06 luglio 2013

GoGo Team

Piano di Qualifica

Informazioni sul documento

Nome Documento	Piano di Qualifica
Versione	4.0
Stato	*Formale*
Uso	*Esterno*
Data Creazione	4 dicembre 2012
Data Ultima Modifica	06 luglio 2013
Redazione	Davide Ceccon
Approvazione	Matteo Belletti
Verifica	Sara Lazzaretto
Lista distribuzione	GoGo Team
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Il proponente Zucchetti S.p.A.

Sommario

Questo documento contiene le norme e le procedure di validazione adottate dal gruppo *GoGo Team* per il progetto **MyTalk**, proposto dall'azienda *Zucchetti S.p.A.*

Registro delle modifiche

Versione	Autore	Data	Descrizione
4.0	Matteo Belletti	2013-07-06	Approvazione. Approvazione del documento, cambio di stato in "Formale a uso esterno" e avanzamento di versione.
3.3	Davide Ceccon	2013-07-04	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Sara Lazzaretto in data 2013-07-02.
3.2	Davide Ceccon	2013-07-02	Aggiunta contenuti nella sessione *Test di unità* e modifica contenuti nella sessione *Test di sistema*
3.1	Davide Ceccon	2013-07-01	Aggiunto *Test di sistema*
3.0	Valentina Pasqualotto	2013-06-10	Approvazione. Approvazione del documento, cambio di stato in "Formale a uso esterno" e avanzamento di versione.
2.6	Matteo Belletti	2013-05-30	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Elena Zerbato in data 2013-06-06.
2.5	Matteo Belletti	2013-05- 26	Aggiunto il capitolo *Rapporto Misurazioni.*
2.4	Matteo Belletti	2013-05- 25	Aggiunto *Test di unità*
2.3	Matteo Belletti	2013-02-15	Aggiunta contenuti nella sessione *Gestione amministrativa delle revisioni.*
2.2	Matteo Belletti	2013-02-15	Aggiunta contenuti nella sessione *Visione generale delle strategie di verifica*
2.1	Matteo Belletti	2013-01-28	Correzione a seguito delle segnalazioni effettuate in sede di RP dal *Committente* prof. Tullio Vardanega in data 2013-02-07.

2.0	Francesco Zattarin	2013-01-28	Approvazione. Approvazione del documento, cambio di stato in "Formale a uso esterno" e avanzamento di versione
1.7	Davide Ceccon	2013-01-27	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Davide Ceccon in data 2013-01-26.
1.6	Sara Lazzaretto	2013-01-25	Correzione. Correzione degli errori lessicali ed ortografici.
1.5	Sara Lazzaretto	2013-01-24	Aggiunta Contenuti. Aggiunto nella sotto sessione *Pianificazione dello sviluppo software* modello a V.
1.4	Sara Lazzaretto	2013-01-23	Aggiunta Contenuti. Stesura della sessione *Interpretazione delle metriche*.
1.3	Elena Zerbato	2013-01-21	Correzione. Correzione degli errori lessicali ed ortografici.
1.2	Sara Lazzaretto	2013-01-20	Aggiunta Contenuti. Aggiunta appendice B *Attività di test*.
1.1	Sara Lazzaretto	2013-01-15	Correzione. Correzione a seguito delle segnalazioni effettuate in sede di RR dal *Committente* prof. Tullio Vardanega in data 2013-01-11. Rimozione della sottosezione *Strumenti, tecniche, metodi* e spostamento della sezione *Obiettivi di qualità* in appendice A.
1.0	Valentina Pasqualotto	2012-12-20	Approvazione. Approvazione del documento, cambio di stato in "Formale a uso esterno" e avanzamento di versione

0.9	Francesco Zattarin	2012-12-17	Revisione. Revisione a seguito delle segnalazioni effettuate dal verificatore Davide Ceccon in data 2013-01-16.
0.8	Alessandro Bonaldo	2012-12-15	Ampliamento Contenuti. Modifica della sezione *Visione generale della strategia di verifica*.
0.7	Alessandro Bonaldo	2012-12-11	Aggiunta Contenuti Completamento della sezione *Resoconto di attività di verifica*. Correzione. Correzione degli errori lessicali ed ortografici del documento.
0.6	Francesco Zattarin	2012-12-10	Ampliamento Contenuti. Integrazione della sezione *Gestione amministrativa della revisione* e inizio stesura della sezione *Resoconto di attività di verifica*.
0.5	Francesco Zattarin	2012-12-08	Correzione. Correzione degli errori lessicali ed ortografici della sezione *Visione generale della strategia di verifica*. Aggiunta Contenuti. Ampliamento della sezione *Risorse*.
0.4	Alessandro Bonaldo	2012-12-07	Aggiunta Contenuti. Completamento della sezione *Visione generale della strategia di verifica* e impostazione della sezione *Risorse*.
0.3	Francesco Zattarin	2012-12-06	Aggiunta Contenuti. Ampliamento della sezione *Visione generale della strategia di verifica* e stesura della sezione *Gestione amministrativa della revisione*.

0.2	Francesco Zattarin	2012-12-05	Aggiunta Contenuti. Stesura della sezione *Introduzione* e parte della sezione *Visione generale della strategia di verifica*.
0.1	Alessandro Bonaldo	2012-12-04	Prima Stesura. Contenuti documento: scheletro di base dell'intero documento.

Tabella 1: Versionamento del documento

Storico

pre-RR

Versione 1.0	Nominativo
Redazione	Alessandro Bonaldo, Francesco Zattarin
Verifica	Davide Ceccon
Approvazione	Valentina Pasqualotto

Tabella 2: Storico ruoli pre-RR

RR ->RP

Versione 2.0	Nominativo
Redazione	Elena Zerbato, Sara Lazzaretto
Verifica	Davide Ceccon
Approvazione	Francesco Zattarin

Tabella 3: Storico ruoli RR ->RP

RP ->RQ

Versione 3.0	Nominativo
Redazione	Matteo Belletti
Verifica	Elena Zerbato
Approvazione	Valentina Pasqualotto

Tabella 4: Storico ruoli RR ->RP

RQ ->RA

Versione 4.0	**Nominativo**
Redazione	Davide Ceccon
Verifica	Sara Lazzaretto
Approvazione	Matteo Belletti

Tabella 5: Storico ruoli RQ ->RA

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di illustrare le strategie adottate per implementare i processi di verifica e validazione del lavoro svolto da *GoGo Team* per assicurare la qualità del progetto **MyTalk** e dei processi volti alla sua produzione. In futuro il presente documento potrà essere aggiornato in seguito a scelte progettuali del gruppo e/o variazione dei requisiti da parte del *Proponente*.

1.2 Scopo del prodotto

Il prodotto **MyTalk** è volto ad offrire la possibilità agli utenti di comunicare tra loro, trasmettendo il segnale audio e video, attraverso il browser$_{|g|}$ mediante l'utilizzo di soli componenti standard, senza che sia necessario installare plugin$_{|g|}$ o programmi aggiuntivi (*es.* Skype). Attualmente, infatti, la comunicazione istantanea tra utenti avviene solo tramite componenti non presenti di default nei browser$_{|g|}$. Il software$_{|g|}$ dovrà risiedere in una singola pagina web$_{|g|}$ e dovrà essere basato sulla tecnologia WebRTC$_{|g|}$ (http://www.webrtc.org).

1.3 Glossario

Al fine di migliorare la comprensione al lettore ed evitare ambiguità rispetto ai termini tecnici utilizzati nel documento, viene allegato il file *Glossario_v3.0.pdf*, nel quale vengono descritti i termini contrassegnati dal simbolo $_{|g|}$ alla fine della parola. Per i termini composti da più parole, oltre al simbolo $_{|g|}$, è presente anche la sottolineatura.

1.4 Riferimenti

1.4.1 Normativi

- Capitolato d'appalto: **MyTalk**, *software$_{|g|}$ di comunicazione tra utenti senza requisiti di installazione*, rilasciato dal proponente *Zucchetti S.p.A.*, reperibile all'indirizzo http://www.math.unipd.it/~tullio/IS-1/2012/Progetto/C1.pdf.
- Analisi dei requisiti (allegato *AnalisiDeiRequisiti_v4.0.pdf*).
- Norme di progetto (allegato *NormeDiProgetto_v4.0.pdf*).

1.4.2 Informativi

- Software Engeneering - Chapter 24: Quality Management, Chapter 26: Process Improvement – Ian Sommerville - 9^{th} Edition (2009).
- Materiale del corso di Ingegneria del Software 2012-2013 – Prof. Tullio Vardanega e Riccardo Cardin (http://www.math.unipd.it/~tullio/IS-1/2012/).
- ISO/IEC$_{|g|}$ 9126:2001 (inglobato da ISO/IEC$_{|g|}$ 25010:2011): Systems and software engineering – Systems and software Quality Requirements and Evaluation

(SQuaRE) – System and software quality models (http://www2.cnipa.gov.it/site/_contentfiles/01379900/1379951_ISO%209126.pdf).

- ISO/IEC$_{|g|}$ 12207:2008: Software Life Cycle Processes (http://en.wikipedia.org/wiki/ISO/IEC_12207).
- ISO/IEC$_{|g|}$ 14598:2001 (inglobato da ISO/IEC$_{|g|}$ 25040:2011): Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Evaluation process (http://www2.cnipa.gov.it/site/_contentfiles/01379900/1379952_ISO%2014598.pdf).
- ISO/IEC$_{|g|}$ 15504:1998: Information Tecnology – Process Assessment, conosciuto come SPICE (Software Process Improvement and Capability dEtermination) (http://www2.cnipa.gov.it/site/_contentfiles/00310300/310320_15504.pdf).
- ISO/IEC$_{|g|}$ 15939:1998: Software Engineering – Software measurement process (2002) (http://www2.cnipa.gov.it/site/_contentfiles/01379900/1379952_ISO%2014598.pdf).

2 Visione generale della strategia di verifica

2.1 Organizzazione

Allo scopo di garantire la qualità del prodotto in tutte le sue fasi di realizzazione, accertandone la conformità rispetto a quanto emerso durante l'attività di Analisi dei Requisiti (*AnalisiDeiRequisiti_v4.0.pdf*), si intende svolgere una costante verifica attraverso tutte le fasi di sviluppo del progetto. È infatti impensabile prevedere la verifica solo alla fine del lavoro, rischiando di dover rifare o ripensare alcune componenti.
Le operazioni di controllo verranno istanziate quando il prodotto da analizzare avrà raggiunto uno stato per cui presenti differenze sostanziali rispetto allo stato precedente; grazie al registro delle modifiche le verifiche saranno circoscritte ai soli cambiamenti permettendo di ottimizzare il tempo e le risorse impiegate.
L'approccio scelto per la correzione delle anomalie è quello del *Broken Windows Theory*, secondo il quale ogni difetto individuato va corretto immediatamente onde evitarne il propagarsi all'interno del progetto.
I processi di verifica dovranno essere il più automatizzati possibile in modo tale da garantirne la ripetibilità e l'efficienza. Il modello di ciclo di vita scelto è quello incrementale (*PianoDiProgetto_v4.0.pdf*).

L'obiettivo delle attività di verifica è quello di trovare e rimuovere i problemi presenti. Un problema può verificarsi a vari livelli, e per ogni livello assume un nome diverso:

- **fault (difetto)**: è l'origine del problema, il difetto che fa scaturire il malfunzionamento;
- **error (errore)**: è lo stato per cui il software$_{|g|}$ si trova in un punto sbagliato del flusso di esecuzione o con valori sbagliati rispetto a quanto previsto dalla specifica;
- **failure (fallimento, guasto)**: è un comportamento difforme dalla specifica, cioè la manifestazione dell'errore all'utente del software$_{|g|}$.

Esiste una relazione di causa-effetto fra questi tre termini:

$$\text{DIFETTO} \longrightarrow \text{ERRORE} \longrightarrow \text{FALLIMENTO}$$

Non sempre un errore dà origine ad un fallimento: ad esempio potrebbero esserci alcune variabili che si trovano in stato erroneo ma non vengono lette, o non viene percorso il ramo di codice che le contiene. È necessario prestare particolare attenzione a questo tipo di errori (detti anche quiescenti), avvalendosi anche di strumenti per il rilevamento dei bug$_{|g|}$ come FindBugs.

2.2 Pianificazione strategica e generale

Il processo di verifica viene strutturato in tre fasi:

1. **Attività di pre-verifica**: riguardano la pianificazione e la preparazione delle attività di verifica. Consistono nella scelta delle persone che si occuperanno della verifica e nella distribuzione dei documenti o componenti software$_{|g|}$ da controllare. I verificatori lavorano indipendentemente per trovare errori, omissioni e scostamenti rispetto agli standard previsti.

2. **Incontro di verifica**: durante questa fase, un autore del documento o componente software$_{|g|}$ rivede l'oggetto dell'analisi con il Verificatore. Deve essere annotato un elenco delle azioni correttive concordate durante l'incontro. Verranno usate diverse checklist per diverse aree di verifica (documentazione, vari linguaggi di programmazione, etc.).

3. **Attività di post-verifica**: i problemi emersi dall'incontro di verifica devono essere risolti. Questo può includere la correzione di bug$_{|g|}$ e la riscrittura di alcune parti di codice o di documenti. La persona che apporta i cambiamenti deve controllare che tutti i punti inseriti nell'elenco redatto al punto precedente siano stati coperti. Talvolta potrebbe essere richiesta una nuova verifica per accertarsi che le correzioni effettuate corrispondano alle richieste di modifica.

Durante le attività di verifica è inevitabile che gli errori commessi dagli individui vengano esposti a tutto il gruppo. È quindi molto importante che si sviluppi una mentalità per la quale la segnalazione degli errori non diventi motivo di screditamento per l'individuo, ma occasione di crescita per l'intero gruppo di progetto.
Verranno di seguito illustrate le modalità di verifica riguardanti le diverse fasi del progetto.

2.2.1 Verifica della documentazione

Viene svolta mediante due tecniche:

- **Walkthrough**: è un processo informale, non pianificato, che non richiede preparazione ma è molto dispendioso. Consiste nella rilettura completa del deliverable$_{|g|}$ da parte dell'autore stesso o da parte del Verificatore allo scopo di trovare errori e ogni modifica da apportare viene discussa. Nonostante il dispendio di tempo e risorse che comporta questa tecnica, essa si rende indispensabile all'inizio delle diverse fasi di progetto per la correzione dei difetti. Al fine di ridurre l'utilizzo di questa tecnica nelle fasi più avanzate, a favore della più efficiente inspection (definita nel prossimo punto), ogni problematica individuata dovrà essere aggiunta a una lista di controllo.

- **Inspection**: è un processo formale, volto all'individuazione di errori da ricercare in parti ritenute critiche in base all'esperienza precedentemente accumulata attraverso le revisioni. La lista di problematiche da ricercare viene definita gradualmente durante la fase di walkthrough e va a comporre una lista di controllo. L'inspection è da preferire al walkthrough, ma richiede un sufficiente livello di dettaglio nella lista di controllo.
  Riguardo all'attività di verifica della documentazione, elenchiamo di seguito i punti da controllare. Per le parti scritte in LaTeX:

  - presenza dello spazio dopo un segno di punteggiatura;
  - assenza dello spazio dopo la parentesi aperta, prima della parentesi chiusa, all'inizio e alla fine delle virgolette e degli apici;
  - corretto uso della punteggiatura, soprattutto negli elenchi puntati e numerati;
  - corretto utilizzo delle lettere maiuscole o minuscole negli elenchi puntati e numerati;

- assenza di parti di documento da definire marcate da sequenze come “Aggiungere”;
- funzionamento dei link interni ed esterni al documento (assenza della sequenza ’??’ generata da LaTeXin questi casi);
- corretta sillabazione alla fine delle righe;
- assenza di termini definiti nel glossario e non contrassegnati da simbolo $_{|g|}$, con sottolineatura per i termini formati da più parole;
- didascalie coerenti con l’immagine a cui si riferiscono;
- corretto posizionamento delle immagini;
- assenza di errori di battitura (tramite Aspell) e corretto utilizzo degli accenti.

Per una trattazione completa delle caratteristiche che un documento deve possedere, si rimanda al documento *NormeDiProgetto_v4.0.pdf*.

2.2.2 Analisi dei Requisiti

La verifica inizia quando i documenti:

- piano di progetto
- norme di progetto
- analisi dei requisiti
- studio di fattibilità
- piano di qualifica

hanno una struttura definita e sufficienti contenuti su cui poter verificare. La verifica avviene mediante controllo ortografico, grammaticale e concettuale dei documenti da consegnare alla prima revisione; in caso ci fossero problemi il Verificatore dovrà comportarsi come descritto nel documento *NormeDiProgetto_v4.0.pdf*. Il Verificatore deve accertarsi della correttezza logica e formale dei requisiti, della loro tracciabilità, non ambiguità e conformità con quanto richiesto dal *Proponente*. Il tracciamento dei requisiti si avvia con la fase di analisi dei requisiti e permane fino alla fase di codifica. Il suo scopo è quello di evidenziare le relazioni tra i requisiti emersi e ciò che è stato prodotto nelle varie fasi al fine di garantire lo sviluppo di un prodotto finale che soddisfi tutti i requisiti individuati in partenza.
Viene rivolta particolare attenzione ai casi d’uso, che sono la fonte principale di errori di vario tipo: essi devono possedere un codice identificativo univoco e un titolo non ambiguo e sufficientemente descrittivo. Inoltre, gli attori devono avere l’indicazione del ruolo svolto nel particolare caso d’uso. Ogni caso d’uso è associato ad uno o più requisiti. Viene effettuata una prima verifica che controlla che ciascun ingresso di una fase venga messo in relazione con una specifica uscita in quella fase.
Un’ulteriore fonte di errore è l’uso scorretto degli exend per modellare le condizioni. La soluzione sarà quella di minimizzarli.
Nel caso in cui vengano riscontrati errori il Verificatore avrà cura di segnalarli al redattore del documento, il quale dovrà apportare le modifiche richieste e ripresentare il documento stesso per un’ulteriore attività di verifica ed eventuale validazione.

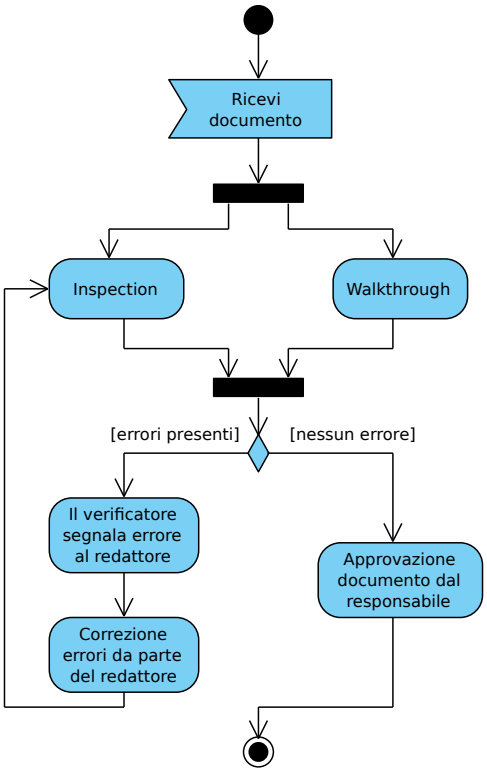

Figura 1: Verifica dei documenti

2.2.3 Progettazione architetturale

Lo scopo del processo di verifica consiste nell'accertarsi che ogni requisito evidenziato in sede di analisi sia tracciabile nei componenti individuati in sede di progettazione e che ogni componente soddisfi almeno un requisito. Durante questa attività sarà scopo del gruppo il perseguimento delle seguenti proprietà:

- **semplicità**: i componenti devono contenere solo quello che è necessario al loro funzionamento;
- **incapsulamento**: il funzionamento interno di una classe viene nascosto all'esterno, proteggendo così gli utenti di quella classe da eventuali modifiche. Per ottenere ciò è necessario progettare robuste interfacce per le classi;
- **coesione**: misura quanto sono collegati tra di loro le componenti di un modulo$_{|g|}$. Se i metodi di una classe svolgono compiti simili, il grado di coesione di tale classe è alto. L'obiettivo sarà quello di massimizzare questo aspetto;
- **accoppiamento**: indica quanto una componente fa affidamento sull'altra, dipendendo da essa. Un basso grado di accoppiamento favorisce la manutenibilità del software$_{|g|}$.

Gli obiettivi di qualità vengono descritti nell'appendice A (cap. A.2). Il verificatore effettua una verifica incrociata tra i requisiti riportati nel documento *AnalisiDeiRequisiti_v4.0.pdf* e le componenti descritte nel documento *SpecificaTecnica_v3.0.pdf*, per accertare che le componenti architetturali prodotte durante la progettazione ad alto livello e i metodi e gli attributi prodotti nella progettazione di dettaglio soddisfino i

requisiti utente. Le attività di tracciamento, controllo diagrammi UML$_{|g|}$ e verifica della progettazione vengono eseguite in parallelo assicurando il soddisfacimento delle quattro proprietà descritte sopra. Inoltre, i verificatori devono controllare che:

- le componenti del sistema non siano eccessivamente complesse;
- non sia possibile decomporre ulteriormente le parti individuate;
- i pattern siano usati correttamente;

I diagrammi UML$_{|g|}$ devono possedere le seguenti proprietà:

- seguire le specifiche descritte in *NormeDiProgetto_v4.0.pdf*;
- conformità allo standard 2.0;
- correttezza tra associazioni;
- complessità non eccessiva: se è possibile individuare dei gruppi logici indipendenti in uno stesso diagramma, allora la complessità è troppo elevata. Ciò è causa di errori, diminuisce la comprensibilità e riduce il tempo di vita del diagramma stesso, in quanto una delle componenti rappresentate potrebbe essere soggetta a cambiamenti;
- le relazioni di dipendenza devono essere corrette;
- le classi devono essere posizionate correttamente (al posto giusto e nel package$_{|g|}$ giusto).

I pattern utilizzati devono essere descritti e motivati, specificando i vantaggi e le implicazioni derivanti dalla loro adozione.

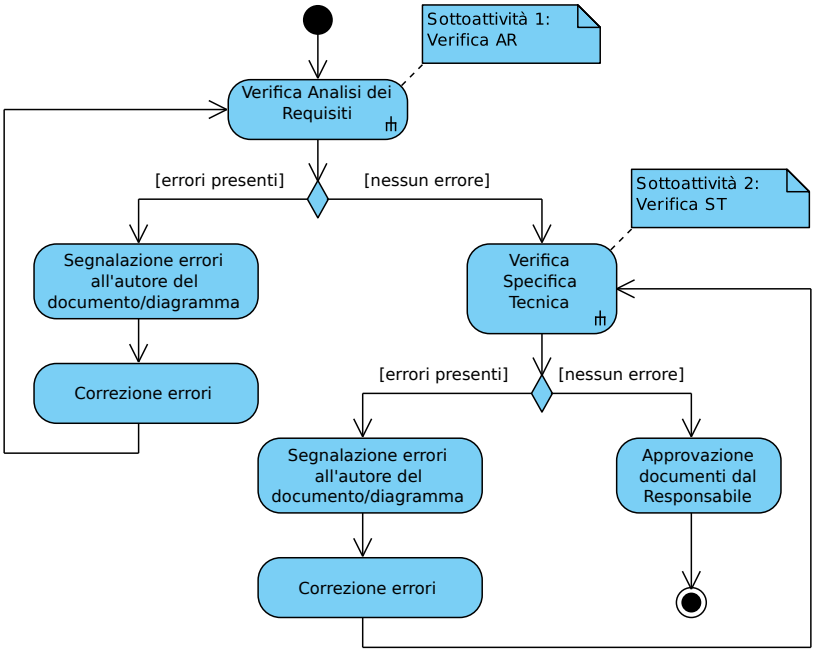

Figura 2: Verifica della progettazione

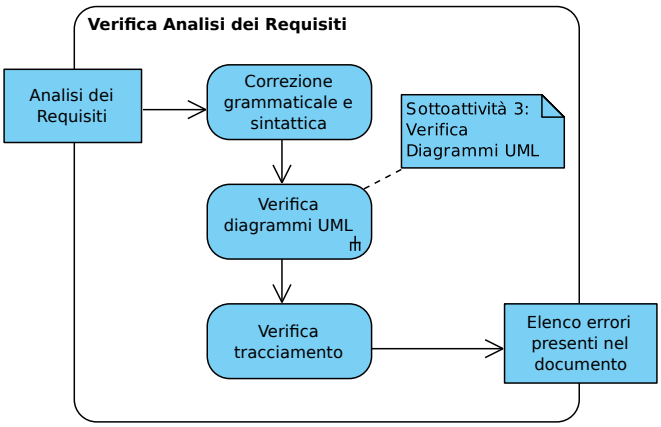

Figura 3: Sottoattività 1: Verifica dell'Analisi dei Requisiti

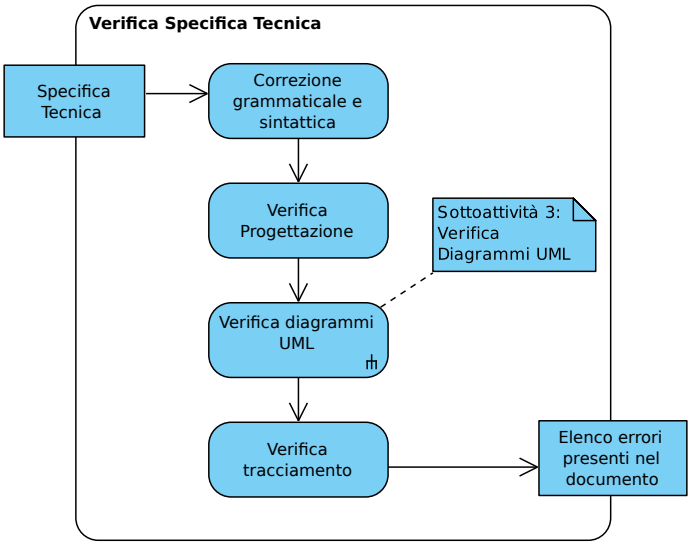

Figura 4: Sottoattività 2: Verifica della Specifica Tecnica

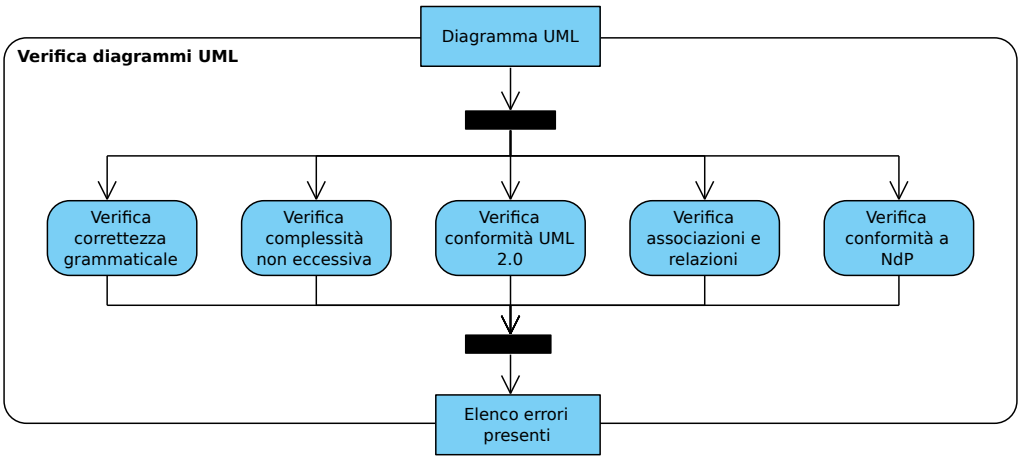

Figura 5: Sottoattività 3: Verifica dei diagrammi UML

2.2.4 Programmazione di dettaglio e codifica

Parallelamente all'attività di progettazione di dettaglio viene portata avanti la codifica, che deve essere accompagnata da test sul codice prodotto non appena questo è pronto. Devono essere effettuati test di unità, scritti dai programmatori stessi, per verificare che il comportamento del codice che hanno scritto corrisponda a quanto ci si aspetta. Un test su un'unità è composto da:

- l'oggetto su cui viene eseguito il test;
- la strategia utilizzata per effettuare la prova;
- le risorse software$_{|g|}$ necessarie;
- il piano di esecuzione del test stesso, che deve prevedere gli ingressi e le uscite attese del test.

È necessario che i test pianificati assicurino le seguenti due proprietà al fine di individuare gli errori:

- **Statement coverage**: il test deve coprire tutte le linee di codice del modulo sotto esame;
- **Branch coverage**: il test deve coprire tutti i rami del flusso di controllo almeno una volta;

Tali test saranno di tipo white box: si entra in dettaglio sul codice sorgente delle singole unità analizzando come, a partire dagli input, gli output sono prodotti. Si verifica così la correttezza logica di ogni metodo.
I verificatori dovranno poi eseguire un'attività di inspection su quanto scritto dai programmatori secondo la seguente lista di controllo:

- le variabili globali non sono ammesse;
- le variabili vanno inizializzate al momento della dichiarazione;
- i nomi delle variabili, delle classi e dei metodi devono rispettare le *NormeDiProgetto_v4.0.pdf*;
- non sono ammesse variabili e metodi non utilizzati;
- minimizzare, se non eliminare, la presenza di warning$_{|g|}$; l'uso del `break`$_{|g|}$ non è ammesso se non all'interno dei costrutti `switch`$_{|g|}$, che vanno minimizzati in quanto aumentano la complessità ciclomatica di 1 per ogni `case`;
- le variabili che controllano l'uscita dai cicli non devono poter essere modificate dall'esterno del ciclo;
- la derivazione della classi non deve essere abusata al fine di evitare ambiguità sull'overriding$_{|g|}$ di metodi;
- deve essere favorita la lazy evaluation$_{|g|}$ delle condizioni booleane;
- il codice deve essere il più possibile comprensibile; qualora risulti di difficile comprensione il programmatore deve inserire dei commenti. I commenti non vanno tuttavia inseriti per descrivere righe il cui significato è ovvio;
- l'indentazione deve essere consistente e favorire la lettura del codice.

Lo stile di codifica deve comunque essere conforme a quanto specificato nel documento *NormeDiProgetto_v4.0.pdf*. Vanno infine garantite le caratteristiche di qualità descritte dallo standard ISO/IEC 9126 e riportate in appendice A.2 e il rispetto dei valori per le metriche stabiliti nel presente documento (cap. 5).
La verifica durante l'attività di codifica verrà effettuata dai Programmatori, che dovranno scrivere il codice del test sui metodi da loro prodotti. È inoltre responsabilità dei Programmatori lo svolgimento dei test di unità e il rispetto degli obiettivi di qualità elencati nella sezione Qualità (cap. A).

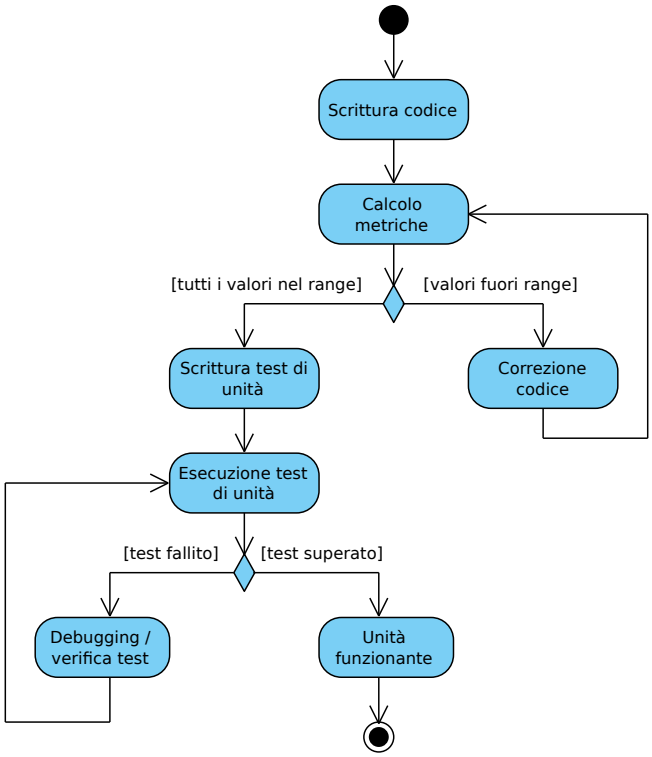

Figura 6: Verifica della codifica

2.3 Test di integrazione

Quando è terminata la fase di sviluppo e testing di unità che dovranno essere integrate, si procede all'integrazione (aggiungendo il codice necessario) e al successivo test di integrazione, senza attendere che vengano ultimati i test su tutte le unità, al fine di ottimizzare i tempi. Tale test ha lo scopo di verificare che le componenti interagiscano nella maniera desiderata.
Il modello di sviluppo scelto è di tipo top-down$_{|g|}$, che prevede che la realizzazione delle componenti di livello più alto preceda quella delle componenti di livello più basso; sarà quindi disponibile in breve tempo uno scheletro del sistema. Gli errori individuati sono pertanto da imputare all'aggiunta dell'ultima componente, rendendo l'attività di testing un'attività incrementale. I test di integrazione sono di tipo black-box. I test devono essere contraddistinti da un ID, devono elencare le componenti in esame e le modalità di testing. Nell'appendice B.2 vengono elencati i test di integrazione svolti dal gruppo per lo sviluppo di **MyTalk**.

2.3.1 Validazione

Quando tutte le componenti sono state testate e integrate si procede con il test di sistema, che controlla che il prodotto svolga correttamente i suoi compiti e rispetti i requisiti fissati in sede di analisi.
Il controllo di sistema si divide in due categorie:

- **alfa-test**: effettuato all'interno del gruppo, consiste nell'utilizzo del software$_{|g|}$, andando a testare tutte le funzionalità e inserendo istruzioni di controllo a tempo di esecuzione, segnalando eventuali malfunzionamenti;

- **beta-test**: consiste in un utilizzo normale del prodotto, alla ricerca degli ultimi problemi residui.

GoGo Team si impegna a garantire il corretto funzionamento del prodotto software$_{|g|}$ realizzato. Nel caso in cui vengano riscontrate anomalie o discrepanze tra le caratteristiche del prodotto e le richieste del cliente sarà cura del fornitore eliminare tali difetti, interamente a proprio carico.

2.4 Responsabilità

La responsabilità dell'attività di verifica viene affidata ai seguenti ruoli:

- Responsabile: controlla che l'evoluzione del progetto rispetti le tempistiche prefissate ed è garante della qualità dei processi interni;
- Verificatore: segue le strategie di analisi concordate e si occupa della parte di verifica, segnalando formalmente gli errori riscontrati.

Gli strumenti, le tecniche e i metodi di verifica vengono descritti nel documento *NormeDiProgetto_v4.0.pdf*.

3 Risorse

3.1 Risorse necessarie

L'attività di verifica richiederà l'utilizzo di risorse umane, hardware$_{|g|}$ e software$_{|g|}$.

3.1.1 Risorse umane

I ruoli per assicurare la qualità del prodotto sono i seguenti:

- **Responsabile di Progetto**: supervisiona la qualità dei processi interni e coordina le attività di verifica. È responsabile nei confronti del committente della corretta realizzazione del prodotto e valuta le proposte di modifica avanzate dal Verificatore assegnando alla persona opportuna il compito di correzione.
- **Amministratore di Progetto**: coordina le attività di verifica, definisce i piani di gestione di qualità, stabilisce le norme da seguire per la risoluzione di anomalie e discrepanze.
- **Verificatore**: esegue l'attività di verifica su ogni prodotto seguendo il metodo black-box utilizzando i test definiti dal Programmatore, riassume gli esiti di tali attività e, in caso di discrepanze, li presenta al Responsabile di Progetto e al Programmatore stesso.
- **Programmatore**: si occupa della stesura del codice, della creazione dei test, della loro esecuzione e nel caso siano presenti errori, del loro debugging$_{|g|}$ con successiva correzione. Deve inoltre correggere gli errori riscontrati dai Verificatori nel lavoro da lui svolto.

3.1.2 Risorse hardware

Saranno necessari:

- computer con installato software$_{|g|}$ necessario allo sviluppo del progetto in tutte le sue fasi;
- luogo fisico in cui incontrarsi per lo sviluppo del progetto, possibilmente con una connessione ad Internet.

3.1.3 Risorse software

Durante la fase di realizzazione del progetto saranno necessari:

- strumenti che consentano l'analisi statica del codice per poter misurare le metriche descritte in sezione 5;
- framework$_{|g|}$ per eseguire test di unità;
- strumenti per automatizzare i test;
- debugger$_{|g|}$ per i linguaggi di programmazione scelti;
- browser$_{|g|}$ come piattaforma di testing dell'applicazione da sviluppare;
- piattaforma di versionamento per la creazione e la gestione di ticket$_{|g|}$.

4 Gestione amministrativa della revisione

4.1 Comunicazione e risoluzione delle anomalie

Un'anomalia è uno scostamento del comportamento del programma rispetto alle aspettative prefissate.
Per la segnalazione e il trattamento delle anomalie si utilizza il servizio di ticketing offerto dalla piattaforma di sviluppo SourceForge (http://www.sourceforge.net). Quando il Verificatore riscontra un'anomalia deve aprire un ticket$_{|g|}$su SourceForge. Le modalità di risoluzione di quest'ultimo e la sua struttura vengono descritte in modo dettagliato nel documento *NormeDiProgetto_v4.0.pdf*. Quando un documento/modulo$_{|g|}$ viene rilasciato in una nuova versione, il Verificatore controlla il registro delle modifiche e, in base ad esso, effettua una verifica alla ricerca di anomalie da correggere. Se ne trova, apre un ticket$_{|g|}$ su SourceForge e lo comunica all'Amministratore; sarà poi compito della persona che ha modificato il documento/modulo$_{|g|}$ apportare le dovute modifiche, che devono essere approvate dall'Amministratore.

4.2 Trattamento della discrepanza

Una discrepanza è una differenza tra il lavoro svolto e quanto era stato pianificato, sia per quanto riguarda i requisiti specificati nel capitolato d'appalto che per quanto riguarda le norme di progetto.
Risolvere una discrepanza comporta necessariamente dei costi, che dovranno venire quantificati dal Responsabile di Progetto sotto il punto di vista delle risorse umane, economiche e temporali. Qualunque sia l'entità della discrepanza, essa dovrà essere registrata e tracciata tramite ticket$_{|g|}$.

4.3 Procedure di controllo di qualità del processo

L'organizzazione dei processi fa riferimento al ciclo di Deming, che ha come scopo il miglioramento continuo, secondo il principio del PDCA. Gli obiettivi sono promuovere una cultura volta al perseguimento della qualità e all'utilizzo ottimale delle risorse e dare una maggiore coesione tra le fasi di analisi, progettazione, realizzazione, verifica e collaudo. Tale metodo è suddiviso in quattro fasi:

1. **Plan - Pianificare**
   (a) Definire il problema/impostare il progetto;
   (b) Documentare la situazione di partenza;
   (c) Analizzare il problema;
   (d) Pianificare le azioni da realizzare.
2. **Do - Realizzare**
   (a) Addestrare le persone incaricate della realizzazione;
   (b) Realizzare le azioni che sono state pianificate.
3. **Check -Verificare**
   (a) Verificare i risultati e confrontarli con gli obiettivi;

- Se si è raggiunto l'obiettivo: passare alla lettera a della fase Act;
- Se non si è raggiunto l'obiettivo: passare alla lettera b della fase Act.

4. **Act - Mantenere o Migliorare**
   (a) Obiettivo raggiunto:
      i. Standardizzare, consolidare e addestrare gli operatori;
      ii. Procedere a un nuovo PDCA per un ulteriore miglioramento sul tema.
   (b) Obiettivo non raggiunto:
      i. Ripetere il ciclo PDCA sullo stesso problema, analizzando criticamente le varie fasi del ciclo precedente per individuare le cause del non raggiungimento dell'obiettivo.

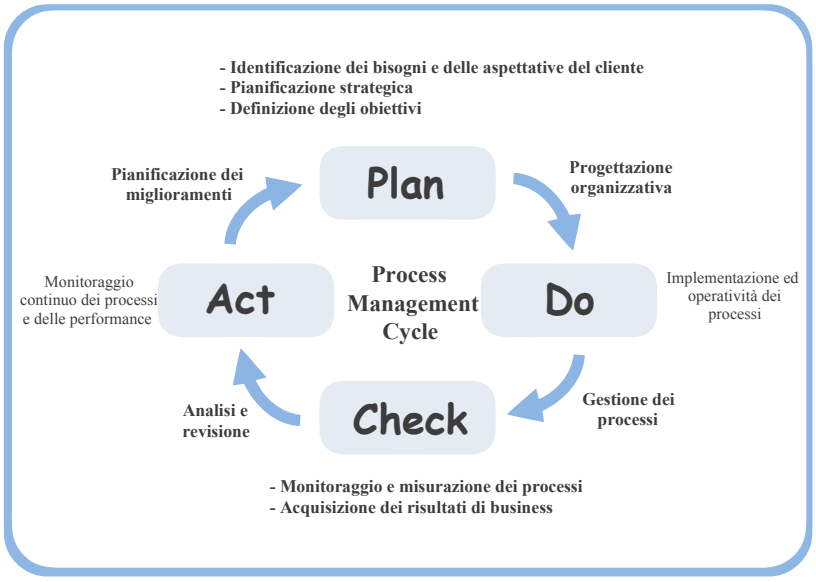

Figura 7: Ciclo di Deming

I parametri che permetteranno di valutare la qualità del processo saranno principalmente:

- il tempo impiegato per essere portato a termine;
- la quantità di risorse impiegate;
- l'aderenza del processo alla pianificazione iniziale;
- il numero di iterazioni che è stato fatto;
- la soddisfazione dei requisiti richiesti;
- il numero di difetti trovati durante la fase di testing;
- l'efficacia dell'attività di correzione dei difetti.

Tali parametri devono essere quantificati sia durante che al termine del processo, al fine di individuare eventuali problemi e capire, attraverso un'analisi condivisa dai membri del gruppo, in quali aree c'è bisogno di un miglioramento. Alla successiva istanziazione del processo, i dati raccolti la volta precedente vanno capiti e migliorati in modo da rendere più efficiente ed efficace il processo stesso.

Il gruppo *GoGo Team* adotta come riferimento gli standard ISO/IEC$_{|g|}$ 15939 e ISO/IEC$_{|g|}$ 14598 per la misurazione e valutazione del software$_{|g|}$ nel suo intero ciclo di vita. In particolare, ISO/IEC$_{|g|}$ 15939 definisce il processo di misura un "ciclo iterativo che prevede la misurazione, la raccolta di feedback e l'utilizzo di questi ultimi per impostare azioni correttive per migliorare il processo di produzione PDCA". La norma suddivide il processo di verifica nelle seguenti quattro attività:

1. definire l'esigenza della misura in base ai requisiti da rispettare e alle risorse a disposizione;
2. pianificare la misurazione definendo tecniche, metriche e procedure;
3. rilevare le misure memorizzando i dati per creare uno storico;
4. valutare le misure rilevate generando report e pianificando azioni correttive.

Lo scopo delle misurazioni è quello di prevedere e stimare i costi per lo sviluppo e la qualità del prodotto. Per tutta la durata del ciclo di vita esse verranno effettuate in diversi momenti:

- in fase di progettazione: hanno l'obiettivo di quantificare l'impegno richiesto dalla manutenzione futura e di prevenire problemi quando il software$_{|g|}$ sarà in uso;
- in fase di collaudo e test: viene effettuato un confronto con le specifiche date e si cerca di individuare problemi non considerati precedentemente;

Vengono di seguito definiti i termini *misurazione*, *misura*, *metrica* e *indicatore*:

- *misurazione*: l'uso di una metrica per assegnare un valore su una scala predefinita;
- *misura*: risultato della misurazione;
- *metrica*: insieme di regole per fissare le entità da misurare, gli attributi rilevanti, l'unità di misura, la procedura per assegnare e interpretare i valori;
- *indicatore*: misura ottenuta indirettamente a partire da altre misure o tramite stime o predizioni; si utilizza un indicatore per quantificare aspetti difficilmente misurabili.

La norma ISO/IEC$_{|g|}$ 14598 descrive il processo di valutazione della qualità del software$_{|g|}$, in accordo con la norma ISO/IEC$_{|g|}$ 9126 descritta in appendice A. Essa verrà utilizzata per valutare il software$_{|g|}$ durante tutto il suo ciclo di vita. Ogni valutazione deve possedere una descrizione, uno scopo, deve identificare i prodotti da valutare e i risultati attesi per ogni caratteristica presa in esame. La figura seguente schematizza il processo di valutazione.

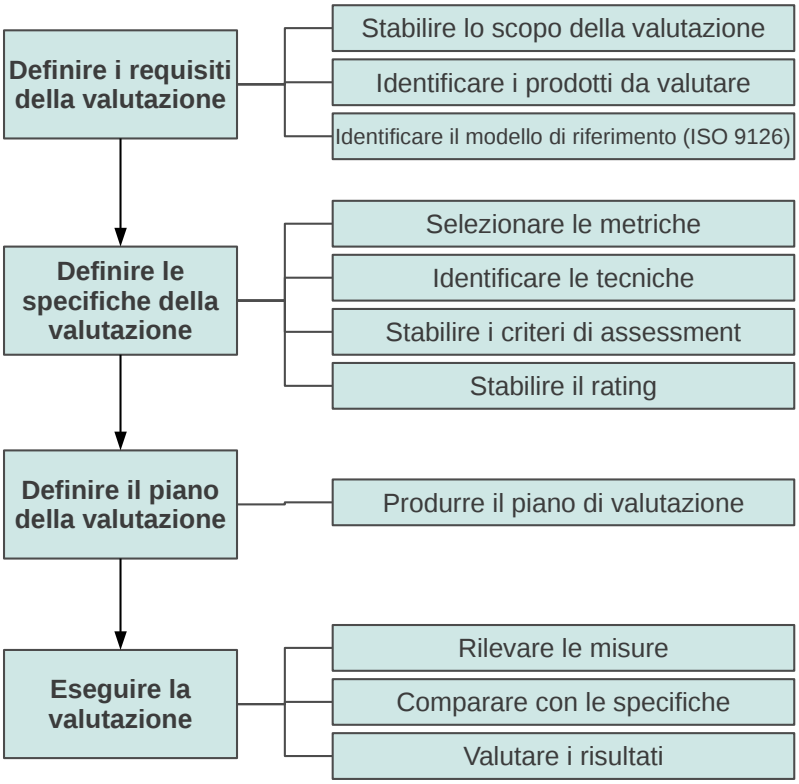

Figura 8: Il processo di valutazione

Poiché le caratteristiche definite nel modello ISO/IEC$_{|g|}$ 9126 non sono tutte richieste per il prodotto che il gruppo deve sviluppare, verrà indicato nell'appendice A.2 quali aspetti vengono ritenuti rilevanti e il profilo di qualità atteso per ognuno di loro. In ogni caso, tutte le componenti del prodotto devono essere sottoposte a valutazione.

5 Interpretazione delle metriche

Una metrica è una misura di una qualche proprietà relativa ad una porzione di software$_{|g|}$, allo scopo di fornire informazioni significative sulla qualità del codice prodotto. Non bisogna tuttavia basarsi solamente sulle metriche, che sono solamente indicatori a posteriori della bontà del lavoro svolto: un'importanza ancora maggiore la riveste il controllo sulla qualità del processo.
Il plugin Metrics di Eclipse permette di monitorare, a tempo di compilazione, numerose metriche relative al codice Java$_{|g|}$ prodotto. Esso dà inoltre la possibilità di impostare dei limiti per le metriche, segnalando eventuali violazioni. L'assenza di valori out of range (fuori limite), tuttavia, non è una condizione sufficiente ad indicare la bontà del codice scritto, che non può prescindere da una buona progettazione e che deve essere il più possibile corretto per costruzione. Le metriche che Metrics è in grado di calcolare sono le seguenti:

Dimensionali

- NOF (Number Of Fields): numero di campi dati per classe;
- NSC (Number of Children): numero di sottoclassi di primo livello di una classe; è un indicatore del riuso del codice;
- NOC (Number of Classes): numero di classi in un particolare scope$_{|g|}$;
- NSF (Number of Static attributes): numero di attributi statici in un particolare scope$_{|g|}$;
- NOM (Number of Methods): numero di metodi in un particolare scope$_{|g|}$;
- NOI (Number Of Interfaces): numero di interfacce in un particolare scope$_{|g|}$;
- NSM (Number of Static Methods): numero di metodi statici$_{|g|}$ in un particolare scope$_{|g|}$;
- NOP (Number Of Packages): numero di package$_{|g|}$ in un particolare scope$_{|g|}$;
- MLOC (Method Lines Of Code): numero di linee di codice di un metodo, al netto delle linee vuote o commentate;
- TLOC (Total Lines Of Code): numero di linee di codice totali, al netto delle linee vuote o commentate;
- NBD (Nested Block Depth): profondità di annidamento dei blocchi;
- PAR (number of PARameters): numero di parametri in un particolare scope$_{|g|}$. L'obiettivo sarà quello di limitare questo numero a 5.

Anche se si ritiene ragionevole porsi l'obiettivo di mantenere la lunghezza dei moduli$_{|g|}$ al di sotto delle 400 righe per agevolarne la manutenibilità, le metriche riguardanti il numero di linee di codice non devono essere viste troppo rigidamente. Piuttosto, la regola da seguire è quella della semplicità: ogni metodo dovrebbe fare una e una sola cosa. Quindi, invece di porre un limite alla lunghezza dei metodi (magari *42* righe?),

è più logico chiedersi quanti compiti svolge un singolo metodo. Se la risposta alle domande "il mio codice è molto indentato?" e "ho inserito spazi vuoti per separare logicamente gruppi di operazioni?" è sì, allora è il caso di spezzare il metodo in altri più semplici. Seguendo queste linee guida, nella maggior parte dei casi la lunghezza non dovrebbe comunque superare le poche decine di righe.

Riguardanti i tipi

- *CA (Afferent Coupling)*: numero di classi esterne al package$_{|g|}$ corrente che dipendono da classi interne al package$_{|g|}$; è equivalente a fan-in. Un valore superiore a 2 indica un codice riusato. Più alto è il valore, maggiore è il riuso. Valori eccessivi indicano però un accoppiamento eccessivo: queste componenti vengono chiamate spesso perché svolgono più di un lavoro;

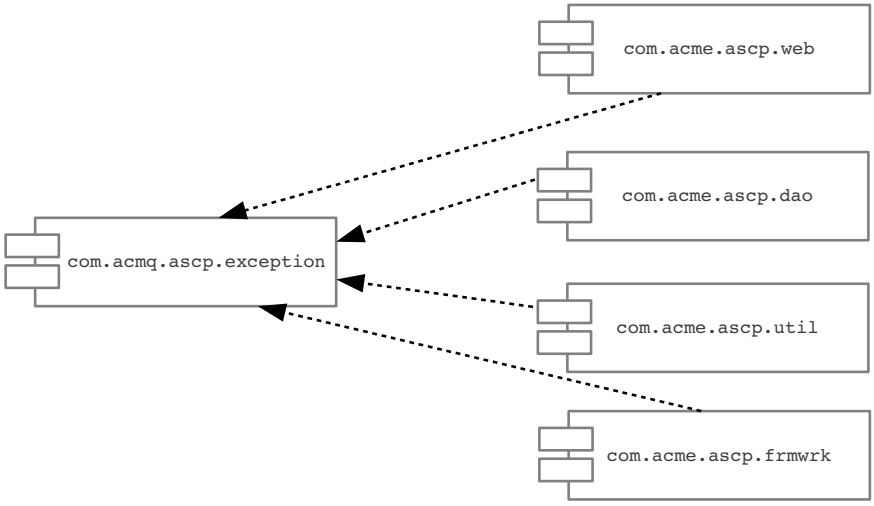

Figura 9: Afferent Coupling

- *CE (Efferent Coupling)*: numero di classi interne al package$_{|g|}$ corrente che dipendono da classi esterne al package$_{|g|}$; è equivalente a fan-out. Alti valori indicano un'eccessiva dipendenza del modulo$_{|g|}$ e ne complicano il testing: questo valore va quindi mantenuto basso.

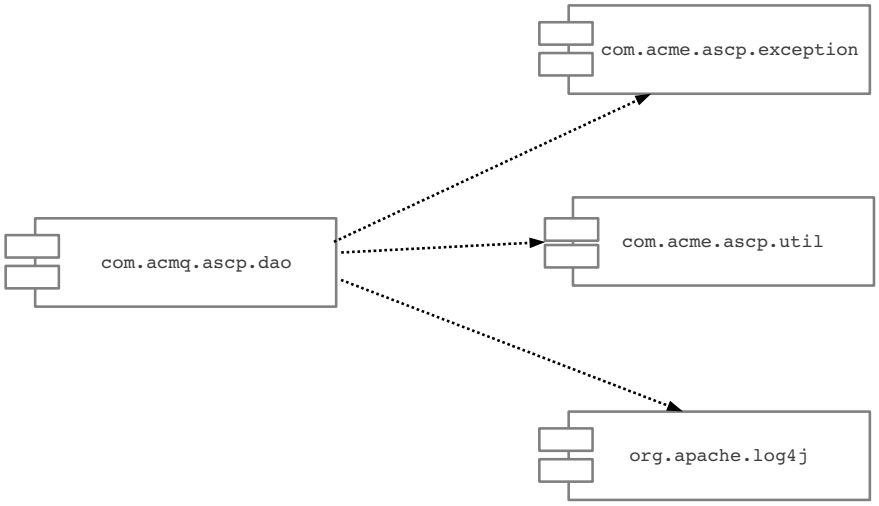

Figura 10: Efferent Coupling

L'accoppiamento fa riferimento ai legami esistenti tra unità (classi) separate in un programma. Se due classi dipendono strettamente l'una dall'altra, si dicono fortemente accoppiate (strong coupling). L'obiettivo è quello di minimizzare l'accoppiamento (loose coupling), agevolando così la manutenibilità e la facilità di comprensione del software$_{|g|}$, dato che non è necessario preoccuparsi di reperire le informazioni sulle altre classi associate. Seguendo questo principio, eventuali modifiche apportate ad una classe avranno poche o nessuna ripercussione sulle altre classi che dipendono da lei.

Architetturali

- *VG (McCabe Cyclomatic Complexity)*: complessità ciclomatica di McCabe. Misura il numero di cammini in un pezzo di codice. Ad ogni occorrenza di un branch, il numero incrementa di uno. I costrutti che incrementano la complessità ciclomatica sono: **if, else, while, do...while, switch** (+ 1 per ogni caso), **for, foreach, catch**. La formula utilizzata per calcolare la complessità ciclomatica è mostrata nell'equazione (n) ed è definita in riferimento ad un grafo contenente i blocchi base di un programma, con un arco tra due blocchi se il controllo può passare da uno all'altro:

$$V(G) = e - n + 2p \tag{1}$$

dove V(G) = complessità ciclomatica di G

e = numero of archi del grafo

n = numero di nodi del grafo

p = numero di componenti connesse

Verrà adottata la seguente tabella di riferimento per la complessità:
L'obiettivo del gruppo sarà quello di mantenere una complessità ciclomatica $\leqslant$

Complessità ciclomatica	Tipo di procedura	Rischio
1-4	Semplice	Basso
5-10	Ben strutturata e stabile	Medio Basso
11-20	Complessa	Moderato
21-50	Molto complessa, preoccupante	Alto
>50	Soggetta a errori, problematiche, instabile	Molto alto

Tabella 6: Stima del costo e delle ore totali.

5;

- *LCOM (Lack of Cohesion Of Methods)*: mancanza di coesione dei metodi. La coesione di una classe è influenzata da quanto i suoi metodi sono collegati alle sue variabili d'istanza. In caso di perfetta coesione (tutti i metodi accedono a tutti gli attributi) il valore sarà 0, mentre in caso di totale assenza di coesione (ogni metodo accede a una singola variabile) il valore sarà 1 e sarà opportuno dividere la classe.

$$LCOM = \frac{\frac{1}{n}(\sum_{i=0}^{m} m(A)_i) - m}{1 - m} \quad (2)$$

  Sia m(A) il numero di metodi che accedono ad un particolare attributo. Si calcola la media per tutti gli n attributi, si sottrae il numero m di metodi e si divide per 1 - m. Attraverso la coesione si è in grado di stabilire quali e quanti siano i compiti per i quali una classe (o metodo) è stata disegnata. Più una classe ha una responsabilità ristretta a un solo compito, più si dice che è coesa. L'obiettivo da prefiggersi è quello di avere una coesione elevata: ciò semplifica la comprensione di una classe (o metodo), ne semplifica la nomenclatura e ne favorisce il riutilizzo;

- *RMD (Normalized Distance)*: il suo numero dovrebbe essere prossimo allo zero per indicare un buon design dei package$_{|g|}$. E' data da

$$|RMA + RMI - 1|; \quad (3)$$

- *RMA (Abstractness)*: numero di classi astratte e interfacce diviso il numero totale di classi in un particolare scope$_{|g|}$. 0 indica un package$_{|g|}$ interamente concreto, mentre 1 indica un package$_{|g|}$ interamente astratto. Classi che hanno elevati valori di CA e CE dovrebbero essere più astratte allo scopo di mascherare i dettagli implementativi e facilitare eventuali modifiche al codice;

- *RMI (Instability)*:
$$\frac{CE}{CA+CE} \tag{4}$$
0 indica un package$_{|g|}$ completamente stabile, mentre 1 indica un package$_{|g|}$ completamente instabile. Da questa metrica si evince come l'efferent coupling vada contro la stabilità del package$_{|g|}$: più un package$_{|g|}$ fa affidamento sugli altri, più è vulnerabile rispetto ai cambiamenti degli altri package$_{|g|}$;
- *NORM (Number of Overridden Methods)*: numero di metodi che sono overriding$_{|g|}$ di metodi presenti nelle classi superiori, in un particolare scope$_{|g|}$;
- *DIT (Depth of Inheritance Tree)*: profondità dell'albero di ereditarietà, intesa come la lunghezza massima dal nodo alla radice. Una profondità bassa può indicare una complessità inferiore, ma anche la possibilità che non ci sia abbastanza riuso del codice tramite l'ereditarietà. Dualmente, una profondità alta aumenta il potenziale riuso del codice, ma introduce maggiore complessità e possibili errori. Anche se non esistono valori standard per la profondità dell'albero di ereditarietà, assumiamo come limite superiore un valore pari a 6;
- *SIX (Specialization IndeX)*: indice di specializzazione, definito come
$$\frac{NORM * DIT}{NOM} \tag{5}$$
Si riferisce alla quantità di uso dell'ereditarietà; SIX fornisce un'indicazione approssimata del grado di specializzazione per ogni sottoclasse in un software$_{|g|}$ Object Oriented;
- *WMC (Weighted Methods per Class)*: somma della complessità ciclomatica per tutti i metodi di una classe.

6 Resoconto attività di verifica

In questa sezione vengono descritte le procedure adottate durante il processo di verifica e i risultati ottenuti.

6.1 Dettaglio delle verifiche tramite analisi

6.1.1 Analisi dei requisiti

Le tabelle presenti nel documento *AnalisiDeiRequisiti_v4.0.pdf* evidenziano i seguenti requisiti:

- **totali**: 141
- **Funzionali Utente**: 90, di cui 55 obbligatori, 5 desiderabili, 30 opzionali;
- **Funzionali Amministratore**: 30, di cui 27 obbligatori, 3 opzionali;
- **di Vincolo**: 18, di cui 11 obbligatori, 2 desiderabili, 5 opzionali;
- **di Qualità**: 3, di cui 1 obbligatorio e 2 desiderabili.

Gli use case$_{|g|}$ individuati sono 119, di cui 89 lato utente e 30 lato amministratore. È stato poi verificato che tutti i bisogni emersi siano stati coperti da requisiti, e che tutti i requisiti siano coperti dagli use case$_{|g|}$.

6.1.2 Progettazione ad alto livello

Nella fase di progettazione si verifica, tramite tracciamento, che ad ogni requisito funzionale espresso nell'analisi dei requisiti corrisponda una componente architetturale e viceversa, garantendo il pieno soddisfacimento dei requisiti stessi.
Per consultare l'esito del tracciamento componenti - requisiti effettuato si rimanda al documento *SpecificaTecnica_v3.0.pdf*.

6.2 Dettaglio della verifiche tramite i test

Si rimanda all'appendice B (Attività di test) per le misurazioni e i test svolti con i rispettivi risultati.

6.3 Dettaglio dell'esito delle revisioni

Le revisioni di progetto sono le scadenze alle quali il gruppo dovrà presentare, di volta in volta, la documentazione necessaria prodotta al fine di coordinare tutte le risorse appartenenti al gruppo e procedere con lo sviluppo del progetto. Il numero totale di revisioni per l'anno accademico 2012/2013 è 4:

- Revisione dei Requisiti
- Revisione di Progetto
- Revisione di Qualifica
- Revisione di Accettazione

Ad ogni revisione il gruppo sottoporrà al *Committente* la documentazione elaborata accompagnata da una breve presentazione. *GoGo Team* si propone di partecipare alla revisione di qualifica del 2013-06-11, come specificato nel documento.

6.3.1 Revisione dei requisiti

In seguito alle osservazioni effettuate dal committente in sede di RR, il gruppo *GoGo Team* si è riunito per analizzare gli esiti e correggere gli errori rilevati con lo scopo di presentare una versione migliorativa dei vari documenti.
Le correzioni svolte sono:

- **Su tutti i documenti:**
  - applicata una forma tabulare al registro delle modifiche;
  - aggiunto lo storico delle revisioni.
- **Analisi dei requisiti:**
  - revisione dei casi d'uso dell'ambito utente con conseguenti modifiche alla struttura e con conseguente aggiunta di essi;
  - revisione dei casi d'uso dell'ambito amministratore con conseguenti modifiche alla struttura e con conseguente aggiunta di essi;
  - aggiornamento dei requisiti e del relativo tracciamento in base alle modifiche e alle aggiunte effettuate ai casi d'uso;
  - inserito il tracciamento tra requisiti e casi d'uso.
- **Piano di qualifica:**
  - riorganizzazione dell'indice;
  - spostamento del capitolo 3 in appendice;
  - spostamento del capitolo 2.4 in *NormeDiProgetto_v4.0.pdf*.
- **Piano di progetto:**
  - Il documento è stato integrato con i contenuti richiesti
  - correzione di alcune terminologie utilizzate;
  - è stata rivista l'assegnazione delle ore giornaliere di lavoro che ogni membro del gruppo dovrà dedicare al progetto;
  - applicata una forma tabulare all'analisi dei rischi.
- **Norme di progetto:**
  - il documento è stato integrato con i contenuti richiesti (norme relative alla progettazione, procedure per la gestione dei cambiamenti, garanzia dell'assenza di conflitto di interessi nell'attività di verifica);
  - il capitolo riguardante il versionamento è stato modificato;
  - sono state incorporate nel documento le specifiche degli strumenti e le procedure di utilizzo che erano state precedentemente inserite nella versione 1.0 del documento. *PianoDiQualifica_v3.0.pdf*.

- **Studio di fattibilità:** approfondita l'analisi della criticità;
- **Glossario:** correzione di alcune terminologie utilizzate.

6.3.2 Revisione di progettazione ad alto livello

In seguito alle osservazioni effettuate dal committente in sede di RP, il gruppo *GoGo Team* si è riunito per analizzare gli esiti e correggere gli errori rilevati con lo scopo di presentare una versione migliorativa dei vari documenti.
Le correzioni svolte sono:

- **Analisi dei requisiti:**
  - modifica dell'immagine della gerarchia utente (paragrafo 3.1);
  - aggiunta di scenari alternativi ad alcuni casi d'uso;
  - modifica dei requisiti FOB0 e QOB1.
- **Piano di qualifica:**
  - rimozione del modello a V nel capitolo 2, non compatibile con il modello di ciclo di vita adottato;
  - estensione dei capitoli 2 (sottosezione "Pianificazione Strategica e Generale") e 4;
  - spostamento nel documento NormeDiProgetto_v4.0.pdf delle risorse software del capitolo 3;
  - estensione capitolo in appendice: Obiettivi di qualità, fissato metriche e valori attesi.
- **Piano di progetto:**
  - ripartizione ore per ruolo;
  - aggiunta preventivo a finire;
  - motivazione spostamento RQ.
- **Norme di progetto:**
  - definizione e stesura delle norme relativa ad ogni singola attività;
  - riorganizzazione del documento;
  - aggiornamento degli strumenti utilizzati.
- **Specifica tecnica:**
  - Aggiunta dei diagrammi dei package;
  - Aggiunta del DAO;
  - Aggiunta e modifica delle classi interne;
  - Spostamento di alcuni package.
- **Glossario:** Nessuna correzione specifica da operare nel documento, che è stato comunque ampliato con qualche nuova definizione.

6.3.3 Revisione di qualifica

In seguito alle osservazioni effettuate dal committente in sede di RQ, il gruppo *GoGo Team* si è riunito per analizzare gli esiti e correggere gli errori rilevati con lo scopo di presentare una versione migliorativa dei vari documenti.
Le correzioni svolte sono:

- **Analisi dei requisiti:**
  - modifica dell'immagine della gerarchia utente (paragrafo 3.1);
  - eliminazione del requisito FOB0.
- **Piano di qualifica:** Nessuna correzione specifica da operare nel documento.
- **Piano di progetto:**
  - revisione e aggiornamento del preventivo a finire, ora consuntivo.
- **Norme di progetto:**
  - modificata la struttura del documento
- **Specifica tecnica:**
  - modifica dell'architettura per eliminare le dipendenze circolari;
  - aggiunto il design pattern DAO e modificata la descrizione del design pattern MVP.
- **Definizione di prodotto:**
  - aggiornate le nuove classi e i relativi metodi;
  - modifica dei diagrammi delle classi.
- **Manuale Amministratore:**
  - rivista la descrizione dello scopo del documento;
  - aggiunto l'oggetto predefinito della mail;
  - assegnato un codice di errore ad ogni errore.
- **Manuale installazione:**
  - riviste le istruzioni relative all'esecuzione di Tomcat;
  - inserimento Javadoc.
- **Manuale utente:**
  - rivista la descrizione dello scopo del documento;
  - aggiunto l'oggetto predefinito della mail;
  - assegnato un codice di errore ad ogni errore;
  - rivisti i riferimenti alle figure;
  - riviste le descrizioni relative al primo accesso all'applicazione e alle funzionalità di chiamata e ricezione.
- **Glossario:** Nessuna correzione specifica da operare nel documento.

6.4 Revisione documentazione

È stata effettuata, come prima verifica, un'attività di walkthrough su tutta la documentazione, che ha permesso di individuare errori grammaticali e di sintassi. Tali errori sono stati comunicati dal Verificatore al Redattore del documento, il quale ha poi provveduto alla loro correzione. Sono stati inoltre rivisti e corretti i riferimenti interni ed esterni ai documenti scritti in LaTeXche non erano funzionanti e mostravano i doppi punti di domanda (??).
È stato infine svolto un controllo ortografico su tutti i file `*.tex` prodotti tramite Aspell (eseguendo il comando riportato in *NormeDiProgetto_v4.0.pdf*), che ha rilevato numerosi errori di battitura sfuggiti al Verificatore dopo una prima lettura completa.

Appendici

A Obiettivi di qualità

In questa appendice vengono illustrati gli standard che *GoGo Team* adotta come riferimenti: tali standard sono ISO/IEC$_{|g|}$ 15504 per la qualità di processo e ISO/IEC$_{|g|}$ 9126 per la qualità di prodotto.

A.1 Qualità dei processi

La qualità del processo viene vista come un'esigenza, e la norma ISO/IEC$_{|g|}$ 15504 (che ingloba SPICE, Software Process Improvement Capability dEtermination) definisce un modello per la valutazione dei processi in un'organizzazione del settore IT (Information Technology), denominato SPY (SW Process Assessment & Improvement, vedi figura 11).

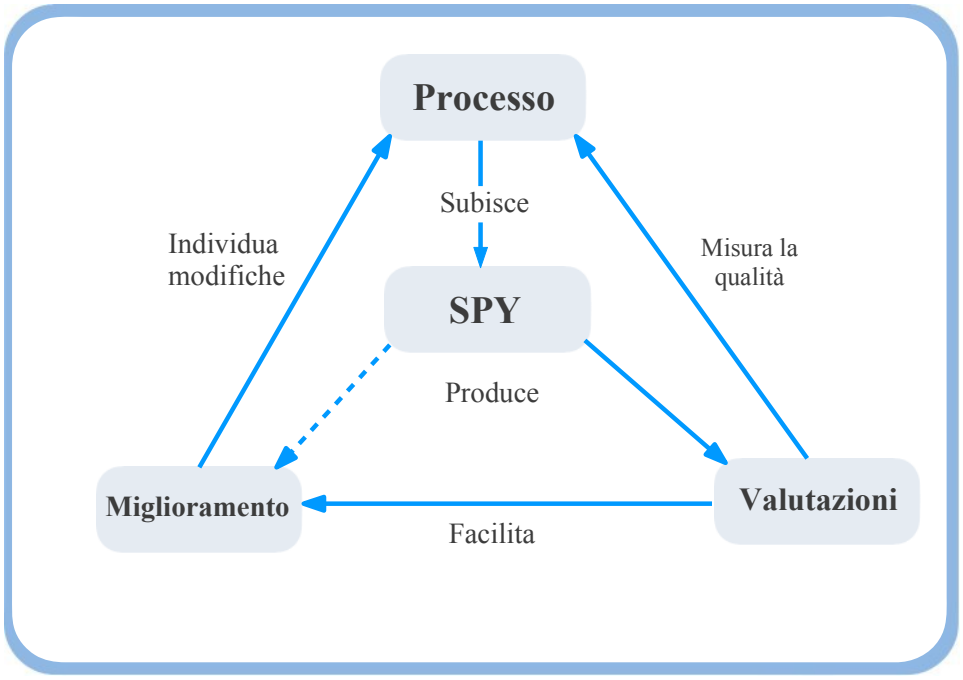

Figura 11: Modello SPY

Tale standard definisce nove attributi di qualità:

1. **Process performance**: un processo raggiunge i suoi obiettivi, trasformando input identificabili in output identificabili.
2. **Performance management**: l'attuazione di un processo è pianificata e controllata al fine di produrre risultati che rispondano agli obiettivi attesi.
3. **Work product management**: l'attuazione di un processo è pianificata e controllata al fine di produrre risultati che siano propriamente documentati, controllati e verificati.

4. **Process definition**: l'attuazione di un processo si basa su approcci standardizzati.
5. **Process resource**: il processo può contare sulle risorse adeguate (umane, infrastrutture, etc.) per essere attuato.
6. **Process measurement**: i risultati raggiunti e le misure rilevate durante l'attuazione di un processo sono stati usati per assicurarsi che l'attuazione di tale processo supporti efficacemente il raggiungimento di specifici obiettivi.
7. **Process control**: un processo è controllato attraverso la raccolta, analisi ed utilizzo delle misure di prodotto e di processo rilevate, al fine di correggere, se necessario, le sue modalità di attuazione.
8. **Process change**: le modifiche alla definizione, gestione, attuazione di un processo sono controllate.
9. **Continuous improvement**: le modifiche ad un processo sono identificate ed implementate al fine di assicurare il continuo miglioramento nel raggiungimento degli obiettivi rilevanti per l'organizzazione.

La norma definisce poi quattro livelli di possesso di un attributo:

- **N** – non posseduto (0-15% di possesso): non c'è evidenza, o ve ne è poca, del possesso di un attributo;
- **P** – parzialmente posseduto (16-50% di possesso): vi è evidenza di approccio sistematico al raggiungimento del possesso di un attributo e del raggiungimento di tale possesso, ma alcuni aspetti del possesso possono essere non prevedibili;
- **L** – largamente posseduto (51-85% di possesso): vi è evidenza di approccio sistematico al raggiungimento del possesso di un attributo e di un significativo livello di possesso di tale attributo, ma l'attuazione del processo può variare nelle diverse unità operative della organizzazione;
- **F** – (Fully) pienamente posseduto (86-100% di possesso): vi è evidenza di un completo e sistematico approccio e di un pieno raggiungimento del possesso dell'attributo e non esistono significative differenze nel modo di attuare il processo tra le diverse unità operative.

Vi sono infine cinque livelli di maturità dei processi:

- **Livello 1 – processo semplicemente attuato**: il processo viene messo in atto e raggiunge i suoi obiettivi. Non vi è evidenza di un approccio sistematico ad alcuno degli attributi definiti. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "process performance".
- **Livello 2 – processo gestito**: il processo è attuato ma anche pianificato, tracciato, verificato ed aggiustato se necessario, sulla base di obiettivi ben definiti. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Performance management" e "Work product management".

- **Livello 3 – processo definito**: il processo è attuato, pianificato e controllato sulla base di procedure ben definite, basate sui principi del software engineering$_{|g|}$. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process definition" e "Process resource".

- **Livello 4 – processo predicibile**: il processo è stabilizzato ed è attuato all'interno di definiti limiti riguardo i risultati attesi, le performance, le risorse impiegate, etc. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process measurement" e "Process control".

- **Livello 5 – processo ottimizzante**: il processo è predicibile ed in grado di adattarsi per raggiungere obiettivi specifici e rilevanti per l'organizzazione. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process change" e "Continuous improvement".

I vantaggi derivanti dall'applicazione di questo standard sono molteplici: per l'azienda, permette di sfruttare conoscenza proveniente da best practice$_{|g|}$, facilita il passaggio di consegne da una persona all'altra, porta a riduzione dei costi, migliori tempi di consegna, maggiore qualità del prodotto, ritorno degli investimenti e soddisfazione del cliente. Per il cliente, invece, vi è una maggiore percezione della qualità delle aziende sulla base del livello CMMI$_{|g|}$ raggiunto.

A.2 Qualità del prodotto

Si è scelto di seguire lo standard `ISO/IEC`$_{|g|}$ `9126`, che definisce la qualità del prodotto software$_{|g|}$ come l'insieme delle caratteristiche che incidono sulla capacità del prodotto di soddisfare requisiti espliciti o impliciti.
Lo standard `ISO/IEC`$_{|g|}$ `9126` distingue poi tre tipi di qualità:

- qualità interna: rappresenta le qualità intrinseche del prodotto, cioè quelle misurabili direttamente a partire dal codice sorgente. Esse scaturiscono dai requisiti utente e dalla specifica tecnica;

- qualità esterna: è rappresentata dalle prestazioni del prodotto e dalle funzionalità che offre: riguarda quindi il comportamento dinamico del software$_{|g|}$. Essa deriva dai requisiti utente;

- qualità in uso: è rappresentata dal livello con cui il software$_{|g|}$ si rende utile all'utente, cioè la soddisfazione che deriva dal suo utilizzo e il grado di efficienza/efficacia che fornisce.

Vengono individuate sei caratteristiche indicatrici di qualità del prodotto software$_{|g|}$, ciascuna delle quali suddivisa in sottocaratteristiche.

A.2.1 Funzionalità

È la capacità di fornire servizi in grado di soddisfare, in determinate condizioni, requisiti funzionali espliciti o impliciti. Le sue sottocaratteristiche sono:

- **adeguatezza**: presenza di funzioni appropriate per compiti specifici che supportano gli obiettivi dell'utente;

- **accuratezza**: capacità di fornire risultati corretti, in accordo con i requisiti dati dall'utente;
- **interoperabilità**: capacità di interagire con altri sistemi;
- **sicurezza**: capacità di proteggere programmi e dati da accessi non autorizzati e consentire quelli autorizzati.

Nel Capitolato d'appalto non viene richiesto alcun livello di sicurezza, dato che l'obiettivo è quello di capire il funzionamento e le potenzialità delle librerie WebRTC$_{|g|}$. La metrica, i valori attesi e gli strumenti adottati sono i seguenti:

- **metrica**: percentuale di requisiti soddisfatti;
- **valore atteso**: l'obiettivo minimo è il soddisfacimento di tutti i requisiti obbligatori;
- **strumenti**: per garantire che il prodotto possieda tutte le funzionalità richieste sarà necessario il superamento di tutti i test effettuati con JUnit. Per quanto riguarda l'interfaccia, il suo funzionamento verrà garantito dalla verifica tramite normale utilizzo.

A.2.2 Affidabilità

È la capacità di mantenere le prestazioni stabilite nelle condizioni e nei tempi fissati. Le sue sottocaratteristiche sono:

- **Maturità (robustezza)**: capacità di evitare fallimenti dell'applicazione (*failure*) a seguito di malfunzionamenti (*fault*);
- **Tolleranza errori**: capacità di mantenere determinati livelli di prestazione in caso di malfunzionamenti;
- **Recuperabilità**: capacità e velocità, in caso di malfunzionamento, di ripristinare i livelli di prestazione predeterminati e di recuperare dati.

La metrica, i valori attesi e gli strumenti adottati sono i seguenti:

- **metrica**: percentuale di test di sistema soddisfatti;
- **valore atteso**: il superamento di tutti i test di sistema;
- **strumenti**: Chrome Driver / test manuali.

A.2.3 Usabilità

È la capacità di essere compreso, appreso e usato con soddisfazione dall'utente in determinate condizioni d'uso. Le sue sottocaratteristiche sono:

- **Comprensibilità**: capacità di ridurre l'impegno richiesto agli utenti per capirne il funzionamento e le modalità di utilizzo;
- **Apprendibilità**: capacità di ridurre l'impegno richiesto agli utenti per imparare ad usarlo;

- **Operabilità**: capacità di mettere in condizione gli utenti di farne uso per i propri scopi;
- **Attrattività/Piacevolezza**: capacità di essere piacevole per l'utente che ne fa uso.

La metrica, i valori attesi e gli strumenti adottati sono i seguenti:

- **metrica**: non è possibile stabilire una metrica per questo aspetto;
- **valore atteso**: un utente con competenze di utilizzo del computer di base deve essere in grado di utilizzare il software$_{|g|}$;
- **strumenti**: prova da parte di un utente non informatico.

A.2.4 Efficienza

È il rapporto fra prestazioni e quantità di risorse utilizzate, in condizioni definite di funzionamento. Le sue sottocaratteristiche sono:

- **comportamento rispetto al tempo**: tempi di risposta, tempi di elaborazione e throughput rates$_{|g|}$ adeguati per eseguire le funzioni richieste, sotto determinate condizioni;
- **uso di risorse**: utilizzo di una quantità e di una tipologia di risorse adeguate per eseguire le funzioni richieste, sotto determinate condizioni.

La metrica, i valori attesi e gli strumenti adottati sono i seguenti:

- **metrica**: il tempo di risposta dell'applicazione, la latenza$_{|g|}$, il tempo necessario alla connessione;
- **valore atteso**: data la novità del tipo di applicazione da sviluppare, al momento non è possibile stabilire un valore atteso. L'applicazione deve comunque rispondere in tempi ragionevoli;
- **strumenti**: il metodo `getStats`, che calcola le statistiche relative alla connessione delle librerie WebRTC$_{|g|}$.

A.2.5 Manutenibilità

È la capacità di essere modificato con un impegno contenuto. Le sue sottocaratteristiche correlate sono:

- **analizzabilità**: capacità di limitare l'impegno richiesto per diagnosticare carenze o cause di malfunzionamenti, o per identificare parti da modificare;
- **modificabilità**: capacità di limitare l'impegno richiesto per modificare il programma, rimuovere errori o sostituire componenti;
- **stabilità**: capacità di ridurre il rischio di comportamenti inaspettati a seguito dell'effettuazione di modifiche;

- **testabilità**: capacità di essere facilmente testato per validare le modifiche apportate.

La metrica, i valori attesi e gli strumenti adottati sono i seguenti:

- **metrica**: i punti di riferimento sono quelli descritti nella sezione Metriche (cap. 5);
- **valori attesi**: rispetto dei valori indicati nella sezione Metriche (cap. 5) per le singole metriche;
- **strumenti**: il plugin$_{|g|}$ Metrics per Eclipse.

A.2.6 Portabilità

È la facilità con cui il software$_{|g|}$ può essere trasferito da un ambiente operativo ad un altro. Le sue sottocaratteristiche sono:

- **adattabilità**: capacità di adattarsi a nuovi ambienti operativi limitando la necessità di apportare modifiche;
- **installabilità**: capacità di ridurre l'impegno richiesto per installarlo in un particolare ambiente operativo;
- **coesistenza**: capacità di coesistere con altri software$_{|g|}$ nel medesimo ambiente, condividendo risorse;
- **sostituibilità**: capacità di essere utilizzato al posto di un altro software$_{|g|}$ per svolgere gli stessi compiti nello stesso ambiente.

Il requisito obbligatorio da soddisfare per il prodotto **MyTalk** è il funzionamento sul browser$_{|g|}$ Google Chrome$_{|g|}$. La portabilità su piattaforme diverse dipende quindi dall'implementazione delle librerie WebRTC$_{|g|}$ e dalla capacità degli altri browser$_{|g|}$ di supportarle.

B Attività di test

Questi test rientrano nell'analisi dinamica è quindi necessario testare ogni singola componente, sia semplice che complessa, in quanto ogni singola unità funzionante non assicura l'integrazione tra esse. I test devono possedere due fondamentali caratteristiche:

- **Determinismo**: a priori bisogna sapere se il risultato ottenuto dal test è giusto o sbagliato;
- **Ripetibilità**: eseguendo più volte lo stesso test sullo stesso codice, il risultato non deve variare.

Per garantire queste proprietà dei test risulta obbligatoria l'automatizzazione dei processi di verifica del codice prodotto.

B.1 Test di sistema

La presente sezione contiene l'insieme dei test che si prevede verranno attuati prima del rilascio del prodotto finito. L'esecuzione dei test deve garantire il rispetto dei requisiti (allegato *AnalisiDeiRequisiti_v4.0.pdf*).
La copertura del codice deve essere garantita con una percentuale $\geqslant 60$.
I test che verranno riportati sono quelli relativi alle funzionalità del prodotto. Il codice del test riporta anche la chiave del requisito a cui il test si riferisce.

B.1.1 Ambito utente

Codice requisito	Codice verifica	Modalità di verifica	Stato della verifica
FOB 0	TS - FOB 0	Prova dinamica: il server$_{\lvert g \rvert}$ su cui risiede il sistema dovrà dimostrarsi funzionante rispondendo alle richieste dei vari componenti.	Pianificata
FOB 1	TS - FOB 1	Prova dinamica: viene verificato se le credenziali inserite dall'utente test risultano essere valide.	Pianificata
FOB 2	TS - FOB 2	Prova dinamica: viene verificato se il sistema accetta positivamente l'inserimento dei dati da parte dell'utente test.	Pianificata
FOB 2.1.2	TS - FOB 2.1.2	Prova dinamica: viene verificato se lo username inserito dall'utente test è univoco o se è già utilizzato da altri utenti.	Pianificata
FOB 2.2.1	TS - FOB 2.2.1	Prova dinamica: viene verificato se la password è formata da almeno 8 caratteri alfanumerici.	Pianificata
FOB 2.2.3	TS - FOB 2.2.3	Prova dinamica: viene verificato che l'utente test abbia inserito una conferma della password e che la password e la sua conferma siano identiche.	Pianificata
FOB 2.6	TS - FOB 2.6	Prova dinamica: viene verificato che il numero di telefono sia completamente numerico e formato da massimo 11 cifre.	Pianificata
FOB 3.1	TS - FOB 3.1a	Prova dinamica: viene verificato se dopo aver effettuato l'autenticazione di un utente test, questo è effettivamente autorizzato a visualizzare i propri dati personali.	Pianificata
FOB 3.1	TS - FOB 3.1b	Prova dinamica: viene verificato se dopo aver effettuato l'autenticazione di un utente test, questo riesce a visualizzare i propri dati personali in modo corretto.	Pianificata
FOB 3.2	TS - FOB 3.2	Prova dinamica: viene verificato se dopo aver effettuato l'autenticazione di un utente test, questo è effettivamente autorizzato a modificare i propri dati personali.	Pianificata

Codice requisito	Codice verifica	Modalità di verifica	Stato della verifica
FOB 4.1.1	TS - FOB 4.1.1	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a ricevere chiamate da altri utenti.	Pianificata
FOB 4.1.1.1.1	TS - FOB 4.1.1.1.1	Prova dinamica: viene verificato che il sistema durante una chiamata permetta lo scambio del segnale audio e video.	Pianificata
FOB 4.1.2	TS - FOB 4.1.2	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a rifiutare chiamate in entrata.	Pianificata
FOB 4.2.1	TS - FOB 4.2.1	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a chiamare un singolo utente.	Pianificata
FOB 4.2.1.1	TS - FOB 4.2.1.1a	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a chiamare un altro utente conoscendone lo username.	Pianificata
FOB 4.2.1.1	TS - FOB 4.2.1.1b	Prova dinamica: viene verificato se il sistema accetta positivamente l'inserimento dello username da parte dell'utente test.	Pianificata
FOB 4.2.1.2	TS - FOB 4.2.1.2a	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a chiamare un altro utente conoscendone l'indirizzo $IP_{\|g\|}$.	Pianificata
FOB 4.2.1.2	TS - FOB 4.2.1.2b	Prova dinamica: viene verificato se il sistema accetta positivamente l'inserimento dell'indirizzo $IP_{\|g\|}$ da parte dell'utente test.	Pianificata
FOB 4.2.1.3	TS - FOB 4.2.1.3	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a chiamare un altro utente selezionandolo dalla lista.	Pianificata
FOB 6	TS - FOB 6	Prova dinamica: viene verificato se dopo l'autenticazione di un utente test, questo riesce effettivamente a uscire dalla sessione autenticata.	Pianificata

Tabella 7: Test relativi all'utente

B.1.2 Ambito amministratore

Codice requisito	Codice verifica	Modalità di verifica	Stato della verifica
FAOB 2.1	TS - FAOB 2.1	Prova dinamica: viene verificato se l'amministratore autenticato visualizza le chiamate effettuate dagli utenti nella settimana.	Pianificata
FAOB 2.1.1	TS - FAOB 2.1.1	Prova dinamica: viene verificato se l'amministratore autenticato può filtrare le chiamate in base al giorno di effettuazione.	Pianificata
FAOB 2.1.2	TS - FAOB 2.1.2	Prova dinamica: viene verificato se l'amministratore autenticato può filtrare le chiamate in base al giudizio espresso dagli utenti.	Pianificata
FAOB 2.2	TS - FAOB 2.2	Prova dinamica: viene verificato se l'amministratore autenticato può visualizzare le statistiche sulle chiamate.	Pianificata
FAOB 2.2.2	TS - FAOB 2.2.2	Prova dinamica: viene verificato se l'amministratore autenticato può visualizzare il numero di pacchetti trasmessi.	Pianificata
FAOB 2.2.3	TS - FAOB 2.2.3	Prova dinamica: viene verificato se l'amministratore autenticato può visualizzare il numero di byte trasmessi.	Pianificata
FAOB 2.2.4	TS - FAOB 2.2.4	Prova dinamica: viene verificato se l'amministratore autenticato può visualizzare il giudizio della chiamata.	Pianificata
FAOB 2.2.5	TS - FAOB 2.2.5	Prova dinamica: viene verificato se l'amministratore autenticato può visualizzare il numero di pacchetti persi.	Pianificata

Tabella 8: Test relativi all'amministratore

B.1.3 Requisiti di vincolo

Codice requisito	Codice verifica	Modalità di verifica	Stato della verifica
VOB 1	TS - VOB 1	Prova dinamica: viene verificato tramite l'utilizzo del browser$_{\|g\|}$ Chrome versione 27 o successiva se il sistema è funzionante tramite interfaccia web$_{\|g\|}$.	Pianificata
VOB 3	TS - VOB 3	Prova dinamica: viene verificato se, al primo accesso, il sistema richiede l'installazione di plugin$_{\|g\|}$ diversi dalle librerie WebRTC$_{\|g\|}$.	Pianificata
VOB 7	TS - VOB 7	Prova dinamica: viene verificato tramite l'utilizzo di un browser$_{\|g\|}$ se il sistema è effettivamente contenuto in un'unica pagina web$_{\|g\|}$.	Pianificata
VOB 9	TS - VOB 9	Prova dinamica: viene verificato se il progetto è effettivamente stato pubblicato sul repository$_{\|g\|}$ SourceForge.	Pianificata
VOB 10	TS - VOB 10	Prova dinamica: viene verificato se il server$_{\|g\|}$ su cui risiede il sistema è funzionante, quindi viene verificato se il server$_{\|g\|}$ risponde alle richieste degli utenti.	Pianificata

Tabella 9: Requisiti di vincolo

B.2 Test di integrazione

Di seguito, per ogni componente descritta nel documento *SpecificaTecnica_v3.0.pdf*, sono riportate le funzioni che il gruppo ha reputato necessario testare ai fini di garantire il corretto funzionamento dell'applicazione una volta rilasciata.

B.2.1 Ambito utente

Componente	Codice verifica	Test di integrazione
Login	TI - Login	• Autenticazione se le credenziali sono corrette • Opportuno avviso con credenziali errate • Uscita dalla sezione
Gestione dati	TI - gestione dati	• L'utente può visualizzare i propri dati • L'utente può scegliere un eventuale dato da modificare • L'utente può inserire il nuovo dato • Se il dato inserito è valido viene modificato (*AnalisiDeiRequisiti_v4.0.pdf*) • Il sistema avvisa l'utente se il dato è errato
Communication	TI - Communication	• L'utente può selezionare un utente da chiamare inserendone lo username. • L'utente può selezionare un utente da chiamare inserendone l'indirizzo IP. • Se l'utente indicato mediante inserimento dati non esiste si riceve un'opportuna comunicazione. • L'utente può selezionare un utente da chiamare da una lista di utenti registrati al server. • L'utente indicato, se esiste, viene chiamato. • Se l'utente selezionato accetta la chiamata il canale di comunicazione viene aperto. • Se l'utente indicato rifiuta la chiamata il canale di comunicazione non viene aperto e l'utente che ha chiamato riceve un opportuno avviso. • Se uno dei due utenti chiude la chiamata il canale di comunicazione viene interrotto. • Al termine della chiamata vengono visualizzate statistiche sull'uso. • Al termine della chiamata l'utente può esprimere un giudizio.

Tabella 10: Tabella dei requisiti di integrazione per l'utente.

B.2.2 Ambito amministratore

Componente	Codice verifica	Test di integrazione
Login	TI - Login	• Autenticazione se le credenziali sono corrette • Opportuno avviso se le credenziali sono errate • Uscita dalla sezione
Statistiche	TI - Statistic	• l'amministratore può visualizzare le statistiche di tutte le chiamate effettuate nell'ultima settimana.

Tabella 11: Tabella dei requisiti di integrazione per l'amministratore

C Test di Unità

Di seguito sono riportati gli esiti dei test di unità effettuati per verificare la correttezza logica dei singoli metodi. I package sottoposti a tale verifica sono quelli riguardanti il Presenter e il Model.
L'interfaccia grafica, invece, sarà sottoposta ai soli test di sistema poiché è risultato impossibile istanziare gli oggetti delle classi grafiche da testare all'interno del framework GWT, anche utilizzando la classe GWTTest, utilizzata per la verifica di alcune classi del Presenter.

C.1 Presenter

L'obiettivo di questi test è quello di verificare il corretto funzionamento della logica dei metodi delle classi che compongono il Presenter. Per l'esecuzione dei test sono state utilizzati i seguenti framework$_{|g|}$ di testing:

- JUnit 4: permette di scrivere ed eseguire in modo automatizzato i test case$_{|g|}$;
- Mockito 1.9.5: permette di simulare oggetti appartenenti a classi esterne a quella testata ed utilizzati da quest'ultima;
- GWT Test Utils 0.35: permette di accedere alle funzionalità offerte dal framework$_{|g|}$ Google Web Toolkit, altrimenti non utilizzabili tramite semplici test JUnit. In particolare, si vuole controllare la corretta gestione dei cookie$_{|g|}$.

C.1.1 Package mytalk.client.model.localDataUser

- **Classe** `mytalk.client.model.localDataUser.ManageCookies`

  Classe che realizza i test:
  `mytalk.client.model.localDataUser.ManageCookiesTest`

  Descrizione:
  Verifica la corretta creazione, lettura e distruzione dei cookie$_{|g|}$.

  Metodi di test:

  * `testCreateSessionCookie()`: viene creato il cookie$_{|g|}$ richiamando il metodo `createSessionCookie()` e ne viene verificata l'effettiva creazione richiamando il metodo della classe `Cookies` di `GWT` `getCookie()`.
  * `testGetCookieUsername()`: vengono creati due cookie$_{|g|}$, uno con contenuto vuoto e uno con la stringa `\username@email.it"`, e si verifica che il metodo `getCookieUsername()` ritorni il contenuto corretto. Viene infine testato il caso in cui il cookie$_{|g|}$ non esista, controllando che il valore di ritorno di `getCookieUsername()` sia `null`.
  * `testGetCookieIP()`: viene creato un cookie$_{|g|}$ con un indirizzo IP$_{|g|}$ come contenuto e si verifica che il metodo `getCookieIP()` ritorni l'indirizzo corretto.
  * `testDeleteCookies()`: viene creato un cookie$_{|g|}$ tramite il metodo `createSessionCookie(String name, String, content)` e ne viene verificata la corretta creazione utilizzando il metodo

`getCookie(String name)`, entrambi offerti dalla classe `Cookies` di `GWT`. Successivamente, viene richiamato il metodo `deleteCookies()` della classe `ManageCookies` per eliminare il cookie$_{|g|}$ creato e ne viene verificata l'effettiva eliminazione controllando che il metodo `getCookie(String name)` ritorni un riferimento nullo.

Stato: Superato

Bug individuati: 0

C.1.2 Package mytalk.client.presenter.user.logicUser

- **Classe** `mytalk.client.presenter.user.logicUser.DataUserLogic`

  Classe che realizza i test:
  `mytalk.client.presenter.user.logicUser.DataUserLogicTest`

  Descrizione:
  Viene controllato che la classe:

  * modifichi i dati nel nodo esatto;
  * restituisca opportuni messaggi di errore;
  * comunichi correttamente con le classi `WebSocket` e `UpdateViewLogic`.

  Prima di iniziare il test, vengono creati due vettori, uno con dati corretti e uno con dati sbagliati.

  Metodi di test:

  * `testCheckNewData()`: viene richiamato più volte il metodo da testare `checkNewData(Vector<String> input)` con diverse combinazioni, corrette ed errate, di dati nel vettore passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua. Ad ogni invocazione, viene controllato che il valore del campo dati `loginOk` corrisponda a quanto atteso.
  * `testLogoutUser()`: viene inizialmente creato un cookie$_{|g|}$ per simulare una sessione attiva, successivamente viene richiamato il metodo `logoutUser()` e si controlla che invochi correttamente il metodo `mytalk.client.presenter.client.user.serverComUser.logoutUser(String data)`, con la stringa uguale al contenuto del cookie$_{|g|}$. Si verifica poi l'invocazione dei metodi `mytalk.client.presenter.user.logicUser.removeCookies()` e `mytalk.client.presenter.user.logicUser.loadViewLogUser()`.
  * `testGetDataUser()`: viene verificata l'effettiva invocazione del metodo `mytalk.client.presenter.client.user.serverComUser.getUserData()`.

  Stato: Superato

  Bug individuati : 1

  Errore di battitura nel messaggio di errore ritornato all'utente nel metodo `checkStringData()`. Tale errore è stato corretto.

- **Classe** `mytalk.client.presenter.user.logicUser.LogUserLogic`

  Classe che realizza i test:
  `mytalk.client.presenter.user.logicUser.LogUserLogicTest`

  Descrizione:
  Viene controllato che la classe:

  * effettui un controllo corretto dei dati di login inseriti dall'utente;
  * restituisca i corretti messaggi di errore;
  * comunichi correttamente con le classi `mytalk.client.presenter.user.serverComUser.WebSocketUser` e `mytalk.client.presenter.user.logicUser.UpdateViewLogic`.

  Metodi di test:

  * `testValidateData()`: viene richiamato il metodo `validateData()` della classe `LogUserLogic` per tre volte: la prima con dei dati in input corretti, la seconda con la password sbagliata e la terza con l'indirizzo e-mail sbagliato. I dati in input sono contenuti in un `Vector` composto da due valori: nome utente (un indirizzo e-mail) e password (di almeno 8 caratteri per essere corretta). Viene verificato che il campo dati booleano `loginOk` della classe `LogUserLogic` contenga il valore appropriato e che venga richiamato il metodo `mytalk.client.presenter.user.serverComUser.authenticateUser(Vector<String> loginInput)` nel caso in cui i dati di login siano corretti, mentre nel caso di dati di login errati si verifica che venga richiamato il metodo `mytalk.client.presenter.user.logicUser.updateViewLogUser(boolean update, String message)`.

  Stato: Superato

  Bug individuati: 0

- **Classe** `mytalk.client.presenter.user.logicUser.UpdateViewLogic`

  Classe che realizza i test:
  `mytalk.client.presenter.user.logicUser.UpadateViewLogicTest`

  Descrizione:
  Si controlla che vengano richiamati correttamente i metodi della classe `PageUserView`.

  Metodi di test:

  * `testLoginResult()`: viene richiamato il metodo `loginResult boolean loginSuccess, Vector<String> loginInput`, prima con `loginSuccess` a `true` e poi a `false`, e si controlla che venga invocato correttamente il metodo `updateViewLogUser`. Nel caso della chiamata con `loginSuccess` a `true`, si controlla anche che venga creato il cookie$_{|g|}$ di sessione.

* `testCheckLoggedUser()`: viene creato un cookie$_{|g|}$ di sessione tramite il metodo `setCookie` della classe di GWT `Cookies` per simulare una sessione attiva, successivamente viene richiamato il metodo da testare `checkLoggedUser()` e si controlla che richiami il metodo `updateViewLogUser(boolean update, String message)` con i parametri corretti. Il cookie$_{|g|}$ viene poi rimosso e viene richiamato nuovamente il metodo da testare, che in caso di cookie$_{|g|}$ non esistente deve invece richiamare il metodo `loadViewLogUser()`.
* `testRegisterResult()`: viene richiamato due volte il metodo da testare `registerResult(boolean registerSuccess)`, la prima volta con argomento `true` e la seconda con argomento `false`. Nel caso di `registerSuccess == true`, deve essere richiamato il metodo `updateViewRegister(boolean registerSuccess, Vector<String> messages)` con l'argomento `messages` uguale ad un vettore vuoto, mentre nel caso di `registerSuccess == false` il metodo citato deve essere richiamato con l'argomento `messages` contenente, in posizione 0, il messaggio di errore "username già attivo".
* `testSetUsernameLabel()`: si verifica che il metodo da testare richiami il metodo `mytalk.client.view.user.setUsernameLabel(String username)` con la stringa corretta, ricavata dal cookie$_{|g|}$ di sessione appositamente creato per il test.
* `testResultDataUser()`: viene richiamato due volte il metodo da testare `resultDataUser(boolean update)`, la prima volta con argomento `true` e la seconda con argomento `false`. Nel caso di `update == true`, deve essere richiamato il metodo `updateViewDataUser(boolean update, Vector<String> messages)` con l'argomento `messages` uguale ad un vettore vuoto, mentre nel caso di `update == false` il metodo citato deve essere richiamato con l'argomento `messages` contenente, in posizione 5, il messaggio di errore "password errata".

I metodi:

* `testUpdateViewLogUser();`
* `testNotifyCall();`
* `testNotifyRefuseCall();`
* `testSetUserList();`
* `testLoadViewRegister();`
* `testLoadViewLogUser();`
* `testUpdateViewReigister();`
* `testLoadViewUserData();`
* `testSetLocalVideo();`
* `testLoadViewCommunication();`
* `testUpdateViewDataUser();`
* `testSetUserDataLabel();`
* `testCallActive();`

Invocano il corrispondente metodo da testare nella classe `UpdateViewLogic` e verificano, attraverso l'oggetto di mock `mockPageUserView`, che venga

richiamato il metodo corretto con i parametri corretti, ove questi siano richiesti.

Stato: Superato

Bug individuati: 1
Il metodo `checkLoggedUser()`: salva in una stringa il valore ritornato dal metodo `getCookieUsername()` della classe `ManageCookies`. Tale metodo, se il cookie$_{|g|}$ è inesistente, ritorna `null`. Veniva assunto erroneamente che ritornasse stringa vuota, quindi veniva effettuato il controllo sul contenuto della stringa tramite il metodo `equals()`, che richiamato su un riferimento nullo provoca `NullPointerException`. Tale problema è stato risolto.

- **Classe** `mytalk.client.presenter.user.logicUser.RegisterLogic`

  Classe che realizza i test:
  `mytalk.client.presenter.user.logicUser.RegisterLogicTest`

  Descrizione:
  Viene controllato che la classe:

  * effettui un controllo corretto dei dati di registrazione inseriti dall'utente;
  * restituisca i corretti messaggi di errore;
  * comunichi correttamente con le classi `WebSocketUser` e `UpdateViewLogic`;

  Metodi di test:

  * `testValidateData()`: viene richiamato il metodo `validateData()` della classe `RegisterLogic` per due volte, la prima con dei dati in input corretti e la seconda con dei dati non corretti. I dati in input sono contenuti in un `Vector` composto da due valori: nome utente (un indirizzo e-mail) e password (di almeno 8 caratteri per essere corretta). Viene verificato che il campo dati booleano `loginOk` della classe `RegisterLogic` contenga il valore appropriato.

  Stato: Superato

  Bug individuati: 0

- **Classe** `mytalk.client.presenter.user.logicUser.CommunicationLogic`

  Classe che realizza i test:
  mytalk.client.presenter.user.logicUser.CommunicationLogicTest

  Descrizione: si verifica che vengano chiamati correttamente gli opportuni metodi, con i valori desiderati, delle classi `mytalk.client.presenter.user.communication.PeerConnection`, `mytalk.client.presenter.user.communication.MediaStream`, `mytalk.client.presenter.user.logicUser.UpdateViewLogic` e `mytalk.client.presenter.user.serverComUser.WebSocketUser` quando viene invocato un metodo della classe `mytalk.client.presenter.user.logicUser.CommunicationLogic`.

  Metodi di test:

* `testReceivedSession()`: viene creato un `Vector<String>` contenente un certo numero di elementi e si verifica che il metodo da testare richiami, per ogni elemento del vettore, il metodo `mytalk.client.presenter.user.communication.PeerConnection.addCandidate(String candidate)`. Vengono inoltre impostati, tramite mock, i metodi `getCandidates()` e `getRemoteURL()` della classe `PeerConnection` per ritornare delle stringhe arbitrarie; queste stringhe devono essere passate, oltre al nome del mittente, rispettivamente ai metodi `mytalk.client.presenter.user.serverComUser.WebSocketUser.sendAnswer(String mittente, String iceCandidate)` e `mytalk.client.presenter.user.logicUser.UpdateViewLogic.callActive(String remoteVideo, String nome)`.
* `testReceivedSessionPerformed()`: viene verificata la corretta invocazione dei metodi `mytalk.client.presenter.user.communication.IPeerConnection.receivedCommunication(Vector<String> candidate)` e `mytalk.client.presenter.user.communication.IPeerConnection.startStat()`.
* `testLogoutUser()`: viene creato un cookie$_{|g|}$ di sessione tramite il metodo GWT `Cookies.setCookie(String name, String content)`, viene richiamato il metodo da testare e si controlla la corretta invocazione dei metodi della classe `mytalk.client.presenter.user.logicUser.UpdateViewLogic removeCookies()` e `loadViewLogUser()`.
* `testGetLocalURL()`: attraverso il mock `mockMediaStream` si imposta il comportamento del metodo `mytalk.client.presenter.user.communication.MediaStream.localURL()` per fargli ritornare una certa stringa; si verifica poi che venga invocato il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.setLocalVideo(String url)` con quella stringa.
* `testGetLocalURLnull()`: come il precedente, ma viene fatto ritornare il valore `null` al metodo `mytalk.client.presenter.user.communication.MediaStream.localURL()`. Si testa che non venga richiamato il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.setLocalVideo(String url)`.
* `testCall()`: attraverso il mock `mockPeerConnection` viene fatta ritornare una particolare stringa al metodo `mytalk.client.presenter.user.communication.PeerConnection.getOffer()`. Si verifica che quella stringa venga passata correttamente al metodo `mytalk.client.presenter.serverComUser.WebSocketUser call(String utente, String offer)` e si controlla la corretta invocazione dei metodi `initialize()` e `addStream(MediaStream stream)` della classe `PeerConnection`.
* `testSetUserList()`: si controlla la corretta invocazione del metodo `mytalk.client.presenter.serverComUser.WebSocketUser.setUserList()`.

* `testAccept()`: attraverso il mock `mockPeerConnection` viene fatta ritornare una particolare stringa al metodo `mytalk.client.presenter.user.communication.PeerConnection.getAnswer()`. Si verifica che quella stringa venga passata correttamente al metodo `mytalk.client.presenter.serverComUser.WebSocketUser accept(String utente, String answer)`.
* `testRefuse()`: si controlla la corretta invocazione del metodo `mytalk.client.presenter.serverComUser.WebSocketUser.refuse(String utente)`.
* `testSearch()`: viene invocato il metodo di test prima con la stringa ``Ip'' e poi con la stringa ``EMail'' e si verifica la corretta invocazione dei metodi `SearchUserByIP(String ip)` e `SearchUserByEmail(String user` della classe `mytalk.client.presenter.user.serverComUser.WebSocketUser`.
* `testCallEnter()`: si verifica la corretta invocazione del metodo `mytalk.client.presenter.user.communication.PeerConnection.answer(String descrizione, IMediaStream stream)`.
* `testGetStats()`: viene creato un `Vector<Double>` al quale vengono aggiunti tre valori. Attraverso il mock `mockPeerConnection`, si impostano i metodi `getBytesSent()`, `getPacketsSent()` e `getPacketsLost()` della classe `mytalk.client.presenter.user.communication.PeerConnection` per ritornare gli stessi valori contenuti nel `Vector` creato. Si verifica quindi che venga richiamato il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.updateFormInfoChiamata(Vector<String> stats)` con i valori desiderati.
* `testClose()`: viene creato un `Vector<Double>` al quale vengono aggiunti cinque valori. Attraverso il mock `mockPeerConnection`, si impostano i metodi `getPacketsSent()`, `getPacketsLost()`, `getBytesSent()`, `getInizio()` e `getFine()` della classe `mytalk.client.presenter.user.communication.PeerConnection` per ritornare gli stessi valori contenuti nel `Vector` creato. Si verifica quindi che venga richiamato il metodo `mytalk.client.presenter.serverComUser.WebSocketUser.sendStats(String ricevente, Vector<Double> statistiche)` con i valori desiderati.

Stato: Superato

Bug individuati: 0

C.1.3 Package mytalk.client.presenter.user.logicUser.common

- **Classe** `mytalk.client.presenter.client.user.logicUser.common.CommonFunctions`

  Classe che realizza i test:
  `mytalk.client.presenter.client.user.logicUser.common.CommonFunctionsTest`

Descrizione:
Vengono controllate le varie stringhe passategli dalle classi. Prima di iniziare il test, vengono create opportune stringhe contenenti dati corretti e dati sbagliati.

Metodi di test:

* `testCheckEmail()`: viene richiamato più volte il metodo da testare `testCheckEmail(String input)` con diverse combinazioni, corrette ed errate, di dati nella stringa passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua.
* `ttestCheckPassword()`: viene richiamato più volte il metodo da testare `testCheckEmail(String input)` con diverse combinazioni, corrette ed errate, di dati nella stringa passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua.
* `testCheckNameSurname()`: viene richiamato più volte il metodo da testare `testCheckNameSurname(String input)` con diverse combinazioni, corrette ed errate, di dati nella stringa passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua.
* `testCheckPasswordControl()`: viene richiamato più volte il metodo da testare `testCheckPasswordControl(String input)` con diverse combinazioni, corrette ed errate, di dati nella stringa passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua.
* `testCheckCompany()`: viene richiamato più volte il metodo da testare `testCheckCompany(String input)` con diverse combinazioni, corrette ed errate, di dati nella stringa passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua.
* `testCheckNumber()`: viene richiamato più volte il metodo da testare `testCheckNumber)(String input)` con diverse combinazioni, corrette ed errate, di dati nella stringa passato in input, in modo da raggiungere una copertura totale dei controlli che il metodo effettua.

Stato: Superato

Bug individuati : 0

C.1.4 Package mytalk.client.presenter.user.serverComUser

- **Classe** `mytalk.client.presenter.user.serverComUser.WebSocketUser`

  Classe che realizza i test:
  mytalk.client.presenter.user.serverComUser.WebSocketUserTest

  Descrizione: viene verificate la corretta interpretazione dei messaggi provenienti dal server. Si creano diversi tipi di messaggi $XML_{|g|}$ in modo da percorrere tutti i possibili cammini all'interno dei metodi deputati alla loro interpretazione.

  Metodi di test:

* `testCallEnter()`: si verifica la corretta invocazione del metodo `mytalk.client.presenter.user.communication.PeerConnection.answer(String descrizione, IMediaStream stream)`.
* `testParseXML_c_callNegotiation_offer()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `callNegotiation` e tipo uguale a `offer`. Poiché non c'è invocazione di altri metodi, viene verificata tramite EclEmma la corretta copertura del parser.
* `testParseXML_c_callNegotiation_answer_offline`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `callNegotiation`, tipo uguale a `answer` e tag relativo allo stato dell'utente (`<st>`) impostato a `offline`. Viene verificata la corretta invocazione del metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.updatePanelSearch(boolean b, String utente)` con argomenti `false` e stringa vuota.
* `testParseXML_c_callNegotiation_answer_accepted()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `callNegotiation`, tipo uguale a `answer` e tag relativo allo stato dell'utente impostato a `accepted`. Vengono inoltre impostati il tag utente (`<us>`) e il tag remote description (`<rd>`). Si verifica che venga invocato correttamente il metodo `mytalk.client.presenter.user.logicUser.acceptedCall(String answer, String utente` con i valori contenuti nel tag remote description e utente.
* `testParseXML_c_callNegotiation_answer_refused()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `callNegotiation`, tipo uguale a `answer` e tag relativo allo stato dell'utente impostato a `refused`. Vengono inoltre impostati il tag utente (`<us>`) e il tag relativo allo stato (`<st>`), che contiene `refused`. Viene verificata la corretta invocazione del metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.notifyRefuseCall(String s` con argomento uguale al nome dell'utente impostato.
* `testParseXML_c_userList()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `userList` e tag relativi agli utenti (`<m>`) con dei nomi utente. Viene creato localmente un `Vector` contenente i nomi utente inseriti nei tag, e si controlla che venga passato un vettore uguale al metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.setUserList(Vector<String> listaUtenti)`.
* `testParseXML_c_callExchange_answerDescription()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `callExchange` e tipo uguale ad `answerDescription`. I tag relativi agli utenti (`<m>`) e agli ice candidates (`<ic>`) vengono popolati. Viene creato localmente un `Vector` contenente gli ice candidates inseriti nei tag, e si controlla che venga passato un vettore uguale al metodo

`mytalk.client.presenter.user.logicUser.CommunicatinLogic.receivedSessionPerformed(Vector<String> candidate)`.

* `testParseXML_c_callExchange_offerDescription()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `callExchange` e tipo uguale ad `offerDescription`. I tag relativi all'utente corrente (`<us>`), agli utenti (`<m>`) e agli ice candidates (`<ic>`) vengono popolati. Viene creato localmente un `Vector` contenente gli ice candidates inseriti nei tag, e si controlla che al metodo `mytalk.client.presenter.user.logicUser.CommunicatinLogic.receivedSession(String mittente, Vector<String> candidate)` vengano passati il nome utente contenuto nel tag `<us>` e un vettore uguale a quello creato.
* `testParseXML_c_find_es_true()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `find`. Vengono popolati i tag relativi agli utenti (`<m>`) e all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.CommunicatinLogic.call(final String user)` venga richiamato con argomento uguale al contenuto del tag `<m>` e che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.updatePanelSearch(boolean b, String utente)` venga invocato con argomenti uguali a `true` e al contenuto del tag `<m>`.
* `testParseXML_c_find_es_false()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla comunicazione (`<c>`), con l'operazione uguale a `find`. Vengono popolati i tag relativi agli utenti (`<m>`) e all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.updatePanelSearch(boolean b, String utente)` venga invocato con argomenti uguali a `false` alla stringa vuota.
* `testParseXML_ud_log_es_true()`: viene testato il parsing della stringa $XML_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale a `log`. Vengono popolati i tag relativi agli utenti (`<m>`) e all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.loginResult(boolean loginSuccess, Vector<String> userInput)` venga invocato con argomento uguale a `true`.
* `testParseXML_ud_log_es_false()`: viene testato il parsing della stringa $XML_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale a `log`. Vengono popolati i tag relativi agli utenti (`<m>`) e all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.loginResult(boolean loginSuccess, Vector<String> userInput)` venga invocato con argomento uguale a `false`.
* `testParseXML_ud_add_es_true()`: viene testato il parsing della stringa $XML_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale ad `add`. Viene popolato il tag relativo all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.registerResult(boolean registerSuccess)` venga invocato con argomento uguale a `true`.

* `testParseXML_ud_add_es_false()`: viene testato il parsing della stringa XML$_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale ad `add`. Viene popolato il tag relativo all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.registerResult(boolean registerSuccess)` venga invocato con argomento uguale a `false`.
* `testParseXML_ud_mod_es_true()`: viene testato il parsing della stringa XML$_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale a `mod`. Viene popolato il tag relativo all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.resultDataUser(boolean update)` venga invocato con argomento uguale a `true`. Inoltre, nel metodo `setUp()` della classe di test, viene creato un cookie$_{|g|}$ che la classe `WebSocketUser` utilizza per recuperare il valore da inserire nel nuovo cookie$_{|g|}$. Si verifica che il nuovo cookie$_{|g|}$ venga effettivamente creato con questo valore.
* `testParseXML_ud_mod_es_false()`: viene testato il parsing della stringa XML$_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale a `mod`. Viene popolato il tag relativo all'esito (`<es>`). Si controlla che il metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.resultDataUser(boolean update)` venga invocato con argomento uguale a `false`.
* `testParseXML_ud_udt()`: viene testato il parsing della stringa XML$_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale a `udt`. Vengono popolati i tag relativi agli utenti (`<m>`) e quelli relativi agli errori (`<er>`), oltre ai tag deputati alla raccolta dei dati utente: nome (`<nm>`), cognome (`<sn>`), società (`<sc>`), telefono (`<tl>`) e password (`<p>`). Si crea un `Vector` contenente i valori presenti nei campi dei dati utente e si verifica che ne venga passato uno uguale al metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.setUserDataLabel(Vector<String> userData)`.
* `testParseXML_ud_udt_null()`: viene testato il parsing della stringa XML$_{|g|}$ relativa ai dati utente (`<ud>`), con l'operazione uguale a `udt`. Vengono popolati i tag relativi agli utenti (`<m>`) e quelli relativi agli errori (`<er>`), oltre ai soli tag obbligatori tra quelli deputati alla raccolta dei dati utente: nome (`<nm>`), cognome (`<sn>`) e password (`<p>`). Si crea un `Vector` contenente i valori presenti nei campi dei dati utente e si verifica che ne venga passato uno uguale al metodo `mytalk.client.presenter.user.logicUser.UpdateViewLogic.setUserDataLabel(Vector<String> userData)`.
* `testParseXML_ip()`: viene testato il parsing della stringa XML$_{|g|}$ relativa all'indirizzo IP$_{|g|}$ (`<ip>`). Viene popolato il tag relativo all'utente (`<us>`), nel quale viene inserito un indirizzo IP$_{|g|}$. Si controlla la corretta creazione del cookie$_{|g|}$ di sessione relativo all'indirizzo IP$_{|g|}$, che deve contenere l'indirizzo presente nel tag (`<us>`), tramite il metodo della classe GWT `Cookies.getCookie(String cookieName)`.

Stato: Superato

Bug individuati: 0

- **Classe** `mytalkadmin.client.presenter.serverComUser.WebSocketAdmin`

  Classe che realizza i test:
  mytalkadmin.client.presenter.serverComUser.WebSocketAdminTest

  Descrizione: viene verificata la corretta interpretazione dei messaggi provenienti dal server. Si creano diversi tipi di messaggi $XML_{|g|}$ in modo da percorrere tutti i possibili cammini all'interno dei metodi deputati alla loro interpretazione.

  Metodi di test:

  * `testParseXML_c_userList()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla richiesta della lista utenti. Vengono inseriti alcuni utenti nella stringa da passare al metodo da testare e si verifica che tale metodo invochi `mytalkadmin.client.presenter.logicAdmin.UpdateViewLogic.setUserList(Vector<String> listaUtenti)`
  * `testParseXML_c()`: viene testato il parsing della stringa $XML_{|g|}$ relativa alla ricerca che può effettuare l'amministratore. Vengono inserite nella stringa due liste di ricerca, che vengono copiate in un `Vector<Vector<String>>` (ogni lista è un vettore, che viene inserito in un altro vettore). Si controlla che il metodo da testare recuperi correttamente i contenuti dei tag e che li inserisca nel `Vector`; tale vettore deve essere passato al metodo `mytalkadmin.client.presenter.logicAdmin.UpdateViewLogic.setListData(Vector<Vector<String>> list)`.
  * `testParseXML_c_null()`: viene testato il parsing della stringa $XML_{|g|}$ che contiene solamente il tag `<c>`. Si verifica che al metodo `mytalkadmin.client.presenter.logicAdmin.UpdateViewLogic.setListData(Vector<Vector<String>> list)` venga passato un `Vector<Vector<String>>` vuoto.
  * `testParseXML_ud_log_es_true()`: viene testato il parsing della stringa $XML_{|g|}$ relativa all'autenticazione dell'amministratore con esito positivo. Si verifica che il metodo `mytalkadmin.client.presenter.logicAdmin.UpdateViewLogic.loginResult(boolean loginSuccess, Vector<String> loginInput` venga invocato con argomento uguale a `true`.
  * `testParseXML_ud_log_es_false()`: viene testato il parsing della stringa $XML_{|g|}$ relativa all'autenticazione dell'amministratore con esito negativo. Si verifica che il metodo `mytalkadmin.client.presenter.logicAdmin.UpdateViewLogic.loginResult(boolean loginSuccess, Vector<String> loginInput` venga invocato con argomento uguale a `false`.

  Stato: Superato

  Bug individuati: 0

C.1.5 Package mytalk.server.presenter

- **Classe** `mytalk.server.presenter.XMLField`

  Classe che realizza i test:
  `mytalk.server.presenter.XMLFieldTest`

  Descrizione: viene verificata la correttezza dell'oggetto per la raccolta dei metodi e dei riferimenti alle stringhe XML elaborate.

  Metodi di test:

  * `testLoadXMLFromString()`: si verifica la corretta creazione dell'oggetto per il riferimento dei nodi della stringa XML passata. Viene richiamato il metodo `loadXMLFromString(String xml)` della classe `XMLField`. Viene verificato che il metodo ritorni l'oggetto corretto.
  * `testExtractLogInfo()`: si verifica la corretta estrazione delle informazioni che servono per gestire l'accesso al servizio. Viene richiamato il metodo `extractLogInfo(Element e)` della classe `XMLField` per due volte. In entrambi i casi inizialmente viene chiamato il metodo `loadXMLFromString(String xml)` passandogli un particolare parametro `xml` e sull'oggetto `Document` ottenuto viene ricavato l'oggetto `e` che rappresenta l'elemento XML da passare come parametro al metodo `extractLogInfo(Element e)`. La prima volta `xml` contiene tutti gli elementi corretti ma non l'indirizzo IP, la seconda volta invece lo contiene. Viene verificato che il metodo ritorni l'oggetto corretto.
  * `testExtractRegInfo()`: si verifica la corretta estrazione delle informazioni che servono per gestire la registrazione al servizio. Viene richiamato il metodo `extractRegInfo(Element e)` della classe `XMLField` per tre volte. In tutti i casi inizialmente viene chiamato il metodo `loadXMLFromString(String xml)` passandogli un particolare parametro `xml` e sull'oggetto `Document` ottenuto viene ricavato l'oggetto `e` che rappresenta l'elemento XML da passare come parametro al metodo `extractRegInfo(Element e)`. La prima volta `xml` contiene tutti gli elementi corretti riguardanti la registrazione di un nuovo utente, la seconda volta ne contiene solo alcuni e la terza nessuno. Viene verificato che il metodo ritorni l'oggetto corretto.
  * `testExtractCommInfo()`: si verifica la corretta estrazione delle informazioni che servono per gestire la comunicazione attraverso il servizio. Viene richiamato il metodo `extractCommInfo(Element e)` della classe `XMLField`. Inizialmente viene chiamato il metodo `loadXMLFromString(String xml)` passandogli un particolare parametro `xml` e sull'oggetto `Document` ottenuto viene ricavato l'oggetto `e` che rappresenta l'elemento XML da passare come parametro al metodo `extractCommInfo(Element e)`. L'oggetto `xml` contiene tutti gli elementi corretti riguardanti una comunicazione. Viene verificato che il metodo ritorni l'oggetto corretto.
  * `testExtractFindInfo()`: si verifica la corretta estrazione delle informazioni che servono per gestire la comunicazione attraverso il servizio. Viene richiamato il metodo `extractFindInfo(Element e)` della classe

`XMLField` per due volte. In entrambi i casi inizialmente viene chiamato il metodo `loadXMLFromString(String xml)` passandogli un particolare parametro `xml` e sull'oggetto `Document` ottenuto viene ricavato l'oggetto `e` che rappresenta l'elemento XML da passare come parametro al metodo `extractFindInfo(Element e)`. La prima volta `xml` contiene l'elemento corretto relativo all'indirizzo IP, la seconda volta invece non lo contiene. Viene verificato che il metodo ritorni l'oggetto corretto.

* `testExtractStatReq()`: si verifica la corretta estrazione delle informazioni di riferimento per l'estrazione delle statistiche. Viene richiamato il metodo `extractStatReq(Element e)` della classe `XMLField` per quattro volte. In tutti i casi inizialmente viene chiamato il metodo `loadXMLFromString(String xml)` passandogli un particolare parametro `xml` e sull'oggetto `Document` ottenuto viene ricavato l'oggetto `e` che rappresenta l'elemento XML da passare come parametro al metodo `extractStatReq(Element e)`. La prima volta `xml` contiene l'attributo `time` e l'elemento `email` corretti, la seconda volta contiene l'attributo `time` e l'elemento `grade` corretti, la terza volta contiene l'attributo `time` e l'elemento `dateStart` corretti e la quarta non contiene nessun elemento e nessun attributo. Viene verificato che il metodo ritorni l'oggetto corretto.

Stato: Superato

Bug individuati: 0

C.1.6 Package mytalk.server.presenter.administrator.logicAdmin

- **Classe** `mytalk.server.presenter.administrator.logicAdmin.ManageWSA`

Classe che realizza i test:
`mytalk.server.presenter.administrator.logicAdmin.ManageWSATest`

Descrizione: viene verificata la correttezza dell'oggetto per la gestione delle richieste in formato XML relative all'amministratore da inoltrare al Model.

Metodi di test:

* `testGetOperation()`: si verifica il corretto recupero dell'informazione riguardante l'identificativo dell'ultima operazione effettuata. Viene richiamato il metodo `getOperation()` della classe `ManageWSA`. Inizialmente si invoca il metodo `testRequest(String xml)` passandogli un particolare parametro `xml` che contiene il tipo di operazione da effettuare. Poi sullo stesso oggetto si invoca il metodo `getOperation()`. Viene verificato che il metodo ritorni l'identificativo corretto dell'operazione.

Nei prossimi test si verifica, a seconda del tipo di operazione rilevato, la corretta chiamata del metodo privato che gestisce l'operazione individuata. In ogni test viene invocato il metodo `request(String xml)` della classe

`ManageWSA` passandogli, ogni volta, un diverso parametro `xml`. Per ogni test viene verificato che il metodo ritorni il messaggio corretto.

* `testRequestEmpty()`: l'oggetto `xml` non contiene nessun elemento di tipo comunicazione o amministratore.
* `testRequestOpLog()`: si verifica la corretta chiamata del metodo privato che verifica i dati di login dell'amministratore. Il metodo `request(String xml)` viene invocato tre volte. In tutti i casi `xml` contiene un elemento di tipo amministratore. La prima volta contiene un elemento corretto di tipo amministratore, la seconda contiene un elemento di tipo amministratore ma con indirizzo email non presente nel database e la terza contiene un elemento di tipo amministratore ma con password errata.
* `testRequestOpGetStat()`: si verifica la corretta chiamata del metodo privato che verifica i valori di riferimento per l'estrazione delle statistiche. Il metodo `request(String xml)` viene invocato due volte. In tutti i casi `xml` contiene un elemento di tipo comunicazione. La prima volta contiene un elemento corretto di tipo comunicazione e la seconda contiene un elemento di tipo comunicazione ma con la data troppo vecchia per poter essere visualizzata.
* `testRequestOpUserList()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta per l'ottenimento della lista utenti. L'oggetto `xml` contiene un elemento corretto di tipo comunicazione.

Stato: Superato

Bug individuati: 0

- **Classe** `mytalk.server.presenter.administrator.logicAdmin.WSAdmin`

  Classe che realizza i test:
  `mytalk.server.presenter.administrator.logicAdmin.WSAdminTest`

  Descrizione: si verifica la correttezza dell'oggetto servlet per la gestione delle connessioni WebSocket e dello scambio d'informazioni.

  Metodi di test:

  * `testCreateWebSocketInbound()`: si verifica la corretta creazione dell'istanza per l'elaborazione della connessione in ingresso. Viene invocato il metodo `createWebSocketInbound(String arg0, HttpServletRequest arg1)` della classe `WSAdmin`. Viene verificato che il metodo ritorni l'oggetto corretto.
  * `testOnOpen()`: si verifica la corretta gestione della richiesta di apertura di una connessione HTTP per la creazione di una WebSocket. Viene invocato il metodo `onOpen(WsOutbound outbound)` della classe `WSAdmin`. Viene verificato che il metodo gestisca correttamente la creazione della connessione.
  * `testOnTextMessageNull()`: si verifica la corretta ricezione di un messaggio testuale da parte del client. Viene invocato il metodo `onTextMessage(CharBuffer buffer)` della classe `WSAdmin` specificando il riferimento al client a cui spedire il messaggio. Viene verificato che il metodo crei il messaggio corretto.

* `testOnTextMessage()`: si verifica la corretta ricezione di un messaggio testuale da parte del client. Viene invocato il metodo `onTextMessage(CharBuffer buffer)` della classe `WSAdmin` senza specificare il riferimento al client a cui spedire il messaggio. Viene verificato che il metodo crei il messaggio corretto.

Stato: Superato

Bug individuati: 0

C.1.7 Package mytalk.server.presenter.user.logicUser

- **Classe** `mytalk.server.presenter.user.logicUser.ManageWSU`

  Classe che realizza i test:
  `mytalk.server.presenter.user.logicUser.ManageWSUTest`

  Descrizione: viene verificata la correttezza dell'oggetto per la gestione dei messaggi scambiati tra client e server.

  Metodi di test:

  * `testGetOperation()`: si verifica il corretto recupero dell'informazione riguardante l'identificativo dell'ultima operazione effettuata. Viene richiamato il metodo `getOperation()` della classe `ManageWSU`. Inizialmente si invoca il metodo `testRequest(String xml)` passandogli un particolare parametro `xml` che contiene il tipo di operazione da effettuare. Poi sullo stesso oggetto si invoca il metodo `getOperation()`. Viene verificato che il metodo ritorni l'identificativo corretto dell'operazione.
  * `testGetRefUser()`: si verifica il corretto recupero dell'informazione riguardante l'indirizzo email dell'utente di riferimento. Viene richiamato il metodo `getRefUser()` della classe `ManageWSU`. Inizialmente si invoca il metodo `testRequest(String xml)` passandogli un particolare parametro `xml` che contiene l'indirizzo email dell'utente di riferimento. Poi sullo stesso oggetto si invoca il metodo `getRefUser()`. Viene verificato che il metodo ritorni l'indirizzo email corretto dell'utente di riferimento.
  * `testGetDestUser()`: si verifica il corretto recupero dell'informazione riguardante l'indirizzo email dell'utente destinatario. Viene richiamato il metodo `getDestUser()` della classe `ManageWSU`. Inizialmente si invoca il metodo `testRequest(String xml)` passandogli un particolare parametro `xml` che contiene l'indirizzo email dell'utente destinatario. Poi sullo stesso oggetto si invoca il metodo `getDestUser()`. Viene verificato che il metodo ritorni l'indirizzo email corretto dell'utente destinatario.

  Nei prossimi test si verifica, a seconda del tipo di operazione rilevato, la corretta chiamata del metodo privato che gestisce l'operazione individuata. In ogni test viene invocato il metodo `request(String xml)` della classe `ManageWSU` passandogli, ogni volta, un diverso parametro `xml`. Per ogni test viene verificato che il metodo ritorni il messaggio corretto.

* `testRequestEmpty()`: l'oggetto `xml` non contiene nessun elemento di tipo comunicazione o utente.
* `testRequestOpLog()`: si verifica la corretta chiamata del metodo privato che verifica i dati di login dell'utente. Il metodo `request(String xml)` viene invocato tre volte. In tutti i casi `xml` contiene un elemento di tipo utente. La prima volta con l'elemento email sbagliato, la seconda con l'elemento password errato e la terza con l'elemento corretto.
* `testRequestOpULog()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta di logout dell'utente. L'oggetto `xml` contiene un elemento corretto di tipo utente.
* `testRequestOpAdd()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta di aggiunta di un nuovo utente. Il metodo `request(String xml)` viene invocato due volte. In tutti i casi `xml` contiene un elemento di tipo utente. La prima volta contiene un utente già presente nel database, la seconda contiene un utente non presente.
* `testRequestOpMod()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta di modifica dei dati utente. Il metodo `request(String xml)` viene invocato due volte. In tutti i casi `xml` contiene un elemento di tipo utente per la modifica dell'indirizzo email. La prima volta contiene un indirizzo email già presente nel database, la seconda contiene un indirizzo email non presente.
* `testRequestOpUdt()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta di raccolta delle informazioni dell'utente. Il metodo `request(String xml)` viene invocato tre volte. In tutti i casi `xml` contiene un elemento di tipo utente. La prima volta non contiene il numero di telefono, la seconda contiene un elemento corretto e la terza non contiene il numero di telefono e la società.
* `testRequestOpDel()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta di eliminazione dell'utente. Il metodo `request(String xml)` viene invocato due volte. In tutti i casi `xml` contiene un elemento di tipo utente. La prima volta contiene un utente presente nel database, la seconda contiene un utente non presente.
* `testRequestOpUserList()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta per l'ottenimento della lista utenti. L'oggetto `xml` contiene un elemento corretto di tipo comunicazione.
* `testRequestOpCNeg()`: si verifica la corretta chiamata del metodo privato che effettua la gestione per la negoziazione della comunicazione. Il metodo `request(String xml)` viene invocato due volte. In tutti i casi `xml` contiene un elemento di tipo comunicazione. La prima volta contiene l'utente destinatario offline, la seconda l'utente online.
* `testRequestOpCEx()`: si verifica la corretta chiamata del metodo privato che effettua la richiesta di inoltro dello scambio della comunicazione. L'oggetto `xml` contiene un elemento corretto di tipo comunicazione.
* `testRequestOpIStat()`: si verifica la corretta chiamata del metodo privato che effettua la registrazione delle statistiche relative alla comunicazione. Il metodo `request(String xml)` viene invocato due volte.

In tutti i casi `xml` contiene un elemento di tipo comunicazione. La prima volta contiene un elemento sbagliato, la seconda un elemento corretto.

* `testRequestOpFu()`: si verifica la corretta chiamata del metodo privato che effettua la ricerca di un utente mediante campi dati indicati. Il metodo `request(String xml)` viene invocato due volte. In tutti i casi `xml` contiene un elemento di tipo comunicazione. La prima volta non contiene l'elemento IP, la seconda invece lo contiene.
* `testRequestOpAnswering()`: si verifica la corretta chiamata del metodo privato che gestisce la richiesta di archiviazione di un nuovo messaggio di segreteria. L'oggetto `xml` contiene un elemento corretto di tipo comunicazione.
* `testRequestOpAnsweringDel()`: si verifica la corretta chiamata del metodo privato che gestisce la richiesta di eliminazione di un messaggio presente in segreteria. L'oggetto `xml` contiene un elemento corretto di tipo comunicazione.

Stato: Superato

Bug individuati: 0

- **Classe** `mytalk.server.presenter.user.logicUser.WSUser`

  Classe che realizza i test:
  `mytalk.server.presenter.user.logicUser.WSUserTest`

  Descrizione: si verifica la correttezza dell'oggetto servlet per la gestione delle comunicazioni server-client.

  Metodi di test:

  * `testCreateWebSocketInbound()`: si verifica la corretta creazione dell'istanza per l'elaborazione della connessione in ingresso. Viene invocato il metodo `createWebSocketInbound(String arg0, HttpServletRequest arg1)` della classe `WSUser`. Viene verificato che il metodo ritorni l'oggetto corretto.
  * `testOnOpen()`: si verifica la corretta gestione della richiesta di creazione di una nuova connessione da parte di un client. Viene invocato il metodo `onOpen(WsOutbound outbound)` della classe `WSUser`. Viene verificato che il metodo gestisca correttamente la creazione della connessione.

  Nei prossimi test si verifica la corretta gestione dell'operazione da eseguire ad ogni richiesta del client. In ogni test viene invocato il metodo `onTextMessage(CharBuffer buffer)` della classe `WSUser`. Per ogni test viene verificato che il metodo gestisca correttamente l'operazione corrispondente.

  * `testOnTextMessageNull()`: gli viene passato il parametro `buffer` nullo.
  * `testOnTextMessageOpLog()`: si verifica la corretta gestione dell'operazione di login dell'utente nel caso in cui l'utente abbia già effettuato il login.

* `testOnTextMessageOpLogNull()`: si verifica la corretta gestione dell'operazione di login dell'utente nel caso in cui l'utente non abbia ancora effettuato il login.
* `testOnTextMessageOpULog()`: si verifica la corretta gestione dell'operazione di logout dell'utente.
* `testOnTextMessageOpCNeg()`: si verifica la corretta gestione dell'operazione di negoziazione della comunicazione.
* `testOnTextMessageOpCEx()`: si verifica la corretta gestione dell'operazione di scambio delle informazioni della comunicazione.
* `testOnTextMessageOpAnsweringNull()`: si verifica la corretta gestione dell'operazione di archiviazione di un nuovo messaggio di segreteria nel caso in cui l'utente destinatario non sia presente nell'apposito registro.
* `testOnTextMessageOpAnswering()`: si verifica la corretta gestione dell'operazione di archiviazione di un nuovo messaggio di segreteria nel caso in cui l'utente destinatario sia già presente nell'apposito registro.
* `testOnTextMessageOpAnsweringDelete()`: si verifica la corretta gestione dell'operazione di archiviazione di un nuovo messaggio di segreteria.
* `testOnTextMessage()`: si verifica la corretta gestione dell'invio dei messaggi presenti in segreteria agli utenti destinatari corretti.

Stato: Superato

Bug individuati: 0

C.2 Model

L'obiettivo di questi test è quello di verificare il corretto funzionamento della logica dei metodi delle classi che compongono il Model. Per l'esecuzione dei test sono stati utilizzati i seguenti framework$_{|g|}$ di testing:

- JUnit 4: permette di scrivere ed eseguire in modo automatizzato i test case$_{|g|}$.

L'obiettivo di questi test è quello di verificare il corretto funzionamento della logica dei metodi delle classi che compongono il Model. Per l'esecuzione dei test sono stati utilizzati i seguenti framework$_{|g|}$ di testing:

- JUnit 4: permette di scrivere ed eseguire in modo automatizzato i test case$_{|g|}$.

C.2.1 Package mytalk.server.model.dao

- **Classe** `mytalk.server.model.dao.DataAccessObject`

  Classe che realizza i test:
  `mytalk.server.model.dao.DataAccessObjectTest`

  Descrizione: viene verificata la corretta gestione delle comunicazioni con il database.

  Metodi di test:

  * `testGetInfo()`: si verifica il corretto recupero delle informazioni dal database secondo specifici parametri di richiesta. Viene richiamato il metodo `getInfo(String[] fields, ObjectTransfer ui, String table)` della classe `DataAccessObject` per due volte. La prima passandogli in input valori contenuti nel database e la seconda passandogli valori non contenuti nel database. I valori in input sono contenuti in un `HashMap` composto da due valori: il nome e il cognome dell'utente. Viene verificato che il metodo ritorni il numero corretto di righe contenenti quei valori.
  * `testGetUserInfoByField()`: si verifica il corretto recupero delle informazioni dell'utente dal database secondo un specifico campo scelto. Viene richiamato il metodo `getUserInfoByField(String k, String v)` della classe `DataAccessObject` per due volte. La prima passandogli in input valori contenuti nel database e la seconda passandogli valori non contenuti nel database. I valori in input indicano il campo in cui cercare e il nome dell'utente da cercare. Viene verificato che il metodo ritorni il numero corretto di righe contenenti quei valori.
  * `testGetUserInfo()`: si verifica il corretto recupero delle informazioni dell'utente dal database secondo l'indirizzo email. Viene richiamato il metodo `getUserInfo(String v)` della classe `DataAccessObject` per tre volte. La prima passandogli in input un valore contenuto nel database, la seconda passandogli un valore non contenuto nel database e la terza passandogli un valore nullo. Il valore in input, nei primi due casi, contiene l'indirizzo email dell'utente da cercare. Viene verificato che il metodo ritorni il numero corretto di righe contenenti quei valori.

* `testGetAdminInfoByField()`: si verifica il corretto recupero delle informazioni dell'amministratore secondo un specifico campo scelto. Viene richiamato il metodo `getAdminInfoByField(String k, String v)` della classe `DataAccessObject` per due volte. La prima passandogli in input valori contenuti nel database e la seconda passandogli valori non contenuti nel database. I valori in input indicano il campo in cui cercare e il nome dell'amministratore da cercare. Viene verificato che il metodo ritorni il numero corretto di righe contenenti quei valori.
* `testGetAdminInfo()`: si verifica il corretto recupero delle informazioni dell'amministratore secondo l'indirizzo email. Viene richiamato il metodo `getAdminInfo(String v)` della classe `DataAccessObject` per tre volte. La prima passandogli in input un valore contenuto nel database, la seconda passandogli un valore non contenuto nel database e la terza passandogli un valore nullo. Il valore in input, nei primi due casi, contiene l'indirizzo email dell'amministratore da cercare. Viene verificato che il metodo ritorni il numero corretto di righe contenenti quei valori.
* `testGetCommInfoByOT()`: si verifica il corretto recupero delle informazioni della comunicazione secondo specifici parametri indicati nell'oggetto passato in input. Viene richiamato il metodo `getCommInfoByOT(ObjectTransfer fil, String time)` della classe `DataAccessObject` per tre volte. La prima passandogli in input parametri non nulli, la seconda passandogli il parametro `time` nullo e la terza passandogli entrambi i parametri nulli. I parametri in input indicano, `fil`, l'oggetto `ObjectTransfer` contenente i valori da cercare e, `time`, il numero di secondi da togliere alla data e ora attuale. Viene verificato che il metodo ritorni il numero corretto di righe corrispondenti ai valori definiti nei parametri.
* `testGetCommInfoByField()`: si verifica il corretto recupero delle informazioni della comunicazione secondo un specifico campo scelto. Viene richiamato il metodo `getCommInfoByField(String k, String v, String time)` della classe `DataAccessObject` per due volte. La prima passandogli in input parametri non nulli e la seconda passandogli il parametro `time` nullo. I parametri in input indicano, `k` il campo in cui cercare, `v` l'indirizzo email dell'utente chiamante ed infine `time` il numero di secondi da togliere alla data e ora attuale. Viene verificato che il metodo ritorni il numero corretto di righe corrispondenti ai valori definiti nei parametri.
* `testGetCommInfo()`: si verifica il corretto recupero delle informazioni di tutte le comunicazioni presenti nel database. Viene richiamato il metodo `getCommInfo()` della classe `DataAccessObject`. Viene verificato che il metodo ritorni il numero corretto di righe.
* `testInsertItem()`: si verifica il corretto inserimento di nuovi dati nel database. Viene richiamato il metodo `insertItem(ObjectTransfer ins)` della classe `DataAccessObject` per quattro volte. Le prime tre volte passandogli in input un oggetto `ObjectTransfer`, rispettivamente di tipo utente, amministratore e comunicazione, contenente rispettivamente i dati di un utente, di un amministratore e di una comunicazione da inserire nel database. La quarta volta viene passato un parametro

nullo. Viene verificato che il metodo ritorni il numero corretto di nuove righe inserite nel database.

* `testUpdateItem()`: si verifica il corretto aggiornamento di dati già presenti nel database secondo parametri di riferimento. Viene richiamato il metodo `updateItem(ObjectTransfer insRef, ObjectTransfer insMod)` della classe `DataAccessObject` per sei volte. La prime due passandogli rispettivamente il parametro `insRef` e il parametro `insMod` nulli. La terza passandogli i parametri di tipo diverso tra loro. Le ultime tre passandogli in input entrambi gli oggetti `ObjectTransfer`, rispettivamente di tipo utente, amministratore e comunicazione, cioè `insRef` contenente rispettivamente i dati dell'utente, dell'amministratore e della comunicazione da aggiornare, `insMod` contenente rispettivamente i nuovi dati dell'utente, dell'amministratore e della comunicazione. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testUpdateUserByField()`: si verifica il corretto aggiornamento dei dati dell'utente secondo un unico riferimento. Viene richiamato il metodo `updateUserByField(String k, String v, ObjectTransfer ot)` della classe `DataAccessObject` per cinque volte. Le prime tre con rispettivamente `k` nullo, `v` nullo e `ot` nullo. La quarta con tipo di ot diverso dall'utente. La quinta con i parametri corretti, quindi con `k` contenente il campo in cui cercare, `v` il nome dell'utente da aggiornare ed `ot` i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testUpdateUserByEmail()`: si verifica il corretto aggiornamento dei dati dell'utente secondo l'indirizzo email. Viene richiamato il metodo `updateUserByEmail(String k, ObjectTransfer ot)` della classe `DataAccessObject` con i parametri corretti: con `k` contenente l'inidirzzo email dell'utente da aggiornare ed `ot` con i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testUpdateUserByID()`: si verifica il corretto aggiornamento dei dati dell'utente secondo l'ID. Viene richiamato il metodo `updateUserByID(String k, ObjectTransfer ot)` della classe `DataAccessObject` con i parametri corretti, quindi con `k` contenente l'ID dell'utente da aggiornare ed `ot` con i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testUpdateAdminByField()`: si verifica il corretto aggiornamento dei dati dell'amministratore secondo un unico riferimento. Viene richiamato il metodo `updateAdminByField(String k, String v, ObjectTransfer ot)` della classe `DataAccessObject` per cinque volte. Le prime tre con rispettivamente `k` nullo, `v` nullo e `ot` nullo. La quarta con tipo di ot diverso dall'amministratore. La quinta con i parametri corretti, quindi con `k` contenente il campo in cui cercare, `v` il nome dell'amministratore da aggiornare ed `ot` con i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.

* `testUpdateAdminByEmail()`: si verifica il corretto aggiornamento dei dati dell'amministratore secondo l'indirizzo email. Viene richiamato il metodo `updateAdminByEmail(String k, ObjectTransfer ot)` della classe `DataAccessObject` con i parametri corretti, quindi con `k` contenente l'inidirzzo email dell'amministratore da aggiornare ed `ot` con i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testUpdateAdminByID()`: si verifica il corretto aggiornamento dei dati dell'amministratore secondo l'ID. Viene richiamato il metodo `updateAdminByID(String k, ObjectTransfer ot)` della classe `DataAccessObject` con i parametri corretti, quindi con `k` contenente l'ID dell'amministratore da aggiornare ed `ot` con i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testUpdateCommByField()`: si verifica il corretto aggiornamento dei dati della comunicazione secondo un unico riferimento. Viene richiamato il metodo `updateCommByField(String k, String v, ObjectTransfer ot)` della classe `DataAccessObject` per cinque volte. Le prime tre con rispettivamente `k` nullo, `v` nullo e `ot` nullo. La quarta con tipo di ot diverso dalla comunicazione. La quinta con i parametri corretti, quindi con `k` contenente il campo in cui cercare, `v` l'ID della comunicazione da aggiornare ed `ot` con i nuovi dati. Viene verificato che il metodo ritorni il numero corretto di righe aggiornate nel database.
* `testDeleteItem()`: si verifica la corretta cancellazione dei dati dal database. Viene richiamato il metodo `deleteItem(ObjectTransfer del)` della classe `DataAccessObject` per quattro volte. Le prime tre volte passandogli in input un oggetto `ObjectTransfer`, rispettivamente di tipo utente, amministratore e comunicazione, contenente rispettivamente i dati di un utente, di un amministratore e di una comunicazione da cancellare dal database. La quarta volta viene passato un parametro nullo. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.
* `testDeleteUserByField()`: si verifica la corretta cancellazione di un utente secondo un unico riferimento. Viene richiamato il metodo `deleteUserByField(String k, String v)` della classe `DataAccessObject` per tre volte. Le prime due con rispettivamente `k` nullo e `v` nullo. La terza con i parametri corretti, quindi con `k` contenente il campo in cui cercare, `v` il nome dell'utente da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.
* `testDeleteUserByID()`: si verifica la corretta cancellazione di un utente secondo l'ID. Viene richiamato il metodo `deleteUserByID(String v)` della classe `DataAccessObject` per due volte. La prima con `v` nullo. La seconda con il parametro corretto, quindi con `v` contenente l'ID dell'utente da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.

  - `testDeleteUserByEmail()`: si verifica la corretta cancellazione di un utente secondo l'indirizzo email. Viene richiamato il metodo `deleteUserByEmail(String v)` della classe `DataAccessObject` per due volte. La prima con `v` nullo. La seconda con il parametro corretto, quindi con `v` contenente l'indirizzo email dell'utente da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.
  - `testDeleteAdminByField()`: si verifica la corretta cancellazione di un amministratore secondo un unico riferimento. Viene richiamato il metodo `deleteAdminByField(String k, String v)` della classe `DataAccessObject` per tre volte. Le prime due con rispettivamente `k` nullo e `v` nullo. La terza con i parametri corretti, quindi con `k` contenente il campo in cui cercare, `v` il nome dell'amministratore da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.
  - `testDeleteAdminByID()`: si verifica la corretta cancellazione di un amministratore secondo l'ID. Viene richiamato il metodo `deleteAdminByID(String v)` della classe `DataAccessObject` per due volte. La prima con `v` nullo. La seconda con il parametro corretto, quindi con `v` contenente l'ID dell'amministratore da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.
  - `testDeleteAdminByEmail()`: si verifica la corretta cancellazione di un amministratore secondo l'indirizzo email. Viene richiamato il metodo `deleteAdminByEmail(String v)` della classe `DataAccessObject` per due volte. La prima con `v` nullo. La seconda con il parametro corretto, quindi con `v` contenente l'indirizzo email dell'amministratore da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.
  - `testDeleteCommByField()`: si verifica la corretta cancellazione di una comunicazione secondo un unico riferimento. Viene richiamato il metodo `deleteCommByField(String k, String v)` della classe `DataAccessObject` per tre volte. Le prime due con rispettivamente `k` nullo e `v` nullo. La terza con i parametri corretti, quindi con `k` contenente il campo in cui cercare, `v` il numero di byte trasmessi della comunicazione da cancellare. Viene verificato che il metodo ritorni il numero corretto di righe cancellate dal database.

  Stato: Superato

  Bug individuati: 0

- **Classe** `mytalk.server.model.dao.ObjectTransfer`

  Classe che realizza i test:
  `mytalk.server.model.dao.ObjectTransferTest`

  Descrizione: viene verificata la correttezza dell'oggetto di trasferimento per le informazioni della classe `DataAccessObject`.

  Metodi di test:

* `testGetTableInfo()`: si verifica il corretto recupero dell'informazione riguardante il tipo di tabella che si incorpora. Viene richiamato il metodo `getTableInfo()` della classe `ObjectTransfer` per tre volte. La prima su un oggetto di tipo utente, la seconda su un oggetto di tipo amministratore e la terza su un oggetto di tipo comunicazione. Viene verificato che il metodo ritorni il tipo corretto.
* `testAddMap()`: si verifica il corretto inserimento di una tabella nell'oggetto. Viene richiamato il metodo `addMap(HashMap<String, String> e)` della classe `ObjectTransfer` per due volte. La prima con `e` nullo e la seconda con il parametro corretto, quindi con `e` contenente la tabella da aggiungere. Viene verificato che l'oggetto di invocazione contenga il numero corretto di tabelle.
* `testGetEmptyMapString()`: si verifica la corretta creazione di una nuova tabella vuota con i valori delle chiavi già configurati secondo il tipo indicato nel parametro . Viene richiamato il metodo `getEmptyMap(String t)` della classe `ObjectTransfer` per tre volte. La prima con `t` == user, la seconda con `t` == admin e la terza con `t` == comm. Viene verificato che il metodo ritorni la tabella di tipo corretto.
* `testGetEmptyMap()`: si verifica la corretta creazione di una nuova tabella vuota con i valori delle chiavi già configurati secondo la tipologia dell'oggetto di invocazione. Viene richiamato il metodo `getEmptyMap()` della classe `ObjectTransfer` per tre volte. La prima su un oggetto di tipo utente, la seconda su un oggetto di tipo amministratore e la terza su un oggetto di tipo comunicazione. Viene verificato che il metodo ritorni la tabella di tipo corretto.
* `testGetSize()`: si verifica il corretto recupero dell'informazione riguardante il numero di tabelle contenute nell'oggetto. Viene richiamato il metodo `getSize()` della classe `ObjectTransfer`. Viene verificato che l'oggetto di invocazione contenga il numero corretto di tabelle.
* `testGetMapInt()`: si verifica il corretto recupero della tabella contenuta nell'oggetto nella posizione indicata dal parametro. Viene richiamato il metodo `getMap(int i)` della classe `ObjectTransfer` per tre volte. La prima con `i` < 0, la seconda con `i` maggiore della dimensione dell'oggetto di invocazione e la terza con `i` ≥ 0. Viene verificato che il metodo ritorni la tabella corretta.
* `testGetMap()`: si verifica il corretto recupero della prima tabella contenuta nell'oggetto. Viene richiamato il metodo `getMap()` della classe `ObjectTransfer`. Viene verificato che il metodo ritorni la tabella corretta.
* `testSetFieldMap()`: si verifica il corretto inserimento nella tabella in posizione indicata della coppia chiave-valore segnalata. Viene richiamato il metodo `setFieldMap(String k, String v, int m)` della classe `ObjectTransfer` per tre volte. La prima con tabella in posizione `m` < 0, la seconda con tabella in posizione `m` maggiore della dimensione dell'oggetto di invocazione e la terza con parametri corretti quindi con `k` contenente la chiave, `v` contenente il valore e `m` contenente la posizione

della tabella da recuperare. Viene verificato che l'oggetto di invocazione contenga il numero corretto di tabelle e che la tabella indicata contenga il numero corretto di coppie chiave-valore.

Stato: Superato

Bug individuati: 0

C.3 Tabella riassuntiva dei test di unità

Data	Classe testata	Esito	Bug rivelati
2013-05-10	mytalk.client.model.localDataUser.ManageCookies	Superato	0
2013-05-10	mytalk.client.presenter.user.logicUser.DataUserLogic	Superato	1
2013-05-15	mytalk.client.presenter.user.logicUser.LogUserLogic	Superato	0
2013-05-15	mytalk.client.presenter.user.logicUser.UpdateViewLogic	Superato	1
2013-05-20	mytalk.client.presenter.user.logicUser.RegisterLogic	Superato	0
2013-05-22	mytalk.client.presenter.user.logicUser.CommunicationLogic	Superato	0
2013-05-22	mytalk.client.presenter.client.user.logicUser.common.CommonFunctions	Superato	0
2013-05-22	mytalk.client.presenter.user.serverComUser.WebSocketUser	Superato	0
2013-05-25	mytalkadmin.client.presenter.serverComUser.WebSocketAdmin	Superato	0
2013-05-03	mytalk.server.presenter.XMLField	Superato	0
2013-05-10	mytalk.server.presenter.admin.logicAdmin.ManageWSA	Superato	0
2013-05-02	mytalk.server.presenter.admin.logicAdmin.WSAdmin	Superato	0
2013-05-15	mytalk.server.presenter.user.logicUser.ManageWSU	Superato	0
2013-05-01	mytalk.server.presenter.user.logicUser.WSUser	Superato	0
2013-05-08	mytalk.server.model.dao.DataAccessObject	Superato	0
2013-05-03	mytalk.server.model.dao.ObjectTransfer	Superato	0

Tabella 12: Riassunto dell'esito dei test di unità

C.4 Tabella riassuntiva dei test di integrazione

Data	Ambito	Codice Verifica	Esito
2013-06-03	Utente	TI - Login	Superato
2013-06-03	Utente	TI - gestione dati	Superato
2013-06-04	Utente	TI - Communicaton	Superato
2013-06-05	Amministratore	TI - Login	Superato
2013-06-06	amministratore	TI - Statistic	Superato

Tabella 13: Riassunto dell'esito dei test di integrazione

C.5 Errori riscontrati nella view

Vengono riportati i problemi individuati nelle componenti utilizzate per realizzare l'interfaccia grafica.

- L'uso dell'oggetto `RootPanel` viene limitato alla sola creazione della pagina, mentre in origine era utilizzato anche per l'aggiunta e la rimozione di componenti della $GUI_{|g|}$. Questo comportava il completo aggiornamento della pagina stessa, provocando una nuova creazione di tutti gli oggetti necessari all'applicazione.
- I bottoni, inizialmente di tipo `SubmitButton` forzavano un aggiornamento della pagina, per questo sono stati sostituiti da bottoni di tipo `Button`, i quali richiamano semplicemente le funzionalità desiderate.
- Corretto il bug che provocava la visualizzazione dell'utente stesso tra la lista dei contatti chiamabili.

D Test di Sistema

Sono stati eseguiti test di sistema a mano per verificare il corretto funzionamento dell'applicativo. Si riporta di seguito i requisiti soddisfatti e correttamente implementati all'interno dell'applicativo.

D.1 Tabella riassuntiva dei test di Sistema

Data	Ambito	Codice Verifica	Esito
2013-07-03	Utente	TS-FOB 0	Superato
2013-07-03	Utente	TS-FOB 1	Superato
2013-07-04	Utente	TS-FOB 2	Superato
2013-07-04	Utente	TS-FOB 2.1.2	Superato
2013-07-04	Utente	TS-FOB 2.2.1	Superato
2013-07-04	Utente	TS-FOB 2.2.3	Superato
2013-07-04	Utente	TS-FOB 2.6	Superato
2013-07-04	Utente	TS-FOB 3.1a	Superato
2013-07-04	Utente	TS-FOB 3.1b	Superato
2013-07-04	Utente	TS-FOB 3.2	Superato
2013-07-04	Utente	TS-FOB 4.1.1	Superato
2013-07-04	Utente	TS-FOB 4.1.1.1.1	Superato
2013-07-04	Utente	TS-FOB 4.1.2	Superato
2013-07-04	Utente	TS-FOB 4.2.1	Superato
2013-07-04	Utente	TS-FOB 4.2.1.1a	Superato
2013-07-04	Utente	TS-FOB 4.2.1.1b	Superato
2013-07-04	Utente	TS-FOB 4.2.1.2a	Superato
2013-07-04	Utente	TS-FOB 4.2.1.2b	Superato
2013-07-04	Utente	TS-FOB 4.2.1.3	Superato
2013-07-04	Utente	TS-FOB 6	Superato
2013-07-05	Amministratore	TS - FAOB 2.1	Superato
2013-07-05	Amministratore	TS - FAOB 2.1.1	Superato
2013-07-05	Amministratore	TS - FAOB 2.1.2	Superato
2013-07-05	Amministratore	TS - FAOB 2.2	Superato
2013-07-05	Amministratore	TS - FAOB 2.2.2	Superato
2013-07-05	Amministratore	TS - FAOB 2.2.3	Superato
2013-07-05	Amministratore	TS - FAOB 2.2.4	Superato
2013-07-05	Amministratore	TS - FAOB 2.2.5	Superato

Tabella 14: Riassunto dell'esito dei test di sistema

E Rapporto delle misurazioni

E.1 Copertura del codice

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
MyTalk	72,1 %	14.218	5.501	19.719
src	59,2 %	7.164	4.938	12.102
mytalk.client	0,0 %	0	6	6
mytalk.client.model.localDataUser	92,5 %	37	3	40
mytalk.client.presenter.user.communication	7,7 %	3	36	39
mytalk.client.presenter.user.logicUser	91,2 %	1.193	115	1.308
mytalk.client.presenter.user.logicUser.common	97,1 %	101	3	104
mytalk.client.presenter.user.serverComUser	44,7 %	770	951	1.721
mytalk.client.view.user	0,0 %	0	3.228	3.228
mytalk.server.model.dao	97,1 %	1.566	47	1.613
mytalk.server.presenter	99,6 %	720	3	723
mytalk.server.presenter.administrator.logicAdmin	92,2 %	884	75	959
mytalk.server.presenter.user.logicUser	80,6 %	1.890	456	2.346
mytalk.shared	0,0 %	0	15	15
test	92,6 %	7.054	563	7.617

Figura 12: Copertura del codice.

La figura 12 illustra le percentuali di copertura raggiunte dai test di unità, suddivise per package, calcolate con il plugin di Eclipse `EclEmma`. Come si può vedere, la copertura raggiunta nel codice sorgente è del 59,2%. A questi test vanno aggiunti altri due test di unità, `WSAdminTest` e `WSUserTest`, eseguibili solo singolarmente e non come parte della suite di test rappresentata nell'immagine. Tali test coprono un ulteriore 5% del codice totale del progetto. Quindi la copertura complessiva raggiunta nel codice sorgente è del 64,2%. È stato quindi raggiunto l'obiettivo del 60% fissato in sezione B.

- È stato scelto di non effettuare i test della componente View tramite JUnit, dato che il codice che gestisce l'interfaccia grafica non contiene logica e si basa in gran parte sulle funzionalità offerte dal framework$_{|g|}$ GWT. Il test sulla View viene di conseguenza effettuato in modo manuale.

- Il package `mytalk.client.prenter.user.serverComUser` raggiunge una percentuale bassa in quanto molti dei suoi metodi hanno il solo compito di concatenare delle stringhe e passarle ad un altro metodo; sono quindi stati testati i soli parser della classe `WebSocketUser`.

- Il package `mytalk.client.presenter.user.communication` contiene due classi, `MediaStream` e `PeerConnection`, che sono composte quasi esclusivamente da metodi scritti in JavaScript$_{|g|}$ nativo. Tale tipologia di metodi è risultata impossibile da testare all'interno del framework$_{|g|}$ GWT; il loro corretto funzionamento è stato quindi verificato tramite i test di integrazione.

- I package `mytalk.shared` e `mytalk.client` sono package generati dal framework$_{|g|}$ GWT e contengono poche linee di codice, non testabili.

Da questo rapporto si può pertanto concludere che la quasi totalità del codice che ci si era prefissato di testare è stato verificato.

E.2 Metriche

Le metriche che verranno considerate per le misurazioni sono riportate nella seguente tabella.

Metrica	Descrizione
TLOC	Total Lines of Code
MLOC	Method Lines of Code
NOC	Number of Classes
NOM	Number of Methods
VG M	McCabe Cyclomatic Complexity (Mean)
VG Sd	McCabe Cyclomatic Complexity (Standard Deviation)
LCOM M	Lack of Cohesion of Methods (Mean)
LCOM Sd	Lack of Cohesion of Methods (Standard Deviation)

Tabella 15: Descrizione metriche considerate

E.2.1 Parte Amministratore

Package	TLOC	MLOC	NOC	NOM
mytalk.client.view.administrator	432	227	3	29
mytalk.client.presenter.administrator.logicAdmin	229	109	3	27
mytalk.client.presenter.administrator.serverComUser	161	94	1	16
mytalk.client.presenter.administrator.common	36	22	1	0
mytalk.client.model.administrator.localDataAdmin	29	14	1	0
mytalk.client	22	1	1	1
mytalk.server	27	15	1	2
mytalk.shared	9	4	1	0

Tabella 16: TLOC, MLOC, NOC e NOM MyTalk Amministratore

Package	VG M	VG Sd	LCOM M	LCOM Sd
mytalk.client.view.administrator	1,138	0,433	0,636	0,157
mytalk.client.presenter.administrator.logicAdmin	1,556	0,916	0,289	0,287
mytalk.client.presenter.administrator.serverComUser	1,625	1,166	0,889	0
mytalk.client.presenter.administrator.common	2,75	0,433	0	0
mytalk.client.model.administrator.localDataAdmin	1,6	0,8	0	0
mytalk.client	1	0	0	0
mytalk.server	2	0	0	0
mytalk.shared	2	0	0	0

Tabella 17: VG e LCOM MyTalk Amministratore

E.2.2 Parte Server

Package	TLOC	MLOC	NOC	NOM
mytalk.server.model.dao	602	406	5	43
mytalk.server.presenter	148	68	1	0
mytalk.server.presenter .administrator.logicAdmin	174	106	3	11
mytalk.server.presenter.user.logicUser	366	267	3	21

Tabella 18: TLOC, MLOC, NOC e NOM MyTalk Server

Package	VG M	VG Sd	LCOM M	LCOM Sd
mytalk.server.model.dao	3,114	2,258	0,2	0,245
mytalk.server.presenter	3,167	2,192	0	0
mytalk.server.presenter .administrator.logicAdmin	2,455	2,311	0,167	0,236
mytalk.server.presenter.user.logicUser	2,857	3,44	0,426	0,309

Tabella 19: VG e LCOM MyTalk Server

E.2.3 Parte Utente

Package	TLOC	MLOC	NOC	NOM
mytalk.client.view.user	1097	501	11	83
mytalk.client.presenter.client.user.logicUser	584	290	5	65
mytalk.client.presenter.client.user.serverComUser	428	300	1	38
mytalk.client.presenter.client.user.communication	131	46	2	27
mytalk.client.presenter.client.user.logicUser.common	55	35	1	0
mytalk.client.model.client.localDataUser	29	14	1	0
mytalk.client	22	1	1	1
mytalk.shared(User)	9	4	1	0

Tabella 20: TLOC, MLOC, NOC e NOM MyTalk Utente

Package	VG M	VG Sd	LCOM M	LCOM Sd
mytalk.client.view.user	1,277	0,7	0,42	0,397
mytalk.client.presenter.client.user.logicUser	1,554	0,895	0,531	0,267
mytalk.client.presenter.client.user.serverComUser	1,85	1,388	0,845	0
mytalk.client.presenter.client.user.communication	1,036	0,186	0,375	0,375
mytalk.client.presenter.client.user.logicUser.common	2,571	0.495	0	0
mytalk.client.model.client.localDataUser	1,6	0,8	0	0
mytalk.client	1	0	0	0
mytalk.shared	2	0	0	0

Tabella 21: VG e LCOM MyTalk Utente

E.3 Rapporto misurazioni con ApacheBench

In questa sezione mostreremo le misurazioni relative alla performance del server effettuate mediante `ApacheBench`

E.3.1 Parte Amministratore

Descrizione	Misurazione
Tempo impiegato per i test (in s)	0,1794
Richieste al secondo	5609,517
Tempo per richiesta (in ms)	17,9267
Tempo per richiesta (media di tutte le richieste simulatanee) [in ms]	0,1794
Velocità di trasferimento	6255,929
Tempo di connessione minimo (in ms)	6,5
Tempo di connessione massimo (in ms)	48,6
Tempo di connessione medio (in ms)	16,3

Tabella 22: Misurazioni con Apache Bench di MyTalk Amministratore

E.3.2 Parte Utente

Descrizione	Misurazione
Tempo impiegato per i test (in s)	0,2261
Richieste al secondo	5108,233
Tempo per richiesta (in ms)	22,6151
Tempo per richiesta (media di tutte le richieste simulatanee) [in ms]	0,2261
Velocità di trasferimento	5646,992
Tempo di connessione minimo (in ms)	7,4
Tempo di connessione massimo (in ms)	59,9
Tempo di connessione medio (in ms)	18,3

Tabella 23: Misurazioni con Apache Bench di MyTalk Utente

E.4 Rapporto bug

In questa sezione illustreremo i bug trovati mediante il plugin di Eclipse `FindBugs`

E.4.1 Parte Amministratore

Errore	Metodo	Corretto
Dead store to local variable	MyTalkAdmin.onModuleLoad()	Si
Methods ignores return value	StatisticLogic.searchIP() StatisticLogic.searchUser() StatisticLogic.searchGrade() StatisticLogic.searchDay()	Si Si Si Si

Tabella 24: Bug trovati tramite FindBugs di MyTalk Amministratore

E.4.2 Parte Server

Errore	Metodo	Corretto
Method call passes null for nonnull parameter	DataAccessObject.updateItem()	falso positivo
Method concatenates strings using + in a loop	DataAccessObject.makeGetQuery() DataAccessObject.makeUpdateQuery() DataAccessObject.makeInsertQuery() DataAccessObject.makeDeleteQuery()	No No No No
Should be a static inner class	DataAccessObject.DBDataAccess WSAdmin.WebSocket WSUser.WebSocket	No No No

Tabella 25: Bug trovati tramite FindBugs di MyTalk Server

I bug che riguardano l'errore `Method concatenates strings using + in a loop` non sono stati corretti in quanto segnalano che sarebbe più efficiente concatenare le stringhe mediante il metodo append() e non mediante l'operatore `+`. I bug che riguardano l'errore `Should be a static inner class` non sono stati corretti in quanto le classi interne segnalate nel nostro caso non possono essere statiche.

E.4.3 Parte Utente

Errore	Metodo	Corretto
Dead store to local variable	MyTalk.onModuleLoad()	Si
Method invokes inefficient new String() constructor:	CommunicationLogic	Si

Tabella 26: Bug trovati tramite FindBugs di MyTalk User