata dal Responsabile, la riunione verrà indetta il giorno stesso o il seguente.

3.2 Esterne

Con riunioni esterne si intende qualsiasi incontro fra *Proponente* / *Committente* e un gruppo di rappresentanza (composto da almeno la maggioranza assoluta) del gruppo di progetto.

3.2.1 Richiesta

La richiesta di indire una riunione esterna può essere avanzata da qualsiasi componente del gruppo; è compito del Responsabile contattare ed organizzare l'evento con l'interessato. Una volta pianificato l'incontro il Responsabile deve comunicare con il resto del gruppo secondo i passi indicati alla sezione 3.1 con tag `[Riunione Esterna]`.

3.2.2 Esito

Ad ogni incontro il Responsabile ha il compito di stilare un verbale (cap. 9.6.1) che evidenzia i chiarimenti emersi durante l'incontro.

4 Analisi

4.1 Attività

Lo scopo di questa attività è di analizzare tutte le informazioni utili all'analisi del prodotto per produrre requisiti semplici e di facile comprensione.
I requisiti emersi dall'analisi dovranno ricoprire i seguenti aspetti:

- specifiche funzionali e prestazionali, incluse le caratteristiche dell'ambiente nel quale il software deve operare;
- ambito di utilizzo;
- tipologia di utenti;
- obiettivi di qualità da raggiungere;
- requisiti per la installazione e la accettazione;
- tracciamento dei requisiti.

È inoltre necessario effettuare un'attività di tracciamento dei requisiti nel documento *SpecificaTecnica_v3.0.pdf*.

Le risorse considerate valide per la definizione dei requisiti sono, in ordine decrescente:

- il capitolato d'appalto;
- incontri con il proponente;
- incontri con il committente;
- incontri con persone che potrebbero utilizzare il prodotto (persone impegnate nell'implementazione e sviluppo di sistemi analoghi ed utenti finali che utilizzeranno il prodotto).

Terminata l'attività di analisi dei requisiti si deve produrre un documento completo ed esplicativo: l'Analisi dei Requisiti.
Successivamente, nel documento di Piano di Qualifica (*PianoDiQualifica_v3.0.pdf*) si definiscono una serie di test da eseguire per provare che il sistemi rispecchi i requisiti evidenziati.

Input: capitolato d'appalto, informazioni reperite dai vari stakeholder;

Output: Analisi dei Requisiti, test di sistema;

Risorse: Analisti, strumentazione;

Misurazioni: avanzamento dell'elaborazione del documento Analisi dei Requisiti rispetto alla totalità delle informazioni in ingresso;

Norme: descritte in seguito.

4.2 Requisiti

4.2.1 Provenienza

Per ogni requisito individuato dev'essere specificata la fonte di provenienza. È possibile utilizzare i seguenti acronimi o abbreviazioni:

CA: capitolato;

IP: incontro proponente;

IF: interno fornitore.

4.2.2 Classificazione

Ogni requisito deve rispettare la seguente forma

$$\{Tipologia\}\{Importanza\}\{X\}$$

Tipologia utente:

{F}: indica un requisito funzionale;

{Q}: indica un requisito di qualità;

{T}: indica un requisito tecnologico;

{V}: indica un requisito di vincolo.

Tipologia amministratore:

{FA}: indica un requisito funzionale.

Importanza:

{OB}: indica un requisito obbligatorio;

{DE}: indica un requisito desiderabile;

{OP}: indica un requisito opzionale.

La X indica il numero di requisito.
Se un requisito è un sotto-requisito di un altro, dev'essere denominato nel seguente modo

$$\{Tipologia\}\{Importanza\}\{X\}.\{Y\}$$

dove X è il numero del requisito principale, mentre Y quello del sotto - requisito. Si consiglia nell'Analisi dei Requisiti la creazione di una tabella di tracciamento dei requisiti. Si lascia agli Analisti la definizione della struttura della tabella.

4.2.3 Diagrammi UML: formalismo grafico

La rappresentazione dei casi d'uso deve seguire il formalismo del linguaggio UML$_{|g|}$ v2.x in modo da creare diagrammi che facilitano la comprensione dei requisiti.
Dev'essere adottata la seguente nomenclatura:

$$\{Tipo\}\ \{A\}...\{Z\}$$

dove *Tipo* indica una tipologia dei seguenti diagrammi:

UC: diagramma dei casi d'uso;

UCA: diagramma dei casi d'uso per amministratore;

e $\{A\}...\{Z\}$ indica una struttura gerarchica di insiemi. (*es.* `A.B.C`: `A` indica il diagramma principale, `B` una specifica/estensione di `A`, `C` una specifica/estensione di `B` e cosi via).

4.2.4 Diagrammi UML: descrizione testuale

Ciascun diagramma dei casi d'uso dev'essere seguito da una descrizione testuale nella quale devono essere esplicitati i seguenti punti:

- Attori principali;
- Descrizione del requisito;
- Precondizione;
- Postcondizione;
- Scenario principale dello svolgersi degli eventi;
- Scenario alternativo.

4.3 Strumenti

Per l'esecuzione di questa attività si utilizzano i seguenti strumenti:

LaTeX: per la stesura del documento Analisi dei Requisiti (dettaglio cap. 13.2.1);

Visual Paradigm: per la stesura dei diagrammi UML$_{|g|}$ (dettaglio cap. 13.2.3);

git: per il versionamento della documentazione (dettaglio cap. 13.3.1);

5 Progettazione architetturale

5.1 Attività

Lo scopo di questa attività è quello di realizzare una visione globale di ciò che dovrà essere il sistema a fronte dei requisiti ricavati dall'attività di analisi.
Terminata l'attività di progettazione architetturale si deve produrre un documento completo ed esplicativo: la Specifica Tecnica.
Le attività necessarie alla redazione del documento sono:

- definizione dell'architettura di prodotto a partire dall'Analisi dei Requisiti (vedi *AnalisiDeiRequisiti_v4.0.pdf*);
- individuazione e studio dei design pattern applicabili;
- individuazione della struttura dei package;
- individuazione delle classi che compongono il sistema e delle relazioni tra esse;
- analisi delle tecnologie da adottare;
- studio di fattibilità;
- tracciamento componenti-requisiti (vedi 5.6).

Si deve inoltre definire, in un documento specifico (*PianoDiQualifica_v3.0.pdf*), vari test da eseguire sulle parti del sistema per verificarne la corretta interazione.

Input: Analisi dei Requisiti;

Output: Specifica Tecnica, test di integrazione;

Risorse: Progettisti, documentazione, strumentazione;

Misurazioni: avanzamento dell'elaborazione del documento Specifica Tecnica rispetto alla totalità dei requisiti definiti nel documento di Analisi dei Requisiti;

Norme: descritte in seguito.

5.2 Diagrammi UML

5.2.1 Tipologia

Si deve utilizzare il linguaggio UML$_{|g|}$ v2.x per definire:

Diagrammi dei package$_{|g|}$: per comprendere la struttura generale del sistema. È fondamentale che ciascun package$_{|g|}$ sia definito in modo chiaro ed univoco in modo da facilitare, in fase di codifica, la comprensione delle interazioni tra i vari package$_{|g|}$.

Diagrammi delle classi: per comprendere la complessità del sistema in relazione al numero delle classi.

Diagrammi delle attività: per definire in modo non ambiguo i flussi delle varie attività, composte dalle rispettive componenti, ritenuti non banali.

Diagrammi di sequenza: per definire in modo preciso le responsabilità dei vari package$_{|g|}$ o classi nel portare a termine un determinato compito. Si deve limitare la rappresentazione di cicli o condizioni alternative ai soli casi in cui la loro omissione comporti un grave scompenso alla comprensione del diagramma.

5.2.2 Classificazione

Ciascuna tipologia di diagramma UML$_{|g|}$ dev'essere identificata mediante la seguente nomenclatura:

$$\{Tipo\}\ \{A\}...\{Z\}$$

dove *Tipo* indica una tipologia dei seguenti diagrammi:

DP: diagramma dei package$_{|g|}$;

DC: diagramma delle classi;

DA: diagramma di attività;

DS: digramma di sequenza.

e $\{A\}...\{Z\}$ indica una struttura gerarchica di insiemi. (*es.* `A.B.C`: `A` indica il diagramma principale, `B` una specifica/estensione di `A`, `C` una specifica/estensione di `B` e cosi via).

5.3 Design pattern

Per descrivere i vari design pattern adottati si dovrà utilizzare la seguente specifica:

- **Descrizione generale:** breve descrizione della struttura del design pattern;
- **Motivazione:** descrivere il motivo della scelta di tale design pattern;
- **Contesto applicativo:** elencare i contesti dove il design pattern è stato applicato.

5.4 Immagini GUI

Per rendere più nitida l'idea di come l'utente finale deve interagire con il sistema, è necessario fornire un'anteprima dell'interfaccia grafica del prodotto per ogni attività di utilizzo.
Ogni immagine, rappresentante una GUI$_{|g|}$ del prodotto, deve essere etichettata secondo la seguente convenzione

$$GUI\ \{A\}...\{Z\}$$

dove $\{A\}...\{Z\}$ indica una struttura gerarchica di insiemi. (*es.* `A.B.C`: `A` indica la figura principale, `B` una specifica/estensione di `A`, `C` una specifica/estensione di `B` e cosi via).

5.5 Norme di progettazione

Per rendere più semplice la comprensione degli schemi e la futura fase di verifica, è necessario prestare attenzione a:

- **Annidamento di chiamate:** non ci devono essere, all'interno dell'applicazione, chiamate annidate con una profondità massima maggiore di cinque.
- **Concorrenza:** nel caso in cui si renda necessario l'utilizzo di flussi concorrenti è necessario fornire il diagramma di flusso e la stima delle risorse necessarie all'implementazione. Nel caso in cui i benefici ottenuti dalla concorrenza non risultino essere equivalenti o inferiori alle risorse, questa dev'essere eliminata.
- **Flussi condizionali:** nei casi in cui vengano utilizzati costruttori condizionali (*es.* **if - else**) il loro annidamento non dovrà essere maggiore di cinque. Questo garantisce una maggiore chiarezza, migliore comprensione futura del codice ed una attività di verifica più semplice.
- **Numero di parametri:** non dovranno essere creati metodi aventi più di cinque parametri.
- **Ricorsione:** si deve evitare il più possibile l'utilizzo della ricorsione. Nel caso in cui l'utilizzo della ricorsione sia necessario, si deve fornire una opportuna dimostrazione di terminazione ed una stima dell'occupazione che questa provoca in memoria; qualora queste stime risultino essere eccessive, si deve procedere con l'eliminazione della ricorsione.

Nel documento Piano di Qualifica (*PianoDiQualifica_v3.0.pdf*) devono essere trattate ed approfondite le metriche e la loro misurazione.

5.6 Tracciamento

Nel documento deve essere riportato, in forma tabellare, il tracciamento di ogni requisito con la componente che ne assicura il soddisfacimento.
È richiesto di riportare, anche questo in forma tabellare, il tracciamento di ciascuna componente con il requisito che ne giustifica l'esistenza.

5.7 Organizzazione dei contenuti

Il documento Specifica Tecnica deve contenere al suo interno:

- analisi dei vari design pattern utilizzati nel modo descritto in 5.3;
- definizione della struttura generale del sistema mediante diagrammi di package$_{|g|}$ corredati da una sintetica spiegazione;
- definizione generale delle sottoparti architetturali:
  1. Diagramma delle classi;
  2. Descrizione del diagramma delle classi composta dai seguenti punti:
     - **Nome:** il nome della classe;

- **Tipo:** attributo che indica se si tratta di un'interfaccia o di una classe;
- **Package:** indica il package$_{|g|}$ a cui appartiene l'interfaccia o la classe;
- **Descrizione:** una descrizione sintetica dell'interfaccia o della classe e delle sue funzionalità;
- **Relazioni con altre componenti:** indica le relazioni che la classe ha con altre interfacce o classi presenti all'interno della rappresentazione grafica della componente architetturale.

3. Diagramma di attività, corredato da una sintetica descrizione, per le sottoparti ritenute non banali che portino informazioni aggiuntive;
4. Diagramma di sequenza;
5. Descrizione del diagramma di sequenza composta dai seguenti punti:
   - Precondizione;
   - Descrizione sintetica;
   - Postcondizione.

5.8 Strumenti

Per l'esecuzione di questa attività si utilizzano i seguenti strumenti:

LaTeX: per la stesura del documento Specifica Tecnica (dettaglio cap. 13.2.1);

Visual Paradigm: per la stesura dei diagrammi UML$_{|g|}$ (dettaglio cap. 13.2.3);

git: per il versionamento della documentazione (dettaglio cap. 13.3.1);

6 Progettazione di dettaglio

6.1 Attività

Lo scopo di questa attività è di realizzare una visione dettagliata di quello che dovrà essere il sistema estendendo in modo consono ed appropriato l'architettura specificata nel documento Specifica Tecnica.
Terminata l'attività di organizzazione in dettaglio dell'intero sistema si deve produrre un documento completo ed esplicativo: la Definizione di Prodotto.
Le attività necessarie alla redazione del documento sono:

- descrizione dettagliata delle singole unità (interfacce, classi, metodi e campi dati) che compongono il programma;
- rappresentazione grafica, mediante diagrammi di sequenza, del comportamento del programma;
- tracciamento componenti-requisiti.

Si deve inoltre definire, in un documento specifico (*PianoDiQualifica_v3.0.pdf*), vari test da eseguire sulle classi in modo da provare il loro corretto comportamento.

Input: Specifica Tecnica;

Output: Definizione di Prodotto, test di unità;

Risorse: Progettisti, strumentazione;

Misurazioni: avanzamento dell'elaborazione del documento Definizione di Prodotto rispetto alla totalità della Specifica Tecnica;

Norme: descritte in seguito.

6.2 Diagrammi UML

6.2.1 Tipologia

Si deve utilizzare il linguaggio UML$_{|g|}$ v2.x per definire:

Diagrammi delle classi: per rendere chiare le relazioni tra le classi all'interno delle singole componenti.

Diagrammi delle attività: per definire in modo non ambiguo i flussi delle varie attività, composte dalle rispettive componenti, ritenuti non banali e non descritti nel documento di Specifica Tecnica.

Diagrammi di sequenza: per definire in modo preciso le responsabilità dei vari package$_{|g|}$ o classi nel portare a termine un determinato compito, nel caso in cui ciò non fosse stato specificato adeguatamente nel documento Specifica Tecnica. Si deve limitare la rappresentazione di cicli o condizioni alternative ai soli casi in cui la loro omissione comporti un grave scompenso alla comprensione del diagramma.

6.2.2 Classificazione

Ciascuna tipologia di diagramma UML$_{|g|}$ dev'essere identificata mediante la seguente nomenclatura:

$$\{Tipo\}\ \{A\}...\{Z\}$$

dove *Tipo* indica una tipologia dei seguenti diagrammi:

DC: diagramma delle classi;

DA: diagramma di attività;

DS: digramma di sequenza.

e $\{A\}...\{Z\}$ indica una struttura gerarchica di insiemi. (*es.* `A.B.C`: `A` indica il diagramma principale, `B` una specifica/estensione di `A`, `C` una specifica/estensione di `B` e cosi via).

6.3 Specifica delle classi

Nel documento è importante specificare ciascuna classe mediante il seguente schema:

- **Funzione:** specifica la funzione svolta dalla classe;
- **Relazione con altre componenti:** specifica la tipologia di relazione della classe con le altre componenti;
- **Attributi:** definiti all'interno della classe corredati da una breve descrizione;
- **Metodi:** definiti nella classe corredati da una descrizione che ne descriva, nel caso si presentasse, eccezioni lanciate o gestite oppure tipi di oggetti di ritorno.

6.4 Tracciamento

Nel documento deve essere riportato, in forma tabellare, il tracciamento di ogni requisito con le varie classi che ne assicurano il soddisfacimento. È richiesto di riportare, anche questo in forma tabellare, il tracciamento di ciascuna classe con il requisito che ne giustifica l'esistenza.

6.5 Organizzazione dei contenuti

Il documento Definizione di Prodotto deve contenere al suo interno:

- diagrammi delle classi per ogni suo componente;
- specifiche e descrizioni per ogni classe specificata.

6.6 Strumenti

Per l'esecuzione di questa attività si utilizzano i seguenti strumenti:

LaTeX: per la stesura del documento Specifica Tecnica (dettaglio cap. 13.2.1);

Visual Paradigm: per la stesura dei diagrammi UML$_{|g|}$ (dettaglio cap. 13.2.3);

git: per il versionamento della documentazione (dettaglio cap. 13.3.1);

7 Codifica

7.1 Attività

Lo scopo di questa attività è quello di produrre il codice che soddisfi, il più fedelmente possibile, le specifiche definite nel documento Definizione di Prodotto.
L'attività termina con la presentazione del codice sorgente del sistema.

Input: Definizione di Prodotto;

Output: codice sorgente;

Risorse: Programmatori, strumentazione;

Misurazioni: avanzamento dell'elaborazione del codice sorgente rispetto alla totalità della Definizione di Prodotto;

Norme: descritte in seguito.

7.2 Intestazione file

Ogni file di codice sorgente deve iniziare con un'intestazione che deve rispettare il seguente schema:

```
/**
 * Name: {Nome del file}
 * Package: {Package di appartenenza}
 * Author: {Autore del file}
 * Date: {Data di approvazione del file}
 *
 * Changes:
 * Version    Date        Changes       Reason
 * {X}.{Y}    AAAA-MM-GG  {Cambiamento} {Motivazione}
 *
 *
 * ------------------------------------------------------------
 * Copyright (C) 2013 GoGoTeam
 *
 * This file is part of MyTalk.
 *
 * MyTalk is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License
 * as published by the Free Software Foundation, version 2 of the
 * License.
 *
 * MyTalk is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with MyTalk.  If not, see
 * <http://www.gnu.org/licenses/>.
 * ------------------------------------------------------------
 *
 */
```

- **Name**: indica il nome del file comprensivo di estensione;
- **Package**$_{|g|}$: deve essere comprensivo della gerarchia del package$_{|g|}$;
- **Author**: indica l'autore del file e non necessariamente il programmatore che sta modificando il file attuale;
- **Date**: indica la data di creazione del file (come indicato nel cap. 9.3);
- **Changes**: indica la lista di avanzamento delle modifiche del file apportate. In dettaglio:
  - **Version**: ultima modifica apportata al file (come indicato nel cap. 12.2.2).
  - **Date**: rappresenta la data dell'avvenuta modifica (come indicato nel cap. 9.3).
  - **Changes**: rappresenta la lista dei cambiamenti effettuati. Questa lista deve fornire le informazioni inerenti i metodi modificati preceduti da una delle seguenti etichette:
    - `[+]` indica che il metodo è stato creato;
    - `[-]` indica che il metodo è stato eliminato;
    - `[x]` indica che il metodo ha subito un cambiamento.
  - **Reason**: motivazione di modifica.

  L'inserimento dell'ultima versione è aggiunto a capo della lista in modo da avere uno storico a ritroso delle modifiche fatte al file.

7.3 Convenzione codifica codice

I programmatori devono seguire la Java Code Convention (`http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`) in modo tale da rendere il codice più leggibile e mantenibile. Tale convenzione può essere integrata o modificata solo su autorizzazione del Responsabile di Progetto.

7.3.1 Struttura interna delle classi

All'interno di una classe dovranno comparire nel seguente ordine:

- Variabili statiche
- Variabili di istanza
- Costruttori
- Metodi

7.3.2 Struttura dei codici

- **Struttura linea di codice:** La lunghezza della linea di codice deve facilitare la lettura e fornire una buona presentazione visiva. Per questo scopo si è scelto l'utilizzo di spazi prima e dopo la maggior parte degli operatori, cercando di non alterare la funzione del codice, ed evitare per ciascuna riga l'inserimento di più istruzioni.

- **Rientri e uso delle parentesi graffe** La parentesi graffa di apertura di un blocco di codice dovrà essere posta sulla stessa riga della dichiarazione del blocco di codice interessato, mentre la parentesi graffa di chiusura va posizionata in una nuova riga, allineata verticalmente rispetto al primo carattere della dichiarazione del blocco di codice interessato.
  Ogni livello di indentazione deve rientrare utilizzando il tasto di tabulazione. Di seguito si riporta un esempio d'uso:

```
1        public void stampa(String) {
2            .........
3        }
```

- **Blocchi funzionali:** Il codice viene diviso in paragrafi utilizzando le righe vuote per separarli. Questa convenzione è stata pensata per favorire la leggibilità e semplificare la comprensione della logica del codice stesso.

- **Commenti:** I commenti vanno sempre scritti in una nuova riga lasciando una riga di spazio prima di essi, anche se inseriti all'interno di un blocco di codice. Il carattere che indica l'inizio del commento deve essere // (doppia barra) oppure nella forma /* */ se il commento necessita di più righe .

7.3.3 Convenzione sui nomi

- **Classi:** l'identificatore deve avere la prima lettera maiuscola e tutte le altre minuscole. Se è composto da più parole, la prima lettera di ogni parola deve essere in maiuscolo e non ci devono essere dei caratteri speciali.

- **Interfacce:** l'identificatore sarà composto dalla lettera *i* maiuscola seguita dal nome della classe che implementerà tale interfaccia. Esempio: classe *Pippo*, interfaccia *IPippo*.

- **Metodi:** l'identificatore dovrebbe dare un'immediata idea del suo comportamento contenendo almeno un verbo scritto in minuscolo. Se è composto, la seconda parola andrà scritta solo con la prima lettera in maiuscolo. Esempio *pagareTassa*.

- **Variabili:** scritte con nomi brevi ed evocativi con la prima lettera in minuscola. Nel caso siano composte da più parole, l'iniziale di ogni parola dovrebbe essere maiuscola.

- **Costanti:** dovrebbero essere scritte in maiuscolo. Nel caso siano composte da più parole si dovrebbe utilizzare il simbolo di underscore.

7.4 Strumenti

Per l'esecuzione di questa attività si utilizzano i seguenti strumenti:

Eclipse: per la realizzazione del codice (dettaglio cap. 13.3.2);

GWT: per la codifica dei linguaggi web (dettaglio cap. 13.3.3);

git: per il versionamento della documentazione (dettaglio cap. 13.3.1);

8 Verifica e validazione

8.1 Attività

L'obiettivo principale di questa attività è quello di dimostrare che il prodotto risultante dall'attività di codifica funzioni correttamente (verifica) e che sia il prodotto atteso (validazione). Al termine dell'attività ci si aspetta di avere dei dati che confermano la correttezza del prodotto.

Input: codice sorgente, stime dei test;

Output: esiti dei test;

Risorse: verificatori, strumentazione;

Misurazioni: stato di avanzamento rispetto alle stime dei test;

Norme: descritte in seguito.

L'esito delle attività di verifica e validazione dovrà produrre i risultati che stimano:

- **Copertura dei requisiti:** non inferiore ai requisiti definiti come obbligatori;
- **Copertura del codice:** non inferiore al 60%;
- **Codice morto:** non presente.

Tali dati dovranno essere creati grazie all'ausilio di Metrics (cap. 14.1.3) e EclEmma (cap. 14.2.1).

8.2 Test di unità e integrazione

La verifica dovrà essere realizzata basandosi sulle definizioni dei test di unità e integrazione forniti dalle attività di progettazione architetturale e di progettazione di dettaglio. Per le varie strategie di verifica si dovrà fare riferimento al Piano di Qualifica (*PianoDiQualifica_v3.0.pdf*).
Nel documento riassuntivo dell'esito dei test si dovrà presentare per ogni tipologia di test effettuato:

- specifica dell'ambiente di testing;
- classi testate o componenti testate dal particolare test;
- descrizione del comportamento del test;
- errori individuati a seguito dell'esecuzione dei test.

Si dovrà riportare l'esito dei test in formato tabellare.
La codifica dei test dovrà essere effettuata su *Eclipse* con l'ausilio del framework *JUnit*(cap. 14.1.5).

8.3 Test di sistema

La validazione dovrà essere realizzata basandosi sulle definizioni dei test di sistema forniti dall'attività di analisi dei requisiti. Per le varie strategie di validazione si dovrà fare riferimento al Piano di Qualifica (*PianoDiQualifica_v3.0.pdf*).
Si dovranno riportare in formato tabellare gli esiti dei test effettuati in corrispondenza ai requisiti testati.

9 Documenti

9.1 Struttura documentazione

Tutti i documenti devono essere realizzati utilizzando il template LaTeX presente nel repository$_{|g|}$all'indirizzo

https://sourceforge.net/p/ggt-mytalk/code/docs/template/

L'utilizzo del template deve limitarsi all'invocazione delle funzionalità che consentono la formattazione dei documenti: ciò permette di effettuare modifiche stilistiche localizzate al codice del template per farle ricadere in maniera automatica in tutti i documenti, evitando quindi la modifica a mano di ogni documento.
Nello specifico, il template regola:

- pacchetti LaTeX usati;
- comandi personalizzati;
- riferimenti testuali interni;
- formattazione copertina documento;
- formattazione pagine interne;
- stile tabelle;
- casi particolari di sillabazione di alcune parole;
- stili di tipografia.

Le modifiche al template possono essere avanzate, tramite i metodi di comunicazione interna (cap. 2.1), da tutti i membri del gruppo al Responsabile di Progetto il quale avrà compito di valutare e, nel caso affermativo, applicare le modifiche richieste.

9.2 Norme tipografiche

Al fine di evitare incongruenze tra i vari documenti, si specificano in questa sezione le informazioni riguardanti l'ortografia, la tipografia e l'assunzione di uno stile uniforme in tutti i documenti.

9.2.1 Stile del testo

- **Grassetto:** viene utilizzato per evidenziare dei passaggi estremamente importanti all'interno del testo e per evidenziare l'elemento trattato negli elenchi.
- **Corsivo:** deve essere utilizzato per porre particolare enfasi a termini significativi e per riportare citazioni da fonti esterne.
- **Sottolineato:** deve essere utilizzato per specificare termini composti da più parole, seguiti anche dall'identificativo $_{|g|}$, la cui definizione compare nel Glossario.
- **Maiuscolo:** deve essere limitato all'indicazione di acronimi e nei casi specificati nei Formati di Riferimento (cap. 9.3);
- **Monospace:** deve essere utilizzato per indicare una normativa adottata, riferimenti (cap. 9.3) e per inserire porzioni di codice all'interno dei documenti.

9.2.2 Punteggiatura

- **Punteggiatura:** qualsiasi segno di punteggiatura non deve seguire un carattere di spaziatura, ma deve essere seguito da un carattere di spazio.
- **Lettera maiuscola:** deve seguire esclusivamente un punto, un punto esclamativo o un punto interrogativo.
- **Parentesi:** una qualsiasi frase racchiusa fra parentesi non deve iniziare con un carattere di spaziatura e non deve chiudersi con un carattere di punteggiatura e/o di spaziatura.

9.2.3 Composizione testo

- **Elenchi:**
  - se la lista contiene termini identificativi, questi devono iniziare con la maiuscola e non terminare con un carattere di punteggiatura;
  - se i termini sono frasi considerate parte integrante di quella che introduce la lista allora devono iniziare con la minuscola e terminare con il punto e virgola, tranne l'ultimo termine che deve finire con il punto;
  - se i termini sono complessi e costituiti da frasi distinte rispetto al periodo introduttivo, si usa la maiuscola e il punto alla fine di ogni termine.
- **Note a piè di pagina:** ne è vietato l'uso per non interrompere il flusso di lettura del documento.
- **Glossario:** le parole accompagnate da $_{|g|}$ sono esclusivamente quelle che presentano una corrispondenza nel Glossario.

9.3 Formati di riferimento

- **Riferimenti:**
  - **Percorsi:** per gli indirizzi web$_{|g|}$ completi e indirizzi e-mail deve essere utilizzato il comando appositamente fornito da LaTeX

    `\url{Percorso}`

    nel formato tipografico monospace.
  - **Link a file PDF$_{|g|}$:** devono essere contrassegnati in corsivo mediante l'uso del comando personalizzato:

    `$\nomePdf$`
  - **Ancore:** i riferimenti alle sezioni interne del medesimo documento devono essere scritte nel formato

    (cap. `\ref{label di riferimento della sezione}`)
- **Data:** se non specificato diversamente, deve essere espressa seguendo il formalismo dello standard `ISO`$_{|g|}$ `8601:2004` nel formato:

`AAAA-MM-GG`

dove:

AAAA: rappresenta l'anno in quattro cifre;

MM: rappresenta il mese in due cifre;

GG: rappresenta il giorno in due cifre.

- **Abbreviazioni:** sono ammesse le sigle per i nomi dei documenti

  AR: Analisi dei Requisiti

  GL: Glossario

  NP: Norme di Progetto

  PQ: Piano di Qualifica

  PP: Piano di Progetto

  SF: Studio di Fattibilità

  e per i nomi delle revisioni

  RR: Revisione dei Requisiti

  RP: Revisione di Progettazione

  RQ: Revisione di Qualifica

  RA: Revisione di Accettazione

- **Nomi ricorrenti:**
  - **Ruoli di progetto:** devono essere formattati utilizzando la prima lettera maiuscola di ogni parola che non sia una preposizione (*es.* Responsabile di Progetto).
  - **Nomi dei documenti:** devono essere formattati utilizzando la prima lettera maiuscola di ogni parola che non sia una preposizione (*es.* Norme di Progetto).
  - **Nomi dei file:** il riferimento deve essere comprensivo dell'estensione del file ed essere formattato con carattere monospace.
  - **Nomi propri:** l'utilizzo dei nomi propri deve seguire il formalismo:

    Cognome Nome per elenchi;
    Nome Cognome per tutti gli altri riferimenti.
  - **Nome del gruppo:** deve essere identificato nel seguente stile:

    GoGo Team
  - **Nome del proponente:** deve essere identificato nel seguente stile:

    Proponente o *Zucchetti S.p.A.*
  - **Nome del committente:** deve essere identificato nel seguente stile:

    Committente o *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*

– **Nome del progetto:** deve essere identificato nel seguente stile:

MyTalk

- **Informazioni di pagina:** devono essere espresse nel formato

`{n} di {totale pagine}`

9.4 Immagini

Tutte le immagini devono essere in formato $JPG_{|g|}$, $PNG_{|g|}$ o $PDF_{|g|}$. Ogni figura inserita deve avere una breve didascalia composta da un identificativo numerico univoco seguito da, ove sia ritenuto necessario, una breve descrizione.

9.5 Tabelle

Ogni tabella deve essere stilata in formato LaTeX. Ciascuna tabella deve avere una didascalia composta da un identificativo numerico univoco seguito da, ove sia ritenuto necessario, una breve descrizione.

9.6 Struttura tipi documenti

9.6.1 Verbali incontri

Per *Verbali degli incontri* si intendono quei documenti redatti dal Responsabile di Progetto in occasione di incontri esterni ed interni. Per tali documenti è prevista una sola stesura in quanto promemoria dell'incontro avvenuto. Non è perciò previsto il versionamento. I Verbali degli incontri devono essere denominati secondo il seguente criterio:

`Verbale{tipo incontro}_{data incontro}`

dove:

- **tipo incontro:** indica il tipo di incontro effettuato e deve essere identificato con I (interno) o E (esterno);
- **data incontro:** indica la data in cui è stato tenuto l'incontro seguendo il formato data indicato in cap. 9.3.

La prima pagina di ogni verbale deve obbligatoriamente contenere i seguenti campi, nell'ordine indicato:

- **data**;
- **luogo:** espresso nel formato

{città}({sigla_provincia}) {sede}

- **ora di ritrovo:** espressa nel formato

Ora dell'incontro: {hh}:{mm}

in cui il campo `hh` indicante le ore e il campo `mm` indicante i minuti vanno espressi nel formato 24h specificato nella normativa `ISO`$_{|g|}$ `8601:2004`;

- **durata dell'incontro:** espressa nel formato

  Durata dell'incontro: {x} min

  dove il valore `x` sta ad indicare l'effettiva durata dell'incontro;

- **partecipanti interni:** lista degli appartenenti al gruppo *GoGo Team* presenti all'incontro

  Partecipanti del gruppo: {nome} {cognome}

- **partecipanti esterni:** rappresentanti della ditta/e

  Partecipanti esterni: {nome} {cognome} {ruolo}

  dove il campo `ruolo` rappresenta il ruolo assunto all'interno dell'azienda a cui fanno capo; nel caso il partecipante sia il *Committente*, il campo viene compilato con `Committente`.

- **contenuto:** la decisione del formato è lasciata al Responsabile di Progetto, il quale adotta lo stile più consono in base al tipo di incontro svolto;

- **firme:** devono essere comprese quelle di tutti i partecipanti del gruppo *GoGo Team* a conferma della presa visione del documento.

9.6.2 Lettera di presentazione

La lettera di presentazione deve contenere:

- Logo del gruppo
- Intestazione nel seguente formato:

  Prof. Tullio Vardanega
  Università degli Studi di Padova
  Via Trieste 63
  35121 Padova (PD)

- Breve introduzione (facoltativa)
- Elenco di tutti i documenti in consegna
- Varie ed eventuali, osservazioni (facoltative)
- Firma del responsabile nel seguente formato

  {*Nome*} {*Cognome*}
  il Responsabile del gruppo GoGo Team
  {*Firma del responsabile*}

9.7 Contenuto dei documenti

Ogni documento ufficiale deve essere composto dalle seguenti sezioni:

- Pagina iniziale con titolo, logo ed informazioni del documento
- Sommario
- Registro delle modifiche
- Indice del documento
- Indice delle tabelle (se presenti)
- Indice delle figure (se presenti)
- Introduzione

Le pagine contengono le seguenti informazioni di base:

Intestazione

- Logo
- Nome documento

Piè di pagina

- Versione
- Ateneo - Anno Accademico
- Contatto mail ufficiale di gruppo
- Licenza
- Pagina

10 Glossario

Il Glossario deve essere organizzato come indicato nella sezione Documenti (cap. 9); esso è unico e riassuntivo per tutti i documenti. Il Glossario riporta, in ordine lessicografico, le definizioni delle parole che possono generare confusione o ambiguità all'interno dei documenti, e che in essi appaiono contraddistinte dal simbolo indicato (cap. 9.2).

10.1 Procedura per inserimento vocaboli

Ogni membro del gruppo può apportare modifiche all'interno del file *Glossario_v1.0.pdf*, Per l'inserimento dei vocaboli si fa riferimento alle seguenti specifiche:

- Prima dell'inserimento è necessario che l'utente sia sicuro che il termine da inserire non sia già presente;
- L'inserimento va effettuato nella posizione esatta rispettando lo stile di formattazione definito.

11 Metodi di condivisione

Per un migliore e più efficiente svolgimento delle attività del gruppo, si è resa necessaria l'adozione di due differenti strumenti per la condivisione delle risorse.

11.1 Dropbox - informativi interni

Il servizio Dropbox (https://www.dropbox.com/) consente una facile condivisione di risorse tra i componenti del gruppo anche quando questi si trovano fuori sede.
Le risorse condivise tramite questo strumento hanno natura puramente informativa e non sono soggette a versionamento.
Il riferimento remoto, che si indica come `root` in seguito, è il seguente

https://www.dropbox.com/home/SWE-GoGoTeam

La suddivisione interna della cartella remota è la seguente:

- `root/documenti`: per i documenti interni informali di riferimento;
- `root/guide`: per tutte le guide tecniche che interesseranno lo sviluppo del prodotto;
- `root/manuali`: per i manuali di sviluppo ed implementazione di utility$_{|g|}$ e linguaggi.

L'accesso alla cartella condivisa viene inoltrata dal Responsabile che provvede, entro un giorno lavorativo, ad abilitare l'accesso all'utente.

11.2 Repository

Per un efficiente svolgimento del lavoro interno del gruppo è necessario uno strumento affidabile e preciso per il supporto allo sviluppo della documentazione e di tutti i file di codifica del progetto.
Il servizio scelto per la gestione del repository$_{|g|}$ è SourceForge (https://sourceforge.net).

11.2.1 Struttura

- **Progetto:** si definisce come `root` l'indirizzo web$_{|g|}$ di riferimento del progetto:

  https://sourceforge.net/p/ggt-mytalk/code/

- **Documentazione:** tutta la documentazione viene allocata nel repository$_{|g|}$ nella cartella:

  `root/docs/`

  All'interno di questa sono allocati i vari file necessari alla documentazione:

  - **Documenti in redazione:** i documenti non formali in fase di stesura devono essere caricati in una propria cartella:

`root/docs/{NomeDocumento}/`

- **Immagini:** tutte le immagini ad uso esclusivo del documento sono locate nella seguente posizione:

  `root/docs/{NomeDocumento}/Immagini`

- **Documenti di rilascio:** tutti i documenti formali identificati in versione di rilascio devono essere caricati all'interno di un'unica cartella:

  `root/docs/{TipoRevisione}/`

- **Codice:** tutti i file di codifica sono allocati in

  `root/code/`

  All'interno della cartella vengono smistati i vari file necessari allo sviluppo dell'applicazione eseguibile.
  Si lascia all'IDE$_{|g|}$ Eclipse (vedi cap. 13.3.2) la completa gestione ed organizzazione dei file secondo la struttura definita dai package dell'applicazione.

11.2.2 Utilizzo per i documenti

La documentazione all'interno del repository$_{|g|}$ deve essere composta da soli documenti in formato LaTeX.
Qualora fosse necessario visionare un qualsiasi documento, ogni utente deve procedere autonomamente alla sua compilazione per produrre un file PDF$_{|g|}$. Per ottenere ciò è necessario eseguire il seguente comando:

```
$ pdflatex root/docs/{NomeDocumento}/{NomeDocumento}.tex
```

12 Procedura per lo sviluppo delle applicazioni

12.1 Gestione incarichi

Il servizio SourceForge$_{|g|}$ usato permette l'utilizzo di servizi di tracking$_{|g|}$ e milestone$_{|g|}$. Questi strumenti dovono essere utilizzati nella fase di sviluppo seguendo una rigida normativa.

1. **Creazione milestone$_{|g|}$:** il Responsabile ha il compito di creare una milestone$_{|g|}$ (figura 1) per le successive revisioni a cui il gruppo deciderà di partecipare. Sarà possibile tenere sotto controllo il suo stato di avanzamento analizzando il numero di ticket$_{|g|}$ completati ed i rimanenti ancora aperti.

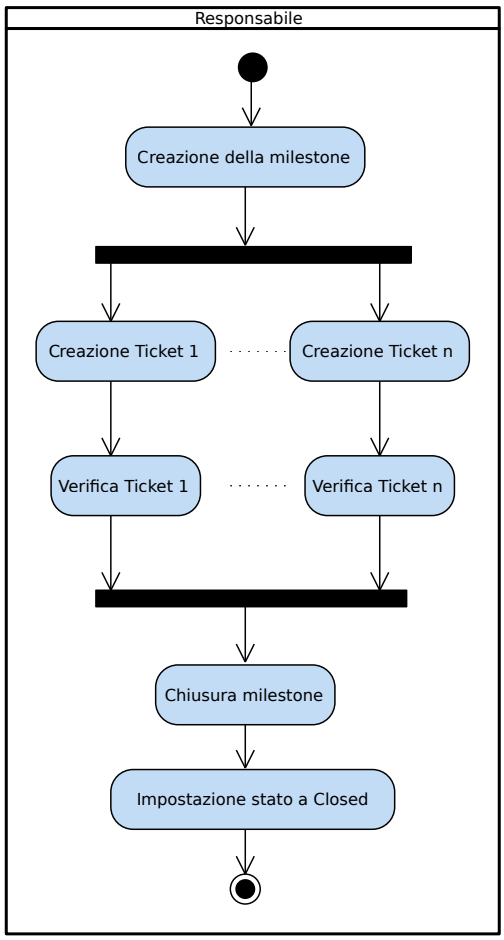

Figura 1: Modello per la creazione di una milestone$_{|g|}$.

2. **Creazione ticket$_{|g|}$:** il Responsabile deve creare una serie di ticket$_{|g|}$ corrispondenti ai compiti da svolgere. Il compito deve essere assegnato al componente del gruppo ritenuto più idoneo alla sua soddisfazione (figura 2).
   La compilazione del ticket$_{|g|}$ deve seguire le seguenti direttive:

   - **Title:** specifica, sinteticamente, l'oggetto del compito;

- **Status:** da impostare come `open`;
- **Owner:** imposta il riferimento al membro del gruppo che avrà incarico di soddisfare il compito indicato;
- **Labels:** indica il campo del compito in oggetto. L'etichetta deve essere composta da un identificativo quale:

  `D:` per i documenti;

  `C:` per il codice applicativo;
- **Private:** identifica il compito come privato, la sua abilitazione viene lasciata a giudizio del Responsabile;
- **Summary:** descrizione del compito da svolgere;

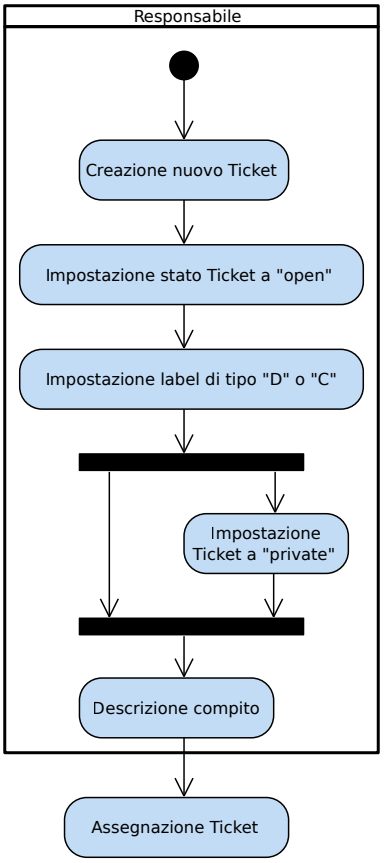

Figura 2: Modello per la creazione di un nuovo ticket$_{|g|}$.

3. **Esecuzione compito:** ciascun membro del gruppo deve visionare i ticket$_{|g|}$ a lui assegnati e modificare il loro stato ad `accepted`.
   Una volta terminato il compito richiesto, l'incaricato deve segnare il ticket$_{|g|}$ corrispondente come `closed` (figura 3).

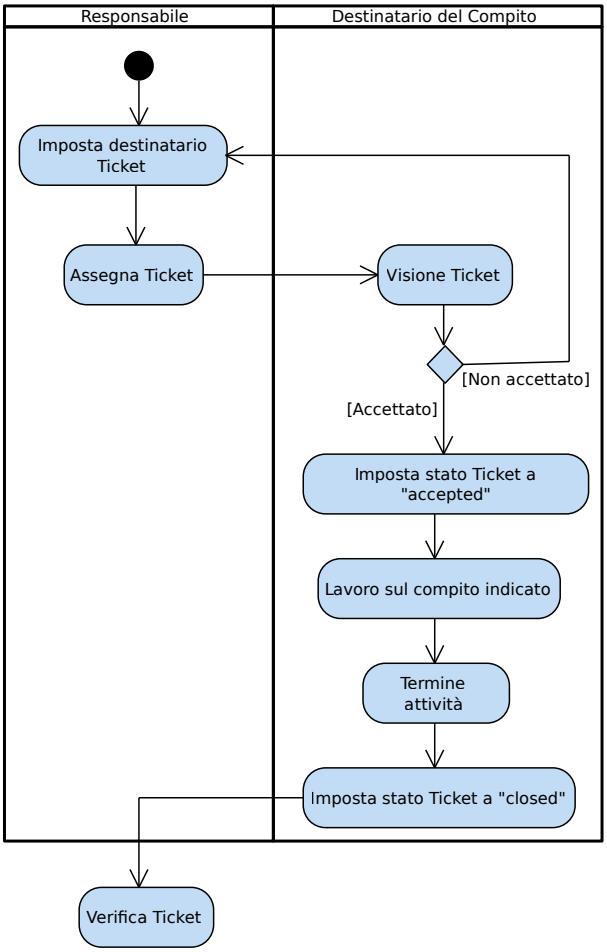

Figura 3: Modello per l'assegnazione di un ticket$_{|g|}$.

4. **Creazione ticket$_{|g|}$ di verifica:** il Responsabile, ad ogni ticket$_{|g|}$ segnalato `closed`, ha il compito di creare un nuovo ticket$_{|g|}$ associato di verifica (figura 4).

   - **Title:** deve essere specificato l'oggetto del compito anteposto dall'identificativo `VERIFICA`:

     `VERIFICA: {Oggetto del compito}`

   - **Status:** da impostare come `open`;
   - **Owner:** imposta il riferimento all'utente Verificatore;
   - **Labels:** si deve aggiungere al label del titcket originario il suffisso `v`;
   - **Private:** identifica il compito come privato, la sua abilitazione viene lasciata a giudizio del Responsabile;
   - **Summary:** descrizione, lasciata a giudizio del Responsabile;

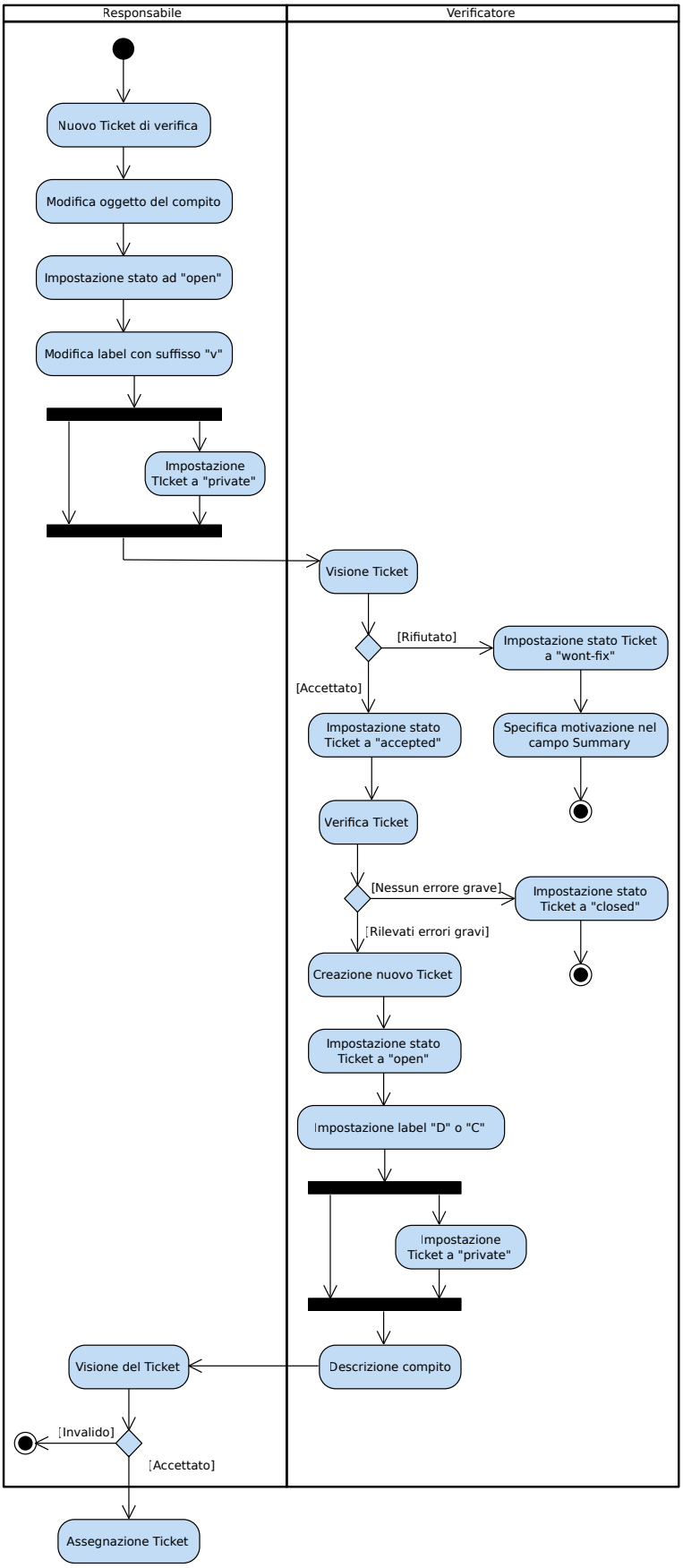

Figura 4: Modello per la creazione e gestione delle verifiche.

5. **Fase di verifica:** il Verificatore può rifiutare o accettare il ticket$_{|g|}$ assegnato (figura 4). Nel primo caso, il Verificatore deve modificare lo stato del ticket$_{|g|}$ a `wont-fix` specificandone il motivo di rifiuto nel campo `Summary`.
Il Verificatore, una volta accettato il compito, deve impostare lo stato del ticket$_{|g|}$ ad `accepted` e procedere con la fase di verifica. Se la verifica ha esito positivo, lo stato del ticket$_{|g|}$ passa a `closed` mentre, nel caso di riscontro di errori gravi, il Verificatore deve creare un nuovo ticket$_{|g|}$:

   - **Title:** specifica, sinteticamente, l'oggetto del compito;
   - **Status:** da impostare come `open`;
   - **Owner:** imposta il riferimento al Responsabile di Progetto;
   - **Label:** label del titcket$_{|g|}$ originario;
   - **Private:** identifica il compito come privato, la sua abilitazione viene lasciata a giudizio del Verificatore;
   - **Summary:** descrizione dell'errore;

   Il Responsabile deve valutare se accettare o rifiutare il ticket$_{|g|}$ di verifica. Nel caso di accettazione, il Responsabile deve modificare il campo `Owner` indicando il riferimento all'utente del gruppo designato per la soddisfazione del compito. Una volta che il ticket$_{|g|}$ viene segnalato come `closed` il Responsabile deve seguire una nuova procedura di verifica (punto 5).

6. **Chiusura milestone$_{|g|}$:** Una volta che tutti i ticket$_{|g|}$ appartenenti alla stessa milestone$_{|g|}$ sono segnati come `closed`, il Responsabile può rendere la milestone$_{|g|}$ `closed` e crearne una nuova ripartendo dal punto 1.

12.2 Norme di versionamento

12.2.1 Documentazione

Al fine di uniformare lo stile di versionamento degli elementi dei documenti interni si è scelta la seguente norma di versionamento:

- **Notazione:** per indicare la versione attuale del documento si usa la seguente notazione:

  `{X}.{Y}`

  dove:

  - `{X}`**:** indica un'evoluzione significativa del documento. Si verifica l'aumento di questo numero quando il documento è in uno stato formale;
  - `{Y}`**:** indica l'apporto di modifiche al documento che possono venire apportate dal Redattore stesso al fine di integrarne o ampliarne il contenuto, su sua iniziativa o su richiesta del Verificatore oppure del *Proponente*/*Committente*;

- **Utilizzo:** la notazione di versionamento è impiegata nei seguenti casi:

- **Nomenclatura documenti:** ogni documento in formato PDF$_{|g|}$ deve seguire la seguente convenzione:

  `NomeFile_v{X}.{Y}.pdf`

  Ogni modifica ed ogni nuova versione devono venire apportate nel registro modifiche del documento inserendo i campi richiesti: Versione, Autore, Data e Descrizione.
  Quando i redattori eseguono delle correzioni al documento a seguito di una revisione da parte dei verificatori, devono riportare nel registro delle modifiche la modifica fatta ed il suo relativo titolo (*es.* Revisione).
- **Riferimenti interni:** ogniqualvolta in un documento sia presente un riferimento ad un altro documento è necessario inserirne il nome e la sua versione:

  `Documento di Riferimento - v{X}.{Y}`
- **Stato del documento:**
  - **Bozza:** indica un documento in prima stesura;
  - **Preliminare:** indica un documento in redazione, al quale si applicano ciclicamente revisioni e correzioni;
  - **Formale:** indica un documento ufficiale e approvato dal responsabile. Inoltre, deve essere specificato il tipo di utilizzo;
  - **Interno:** indica un documento che deve rimanere all'interno di *GoGo Team*;
  - **Esterno:** indica un documento che può venire distribuito a terzi.
- **Intestazione:** deve essere specificata la versione attuale del documento;
- **Pagina:** deve essere specificata la versione attuale del documento nella nota a piè di pagina seguendo il formato indicato in cap. 9.3.

12.2.2 Codifica

Al fine di uniformare lo stile di versionamento degli elementi di codifica si è scelta la seguente norma di versionamento:

`{X}.{Y}`

dove:

- `{X}`: indica un'evoluzione significativa del file di codifica. Si verifica l'aumento di questo numero quando il file è in una versione stabile e completa;
- `{Y}`: indica l'apporto di modifiche al file che possono venire apportate dal Programmatore stesso al fine di integrarne o ampliarne le funzionalità richieste.

13 Ambiente di sviluppo

In questa sezione si descrivono gli strumenti scelti dal gruppo *GoGo Team* per un ottimale sviluppo di progetto.

13.1 Ambiente generale

13.1.1 Sistemi Operativi

L'intero sviluppo del progetto viene svolto in ambienti Unix-Like e Windows, nello specifico, Ubuntu $\geqslant$ 11.10, Mac OS X $\geqslant$ 10.7 e Windows $\geqslant$ 7.
Tale scelta è maturata dopo che i componenti del gruppo hanno analizzato l'ambito di sviluppo del progetto, che è composto da una parte client$_{|g|}$, sviluppata tramite applicativo web$_{|g|}$, e una parte server$_{|g|}$, sviluppata in ambiente JVM$_{|g|}$. Tali tecnologie sono indipendenti dall'ambiente di sviluppo e di impiego.

13.1.2 Browser

Sotto esplicita richiesta da parte del *Proponente*, lo sviluppo dell'applicativo, nella sua parte web client$_{|g|}$, dev'essere svolto in riferimento all'ambiente Google Chrome$_{|g|}$ $\geqslant$ 26.0 (https://www.google.com/intl/it/chrome/browser/).

13.1.3 Server

Sotto consiglio da parte del *Proponente*, lo sviluppo dell'applicativo, nella sua parte server$_{|g|}$, dev'essere svolto in ambiente JVM$_{|g|}$. Si è scelto di utilizzare il server$_{|g|}$ Apache Tomcat$_{|g|}$ $\geqslant$ 7.0.27 (http://tomcat.apache.org/) e linguaggio JVM$_{|g|}$ Java$_{|g|}$ $\geqslant$ 1.6.0 (http://www.oracle.com/technetwork/java/index.html).

13.2 Ambiente di documentazione

13.2.1 Editor testi

La scrittura dei documenti deve essere svolta tramite linguaggio di markup$_{|g|}$ LaTeX (http://www.latex-project.org/) ed utilizzando, per tutti gli ambienti OS$_{|g|}$, l'editor Texmaker $\geqslant$ 1.9.9 (http://www.xm1math.net/texmaker/) oppure l'editor Kile $\geqslant$ 2.1.3 (http://www.kile.sourceforge.net).
La compilazione deve produrre un file in formato PDF$_{|g|}$ che sarà prodotto dal comando, integrato anche nell'editor,

```
pdflatex NomeFile.tex
```

13.2.2 Verifica ortografica

La verifica ortografica per i documenti scritti in linguaggio LaTeX deve essere effettuata tramite lo strumento Aspell $\geqslant$ 0.60.6 (http://aspell.net/).
Il programma Aspell deve essere usato via terminale eseguendo il comando

```
aspell --mode=tex --lang=it check NomeFile.tex
```

13.2.3 Diagrammi UML

Per la stesura dei grafici UML$_{|g|}$ viene utilizzato il programma Visual Paradigm for UML $\geqslant$ 10.0 (http://www.visual-paradigm.com/). Il programma viene utilizzato in licenza *Community Edition* la quale ne permette l'uso per fini non commerciali.
Vengono elencati i motivi che hanno portato la scelta del programma:

- pieno supporto allo standard UML$_{|g|}$ v2.x (http://www.visual-paradigm.com/support/documents/vpumluserguide.jsp);
- facilità di utilizzo e ottima qualità dei diagrammi prodotti (con possibilità di apportare modifiche successive);
- esportazione dei grafici nei formati PDF$_{|g|}$, PNG$_{|g|}$ e JPG$_{|g|}$;
- programma cross-platform$_{|g|}$ disponibile per tutti gli OS$_{|g|}$ adottati.

13.2.4 Gestione della pianificazione

Per la gestione della pianificazione delle attività e dei ruoli durante le diverse fasi di progetto si è deciso di usare Microsoft Project 2010 (http://www.microsoft.com/project/en-us/project-management.aspx). La scelta è stata vincolata dal miglior grado di qualità che il programma esprime nella gestione dei grafici di pianificazione rispetto alla concorrenza.
Il programma viene utilizzato in licenza accademica per mezzo della MSDN-AA (https://msdnaa.studenti.math.unipd.it/) la quale è disponibile per tutti gli studenti iscritti ai corsi di laurea in matematica ed informatica (facenti parte del Dipartimento di Matematica dell'Università degli studi di Padova), per un uso privato non commerciale.

13.3 Ambiente di sviluppo delle applicazioni

13.3.1 Versionamento

Lo strumento di versionamento scelto è git (http://git-scm.com/). git si presenta come uno strumento diffuso nell'ambito del sviluppo software$_{|g|}$, fatto che comporta la presenza di una grande quantità di documentazione a supporto del prodotto, la quale è liberamente accessibile dai componenti del gruppo. Il programma, inoltre, si integra con estrema semplicità negli strumenti di sviluppo scelti dal gruppo.
Il servizio di repository$_{|g|}$ git viene fornito di default da SourceForge (https://sourceforge.net/).

13.3.2 IDE di base

Per la scrittura del codice sarà utilizzato l'IDE$_{|g|}$ Eclipse $\geqslant$ 3.5 (http://www.eclipse.org/).
Questo strumento è stato scelto per la sua elevata possibilità di personalizzazione ed estensione con componenti necessari ad uno sviluppo più agevole delle applicazioni.

13.3.3 Framework di codifica linguaggi web

Per la codifica delle parti web-side viene impiegato il framework$_{|g|}$ GWT (Google Web Toolkit) $\geqslant$ 2.5.0 (https://developers.google.com/web-toolkit/).
GWT è stato scelto per i seguenti motivi:

- sviluppo in linguaggio Java;
- riutilizzo del codice;
- creazione di pagine web dinamiche mediante tecnologia AJAX$_{|g|}$;
- indipendenza dall'ambiente di sviluppo.

Il framework è disponibile come componente aggiuntivo integrabile nell'ambiente Eclipse.

14 Ambiente di verifica e validazione

Vengono descritti di seguito gli strumenti che *GoGo Team* utilizzerà per garantire la qualità del prodotto software$_{|g|}$ sviluppato. Per ogni strumento, è fornita una descrizione delle principali funzionalità.

14.1 Analisi statica

14.1.1 Eclipse ($\geqslant$ 3.5)

Eclipse non è un editor, ma un ambiente di sviluppo; non si può, ad esempio, aprire e compilare un qualunque file. Per usare questo IDE$_{|g|}$ bisogna organizzare il proprio lavoro in progetti. Un progetto è l'insieme di file e impostazioni relativi ad un programma che si sta sviluppando. Integra un debugger$_{|g|}$, che permette di:

- far partire l'esecuzione;
- sospendere l'esecuzione e ispezionare lo stack$_{|g|}$ di attivazione dei metodi (comprese le variabili);
- eseguire solo una riga o un metodo;
- far ripartire il programma.

Per indicare al debugger$_{|g|}$ quando sospendere l'esecuzione si devono indicare una o più righe di codice che (se raggiunte) bloccano il programma, permettendo al programmatore di analizzare lo stato del processo in cerca di errori. Eclipse supporta poi l'evidenziazione della sintassi, l'auto-completamento, può essere integrato con dei plugin$_{|g|}$ e tiene traccia delle gerarchie e dei percorsi dei file, nonché dei metodi e dei campi dati delle classi.

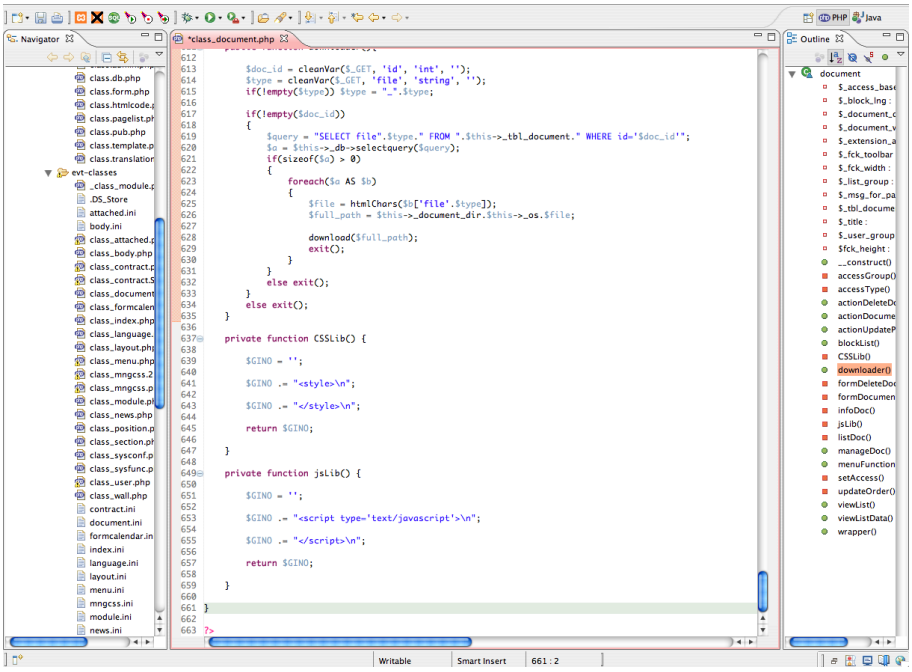

Figura 5: Interfaccia grafica di Eclipse.

14.1.2 GWT ($\geqslant$ 2.5.0)

GWT è un framework$_{|g|}$ integrato per la creazione di applicazioni web$_{|g|}$. Esso agevola lo sviluppo del codice, grazie all'auto-completamento, all'evidenziazione della sintassi (*syntax highlighting*) e alla conoscenza del linguaggio Java$_{|g|}$, poichè fornisce dei suggerimenti. Offre molte librerie AJAX$_{|g|}$ e un debugger$_{|g|}$ integrato.

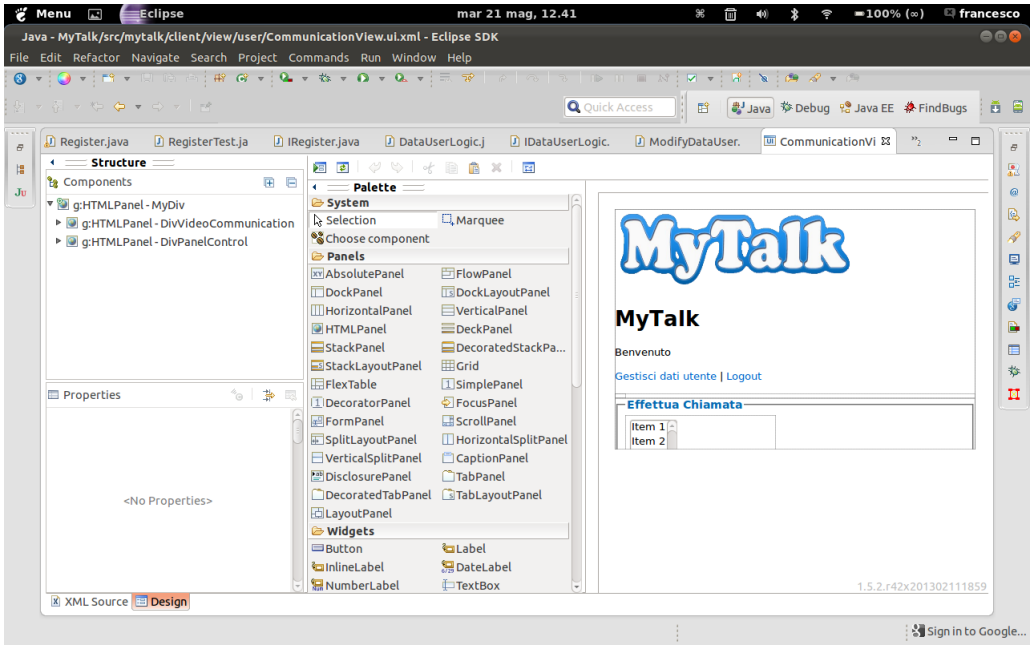

Figura 6: Interfaccia grafica di GWT in ambiente Eclipse.

14.1.3 Metrics Plugin for Eclipse ($\geqslant$ 1.3.6)

Per calcolare le metriche relative al software$_{|g|}$ prodotto verrà utilizzato Metrics Plugin for Eclipse. La spiegazione dettagliata delle metriche, la loro interpretazione e i vincoli da rispettare sono esposti nel *PianoDiQualifica_v3.0.pdf*.

14.1.4 FindBugs Plugin for Eclipse ($\geqslant$ 2.0)

FindBugs cerca errori nei programmi Java$_{|g|}$. I bug$_{|g|}$ sorgono per una serie di motivi:

- caratteristiche del linguaggio difficili;
- metodi API$_{|g|}$ non compresi appieno;
- invarianti non compresi appieno quando il codice viene modificato;
- errori banali: utilizzo errato degli operatori booleani, errori di tipo.

.

FindBugs utilizza l'analisi statica per ispezionare il codice Java$_{|g|}$ per trovare le occorrenze di errori. L'analisi statica di FindBugs avviene controllando il codice di un

programma (l'esecuzione non è necessaria). I bug$_{|g|}$ vengono suddivisi in rank e il loro intervallo va da 1 a 20. Le categorie sono le seguenti (un indice più basso indica un bug$_{|g|}$ più importante):

- Molto preoccupante (scariest) [1;4]
- Preoccupante (scary) [5;9]
- Problematico (troubling) [10;14]
- Rilevante (of concern) [15;20]

La lista di sigle possibili riguardanti i bug$_{|g|}$ trovati è raggiungibile all'indirizzo http://findbugs.sourceforge.net/bugDescriptions.html. Findbugs indica inoltre a quale categoria appartiene il bug$_{|g|}$ rilevato (correctness bug, bad practice$_{|g|}$, dodgy code). Di maggior rilievo sono i bug$_{|g|}$ riferiti a cattive norme di codifica (bad practice$_{|g|}$) o segmenti di codice scritti in maniera ambigua (dodgy code). FindBugs, tuttavia, in alcuni casi fornisce alcuni risultati da considerare falsi positivi (riguardanti possibili ambiguità del codice e bad practice$_{|g|}$). Il Programmatore, in tal caso, deve motivare la scelta di non correggere tale errore. Utilizzando l'attivazione automatica di questo strumento viene effettuata la verifica di bug$_{|g|}$ ogni volta che viene modificata una classe interna del progetto.

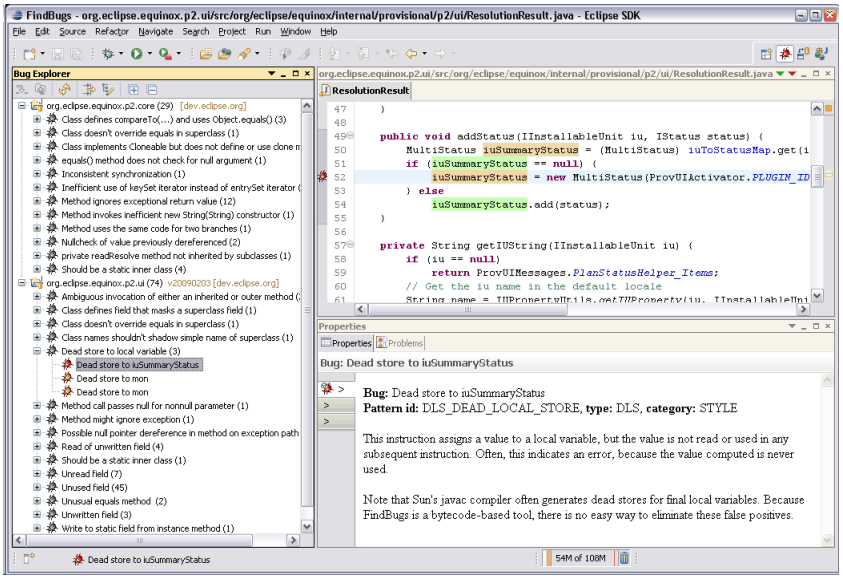

Figura 7: Interfaccia grafica di FindBugs.

14.1.5 JUnit (⩾ 4.11)

JUnit è un framework$_{|g|}$ open source$_{|g|}$ per effettuare test di unità. I *test di unità* sono dei test che vanno a verificare la correttezza direttamente del codice, in ogni sua piccola parte. Questi test vanno scritti ed eseguiti dal programmatore stesso per ogni porzione di codice da lui sviluppata.
I *test funzionali*, invece, sono dei test che vanno a verificare che il software$_{|g|}$ nella sua

completezza funzioni correttamente. Questi test trattano il sistema come se fosse una scatola nera, alla quale vengono forniti degli input e viene verificata la correttezza degli output. Alcune delle caratteristiche principali di JUnit sono:

- Possibilità di effettuare diversi tipi di test con asserzioni diverse.
- Presenza di funzionalità per eseguire delle istruzioni prima o dopo che vengano eseguiti i test, grazie alle annotazioni `@Before` e `@After`. JUnit permette lo sviluppo di test di integrazione grazie all'unione automatica di più test di unità: come sopra, compito del Programmatore è unicamente quello di unire i test di unità da integrare. Un'altra funzione aggiuntiva è l'identificazione e classificazione delle varie tipologie di errori, suddivise tra "errors (difetti) e failures (malfunzionamenti).

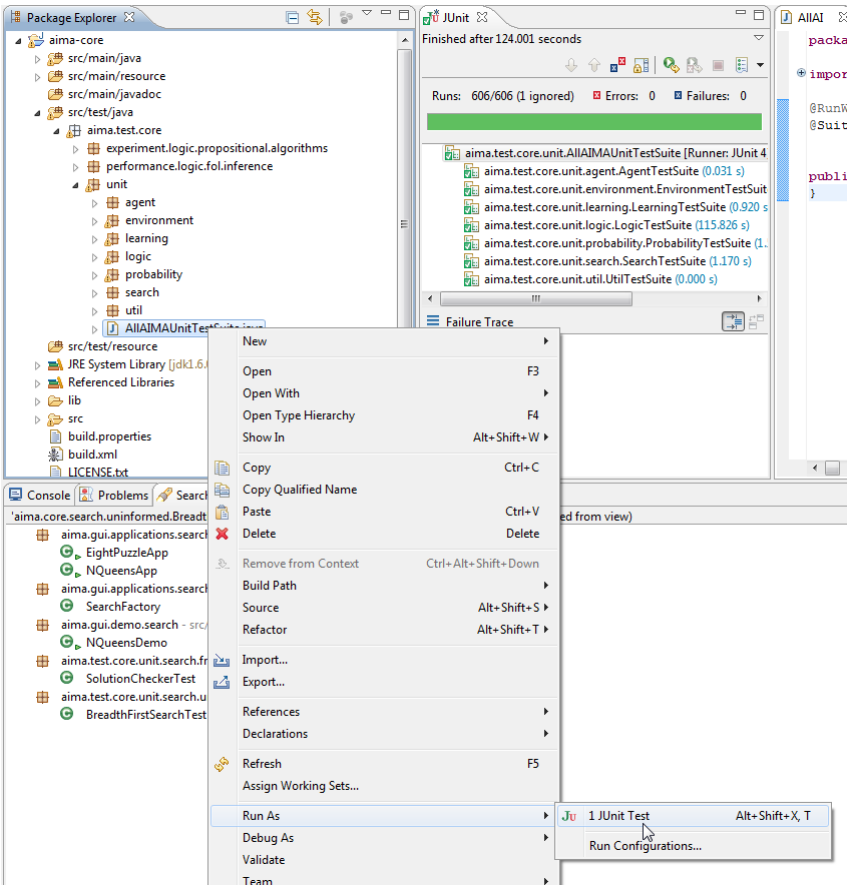

Figura 8: Interfaccia grafica di JUnit.

14.2 Analisi dinamica

14.2.1 EclEmma Plugin for Eclipse ($\geqslant$ 2.2.0)

Strumento che permette di individuare quali righe di codice vengono effettivamente eseguite durante un specifico test. È necessario che il Programmatore verifichi ogni possibile cammino del codice tracciando quali righe non vengono mai eseguite (eliminare le righe inutilizzate o rivedere la logica del metodo). EclEmma fornisce statistiche

sui segmenti di codice testati, per una visione generale del cammino percorso dall'esecuzione dei test. Solo una volta testati tutti i cammini possibili del metodo, è possibile avere un test completo.

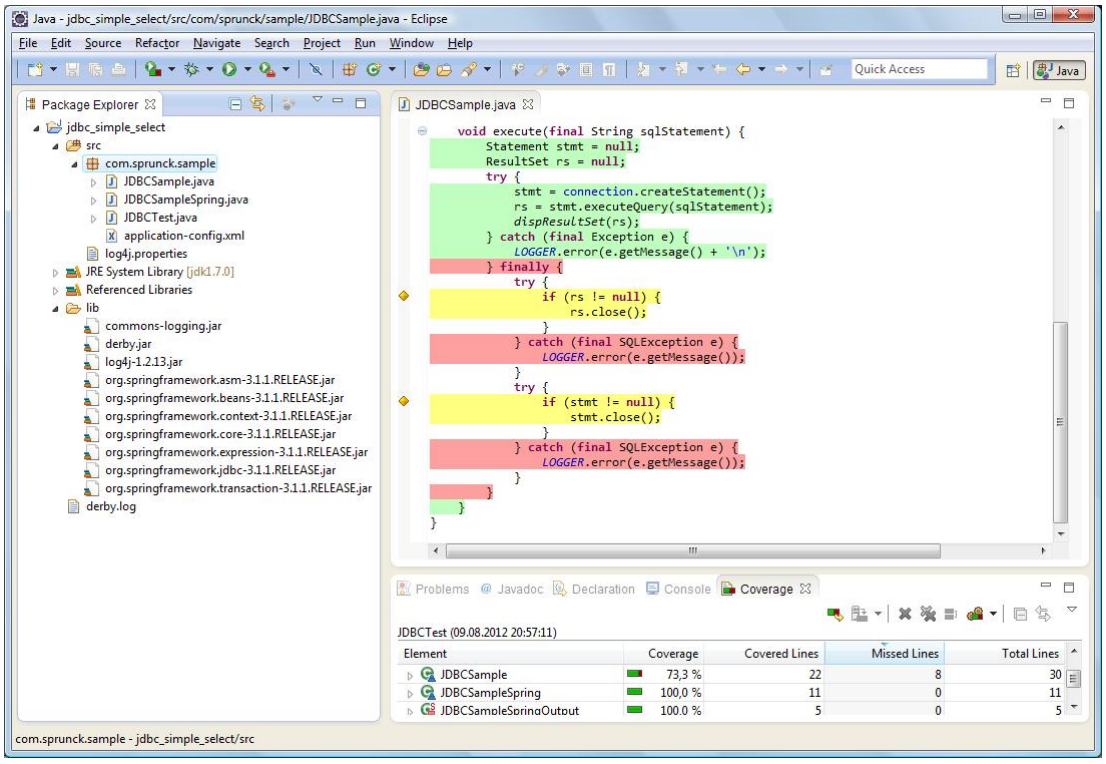

Figura 9: Utilizzo di EclEmma all'interno di Eclipse.

14.2.2 ApacheBench ($\geqslant$ 2.2)

ApacheBench è uno strumento che ci permette di stressare al massimo il server$_{|g|}$, normalmente è disponibile tramite riga di comando su Linux col comando `ab -n numero di richieste da testare -c numero di richieste contemporanee [http[s]://]hostname[:port]/path`.