AtAVi

Piano di qualifica v3.0.0

Sommario
Documento contenente le strategie adottate dal gruppo Co.Code per garantire la qualità del prodotto AtAVi.

Versione	3.0.0
Data di redazione	2017-04-08
Redazione	Pier Paolo Tricomi
	Andrea Magnan
Verifica	Mattia Bottaro
Approvazione	Luca Bertolini
Uso	Esterno
Distribuzione	prof. Tullio Vardanega
	prof. Riccardo Cardin
	Zero12

Diario delle modifiche

Versione	Riepilogo	Autore	Ruolo	Data
3.0.0	Approvazione	Luca Bertolini	*Responsabile*	2017-04-08
2.2.1	Correzione sezioni segnalate dalla verifica	Andrea Magnan	*Amministratore*	2017-04-04
2.2.0	Verifica	Mattia Bottaro	*Verificatore*	2017-04-04
2.1.2	Stesura resoconto RP	Andrea Magnan	*Amministratore*	2017-04-03
2.1.1	Correzione sezioni segnalate dalla verifica	Pier Paolo Tricomi	*Progettisti*	2017-04-03
2.1.0	Verifica	Mattia Bottaro	*Verificatore*	2017-04-03
2.0.4	Aggiunti test di integrazione	Pier Paolo Tricomi	*Progettisti*	2017-04-02
2.0.3	Aggiunti test di unità	Andrea Magnan	*Progettisti*	2017-03-30
2.0.2	Aggiunti test di validazione e di sistema	Pier Paolo Tricomi	*Progettisti*	2017-03-27
2.0.1	Stesura appendice ”Test”	Andrea Magnan	*Progettisti*	2017-03-26
2.0.0	Approvazione	Nicola Tintorri	*Responsabile*	2017-01-31
1.2.1	Correzione sezioni segnalate dalla verifica	Luca Bertolini	*Amministratore*	2017-01-30
1.2.0	Verifica	Simeone Pizzi	*Verificatore*	2017-01-30
1.1.3	Stesura appendici A,B	Andrea Magnan	*Amministratore*	2017-01-29
1.1.2	Stesura sezione ”Qualità di prodotto”	Luca Bertolini	*Amministratore*	2017-01-28
1.1.1	Correzione sezioni segnalate dalla verifica	Luca Bertolini	*Amministratore*	2017-01-27
1.1.0	Verifica	Simeone Pizzi	*Verificatore*	2017-01-26
1.0.2	Stesura sezione ”Qualità di processo”	Andrea Magnan	*Amministratore*	2017-01-26
1.0.1	Eliminate sezioni ”Visione generale della strategia di gestione della qualità”, ”La strategia di gestione della qualità nel dettaglio”, appendici A, B, C, D	Andrea Magnan	*Amministratore*	2017-01-24
1.0.0	Approvazione	Luca Bertolini	*Responsabile*	2017-01-08

Versione	Riepilogo	Autore	Ruolo	Data
0.2.2	Inseriti dati resoconto AR	Andrea Magnan	*Amministratore*	2017-01-07
0.2.1	Correzione sezioni segnalate dalla verifica	Andrea Magnan	*Amministratore*	2016-12-24
0.2.0	Verifica	Pier Paolo Tricomi	*Verificatore*	2016-12-24
0.1.5	Aggiunta struttura resoconto AR	Andrea Magnan	*Amministratore*	2016-12-23
0.1.4	Aggiunta appendice A	Nicola Tintorri	*Amministratore*	2016-12-23
0.1.3	Conclusa stesura sezione 3	Nicola Tintorri	*Amministratore*	2016-12-22
0.1.2	Aggiunte appendici B	Andrea Magnan	*Amministratore*	2016-12-22
0.1.1	Correzione sezioni segnalate dalla verifica	Andrea Magnan	*Amministratore*	2016-12-22
0.1.0	Verifica	Pier Paolo Tricomi	*Verificatore*	2016-12-22
0.0.5	Inizio stesura sezione 3	Nicola Tintorri	*Amministratore*	2016-12-21
0.0.4	Conclusa stesura sezione 2	Andrea Magnan	*Amministratore*	2016-12-21
0.0.3	Inizio stesura sezione 2	Andrea Magnan	*Amministratore*	2016-12-20
0.0.2	Stesura introduzione	Nicola Tintorri	*Amministratore*	2016-12-19
0.0.1	Inizio stesura documento	Nicola Tintorri	*Amministratore*	2016-12-19

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il documento ha lo scopo di definire gli obiettivi di qualità e le strategie che il gruppo Co.Code adotterà per raggiungerli. Verrà inoltre illustrato come il gruppo affronterà le varie fasi di *verifica*$_g$ per poter garantire il miglior risultato qualitativo possibile.

1.2 Scopo del prodotto

Si vuole creare un'applicazione web che permetta ad un ospite, in visita all'ufficio di Zero12, di interrogare un assistente virtuale per annunciare la propria presenza, avvisare l'interessato del suo arrivo sul sistema di comunicazione aziendale (*Slack*$_g$) e nel frattempo essere intrattenuto con varie attività.

1.3 Glossario

Allo scopo di evitare ogni ambiguità nel linguaggio e rendere più semplice e chiara la comprensione dei documenti, viene allegato il "*Glossario v1.0.0*". Le parole in esso contenute sono scritte in corsivo e marcate con una 'g' a pedice (p.es. *Parola*$_g$).

1.4 Riferimenti

1.4.1 Normativi

- Norme di *progetto*$_g$: "*Norme di Progetto v1.0.0*";

1.4.2 Informativi

- Piano di progetto: "*Piano di Progetto v1.0.0*";
- Slide del corso di Ingegneria del *software*$_g$ - Qualità del software : http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L10.pdf;
- Slide del corso di Ingegneria del software - Qualità di processo : http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L11.pdf;
- Slide del corso di Ingegneria del software - Analisi dinamica : http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L14.pdf;
- Indice Gulpease: https://it.wikipedia.org/wiki/Indice_Gulpease;
- Standard ISO/IEC 9126:2001: https://en.wikipedia.org/wiki/ISO/IEC_9126;
- Capability Maturity Model (*CMM*$_g$): https://en.wikipedia.org/wiki/Capability_Maturity_Model;
- Plan-Do-Check-Act (*PDCA*$_g$): https://en.wikipedia.org/wiki/PDCA.

2 Qualità di processo

Da processi scadenti derivano prodotti scadenti. La qualità di processo è quindi un fattore indispensabile per garantire la qualità dei prodotti. Assicurarla, inoltre, permette di:

- favorire l'ottimizzazione delle risorse;
- migliorare la stima dei rischi;
- ridurre i costi.

Per garantire la qualità di processo abbiamo deciso di adottare il Capability Maturity Model (CMM_g) che definisce una scala, suddivisa in cinque livelli, per misurarne la maturità.
Le misurazioni ottenute vengono utilizzate all'interno della strategia di miglioramento continuo della qualità, realizzata tramite Plan-Do-Check-Act ($PDCA_g$).
Per maggiori informazioni su CMM e PDCA consultare le rispettive appendici A e B.
Inoltre, sono stati individuati dallo standard ISO/IEC 12207:2008 i processi ritenuti più importanti nel *ciclo di vita*$_g$ del *prodotto*$_g$. Per ciascuno di essi sono stati individuati gli obiettivi di qualità e i rispettivi intervalli di accettabilità e ottimalità.
Per quantificare gli obiettivi di qualità vengono utilizzate delle metriche, descritte nel documento "*Norme di Progetto v2.0.0*".
Viene assegnato un codice identificativo ad ogni obiettivo al fine di semplificarne il tracciamento. Il metodo di denominazione è descritto nel documento "*Norme di Progetto v2.0.0*".

2.1 Infrastructure Management Process

Questo processo si occupa di mantenere, monitorare e modificare l'infrastruttura per assicurare che essa continui ad eseguire i servizi necessari allo svolgimento del *progetto*$_g$. Con infrastruttura si intendono elementi hardware, *software*$_g$, metodi, strumenti, tecniche e standard impiegati nello sviluppo del prodotto.

2.1.1 Strategie

Per tutto l'arco del progetto l'infrastruttura dovrà possedere le seguenti caratteristiche:

- tutti gli strumenti e le procedure per utilizzarli verranno descritti esaustivamente nel documento "*Norme di Progetto v2.0.0*";
- la piattaforma PragmaDB sarà disponibile ogni qual volta un componente del gruppo ne richiederà l'accesso;
- i dati contenuti in PragmaDB saranno sempre coerenti e aggiornati;
- la piattaforma PragmaDB estenderà le proprie funzionalità in base alle esigenze del progetto;

2.1.2 Obiettivi di qualità

2.1.2.1 Disponibilità PragmaDB - OPC1

Indica la disponibilità di utilizzo della piattaforma di PragmaDB rispetto alla richiesta. Questo valore viene espresso in percentuale.

- **Metrica utilizzata**: percentuale di accessi avvenuti correttamente a PragmaDB (MPC1);
- **Soglia di accettabilità**: $80\% \leq X \leq 100\%$;
- **Soglia di ottimalità**: $90\% \leq X \leq 100\%$;

2.2 Project Planning, Assessment & Control Process

Questo processo (derivante dall'unione dei processi Project Planning Process e Project Assessment & Control Process) si occupa del necessario per la pianificazione del progetto il quale deve contenere la scelta del modello del ciclo di vita del prodotto, la pianificazione dei tempi e costi da sostenere, la descrizione dei compiti e delle attività associate, l'allocazione dei compiti e delle responsabilità e le metriche atte a misurare lo stato del progetto rispetto la pianificazione.

2.2.1 Strategie

Lo sviluppo del progetto dovrà seguire la pianificazione, in particolare:

- il progetto dovrebbe rispettare i tempi indicati nel documento "*Piano di Progetto v2.0.0*";
- il progetto dovrebbe rispettare i costi indicati nel documento "*Piano di Progetto v2.0.0*";
- ruoli e compiti verranno descritti esaustivamente nel documento "*Piano di Progetto v2.0.0*".

2.2.2 Obiettivi di qualità

2.2.2.1 Rispetto dei tempi - OPC2

Indica se i tempi pianificati sono stati rispettati.

- **Metrica utilizzata**: Schedule Variance (MPC2);
- **Soglia di accettabilità**: $\geq -5\%$;
- **Soglia di ottimalità**: $\geq 0\%$.

2.2.2.2 Rispetto dei costi - OPC3

Indica se i costi pianificati, in data corrente, sono rispettati.

- **Metrica utilizzata**: Cost Variance percentuale (MPC3);
- **Soglia di accettabilità**: $\geq -10\%$;
- **Soglia di ottimalità**: $\geq 0\%$.

Eventuali costi non accettabili dovranno essere compensati entro la fine dell'attività di progetto in quanto non è permesso eccedere i costi preventivati oltre la soglia definita.

2.3 Risk Management Process

Questo processo identifica, analizza, tratta e monitora continuamente i rischi che possono sorgere nel ciclo di vita del progetto.

2.3.1 Strategie

Il gruppo dovrà gestire correttamente i rischi, in particolare:

- all'inizio della fase di progetto, i rischi devono essere individuati e descritti nel documento "*Piano di Progetto v2.0.0*";
- per ogni rischio, devono essere definite strategie per il riconoscimento e il trattamento;
- all'inizio di ogni periodo, l'analisi dei rischi permetterà l'individuazione di eventuali nuovi rischi;

- il livello di probabilità che i rischi si presentino dovrà sempre essere tenuto sotto controllo.

2.3.2 Obiettivi di qualità

2.3.2.1 Rischi non preventivati - OPC4

Evidenzia il numero dei rischi presentati e non preventivati nell'arco del progetto; un valore molto alto potrebbe indicare una povera analisi dei rischi.

- **Metrica utilizzata**: numero dei rischi non preventivati (MPC4);
- **Soglia di accettabilità**: 0 - 3;
- **Soglia di ottimalità**: 0.

2.4 System/Software Requirements Analysis Process

Questo processo si occupa di trasformare le idee del *proponente*$_{\text{g}}$ in un insieme di requisiti tecnici atti a guidare la progettazione del *sistema*$_{\text{g}}$.

2.4.1 Strategie

I requisiti identificati dal gruppo e successivamente inseriti nel documento "*Analisi dei Requisiti v2.0.0*" dovranno possedere le seguenti caratteristiche:

- dovranno essere inseriti nella piattaforma PragmaDB;
- si dovrà tenere traccia delle fonti da cui sono stati ricavati;
- si dovrà tenere traccia della loro implementazione;
- dovranno essere approvati dal proponente;
- nessun requisito dovrà risultare superfluo o ambiguo.

2.4.2 Obiettivi di qualità

2.4.2.1 Requisiti obbligatori soddisfatti - OPC5

Indica il numero dei requisiti obbligatori soddisfatti, espresso in percentuale.

- **Metrica utilizzata**: Numero dei requisiti obbligatori soddisfatti (MPC5);
- **Soglia di accettabilità**: 100%;
- **Soglia di ottimalità**: 100%.

2.4.2.2 Requisiti desiderabili soddisfatti - OPC6

Indica il numero dei requisiti desiderabili soddisfatti, espresso in percentuale.

- **Metrica utilizzata**: Numero dei requisiti desiderabili soddisfatti (MPC6);
- **Soglia di accettabilità**: 70%;
- **Soglia di ottimalità**: 100%.

2.4.2.3 Requisiti facoltativi soddisfatti - OPC7

Indica il numero dei requisiti facoltativi soddisfatti, espresso in percentuale.

- **Metrica utilizzata**: Numero dei requisiti facoltativi soddisfatti (MPC7);
- **Soglia di accettabilità**: 0%;
- **Soglia di ottimalità**: 100%.

2.5 System/Software Architectural Design Process

Questo processo si occupa di identificare la corrispondenza tra requisiti di sistema ed elementi del sistema.

2.5.1 Strategie

L'architettura ottenuta svolgendo le attività di questo processo dovrà possedere le seguenti caratteristiche:

- il sistema dovrà presentare basso accoppiamento ed alta coesione;
- ogni componente dovrà essere progettato puntando su incapsulamento, modularizzazione e riuso di codice;
- per ogni elemento del sistema dovrà essere possibile tracciare il requisito associato.

2.5.2 Obiettivi di qualità

2.5.2.1 Structural Fan-In - OPC8

In riferimento ad un modulo del software, indica quanti altri moduli lo utilizzano durante la loro esecuzione; tale indicazione permette di stabilire il livello di riuso implementato.

- **Metrica utilizzata**: SF-IN (MPC8);
- **Soglia di accettabilità**: ≥ 0;
- **Soglia di ottimalità**: ≥ 2;

2.5.2.2 Structural Fan-Out - OPC9

In riferimento ad un modulo del software, indica quanti moduli vengono utilizzati durante la sua esecuzione; tale indicazione permette di stabilire il livello di accoppiamento implementato.

- **Metrica utilizzata**: SF-OUT (MPC9);
- **Soglia di accettabilità**: 0 - 6;
- **Soglia di ottimalità**: 0 - 2.

2.6 Software Detailed Design Process

Questo processo si occupa di fornire, dato un sistema, una sua progettazione di dettaglio che permetta codifica ed esecuzione di test.

2.6.1 Strategie

Le attività di questo processo dovranno possedere le seguenti caratteristiche:

- il livello di dettaglio della progettazione dovrà guidare la codifica e l'esecuzione dei test senza bisogno di informazioni aggiuntive;
- la progettazione di dettaglio, che include architettura di basso livello, relazioni fra le varie unità software concepite e la definizione dettagliata delle interfacce dovrà essere esposta chiaramente nel documento "*Definizione di Prodotto v1.0.0*";
- per ogni elemento dell'architettura a basso livello dovrà essere possibile tracciare il requisito associato.

2.6.2 Obiettivi di qualità

2.6.2.1 Numero di metodi per classe - OPC10

Indica il numero di metodi definiti in una classe; un valore molto alto potrebbe indicare una cattiva decomposizione delle funzionalità a livello di progettazione.

- **Metrica utilizzata**: numero di metodi per classe (MPC10);
- **Soglia di accettabilità**: 1 - 10;
- **Soglia di ottimalità**: 1 - 8.

2.6.2.2 Numero di parametri per metodo - OPC11

Indica il numero di parametri passati ad un metodo; un valore molto alto potrebbe indicare un metodo troppo complesso e ulteriormente scomponibile in metodi più semplici.

- **Metrica utilizzata**: numero di parametri per metodo (MPC11);
- **Soglia di accettabilità**: 0 - 6;
- **Soglia di ottimalità**: 0 - 4.

2.7 Software Construction Process

Questo processo si occupa di produrre unità di software eseguibili che riflettono la progettazione effettuata.

2.7.1 Strategie

Le unità software prodotte dovranno rispettare le seguenti caratteristiche:

- il codice dovrà risultare facilmente manutenibile;
- il codice prodotto dovrà essere facilmente comprensibile e testabile;
- per ogni unità di software dovrà essere possibile tracciare il requisito e l'elemento architetturale associato.

2.7.2 Obiettivi di qualità

2.7.2.1 Complessità ciclomatica - OPC12

Indica la complessità di funzioni, moduli, metodi o classi di un programma. Alti valori di complessità ciclomatica implicano una ridotta manutenibilità del codice.

- **Metrica utilizzata**: indice di complessità ciclomatica (MPC12);
- **Soglia di accettabilità**: 1 - 20;
- **Soglia di ottimalità**: 1 - 10.

2.7.2.2 Livelli di annidamento - OPC13

Indica il numero di procedure e funzioni annidate, ovvero richiamate all'interno di altre procedure o funzioni. Più livelli di annidamento potrebbero rendere il codice di difficile comprensione e potrebbero portare a commettere errori logici durante la realizzazione del codice. In caso di troppi livelli di annidamento sarebbe opportuno riscrivere il metodo, o la funzione, affinchè sia facilmente comprensibile.

- **Metrica utilizzata**: numero di livelli di annidamento (MPC13);
- **Soglia di accettabilità**: 1 - ;
- **Soglia di ottimalità**: 1 - 2.

2.7.2.3 Linee di commento per linee di codice - OPC14

Indica il numero di linee di commento rispetto alle linee totali del codice. Un valore basso indica un codice poco comprensibile e, di conseguenza, difficilmente manutenibile. Un valore troppo alto indica un eccesso di commenti e un appesantimento dei file.

- **Metrica utilizzata**: percentuale linee di commento per linee di codice (MPC14);
- **Soglia di accettabilità**: 10% - 40%;
- **Soglia di ottimalità**: 20% - 30%.

2.7.2.4 Manutenibilità - OPC15

Permette di stabilire il grado di manutenibilità del codice prodotto.

- **Metrica utilizzata**: indice di manutenibilità (MPC15);
- **Soglia di accettabilità**: 10 - 100;
- **Soglia di ottimalità**: 20 - 100.

2.8 System/Software Integration Process

Questo processo si occupa di integrare le unità software tra loro, produrre software coerente con la progettazione e dimostrare che il prodotto soddisfi i requisiti identificati.

2.8.1 Strategie

Le attività previste da questo processo dovranno puntare a raggiungere un alto livello di automazione, in particolare:

- l'integrazione delle varie parti del sistema sarà completamente automatizzata utilizzando lo strumento di continuous integration Jenkins, come definito nel documento "*Norme di Progetto v2.0.0*";
- il livello di integrazione raggiunto del sistema sarà sempre consultabile grazie all'utilizzo dello strumento di continuous integration Jenkins, come definito nel documento "*Norme di Progetto v2.0.0*".

2.8.2 Obiettivi di qualità

2.8.2.1 Componenti integrate - OPC16

Indica il numero di componenti definite in progettazione che sono attualmente implementate e integrate nel sistema. Nel nostro caso, tutte le componenti progettate andranno a costituire il sistema.

- **Metrica utilizzata**: percentuale di componenti integrate nel sistema (MPC16);
- **Soglia di accettabilità**: 100%;
- **Soglia di ottimalità**: 100%.

2.9 System/Software Qualification Testing Process

Questo processo si occupa di assicurare che ogni requisito individuato sia stato implementato nel prodotto.

2.9.1 Strategie

Questo processo deve possedere le seguenti caratteristiche:

- le attività di test previste dal processo verranno svolte su un sistema le cui componenti sono verificate e correttamente integrate fra loro;
- le attività di test dovranno raggiungere il maggior livello di automazione nell'esecuzione tramite lo strumento di continuous integration Jenkins;
- le attività di test dovranno essere eseguite in numero sufficiente in modo tale da garantire un'ottima copertura dei requisiti;
- il software dovrà implementare tutti i requisiti obbligatori.

2.9.2 Obiettivi di qualità

2.9.2.1 Test di unità eseguiti - OPC17

Indica il numero di test di unità eseguiti tra quelli definiti dal gruppo.

- **Metrica utilizzata**: percentuale di test di unità eseguiti (MPC17);
- **Soglia di accettabilità**: 90% - 100%;
- **Soglia di ottimalità**: 100%.

2.9.2.2 Test di integrazione eseguiti - OPC18

Indica il numero di test di integrazione eseguiti tra quelli definiti dal gruppo.

- **Metrica utilizzata**: percentuale di test di integrazione eseguiti (MPC18);
- **Soglia di accettabilità**: 70% - 100%;
- **Soglia di ottimalità**: 80% - 100%.

2.9.2.3 Test di sistema eseguiti - OPC19

Indica il numero di test di sistema eseguiti in modo automatico tra quelli definiti dal gruppo.

- **Metrica utilizzata**: percentuale di test di sistema eseguiti (MPC19);
- **Soglia di accettabilità**: 70% - 100%;
- **Soglia di ottimalità**: 80% - 100%.

2.9.2.4 Test di validazione eseguiti - OPC20

Indica il numero di test di validazione eseguiti manualmente tra quelli definiti dal gruppo.

- **Metrica utilizzata**: percentuale di test di validazione eseguiti (MPC20);
- **Soglia di accettabilità**: 100%;
- **Soglia di ottimalità**: 100%.

2.9.2.5 Test superati - OPC21

Indica il numero di test superati.

- **Metrica utilizzata**: percentuale dei test superati (MPC21);
- **Soglia di accettabilità**: 90% - 100%;
- **Soglia di ottimalità**: 100%.

2.10 Software Verification Process

Questo processo si occupa di controllare che il software prodotto soddisfi correttamente i requisiti identificati.

2.10.1 Strategie

Questo processo deve possedere le seguenti caratteristiche:

- la documentazione verrà verificata mediante inspenction;
- i test dinamici sui vari elementi saranno al più possibile automatizzabili;
- i test dinamici sui vari elementi del software copriranno gran parte delle possibili casistiche di utilizzo;
- l'esito di ogni test deve essere tracciabile.

2.10.2 Obiettivi di qualità

2.10.2.1 Branch coverage - OPC22

Indica il numero di rami decisionali percorsi nei test utilizzati. Questo obiettivo assicura che i branch derivanti da una condizione siano eseguiti da almeno un test.

- **Metrica utilizzata**: percentuale di rami decisionali percorsi (MPC22);
- **Soglia di accettabilità**: 75% - 100%;
- **Soglia di ottimalità**: 85% - 100%.

2.10.2.2 Function coverage - OPC23

Indica il numero di funzioni che sono state chiamate nei test utilizzati.

- **Metrica utilizzata**: numero di funzioni chiamate nei test (MPC23);
- **Soglia di accettabilità**: 70% - 100%;
- **Soglia di ottimalità**: 80% - 100%.

2.10.2.3 Statement coverage - OPC24

Indica il numero di istruzioni che sono state eseguite nei test utilizzati. Maggiore è il valore maggiore sarà il numero di statement eseguiti almeno una volta dai test.

- **Metrica utilizzata**: numero di istruzioni nei test (MPC24);
- **Soglia di accettabilità**: 75% - 100%;
- **Soglia di ottimalità**: 85% - 100%.

3 Qualità di prodotto

È prevista la realizzazione di due tipologie di prodotto: documenti e *software*$_{\text{g}}$. Per ciascuno di essi sono stati individuati degli obiettivi di qualità e i rispettivi intervalli di accettabilità e ottimalità. Per quantificare gli obiettivi di qualità vengono utilizzate delle metriche, descritte nel documento "*Norme di Progetto v2.0.0*".
Viene assegnato un codice identificativo ad ogni obiettivo al fine di semplificarne il tracciamento. Il metodo di denominazione è descritto nel documento "*Norme di Progetto v2.0.0*".

3.1 Documenti

I documenti prodotti dal gruppo Co.Code si dividono in interni ed esterni. I primi definiscono le strategie, gli strumenti e il metodo di lavoro (ways of working) del gruppo affinchè i membri realizzino prodotti simili tra loro secondo delle regole definite. I secondi definiscono tutto ciò che riguarda il software prodotto, partendo dalla progettazione fino a giungere all'analisi dei requisiti e la definizione di prodotto.
Poichè i documenti interni devono essere letti e compresi da tutti i membri del gruppo e quelli esterni devono essere comprensibili affinchè *proponente*$_{\text{g}}$ e committente siano informati correttamente, il gruppo ha deciso di perseguire le strategie e gli obiettivi di qualità definiti di seguito.

3.1.1 Strategie

Tutti i documenti prodotti devono avere le seguenti caratteristiche:

- devono essere comprensibili da utenti con almeno la licenza superiore;
- i termini con significato ambiguo o poco chiaro dovranno essere inseriti nel "*Glossario v2.0.0*";
- saranno sempre aggiornati e allineati allo stato attuale del processo di sviluppo;
- dovranno essere dotati di numero di versione e diario delle modifiche.

3.1.2 Obiettivi di qualità - OPDD1

3.1.2.1 Leggibilità e comprensibilità

Indica il livello di leggibilità e comprensibilità del documento. Maggiore è il valore dell'indice maggiore sarà la leggibilità del documento.

- **Metrica utilizzata**: indice Gulpease (MPDD1);
- **Soglia di accettabilità**: 40 - 100;
- **Soglia di ottimalità**: 60 - 100.

3.2 Software

Sono state individuate dallo standard ISO/IEC 9126:2001 le principali caratteristiche che il software deve soddisfare. Sulla base di esse sono state definiti gli obiettivi di qualità e i relativi intervalli.

3.2.1 Functionality

È la capacità del prodotto software di fornire le funzionalità definite nei requisiti individuati nel documento "*Analisi dei Requisiti v2.0.0*".

3.2.1.1 Strategie

Il software deve soddisfare le seguenti caratteristiche:

- **Suitability**: fornire un appropriato insieme di funzionalità in base alle richieste dell'utente;
- **Accuracy**: fornire i corretti risultati con un adeguato grado di precisione;
- **Security**: proteggere le informazioni e i dati affinchè solo gli utenti autorizzati possano modificarli e/o leggerli.

3.2.1.2 Obiettivi di qualità

3.2.1.2.1 Implementazione funzionale - OPDS1

Indica quanti requisiti funzionali sono stati implementati.

- **Metrica utilizzata**: completezza dell'implementazione funzionale (MPDS1);
- **Soglia di accettabilità**: 100%;
- **Soglia di ottimalità**: 100%.

3.2.1.2.2 Accuratezza rispetto alle attese - OPDS2

Indica quanti risultati sono concordi alle attese.

- **Metrica utilizzata**: percentuale risultati concordi alle attese (MPDS2);
- **Soglia di accettabilità**: 90% - 100%;
- **Soglia di ottimalità**: 100%.

3.2.1.2.3 Controllo degli accessi - OPDS3

Indica quante operazioni illegali non sono state bloccate. Valori grandi indicano un $sistema_{\text{g}}$ poco sicuro e facilmenete violabile. Valori bassi sono d'obbligo per poter garantire la sicurezza dei dati.

- **Metrica utilizzata**: percentuale operazioni illegali non bloccate (MPDS3);
- **Soglia di accettabilità**: 0% - 10%;
- **Soglia di ottimalità**: 0%.

3.2.2 Reliability

È la capacità del prodotto software di svolgere correttamente le sue funzioni in qualunque situazione, anche in caso di situazioni anomale.

3.2.2.1 Strategie

Il software deve soddisfare le seguenti caratteristiche:

- **Maturity**: evitare che si verifichino malfunzionamenti, operazioni illegali e risultati errati in seguito ad errori;
- **Fault tolerance**: mantenere un certo livello di performance nonostante siano presenti errori e guasti o come conseguenza di un uso scorretto dell'applicativo.

3.2.2.2 Obiettivi di qualità

3.2.2.2.1 Densità di failure - OPDS4

Indica quante operazioni di testing sono concluse in failure.

- **Metrica utilizzata**: percentuale failure su test (MPDS4);
- **Soglia di accettabilità**: 0% - 10%;
- **Soglia di ottimalità**: 0%.

3.2.2.2.2 Blocco di operazioni non corrette - OPDS5

Indica quante funzionalità sono in grado di gestire correttamente gli errori che potrebbero verificarsi. Un valore alto è sinonimo di robustezza.

- **Metrica utilizzata**: numero di failure evitati (MPDS5);
- **Soglia di accettabilità**: 90% - 100%;
- **Soglia di ottimalità**: 100%;

3.2.3 Usability

È la capacità del prodotto software di essere capito, compreso e usato dall'utente.

3.2.3.1 Strategie

Il software deve soddisfare le seguenti caratteristiche:

- **Understandability**: permettere all'utente di capire il grado di conformità del software e il suo dominio applicativo;
- **Learnability**: permettere all'utente di capire come utilizzarlo;
- **Operability**: permettere all'utente di utilizzarlo e controllarlo;
- **Attractiveness**: essere piacevole all'utente che lo utilizza.

3.2.3.2 Obiettivi di qualità

3.2.3.2.1 Comprensibilità delle funzioni offerte - OPDS6

Indica quante funzionalità sono state comprese immediatamente dall'utente senza la consultazione del manuale. L'assistente virtuale dovrebbe essere il più user friendly possibile. Infatti, l'interazione dell'utente dovrebbe essere il più fluida possibile senza dover consultare il manuale ad ogni operazione da eseguire.

- **Metrica utilizzata**: percentuale delle funzionalità comprese (MPDS6);
- **Soglia di accettabilità**: 85% - 100%;
- **Soglia di ottimalità**: 95% - 100%.

3.2.3.2.2 Consistenza operazionale in uso - OPDS7

Indica quante funzionalità rispettano le aspettative dell'utente.

- **Metrica utilizzata**: percentuale di funzionalità conformi alle aspettative (MPDS7);
- **Soglia di accettabilità**: 80% - 100%;
- **Soglia di ottimalità**: 90% - 100%.

3.2.4 Efficiency

È la capacità del prodotto software di fornire le proprie funzioni in modo appropriato, in relazione alla quantità di risorse utilizzate.

3.2.4.1 Strategie

Il software deve soddisfare le seguenti caratteristiche:

- **Time behaviour**: svolgere le proprie funzioni in tempi adeguati;
- **Resource utilisation**: eseguire le proprie funzioni utilizzando un'appropriata quantità di risorse.

3.2.4.2 Obiettivi di qualità

3.2.4.2.1 Tempo di risposta - OPDS8

Indica il periodo temporale medio trascorso tra la richiesta al software di una determinata funzionalità e la risposta all'utente.

- **Metrica utilizzata**: tempo medio di risposta (MPDS8);
- **Soglia di accettabilità**: 0 - 20 secondi;
- **Soglia di ottimalità**: 0 - 7 secondi;

3.2.5 Maintainability

È la capacità del prodotto software di essere modificato, ovvero corretto, migliorato o adattato in base a cambiamenti negli ambienti, nei requisiti o nelle specifiche funzionali.

3.2.5.1 Strategie

Il software deve soddisfare le seguenti caratteristiche:

- **Analysability**: poter essere analizzato per poter individuare cause di errori e/o parti da modificare;
- **Changeability**: permettere cambiamenti in alcune sue parti;
- **Stability**: evitare comportamenti indesiderati in seguito a modifiche;
- **Testability**: permettere l'esecuzione di test per validare le modifiche effettuate.

3.2.5.2 Obiettivi di qualità

3.2.5.2.1 Capacità analisi di failure - OPDS9

Indica il numero di failure di cui sono state individuate le cause. Maggiore è il valore più facile sarà individuare gli errori nel software e determinare delle soluzioni per correggerli.

- **Metrica utilizzata**: percentuale di failure con cause individuate (MPDS9)
- **Soglia di accettabilità**: 65% - 100%;
- **Soglia di ottimalità**: 85% - 100%;

3.2.5.2.2 Impatto delle modifiche - OPDS10

Indica il numero di modifiche effettuate in risposta alle failure che ne hanno introdotte di nuove. Valori alti indicano l'individuazione di soluzioni scandenti per la risoluzione delle failure.

- **Metrica utilizzata**: percentuale di failure introdotte con modifiche (MPDS10);
- **Soglia di accettabilità**: 0% - 20%;
- **Soglia di ottimalità**: 0% - 10%.

A Capability Maturity Model

Il Capability Maturity Model (CMM) è stato ideato e introdotto inizialmente dal Dipartimento della Difesa statunitense, per poi essere acquisito, sviluppato e sponsorizzato dalla SEI (*Software*$_g$ Engineering Institute). Tale modello assume che la qualità del software dipende decisamente dal processo utilizzato per il suo sviluppo e per la successiva manutenzione, e consiste nell'applicare le migliori tecniche di gestione dei processi e del miglioramento della qualità. Si basa su:

- linee guida comuni per lo sviluppo e la manutenzione del software;
- struttura per la valutazione consistente dei livelli raggiunti.

A.1 Scopo

Lo scopo principale dell'adozione del modello in esame è quello di migliorare i processi di sviluppo del software in ottica di:

- miglioramento della qualità del software *prodotto*$_g$;
- aumento della produttività dell'organizzazione di sviluppo;
- riduzione dei tempi di sviluppo.

A.2 Struttura

Il CMM è costituito dalla seguente struttura:

- **Livelli di maturità**: Il modello definisce cinque livelli di maturità crescente del processo di sviluppo del software. Il più alto (il quinto) è uno stato ideale in cui i processi vengono sistematicamente gestiti da una combinazione di processi di ottimizzazione e di miglioramento continuo.
- **Aree chiave del processo**: identifica una serie di attività correlate che, se svolte collettivamente, realizzano un insieme di obiettivi considerati importanti;
- **Obiettivi**: indicano lo scopo, i confini e l'intento di ogni area chiave del processo;
- **Caratteristiche comuni**: includono le pratiche che implementano e regolamentano un'area chiave del processo. Ci sono cinque tipologie di caratteristiche comuni:
  - impegno nell'operare;
  - abilità nell'operare;
  - attività eseguite;
  - misurazioni ed analisi;
  - veriche dell'implementazione.
- **pratiche chiave**: descrivono gli elementi dell'infrastruttura e delle pratiche che contribuiscono maggiormente all'implementazione e la regolamentazione di un'area.

A.3 Livelli

I livelli di maturità che costituiscono il CMM sono:

- **Primo livello - iniziale (caotico)**: i processi che rientrano in questo livello sono disorganizzati. Il non essere sufficientemente definiti e documentati non permette loro di essere riutilizzati;

- **Secondo livello - ripetibile**: sono stabiliti processi base di gestione per tracciare i costi, la schedulazione delle attività e le funzionalità sviluppate. Il processo è stabilito per essere ripetibile.
- **Terzo livello - definito**: Il processo di sviluppo software, sia per la parte di gestione che per quella di sviluppo tecnico, è definito, documentato e standardizzato per il riutilizzo.
- **Quarto livello - gestito**: un organizzazione monitora e controlla i propri processi attraverso analisi e Data collection.
- **Quinto livello - ottimizzante**: i processi che rientrano in questo livello sono soggetti ad un continuo miglioramento delle proprie performance attraverso cambiamenti incrementali e miglioramenti tecnologici.

B PDCA

Il Plan-Do-Check-Act (PDCA), conosciuto anche come "Ciclo di Deming" o "Ciclo di miglioramento continuo", è un modello studiato per il miglioramento continuo della qualità in un'ottica a lungo raggio.
Questo modello permette di ricercare la qualità sui processi alla base del *prodotto*$_g$, e non sul prodotto stesso. Questo strumento permette di fissare degli obiettivi di miglioramento a partire dagli esiti delle misurazioni effettuate durante le varie attività di *verifica*$_g$. Una volta fissati gli obiettivi che si desiderano raggiungere, si iterano le quattro attività definite in seguito assicurando un incremento della qualità ad ogni ciclo.

Figura 1: Continuous quality improvement with PDCA

- **Plan - Pianificare:** consiste nel definire gli obiettivi di miglioramento e le strategie da utilizzare per raggiungere la qualità attesa, in dettaglio:
  - identificare il problema o i processi da migliorare raccogliendo dati attraverso misurazioni;
  - analizzare il problema in modo tale da capire quali sono gli effetti negativi definendone l'importanza e la priorità di intervento;
  - definire gli obiettivi di massima in modo chiaro e quantitativo, indicando i benefici ottenibili con il suo raggiungimento. Devono essere definiti anche i tempi, gli indicatori e gli strumenti di controllo.
- **Do - Eseguire:** consiste nell'esecuzione di ciò che è stato pianificato nel punto precedente e nella raccolta dati necessaria all'analisi effettuata nei punti successivi;
- **Check - Verificare:** consiste nel verificare l'esito del processo (per efficienza ed efficacia) confrontandolo con i risultati attesi, così da poter definire se si va nella direzione giusta. Vanno considerate metriche come la Schedule Variance e la completezza dei risultati attesi soddisfatti, vanno elaborati grafici e tabelle per avere una visione chiara di quanto rilevato. Una volta raggiunto l'obiettivo definito nella attività di Plan si può passare a quella di Act, mentre se questo non è soddisfatto è necessario ripetere un nuovo ciclo PDCA sullo stesso problema analizzando i vari stadi del ciclo precedente individuandone le cause del non raggiungimento dell'obiettivo stabilito;
- **Act - Agire:** si standardizza la soluzione individuata ed ogni membro del gruppo di lavoro viene formato e informato. Una volta terminato questo stadio si proseguirà nuovamente dallo stadio 1, con un nuovo problema.

Bisogna tener presente che se l'obiettivo è il miglioramento continuo, le attività devono essere analizzabili, ripetibili e tracciabili. Unendo queste tre caratteristiche è possibile individuare eventuali errori e correggerli.

C Test

Al fine di produrre *software*$_g$ di qualità, il gruppo ha strutturato dei test atti a verificare che le funzionalità del software *prodotto*$_g$ corrispondano alle attese. Tali test sono ottenuti dall'applicazione delle tecniche di analisi dinamica descritte nel documento "*Norme di Progetto v2.0.0*". Inoltre, devono possedere le seguenti caratteristiche:

- devono essere ripetibili al fine di fornire informazioni utili per poter eseguire operazioni di correzione, ove sia necessario;
- devono essere tracciabili al fine di classificare le informazioni ottenute per garantire una più facile consultazione;

Le tipologie di test che verranno eseguiti sono:

- **Test di *validazione*$_g$:** test che hanno lo scopo di verificare che tutte le funzionalità richieste dal *proponente*$_g$ siano soddisfatte. A questo scopo, attraverso una serie di azioni, si andrà a simulare il comportamento generale del software e dell'utente che interagisce con esso;
- **Test di unità:** test che hanno lo scopo di verificare il corretto funzionamento delle unità. Le unità, individuate durante la fase di progettazione, sono le più piccole parti del *sistema*$_g$ dotate di funzionamento proprio. Questo si traduce nel verificare metodi e classi scritte dai *Programmatori*;
- **Test di integrazione:** test che hanno lo scopo di verificare il corretto funzionamento delle varie componenti. In particolare, l'obiettivo è quello di testare le varie componenti prodotte dall'unione delle unità. Nel determinarli, è stato scelto l'approccio top-down, in maniera tale da sottoporre per prime le componenti di livello più alto ai test, integrandole fin da subito. Così facendo anche la logica di alto livello e il flusso di dati vengono sottoposti a test fin da subito; sarà perciò necessario simulare le componenti di livello più basso con degli stub. Una volta codificate, le componenti di più basso livello dovranno a loro volta essere integrate e testate. In questo modo, i difetti rilevati dai test saranno spesso attribuiti all'ultima componente aggiunta. In "*Definizione di Prodotto v1.0.0*" sono descritte come le componenti devono interagire tra loro;
- **Test di sistema:** test che hanno lo scopo di verificare il corretto funzionamento del prodotto software. Inoltre verranno verificate la sua robustezza in presenza di possibili malfunzionamenti e il suo comportamento di fronte a possibili violazioni;
- **Test di regressione:** test che hanno lo scopo di verificare che una modifica dell'implementazione del software non ne comprometta la qualità. Consistono nella ripetizione di test di unità o integrazione sul componente modificato.

C.1 Test di Validazione

I test di *validazione*$_g$ saranno identificati secondo quanto riportato nel documento "*Norme di Progetto v4.0.0*".

Id Test	Descrizione	Stato
TVFO1	L'utente deve *verifica*$_g$re che il *sistema*$_g$ riesca a riconoscerlo come ospite o possibile amministratore. All'utente viene richiesto di: • fornire nome e cognome;	*Non Implementato*

Id Test	Descrizione	Stato
TVFO1.1.2	L'utente deve verificare che il sistema ne permetta l'accesso all'area amministrativa tramite l'assistente virtuale. All'utente viene richiesto di: • comunicare i propri dati identificativi; • verificare che il sistema riconosca l'utente come un possibile amministratore non autenticato; • comunicare l'intento di volersi autenticare come amministratore; • comunicare la frase per lo Speaker Recognition; • verificare l'accesso all'area amministrativa.	*Non Implementato*
TVFO2.1	L'utente deve verificare che il sistema permetta la creazione di una nuova *direttiva*$_{\text{g}}$. All'utente viene richiesto di: • autenticarsi come amministratore; • comunicare l'intento di voler creare una nuova direttiva; • inserire il nome della direttiva; • inserire la funzione della direttiva; • inserire il target della direttiva; • confermare la creazione della direttiva; • verificare che la direttiva sia stata creata correttamente.	*Non Implementato*
TVFO2.1.1.6	L'utente deve verificare che, durante la creazione di una direttiva, l'inserimento di dati non validi (funzione o target della direttiva inesistenti) comporti la visualizzazione di un messaggio d'errore. All'utente viene richiesto di: • autenticarsi come amministratore; • comunicare l'intento di voler creare una nuova direttiva; • inserire il nome della direttiva; • inserire una funzione per la direttiva che non sia valida (inesistente); • inserire un target per la direttiva che non sia valido (inesistente); • verificare la comparsa di un messaggio d'errore.	*Non Implementato*
TVFO2.1.2	L'utente deve verificare che il sistema permetta l'eliminazione di una direttiva. All'utente viene richiesto di: • autenticarsi come amministratore; • comunicare l'intento di voler eliminare una direttiva; • comunicare il nome della direttiva da eliminare; • confermare l'eliminazione della direttiva; • verificare che la direttiva sia stata eliminata correttamente.	*Non Implementato*

Id Test	Descrizione	Stato
TVFO2.1.4	L'utente deve verificare che il sistema permetta la visualizzazione di una direttiva. All'utente viene richiesto di: • autenticarsi come amministratore; • comunicare l'intento di voler visualizzare una direttiva; • verificare che il sistema permetta la visualizzazione di nome, funzione, target,funzionalità e abilitazione della direttiva.	*Non Implementato*
TVFO2.2	L'utente deve verificare che il sistema permetta la modifica dei dati del proprio profilo. All'utente viene richiesto di: • autenticarsi come amministratore; • comunicare l'intento di voler modificare il proprio profilo; • comunicare nome e cognome; • confermare la modifica; • verificare che la modifica sia stata effettuata;	*Non Implementato*
TVFO3.1	L'utente deve verificare che sia possibile comunicare al sistema la persona che si desidera incontrare. All'utente viene richiesto di: • aver comunicato i propri dati al sistema ed essere riconosciuti come ospiti; • comunicare la persona che si desidera raggiungere; • verificare che il sistema abbia capito le informazioni comunicate;	*Non Implementato*
TVFO5	L'utente deve verificare che, nel caso in cui il sistema nel caso non riesca ad interpretare la risposta, chieda nuovamente l'informazione all'utente. All'utente viene richiesto di: • comunicare al sistema qualcosa che non può essere interpretato da esso; • verificare che il sistema richieda nuovamente l'informazione.	*Non Implementato*
TVFO7	L'utente deve verificare che, nel caso in cui esso sia già stato un ospite in passato, il sistema lo riconosca. All'utente viene richiesto di: • aver comunicato i propri dati al sistema ed essere riconosciuti come ospiti; • verificare che il sistema riconosca l'utente come qualcuno che è già stato un ospite in passato.	*Non Implementato*

Tabella 1: Test di Validazione

C.2 Test di Sistema

I test di sistema saranno identificati secondo quanto riportato nel documento "*Norme di Progetto v4.0.0*".

Id Test	Descrizione	Stato
TSFO1	Il sistema deve poter riconoscere un utente come ospite.	*Non Implementato*
TSFO1.1.2.1	Il sistema deve permettere all'utente di autenticarsi come amministratore tramite frase di riconoscimento.	*Non Implementato*
TSFO2.1.1	Il sistema deve permettere all'amministratore di poter creare una nuova direttiva.	*Non Implementato*
TSFO2.1.2	Il sistema deve permettere all'amministratore di poter eliminare una direttiva di cui ha i privilegi.	*Non Implementato*
TSFO2.1.4	Il sistema deve permettere all'amministratore di poter visualizzare le *direttive*$_g$ di cui ha i privilegi.	*Non Implementato*
TSFO2.2.1	L'amministratore deve poter modificare il nome e cognome del suo profilo.	*Non Implementato*
TSFO3.1	Il sistema deve permettere all'ospite di richiedere la persona desiderata.	*Non Implementato*
TSFO5	Il sistema deve richiedere nuovamente le informazioni nel caso in cui non fossero state comprese.	*Non Implementato*
TSFO7	Il sistema deve essere in grado di riconoscere ospiti già stati in visita all'azienda. In questo caso, l'assistente virtuale deve poter prevedere le sue necessità.	*Non Implementato*
TSFO8	Il sistema deve sollecitare la persona desiderata o eventualmente avvisare gli altri membri dell'azienda su richiesta dell'ospite.	*Non Implementato*
TSFO13	Il sistema deve comunicare nell'opportuno canale di *Slack*$_g$ le informazioni raccolte durante l'interazione con l'ospite.	*Non Implementato*
TSVO1.1	Vogliamo testare che il *software*$_g$ funzioni correttamente in un PC con sistema operativo Windows 7 o superiore.	*Non Implementato*
TSVO4	Le pagine HTML devono essere validate.	*Non Implementato*
TSVO5	I fogli di stile CSS devono essere validati.	*Non Implementato*
TSVO10	Vogliamo testare che il software funzioni correttamente con il *browser*$_g$ Google Chrome versione 53 o superiore.	*Non Implementato*

Tabella 2: Test di Sistema

C.3 Test di Integrazione

I test di integrazione saranno identificati secondo quanto riportato nel documento "*Norme di Progetto v4.0.0*".
Segue il diagramma di attività relativo ai test d'integrazione.

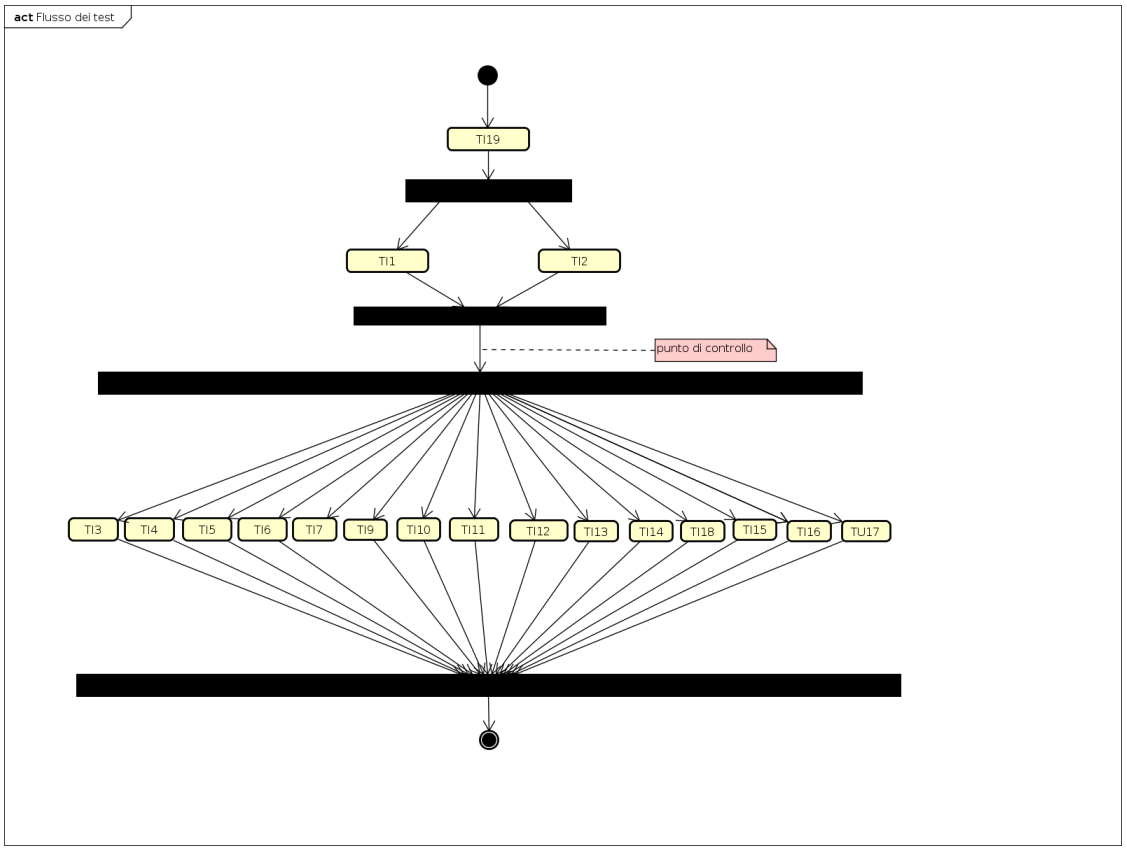

Figura 2: Diagramma di attività dei test d'integrazione

Id Test	Descrizione	Stato
TI1	Vogliamo verificare che `Recorder`, `Logic`, `Utility`, `Recorder`, `TTS`, `ConversationApp` e `ApplicationManager` interagiscano correttamente fra loro.	*Non Implementato*
TI2	Vogliamo verificare che `APIGateway`, `STT`, `VirtualAssistant`, `Users`, `Guests`, `Rules`, `Members`, `Conversations` e `Events` interagiscano correttamente tra di loro. Inoltre, vogliamo verificare che interagiscano correttamente con i servizi e librerie esterne AWS, Speaker Recognition, *Speech to text*$_g$ IBM Watson, api.ai, Slack e `WebAPI`.	*Non Implementato*

Id Test	Descrizione	Stato
TI3	Vogliamo verificare che le seguenti classi, contenute in `Client::ApplicationManager`, interagiscano tra loro correttamente: `ApplicationManager`*`Observer`*`g`, `ApplicationRegistryClient`, `ApplicationRegistryLocalClient`, `ApplicationLocalRegistry`, `Manager`, `State`, `Application`, `ApplicationPackage`.	*Non Implementato*
TI4	Vogliamo verificare che le seguenti classi, contenute in `Client::Logic`, interagiscano tra loro correttamente: `DataArrivedSubject`, `DataArrivedObservable`, `Logic`, `HttpError`, `HttpPromise`, `LogicObserver`.	*Non Implementato*
TI5	Vogliamo verificare che le seguenti classi, contenute in `Client::Recorder`, interagiscano tra loro correttamente: `Recorder`, `RecorderWorker`, `RecorderMsg`, `RecorderWorkerMsg`, `RecorderWorkerConfig`, `RecorderConfig`, `SpeechEndSubject`, `SpeechEndObservable`.	*Non Implementato*
TI6	Vogliamo verificare che le seguenti classi, contenute in `Client::TTS`, interagiscano tra loro correttamente: `TTSConfig`, `Player`, `PlayerObserver`.	*Non Implementato*
TI7	Vogliamo verificare che le seguenti classi, contenute in `Client::Utility`, interagiscano tra loro correttamente: `BoolSubject`, `BoolObservable`, `BoolObserver`.	*Non Implementato*
TI8	Vogliamo verificare che le seguenti classi, contenute in `Back-end::APIGateway`, interagiscano tra loro correttamente: `VocalAPI`, `Enrollement`.	*Non Implementato*
TI9	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Users`, interagiscano tra loro correttamente: `UsersDAODynamoDB`, `User`, `UsersService`.	*Non Implementato*
TI10	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Rules`, interagiscano tra loro correttamente: `Rule`, `RulesDAODynamoDB`, `RuleTarget`, `RuleTaskInstance`, `RulesService`, `TasksDAODynamoDB`, `Task`.	*Non Implementato*
TI11	Vogliamo verificare che le seguenti classi, contenute in `Back-end::VirtualAssistant`, interagiscano tra loro correttamente: `VAService`, `ApiAIVAAdapter`, `VAQuery`, `Agent`, `AgentDAODynamoDB`, `VAEventObject`, `Fulfillment`, `MsgObject`, `ButtonObject`.	*Non Implementato*
TI12	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Member`, interagiscano tra loro correttamente: `MembersSlackDAO`, `Member`.	*Non Implementato*
TI13	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Guests`, interagiscano tra loro correttamente: `Guest`, `GuestDAODynamoDB`.	*Non Implementato*
TI14	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Conversations`, interagiscano tra loro correttamente: `ConversationDAODynamoDB`, `Conversation`, `ConversationMsg`.	*Non Implementato*

Id Test	Descrizione	Stato
TI15	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Events`, interagiscano tra loro correttamente: `SNSRecord`, `SNSMessage`.	*Non Implementato*
TI16	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Notifications`, interagiscano tra loro correttamente: `NotificationChannel`, `Purpose`, `Topic`, `NotificationMessage`, `Attachment`, `Action`, `ConfirmationFields`.	*Non Implementato*
TI17	Vogliamo verificare che le seguenti classi, contenute in `Back-end::Utility`, interagiscano tra loro correttamente: `WebhookRequest`, `ProcessingResult`, `LamdaIdEvent`, `PathIdParam`.	*Non Implementato*
TI18	Vogliamo verificare che le seguenti classi, contenute in `Client::ConversationApp`, interagiscano tra loro correttamente: `ConversationApp`, `ConversationActionObserver`, `ConversationActionObservable`, `ConversaionActionSubject`, `ConversationAction`, `ConversationDispatcher`, `ConversationView`, `MessageStore`.	*Non Implementato*
TI19	Vogliamo verificare che `Client` e `Back-end` interagiscano tra loro correttamente.	*Non Implementato*

Tabella 3: Test di Integrazione

C.4 Test di Unità

I test di unita saranno identificati secondo quanto riportato nel documento "*Norme di Progetto v4.0.0*".

Id Test	Descrizione	Stato
TU1	Vogliamo testare che il metodo imposta il campo `status` della risposta a 200 e il campo `speech` sia uguale al campo `fulfillment.speech` del corpo della richiesta, in caso il token sia presente e valido.	*Non Implementato*
TU2	Vogliamo testare che il metodo imposta il campo `status` della risposta a 403 in caso di mancata autenticazione, ovvero token assente o non valido.	*Non Implementato*
TU3	Vogliamo testare che il metodo solleva un'eccezione alla sua chiamata.	*Non Implementato*
TU4	Vogliamo testare che il metodo accetta un parametro di tipo `Agent` senza generare eccezioni.	*Non Implementato*
TU5	Vogliamo testare che il metodo solleva un eccezione nel caso in cui il parametro non sia di tipo `Agent`.	*Non Implementato*
TU6	Vogliamo testare che se la chiamata al servizio di STT non va a buon fine, venga chiamato il metodo `succeed` del `context`, con un parametro `LambdaResponse` avente `statusCode` pari a 500.	*Non Implementato*
TU7	Vogliamo testare che se lo `status` della risposta ricevuta dall'assistente virtuale sia diverso da 200, venga chiamato il metodo `succeed` di `context` con un oggetto di tipo `LambdaResponse` come parametro, avente il campo `statusCode` uguale a quello ricevuto e corpo del messaggio "Errore nel contattare l'assistente virtuale".	*Non Implementato*
TU8	Vogliamo testare che se action del body della risposta è uguale a `"rule.add"` venga chiamato il metodo privato `addRule`.	*Non Implementato*
TU9	Vogliamo testare che se action del body della risposta è uguale a `"user.add"` venga chiamato il metodo privato `addUser`.	*Non Implementato*
TU10	Vogliamo testare che se action del body della risposta è uguale a `"user.addEnrollment"` venga chiamato il metodo privato `addUserEnrollment`.	*Non Implementato*
TU11	Vogliamo testare che se action del body della risposta è uguale a `"rule.get"` venga chiamato il metodo privato `getRule`.	*Non Implementato*
TU12	Vogliamo testare che se action del body della risposta è uguale a `"rule.getList"` venga chiamato il metodo privato `getRuleList`.	*Non Implementato*
TU13	Vogliamo testare che se action del body della risposta è uguale a `"user.get"` venga chiamato il metodo privato `getUser`.	*Non Implementato*
TU14	Vogliamo testare che se action del body della risposta è uguale a `"user.login"` venga chiamato il metodo privato `loginUser`.	*Non Implementato*
TU15	Vogliamo testare che se action del body della risposta è uguale a `"rule.remove"` venga chiamato il metodo privato `removeRule`.	*Non Implementato*

Id Test	Descrizione	Stato
TU16	Vogliamo testare che se action del body della risposta è uguale a `"user.remove"` venga chiamato il metodo privato `removeUser`.	*Non Implementato*
TU17	Vogliamo testare che se action del body della risposta è uguale a `"user.resetEnrollment"` venga chiamato il metodo privato `resetUserEnrollment`.	*Non Implementato*
TU18	Vogliamo testare che se action del body della risposta è uguale a `"rule.update"` venga chiamato il metodo privato `updateRule`.	*Non Implementato*
TU19	Vogliamo testare che se action del body della risposta è uguale a `"user.update"` venga chiamato il metodo privato `updateUser`.	*Non Implementato*
TU20	Vogliamo testare che, se durante la chiamata al metodo privato `addRule` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU21	Vogliamo testare che, se durante la chiamata al metodo privato `addUser` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU22	Vogliamo testare che, se durante la chiamata al metodo privato `addUserEnrollment` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU23	Vogliamo testare che, se durante la chiamata al metodo privato `getRule` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU24	Vogliamo testare che, se durante la chiamata al metodo privato `getRuleList` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU25	Vogliamo testare che, se durante la chiamata al metodo privato `getUser` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU26	Vogliamo testare che, se durante la chiamata al metodo privato `getUserList` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU27	Vogliamo testare che, se durante la chiamata al metodo privato `loginUser` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*

Id Test	Descrizione	Stato
TU28	Vogliamo testare che, se durante la chiamata al metodo privato `removeRule` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU29	Vogliamo testare che, se durante la chiamata al metodo privato `removeUser` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU30	Vogliamo testare che, se durante la chiamata al metodo privato `resetUserEnrollment` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU31	Vogliamo testare che, se durante la chiamata al metodo privato `updateRule` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU32	Vogliamo testare che, se durante la chiamata al metodo privato `updateUser` si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU33	Vogliamo testare che, se la risposta ricevuta dalla chiamata al *microservizio*$_{g}$ `Rules` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU34	Vogliamo testare che, se la risposta ricevuta dalla chiamata al *microservizio*$_{g}$ `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU35	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU36	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Rules` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU37	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Rules` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*

Id Test	Descrizione	Stato
TU38	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU39	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU40	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU41	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Rules` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU42	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU43	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU44	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Rules` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU45	Vogliamo testare che, se la risposta ricevuta dalla chiamata al microservizio `Users` ha uno status code diverso da 200, il metodo solleva un'eccezione di tipo `Exception` con campo `code` pari allo status code della risposta.	*Non Implementato*
TU46	Vogliamo dimostrare che, se la chiamata al metodo `sns.publish` genera un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari allo `status` dell'errore.	*Non Implementato*
TU47	Vogliamo testare che, se lo status code della risposta di un microservizio è pari a 200 e l'action contenuta nel suo body non corrisponde a nessuna action supportata dal back-end, il metodo rielabori la risposta e la inoltri.	*Non Implementato*
TU48	Vogliamo testare che il metodo accetti un parametro di tipo `Conversation` senza generare eccezioni.	*Non Implementato*

Id Test	Descrizione	Stato
TU49	Vogliamo testare che il metodo sollevi un'eccezione nel caso in cui il parametro non sia di tipo `Conversation`.	*Non Implementato*
TU50	Vogliamo testare che, se il metodo aggiunge correttamente una conversazione, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU51	Vogliamo testare che, se la conversazione non viene aggiunta a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU52	Vogliamo testare che, se il metodo aggiunge correttamente un messaggio ad una conversazione, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU53	Vogliamo testare che, se il messaggio non viene aggiunto alla conversazione a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU54	Vogliamo testare che, nel caso in cui il metodo ottenga la conversazione, l'`Observable` invia tale `Conversation` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU55	Vogliamo testare che, se si verifica un errore nell'ottenere la conversazione, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU56	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `Conversation` presenti nel database tramite il metodo `next`.	*Non Implementato*
TU57	Vogliamo testare che, se si verifica un errore nell'ottenere la lista delle conversazione, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU58	Vogliamo testare che, se il metodo elimina correttamente una conversazione, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU59	Vogliamo testare che, se la conversazione non viene eliminata a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU60	Vogliamo testare che il metodo accetti un parametro di tipo `Guest` senza generare eccezioni.	*Non Implementato*
TU61	Vogliamo testare che il metodo sollevi un eccezione nel caso in cui il parametro non sia di tipo `Guest`.	*Non Implementato*
TU62	Vogliamo testare che, se il metodo aggiunge correttamente un ospite, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU63	Vogliamo testare che, se un ospite non viene aggiunto a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*

Id Test	Descrizione	Stato
TU64	Vogliamo testare che, nel caso in cui il metodo ottenga un ospite, l'`Observable` invia tale `Guest` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU65	Vogliamo testare che, se si verifica un errore nell'ottenere un ospite, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU66	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `Guest` presenti nel database tramite il metodo `next`.	*Non Implementato*
TU67	Vogliamo testare che, se si verifica un errore nell'ottenere la lista degli ospiti, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU68	Vogliamo testare che, se il metodo elimina correttamente l'ospite, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU69	Vogliamo testare che, se l'ospite non viene eliminato a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU70	Vogliamo testare che, se il metodo aggiorna correttamente l'ospite, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU71	Vogliamo testare che, se l'ospite non viene eliminato a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU72	Vogliamo testare che il metodo accetti un parametro di tipo `Member` senza generare eccezioni.	*Non Implementato*
TU73	Vogliamo testare che il metodo sollevi un eccezione nel caso in cui il parametro non sia di tipo `Member`.	*Non Implementato*
TU74	Vogliamo testare che, nel caso in cui il metodo ottenga il membro dell'azienda, l'`Observable` invia tale `Member` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU75	Vogliamo testare che, se si verifica un errore nell'ottenere il membro dell'azienda, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU76	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `Member` presenti nel database tramite il metodo `next`.	*Non Implementato*
TU77	Vogliamo testare che, se si verifica un errore nell'ottenere la lista dei membri dell'azienda, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU78	Vogliamo testare che, anche se viene passato un `Member` corretto, il metodo ritorna un `ErrorObservable` ovvero la chiamata al metodo fallisce sempre.	*Non Implementato*

Id Test	Descrizione	Stato
TU79	Vogliamo testare che, anche se viene passato un `Member` corretto, il metodo ritorna un `ErrorObservable` ovvero la chiamata al metodo fallisce sempre.	*Non Implementato*
TU80	Vogliamo testare che, anche se viene passato l'username di un `Member`, il metodo ritorna un `ErrorObservable`.	*Non Implementato*
TU81	Vogliamo testare che, se si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU82	Vogliamo testare che il metodo imposti il campo `statusCode` della risposta a 200 e il campo `body` contenga la lista dei canali di Slack informato *JSON*$_g$.	*Non Implementato*
TU83	Vogliamo testare che il metodo imposti il campo `statusCode` della risposta a 200 e il campo `body` sia vuoto.	*Non Implementato*
TU84	Vogliamo testare che, se si verifica un errore, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU85	Vogliamo testare che alla chiamata del metodo venga chiamata la funzione di *callback*$_g$ `complete_cb`.	*Non Implementato*
TU86	Vogliamo testare che alla chiamata del metodo venga chiamata la funzione di callback `error_cb`, passandole come parametro l'errore ricevuto.	*Non Implementato*
TU87	Vogliamo testare che alla chiamata del metodo venga chiamata la funzione di callback `next_cb`, passandole come parametro i dati ricevuti.	*Non Implementato*
TU88	Vogliamo testare che il metodo accetti un parametro di tipo `Rule` senza generare eccezioni.	*Non Implementato*
TU89	Vogliamo testare che il metodo sollevi un eccezione nel caso in cui il parametro non sia di tipo `Rule`.	*Non Implementato*
TU90	Vogliamo testare che, se il metodo aggiunge correttamente una direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU91	Vogliamo testare che, se la direttiva non viene aggiunta a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU92	Vogliamo testare che, nel caso in cui il metodo ottenga una direttiva, l'`Observable` invia tale `Rule` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU93	Vogliamo testare che, se si verifica un errore nell'ottenere una direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU94	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `Rule` presenti nel database tramite il metodo `next`.	*Non Implementato*

Id Test	Descrizione	Stato
TU95	Vogliamo testare che, se si verifica un errore nell'ottenere la lista delle direttive, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU96	Vogliamo testare che, se il metodo elimina correttamente la direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU97	Vogliamo testare che, se la direttiva non viene eliminata a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU98	Vogliamo testare che, se il metodo aggiorna correttamente la direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU99	Vogliamo testare che, se la direttiva non viene aggiornata a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU100	Vogliamo testare che, se il metodo aggiunge correttamente la funzione di una direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU101	Vogliamo testare che, se la funziona di una direttiva non viene aggiunta a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU102	Vogliamo testare che, nel caso in cui il metodo ottenga la funzione di una direttiva, l'`Observable` invia tale `Task` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU103	Vogliamo testare che, se si verifica un errore nell'ottenere una funzione, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU104	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `Task` presenti nel database tramite il metodo `next`.	*Non Implementato*
TU105	Vogliamo testare che, se si verifica un errore nell'ottenere la lista delle funzioni, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU106	Vogliamo testare che, se il metodo elimina correttamente la funzione di una direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU107	Vogliamo testare che, se la funzione di una direttiva non viene eliminata a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU108	Vogliamo testare che, se il metodo aggiorna correttamente la funzione di una direttiva, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*

Id Test	Descrizione	Stato
TU109	Vogliamo testare che, se la funzione di una direttiva non viene aggiornata a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU110	Vogliamo testare che, se la chiamata al metodo `stt.recognize` fallisce, viene chiamato il metodo rejected della Promise con un parametro `Exception` avente campo `code` 500.	*Non Implementato*
TU111	Vogliamo testare che il metodo accetti un parametro di tipo `Task` senza generare eccezioni.	*Non Implementato*
TU112	Vogliamo testare che il metodo sollevi un eccezione nel caso in cui il parametro non sia di tipo `Task`.	*Non Implementato*
TU113	Vogliamo testare che il metodo accetti un parametro di tipo `User` senza generare eccezioni.	*Non Implementato*
TU114	Vogliamo testare che il metodo sollevi un eccezione nel caso in cui il parametro non sia di tipo `User`.	*Non Implementato*
TU115	Vogliamo testare che, se il metodo aggiunge correttamente un utente, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU116	Vogliamo testare che, se l'utente non viene aggiunto a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU117	Vogliamo testare che, nel caso in cui il metodo ottenga un utente, l'`Observable` invia tale `User` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU118	Vogliamo testare che, se si verifica un errore nell'ottenere un utente, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU119	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `User` presenti nel database tramite il metodo `next`.	*Non Implementato*
TU120	Vogliamo testare che, se si verifica un errore nell'ottenere la lista degli utenti, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU121	Vogliamo testare che, se il metodo elimina correttamente l'utente, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU122	Vogliamo testare che, se l'utente non viene eliminato a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU123	Vogliamo testare che, se il metodo aggiorna correttamente l'utente, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU124	Vogliamo testare che, se l'utente non viene aggiornato a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*

Id Test	Descrizione	Stato
TU125	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per aggiungere un `Enrollment` ritorna uno `statusCode` diverso da 200, l'`ErrorObservable` notifica `ErrorObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU126	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per creare uno `User` ritorna uno `statusCode` diverso da 200, l'`StringObservable` notifica `StringObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU127	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per eliminare uno `User` ritorna uno `statusCode` diverso da 200, l'`ErrorObservable` notifica `ErrorObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU128	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per effettuare il login ritorna uno `statusCode` diverso da 200, l'`ErrorObservable` notifica `ErrorObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU129	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per ottenere la lista degli `User` ritorna uno `statusCode` diverso da 200, l'`SRUserObservable` notifica `SRUserObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU130	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per ottenere uno `User` ritorna uno `statusCode` diverso da 200, l'`SRUserObservable` notifica `SRUserObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU131	Vogliamo testare che, se la chiamata al servizio di Speaker Recognition per resettare un `Enrollment` ritorna uno `statusCode` diverso da 200, l'`ErrorObservable` notifica `ErrorObserver` chiamando il suo metodo `error`.	*Non Implementato*
TU132	Vogliamo testare che, se il metodo aggiunge correttamente un agente di api.ai, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU133	Vogliamo testare che, se l'agente non viene aggiunto a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU134	Vogliamo testare che, nel caso in cui il metodo ottenga un agente di api.ai, l'`Observable` invia tale `Agent` all'`Observer` iscritto tramite il metodo `next` e lo notifica richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU135	Vogliamo testare che, se si verifica un errore nell'ottenere un agente, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU136	Vogliamo testare che l'`Observable` notifica l'`Observer` con il metodo `complete` solo dopo aver inviato tutti i blocchi di `Agent` presenti nel database tramite il metodo `next`.	*Non Implementato*

Id Test	Descrizione	Stato
TU137	Vogliamo testare che, se si verifica un errore nell'ottenere la lista degli agenti, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU138	Vogliamo testare che, se il metodo elimina correttamente l'agente, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU139	Vogliamo testare che, se l'agente non viene eliminato a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU140	Vogliamo testare che, se il metodo aggiorna correttamente l'agente di api.ai, l'`Observable` notifica l'`Observer` iscritto richiamando una sola volta il metodo `complete`.	*Non Implementato*
TU141	Vogliamo testare che, se l'agente non viene aggiornato a causa di un errore, l'`Observable` notifica l'`Observer` iscritto richiamando il metodo `error`.	*Non Implementato*
TU142	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU143	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU144	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200.	*Non Implementato*
TU145	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU146	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU147	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200.	*Non Implementato*
TU148	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200 e campo `body` contenente la `Rule` cercata.	*Non Implementato*

Id Test	Descrizione	Stato
TU149	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU150	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU151	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200 e campo `body` contenente la lista delle `Rule`.	*Non Implementato*
TU152	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU153	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU154	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200 e campo `body` contenente la lista dei `Task`.	*Non Implementato*
TU155	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU156	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU157	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200 e campo `body` contenente la lista delle `Rule` da applicare ad un determinato caso.	*Non Implementato*
TU158	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*

Id Test	Descrizione	Stato
TU159	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU160	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200.	*Non Implementato*
TU161	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU162	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU163	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200.	*Non Implementato*
TU164	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU165	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU166	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200 e campo `body` contenente l'`User` cercato.	*Non Implementato*
TU167	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU168	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU169	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200 e campo `body` contenente la lista degli `User`.	*Non Implementato*

Id Test	Descrizione	Stato
TU170	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU171	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU172	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200.	*Non Implementato*
TU173	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU174	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU175	Vogliamo testare che, se la chiamata al metodo va a buon fine, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 200.	*Non Implementato*
TU176	Vogliamo testare che, se la chiamata al metodo viene fatta con un parametro aspettato, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 400.	*Non Implementato*
TU177	Vogliamo testare che, se la chiamata al metodo genera un errore del microservizio, viene chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` avente campo `statusCode` pari a 500.	*Non Implementato*
TU178	Se la chiamata al microservizio `Rules` genera un errore, viene chiamata la funzione di callback con un solo parametro diverso da null.	*Non Implementato*
TU179	Se la chiamata al microservizio `Notification` genera un errore, viene chiamata la funzione di callback con un solo parametro diverso da null.	*Non Implementato*
TU180	Se la chiamata ai metodi di `GuestsDAO` genera un errore, viene chiamata la funzione di callback con un solo parametro diverso da null.	*Non Implementato*
TU181	Se la chiamata ai metodi di `ConversationsDAO` genera un errore, viene chiamata la funzione di callback con un solo parametro diverso da null.	*Non Implementato*
TU182	Se le chiamata ai microservizi e le chiamate ai DAO non generano alcun errore, viene chiamata la funzione di callback con due parametri, il primo uguale a null e il secondo contenente la risposta.	*Non Implementato*

Id Test	Descrizione	Stato
TU183	Vogliamo verificare che, se la richiesta $HTTP_g$ genera un errore, viene chiamato il metodo `reject` della Promise.	*Non Implementato*
TU184	Vogliamo verificare che, se la richiesta HTTP va a buon fine, viene chiamato il metodo `fulfill` della Promise.	*Non Implementato*
TU185	Vogliamo testare che, se la richiesta HTTP ad api.ai genera un errore, nel caso in cui status code oppure `status.code` sia diverso da 200, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è impostato a 500.	*Non Implementato*
TU186	Vogliamo testare che, se la richiesta HTTP ad api.ai genera un errore, nel caso in cui `result.fulfillment.data.status` sia impostato ad un valore diverso da 200, venga chiamato il metodo `succeed` del `context` con un parametro `LambdaResponse` il quale campo `statusCode` è uguale allo status di `result.fulfillment.data.status`.	*Non Implementato*
TU187	Vogliamo testare che, se la richiesta HTTP ad api.ai va a buon fine, allora status code, `result.fulfillment.data.status` e `status.code` sono uguali a 200.	*Non Implementato*
TU188	Vogliamo testare che, se l'attributo `paused` è true, non vengono chiamate le funzioni di callback.	*Non Implementato*
TU189	Vogliamo testare che venga aggiunto correttamente l'`ApplicationPackage` passato come parametro.	*Non Implementato*
TU190	Vogliamo testare che sia possibile ottenere l'`ApplicationPackage` a partire dal suo nome passato come parametro.	*Non Implementato*
TU191	Vogliamo testare che sia possibile eliminare l'`ApplicationPackage` a partire dal suo nome passato come parametro.	*Non Implementato*
TU192	Vogliamo testare che sia possibile ottenere l'`ApplicationPackage` a partire dal suo nome passato come parametro.	*Non Implementato*
TU193	Vogliamo testare che venga aggiunto correttamente l'`ApplicationPackage` passato come parametro.	*Non Implementato*
TU194	Vogliamo testare che, alla chiamata del metodo, l'`Observable` notifichi tutti gli `Observer` iscritti passando loro un oggetto composto dai parametri con cui il metodo è stato chiamato.	*Non Implementato*
TU195	Vogliamo testare che l'oggetto ritornato dalla funzione sia effettivamente un `ReactElement`.	*Non Implementato*
TU196	Vogliamo testare che, se l'applicazione è presente all'interno di `State`, non viene interrogato il Client.	*Non Implementato*
TU197	Vogliamo testare che, se l'applicazione non è presente all'interno di `State`, viene interrogato il Client per ottenerla e la vecchia applicazione viene salvata nello `State`.	*Non Implementato*
TU198	Vogliamo testare che venga chiamato `appendChild` sul parametro passato al metodo per poter mostrare l'interfaccia utente.	*Non Implementato*

Id Test	Descrizione	Stato
TU199	Vogliamo testare che, se `action.cmd` è uguale a "clear", viene chiamato il metodo `onClear` e vengono notificati gli `Observer` iscritti all'`Observable`.	*Non Implementato*
TU200	Vogliamo testare che, se `action.cmd` è uguale a "displayMsgs", viene chiamato il metodo `onDisplayMsgs` e vengono notificati gli `Observer` iscritti all'`Observable`.	*Non Implementato*
TU201	Vogliamo testare che, se `action.cmd` è uguale a "msgReceived", viene chiamato il metodo `onMsgReceived` e vengono notificati gli `Observer` iscritti all'`Observable`.	*Non Implementato*
TU202	Vogliamo testare che, se `action.cmd` è uguale a "msgSent", viene chiamato il metodo `onMsgSent` e vengono notificati gli `Observer` iscritti all'`Observable`.	*Non Implementato*
TU203	Vogliamo testare che, se `action.cmd` non corrisponde a nessuna delle action prestabilite, non vengono notificati gli `Observer` e non viene sollevata alcuna eccezione.	*Non Implementato*
TU204	Vogliamo testare che venga aggiunta correttamente l'`Application` passata come parametro.	*Non Implementato*
TU205	Vogliamo testare che sia possibile ottenere l'`Application` a partire dal suo nome passato come parametro.	*Non Implementato*
TU206	Vogliamo testare che richiami il metodo `dispatcher.dispatch` inoltrandogli i parametri ricevuti.	*Non Implementato*
TU207	Vogliamo testare che, se i parametri passati non sono corretti, non viene chiamato il metodo `dispatcher.dispatch` e viene sollevata un'eccezione `Exception`.	*Non Implementato*
TU208	Vogliamo testare che, nel caso in cui il metodo venga chiamato, sia sollevata un'eccezione `Exception`.	*Non Implementato*
TU209	Vogliamo testare che, se la richiesta va a buon fine, viene chiamata la funzione di callback `fulfill`.	*Non Implementato*
TU210	Vogliamo testare che, se la richiesta fallisce, viene chiamata la funzione di callback `reject`.	*Non Implementato*
TU211	Vogliamo testare che, se la promessa viene soddisfatta (fulfill), viene chiamato il metodo `next` del `subject` che si occupa di notificare gli `Observer` iscritti.	*Non Implementato*
TU212	Vogliamo testare che, se la promessa viene respinta (reject), viene chiamato il metodo `error` del `subject` che si occupa di notificare tale errore agli `Observer` iscritti.	*Non Implementato*
TU213	Vogliamo testare che, una volta chiamato il metodo `start`, venga inviata una serie di oggetti `RecorderMsg` a `RecorderWorker` con campo `command` uguale a "record" e che questa serie di messaggi venga interrotta alla chiamata del metodo `stop`.	*Non Implementato*

Tabella 4: Test di Unità

C.5 Tracciamento Test di Validazione-Requisiti

Test	Requisito
TVFO1	RFO1
TVFO1.1.2	RFO1.1.2
TVFO2.1	RFO2.1
TVFO2.1.1.6	RFO2.1.1.6
TVFO2.1.2	RFO2.1.2
TVFO2.1.4	RFO2.1.4
TVFO2.2	RFO2.2
TVFO3.1	RFO3.1
TVFO5	RFO5
TVFO7	RFO7

Tabella 5: Tracciamento Test di Validazione-Requisiti

C.6 Tracciamento Componenti-Test di Integrazione

Componente	Test
Back-end	TI2
Back-end::APIGateway	TI8
Back-end::Conversations	TI14
Back-end::Events	TI15
Back-end::Guests	TI13
Back-end::Members	TI12
Back-end::Notifications	TI16
Back-end::Rules	TI10
Back-end::Users	TI9
Back-end::Utility	TI17
Back-end::VirtualAssistant	TI11
Client	TI1
Client::ApplicationManager	TI3
Client::ConversationApp	TI18
Client::Logic	TI4
Client::Recorder	TI5
Client::TTS	TI6
Client::Utility	TI7

Tabella 6: Tracciamento Componenti-Test di Integrazione

C.7 Tracciamento Metodi-Test di Unità

Metodo	Test
Back-end::AdministrationWebhookService::-webhook()	TU1 TU2
Back-end::APIGateway::VocalAPI::-addRule()	TU33
Back-end::APIGateway::VocalAPI::-addUser()	TU34
Back-end::APIGateway::VocalAPI::-addUserEnrollment()	TU35
Back-end::APIGateway::VocalAPI::-getRule()	TU36
Back-end::APIGateway::VocalAPI::-getRuleList()	TU37
Back-end::APIGateway::VocalAPI::-getUser()	TU38
Back-end::APIGateway::VocalAPI::-getUserList()	TU39
Back-end::APIGateway::VocalAPI::-loginUser()	TU40
Back-end::APIGateway::VocalAPI::-queryLambda()	TU6 TU7 TU8 TU9 TU10 TU11 TU12 TU13 TU14 TU15 TU16 TU17 TU18 TU19 TU20 TU21 TU22 TU23 TU24 TU25 TU26 TU27 TU28 TU29 TU30 TU31 TU32 TU46 TU47
Back-end::APIGateway::VocalAPI::-removeRule()	TU41

Metodo	Test
Back-end::APIGateway::VocalAPI::-removeUser()	TU42
Back-end::APIGateway::VocalAPI::-resetUserEnrollment()	TU43
Back-end::APIGateway::VocalAPI::-updateRule()	TU44
Back-end::APIGateway::VocalAPI::-updateUser()	TU45
Back-end::Conversations::-<<interface>> ConversationsDAO::removeConversation()	TU59
Back-end::Conversations::ConversationObserver::-next()	TU48 TU49
Back-end::Conversations::ConversationsDAODynamoDB::-addConversation()	TU50 TU51
Back-end::Conversations::ConversationsDAODynamoDB::-addMessage()	TU52 TU53
Back-end::Conversations::ConversationsDAODynamoDB::-getConversation()	TU54 TU55
Back-end::Conversations::ConversationsDAODynamoDB::-getConversationList()	TU56 TU57
Back-end::Conversations::ConversationsDAODynamoDB::-removeConversation()	TU58
Back-end::Events::VAMessageListener::-onMessage()	TU178 TU179 TU180 TU181 TU182
Back-end::Guests::GuestObserver::next()	TU60 TU61
Back-end::Guests::GuestsDAODynamoDB::-addGuest()	TU62 TU63
Back-end::Guests::GuestsDAODynamoDB::-getGuest()	TU64 TU65
Back-end::Guests::GuestsDAODynamoDB::-getGuestList()	TU66 TU67
Back-end::Guests::GuestsDAODynamoDB::-removeGuest()	TU68 TU69
Back-end::Guests::GuestsDAODynamoDB::-updateGuest()	TU70 TU71
Back-end::Members::MemberObserver::-next()	TU72 TU73

Metodo	Test
Back-end::Members::MembersDAOSlack::-addMember()	TU78
Back-end::Members::MembersDAOSlack::-getMember()	TU74 TU75
Back-end::Members::MembersDAOSlack::-getMemberList()	TU76 TU77
Back-end::Members::MembersDAOSlack::-removeMember()	TU80
Back-end::Members::MembersDAOSlack::-updateMember()	TU79
Back-end::Notifications::NotificationService::-getChannelList()	TU81 TU82
Back-end::Notifications::NotificationService::-sendMsg()	TU83 TU84
Back-end::ObserverAdapter::complete()	TU85
Back-end::ObserverAdapter::error()	TU86
Back-end::ObserverAdapter::next()	TU87
Back-end::ObserverAdapter::pause()	TU188
Back-end::ObserverAdapter::resume()	TU188
Back-end::Rules::RuleObserver::next()	TU88 TU89
Back-end::Rules::RulesDAODynamoDB::-addRule()	TU90 TU91
Back-end::Rules::RulesDAODynamoDB::-getRule()	TU92 TU93
Back-end::Rules::RulesDAODynamoDB::-getRuleList()	TU94 TU95
Back-end::Rules::RulesDAODynamoDB::-removeRule()	TU96 TU97
Back-end::Rules::RulesDAODynamoDB::-updateRule()	TU98 TU99
Back-end::Rules::RulesService::-addRule()	TU142 TU143 TU144
Back-end::Rules::RulesService::-deleteRule()	TU145 TU146 TU147
Back-end::Rules::RulesService::-getRule()	TU148 TU149 TU150
Back-end::Rules::RulesService::-getRuleList()	TU151

Metodo	Test
	TU152 TU153
Back-end::Rules::RulesService::- getTask()	TU154 TU155
Back-end::Rules::RulesService::- getTaskList()	TU156
Back-end::Rules::RulesService::- queryRule()	TU157 TU158 TU159
Back-end::Rules::RulesService::- updateRule()	TU160 TU161 TU162
Back-end::Rules::TaskObserver::next()	TU111 TU112
Back-end::Rules::TasksDAODynamoDB::- addTask()	TU100 TU101
Back-end::Rules::TasksDAODynamoDB::- getTask()	TU102 TU103
Back-end::Rules::TasksDAODynamoDB::- getTaskList()	TU104 TU105
Back-end::Rules::TasksDAODynamoDB::- removeTask()	TU106 TU107
Back-end::Rules::TasksDAODynamoDB::- updateTask()	TU108 TU109
Back-end::STT::STTWatsonAdapter::- speechToText()	TU110
Back-end::Users::<<interface>>VocalLoginModule::- addEnrollment()	TU125
Back-end::Users::<<interface>>VocalLoginModule::- createUser()	TU126
Back-end::Users::<<interface>>VocalLoginModule::- deleteUser()	TU127
Back-end::Users::<<interface>>VocalLoginModule::- doLogin()	TU128
Back-end::Users::<<interface>>VocalLoginModule::- resetEnrollments()	TU131
Back-end::Users::UserObserver::next()	TU113 TU114
Back-end::Users::UsersDAODynamoDB::- addUser()	TU115 TU116
Back-end::Users::UsersDAODynamoDB::- getUser()	TU117 TU118
Back-end::Users::UsersDAODynamoDB::- getUserList()	TU119

Metodo	Test
	TU120
Back-end::Users::UsersDAODynamoDB::-removeUser()	TU121 TU122
Back-end::Users::UsersDAODynamoDB::-updateUser()	TU123 TU124
Back-end::Users::UsersService::-addUser()	TU163 TU164 TU165
Back-end::Users::UsersService::-getUser()	TU166 TU167 TU168
Back-end::Users::UsersService::-getUserList()	TU169 TU170 TU171
Back-end::Users::UsersService::-removeUser()	TU172 TU173 TU174
Back-end::Users::UsersService::-updateUser()	TU175 TU176 TU177
Back-end::Users::VocalLoginMicrosoftModule::-getList()	TU129
Back-end::Users::VocalLoginMicrosoftModule::-getUser()	TU130
Back-end::VirtualAssistant::-<<interface>> AgentsDAO::getAgentsList()	TU136 TU137
Back-end::VirtualAssistant::-<<interface>> AgentsDAO::removeAgent()	TU138 TU139
Back-end::VirtualAssistant::-<<interface>> AgentsDAO::updateAgent()	TU140 TU141
Back-end::VirtualAssistant::AgentObserver::-next()	TU4 TU5
Back-end::VirtualAssistant::AgentsDAODynamoDB::-addAgent()	TU132 TU133
Back-end::VirtualAssistant::AgentsDAODynamoDB::-getAgent()	TU134 TU135
Back-end::VirtualAssistant::ApiAiVAAdapter::-query()	TU183
Back-end::VirtualAssistant::VAService::-query()	TU184 TU185

Metodo	Test
	TU186 TU187
Client::ApplicationManager::- ApplicationLocalRegistry::query()	TU190
Client::ApplicationManager::- ApplicationLocalRegistry::register()	TU189
Client::ApplicationManager::- ApplicationLocalRegistry::remove()	TU191
Client::ApplicationManager::- ApplicationRegistryLocalClient::query()	TU192
Client::ApplicationManager::- ApplicationRegistryLocalClient::register()	TU193
Client::ApplicationManager::Manager::- runApplication()	TU196 TU197
Client::ApplicationManager::Manager::- setFrame()	TU198
Client::ApplicationManager::State::- addApp()	TU204
Client::ApplicationManager::State::- getApp()	TU205
Client::ConversationApp::ConversationApp::- runCmd()	TU206 TU207
Client::ConversationApp::ConversationDispatcher::- dispatch()	TU194
Client::ConversationApp::ConversationView::- render()	TU195
Client::ConversationApp::MessageStore::- onCmd()	TU199 TU200 TU201 TU202 TU203
Client::Logic::HttpPromise::then()	TU209 TU210
Client::Logic::Logic::sendData()	TU211 TU212
Client::Recorder::Recorder::start()	TU213
Client::Recorder::Recorder::stop()	TU213
Libs::ErrorObserver::next()	TU3

Tabella 7: Tracciamento Metodi-Test di Unità

C.8 Tracciamento Requisiti-Test di Sistema

Requisito	Test
RFO1	TVFO1
RFO1.1.2.1	TVFO1.1.2.1
RFO2.1.1	TVFO2.1.1
RFO2.1.2	TVFO2.1.2
RFO2.1.4	TVFO2.1.4
RFO2.2.1	TVFO2.2.1
RFO3.1	TVFO3.1
RFO5	TVFO5
RFO7	TVFO7
RFO8	TVFO8
RFO13	TVFO13
RVO1.1	TVVO1.1
RVO4	TVVO4
RVO5	TVVO5
RVO10	TVVO10

Tabella 8: Tracciamento Requisiti-Test di Sistema

C.9 Tracciamento Requisiti-Test di Validazione

Requisito	Test
RFO1	TVFO1
RFO1.1.2	TVFO1.1.2
RFO2.1	TVFO2.1
RFO2.1.1.6	TVFO2.1.1.6
RFO2.1.2	TVFO2.1.2
RFO2.1.4	TVFO2.1.4
RFO2.2	TVFO2.2
RFO3.1	TVFO3.1
RFO5	TVFO5
RFO7	TVFO7

Tabella 9: Tracciamento Requisiti-Test di Validazione

C.10 Tracciamento Test di Integrazione-Componenti

Test	Componente
TI1	`Client`
TI2	`Back-end`
TI3	`Client::ApplicationManager`
TI4	`Client::Logic`
TI5	`Client::Recorder`
TI6	`Client::TTS`
TI7	`Client::Utility`
TI8	`Back-end::APIGateway`
TI9	`Back-end::Users`
TI10	`Back-end::Rules`
TI11	`Back-end::VirtualAssistant`
TI12	`Back-end::Members`
TI13	`Back-end::Guests`
TI14	`Back-end::Conversations`
TI15	`Back-end::Events`
TI16	`Back-end::Notifications`
TI17	`Back-end::Utility`
TI18	`Client::ConversationApp`

Tabella 10: Tracciamento Test di Integrazione-Componenti

C.11 Tracciamento Test di Sistema-Requisiti

Test	Requisito
TSFO1	RFO1
TSFO1.1.2.1	RFO1.1.2.1
TSFO2.1.1	RFO2.1.1
TSFO2.1.2	RFO2.1.2
TSFO2.1.4	RFO2.1.4
TSFO2.2.1	RFO2.2.1
TSFO3.1	RFO3.1
TSFO5	RFO5
TSFO7	RFO7
TSFO8	RFO8
TSFO13	RFO13
TSVO1.1	RVO1.1
TSVO4	RVO4
TSVO5	RVO5
TSVO10	RVO10

Tabella 11: Tracciamento Test di Sistema-Requisiti

C.12 Tracciamento Test di Unità-Metodi

Test	Metodi
TU1	Back-end::AdministrationWebhookService::-webhook()
TU2	Back-end::AdministrationWebhookService::-webhook()
TU3	Libs::ErrorObserver::next()
TU4	Back-end::VirtualAssistant::AgentObserver::-next()
TU5	Back-end::VirtualAssistant::AgentObserver::-next()
TU6	Back-end::APIGateway::VocalAPI::-queryLambda()
TU7	Back-end::APIGateway::VocalAPI::-queryLambda()
TU8	Back-end::APIGateway::VocalAPI::-queryLambda()
TU9	Back-end::APIGateway::VocalAPI::-queryLambda()
TU10	Back-end::APIGateway::VocalAPI::-queryLambda()
TU11	Back-end::APIGateway::VocalAPI::-queryLambda()
TU12	Back-end::APIGateway::VocalAPI::-queryLambda()
TU13	Back-end::APIGateway::VocalAPI::-queryLambda()
TU14	Back-end::APIGateway::VocalAPI::-queryLambda()
TU15	Back-end::APIGateway::VocalAPI::-queryLambda()
TU16	Back-end::APIGateway::VocalAPI::-queryLambda()
TU17	Back-end::APIGateway::VocalAPI::-queryLambda()
TU18	Back-end::APIGateway::VocalAPI::-queryLambda()
TU19	Back-end::APIGateway::VocalAPI::-queryLambda()
TU20	Back-end::APIGateway::VocalAPI::-queryLambda()
TU21	Back-end::APIGateway::VocalAPI::-queryLambda()
TU22	Back-end::APIGateway::VocalAPI::-queryLambda()
TU23	Back-end::APIGateway::VocalAPI::-queryLambda()
TU24	Back-end::APIGateway::VocalAPI::-queryLambda()
TU25	Back-end::APIGateway::VocalAPI::-queryLambda()
TU26	Back-end::APIGateway::VocalAPI::-queryLambda()

Test	Metodi
TU27	Back-end::APIGateway::VocalAPI::-queryLambda()
TU28	Back-end::APIGateway::VocalAPI::-queryLambda()
TU29	Back-end::APIGateway::VocalAPI::-queryLambda()
TU30	Back-end::APIGateway::VocalAPI::-queryLambda()
TU31	Back-end::APIGateway::VocalAPI::-queryLambda()
TU32	Back-end::APIGateway::VocalAPI::-queryLambda()
TU33	Back-end::APIGateway::VocalAPI::-addRule()
TU34	Back-end::APIGateway::VocalAPI::-addUser()
TU35	Back-end::APIGateway::VocalAPI::-addUserEnrollment()
TU36	Back-end::APIGateway::VocalAPI::-getRule()
TU37	Back-end::APIGateway::VocalAPI::-getRuleList()
TU38	Back-end::APIGateway::VocalAPI::-getUser()
TU39	Back-end::APIGateway::VocalAPI::-getUserList()
TU40	Back-end::APIGateway::VocalAPI::-loginUser()
TU41	Back-end::APIGateway::VocalAPI::-removeRule()
TU42	Back-end::APIGateway::VocalAPI::-removeUser()
TU43	Back-end::APIGateway::VocalAPI::-resetUserEnrollment()
TU44	Back-end::APIGateway::VocalAPI::-updateRule()
TU45	Back-end::APIGateway::VocalAPI::-updateUser()
TU46	Back-end::APIGateway::VocalAPI::-queryLambda()
TU47	Back-end::APIGateway::VocalAPI::-queryLambda()
TU48	Back-end::Conversations::ConversationObserver::-next()
TU49	Back-end::Conversations::ConversationObserver::-next()
TU50	Back-end::Conversations::ConversationsDAODynamoDB::-addConversation()
TU51	Back-end::Conversations::ConversationsDAODynamoDB::-addConversation()
TU52	Back-end::Conversations::ConversationsDAODynamoDB::-addMessage()
TU53	Back-end::Conversations::ConversationsDAODynamoDB::-addMessage()

Test	Metodi
TU54	Back-end::Conversations::ConversationsDAODynamoDB::-getConversation()
TU55	Back-end::Conversations::ConversationsDAODynamoDB::-getConversation()
TU56	Back-end::Conversations::ConversationsDAODynamoDB::-getConversationList()
TU57	Back-end::Conversations::ConversationsDAODynamoDB::-getConversationList()
TU58	Back-end::Conversations::ConversationsDAODynamoDB::-removeConversation()
TU59	Back-end::Conversations::-<<interface>> ConversationsDAO::removeConversation()
TU60	Back-end::Guests::GuestObserver::next()
TU61	Back-end::Guests::GuestObserver::next()
TU62	Back-end::Guests::GuestsDAODynamoDB::-addGuest()
TU63	Back-end::Guests::GuestsDAODynamoDB::-addGuest()
TU64	Back-end::Guests::GuestsDAODynamoDB::-getGuest()
TU65	Back-end::Guests::GuestsDAODynamoDB::-getGuest()
TU66	Back-end::Guests::GuestsDAODynamoDB::-getGuestList()
TU67	Back-end::Guests::GuestsDAODynamoDB::-getGuestList()
TU68	Back-end::Guests::GuestsDAODynamoDB::-removeGuest()
TU69	Back-end::Guests::GuestsDAODynamoDB::-removeGuest()
TU70	Back-end::Guests::GuestsDAODynamoDB::-updateGuest()
TU71	Back-end::Guests::GuestsDAODynamoDB::-updateGuest()
TU72	Back-end::Members::MemberObserver::-next()
TU73	Back-end::Members::MemberObserver::-next()
TU74	Back-end::Members::MembersDAOSlack::-getMember()
TU75	Back-end::Members::MembersDAOSlack::-getMember()
TU76	Back-end::Members::MembersDAOSlack::-getMemberList()
TU77	Back-end::Members::MembersDAOSlack::-getMemberList()
TU78	Back-end::Members::MembersDAOSlack::-addMember()
TU79	Back-end::Members::MembersDAOSlack::-updateMember()
TU80	Back-end::Members::MembersDAOSlack::-removeMember()
TU81	Back-end::Notifications::NotificationService::-getChannelList()

Test	Metodi
TU82	Back-end::Notifications::NotificationService::-getChannelList()
TU83	Back-end::Notifications::NotificationService::-sendMsg()
TU84	Back-end::Notifications::NotificationService::-sendMsg()
TU85	Back-end::ObserverAdapter::complete()
TU86	Back-end::ObserverAdapter::error()
TU87	Back-end::ObserverAdapter::next()
TU88	Back-end::Rules::RuleObserver::next()
TU89	Back-end::Rules::RuleObserver::next()
TU90	Back-end::Rules::RulesDAODynamoDB::-addRule()
TU91	Back-end::Rules::RulesDAODynamoDB::-addRule()
TU92	Back-end::Rules::RulesDAODynamoDB::-getRule()
TU93	Back-end::Rules::RulesDAODynamoDB::-getRule()
TU94	Back-end::Rules::RulesDAODynamoDB::-getRuleList()
TU95	Back-end::Rules::RulesDAODynamoDB::-getRuleList()
TU96	Back-end::Rules::RulesDAODynamoDB::-removeRule()
TU97	Back-end::Rules::RulesDAODynamoDB::-removeRule()
TU98	Back-end::Rules::RulesDAODynamoDB::-updateRule()
TU99	Back-end::Rules::RulesDAODynamoDB::-updateRule()
TU100	Back-end::Rules::TasksDAODynamoDB::-addTask()
TU101	Back-end::Rules::TasksDAODynamoDB::-addTask()
TU102	Back-end::Rules::TasksDAODynamoDB::-getTask()
TU103	Back-end::Rules::TasksDAODynamoDB::-getTask()
TU104	Back-end::Rules::TasksDAODynamoDB::-getTaskList()
TU105	Back-end::Rules::TasksDAODynamoDB::-getTaskList()
TU106	Back-end::Rules::TasksDAODynamoDB::-removeTask()
TU107	Back-end::Rules::TasksDAODynamoDB::-removeTask()
TU108	Back-end::Rules::TasksDAODynamoDB::-updateTask()
TU109	Back-end::Rules::TasksDAODynamoDB::-updateTask()
TU110	Back-end::STT::STTWatsonAdapter::-speechToText()
TU111	Back-end::Rules::TaskObserver::next()

Test	Metodi
TU112	Back-end::Rules::TaskObserver::next()
TU113	Back-end::Users::UserObserver::next()
TU114	Back-end::Users::UserObserver::next()
TU115	Back-end::Users::UsersDAODynamoDB::-addUser()
TU116	Back-end::Users::UsersDAODynamoDB::-addUser()
TU117	Back-end::Users::UsersDAODynamoDB::-getUser()
TU118	Back-end::Users::UsersDAODynamoDB::-getUser()
TU119	Back-end::Users::UsersDAODynamoDB::-getUserList()
TU120	Back-end::Users::UsersDAODynamoDB::-getUserList()
TU121	Back-end::Users::UsersDAODynamoDB::-removeUser()
TU122	Back-end::Users::UsersDAODynamoDB::-removeUser()
TU123	Back-end::Users::UsersDAODynamoDB::-updateUser()
TU124	Back-end::Users::UsersDAODynamoDB::-updateUser()
TU125	Back-end::Users::<<interface>>VocalLoginModule::-addEnrollment()
TU126	Back-end::Users::<<interface>>VocalLoginModule::-createUser()
TU127	Back-end::Users::<<interface>>VocalLoginModule::-deleteUser()
TU128	Back-end::Users::<<interface>>VocalLoginModule::-doLogin()
TU129	Back-end::Users::VocalLoginMicrosoftModule::-getList()
TU130	Back-end::Users::VocalLoginMicrosoftModule::-getUser()
TU131	Back-end::Users::<<interface>>VocalLoginModule::-resetEnrollments()
TU132	Back-end::VirtualAssistant::AgentsDAODynamoDB::-addAgent()
TU133	Back-end::VirtualAssistant::AgentsDAODynamoDB::-addAgent()
TU134	Back-end::VirtualAssistant::AgentsDAODynamoDB::-getAgent()
TU135	Back-end::VirtualAssistant::AgentsDAODynamoDB::-getAgent()
TU136	Back-end::VirtualAssistant::-<<interface>> AgentsDAO::getAgentsList()
TU137	Back-end::VirtualAssistant::-<<interface>> AgentsDAO::getAgentsList()
TU138	Back-end::VirtualAssistant::-<<interface>> AgentsDAO::removeAgent()
TU139	Back-end::VirtualAssistant::-<<interface>> AgentsDAO::removeAgent()

Test	Metodi
TU140	Back-end::VirtualAssistant::- <<interface>> AgentsDAO::updateAgent()
TU141	Back-end::VirtualAssistant::- <<interface>> AgentsDAO::updateAgent()
TU142	Back-end::Rules::RulesService::- addRule()
TU143	Back-end::Rules::RulesService::- addRule()
TU144	Back-end::Rules::RulesService::- addRule()
TU145	Back-end::Rules::RulesService::- deleteRule()
TU146	Back-end::Rules::RulesService::- deleteRule()
TU147	Back-end::Rules::RulesService::- deleteRule()
TU148	Back-end::Rules::RulesService::- getRule()
TU149	Back-end::Rules::RulesService::- getRule()
TU150	Back-end::Rules::RulesService::- getRule()
TU151	Back-end::Rules::RulesService::- getRuleList()
TU152	Back-end::Rules::RulesService::- getRuleList()
TU153	Back-end::Rules::RulesService::- getRuleList()
TU154	Back-end::Rules::RulesService::- getTask()
TU155	Back-end::Rules::RulesService::- getTask()
TU156	Back-end::Rules::RulesService::- getTaskList()
TU157	Back-end::Rules::RulesService::- queryRule()
TU158	Back-end::Rules::RulesService::- queryRule()
TU159	Back-end::Rules::RulesService::- queryRule()
TU160	Back-end::Rules::RulesService::- updateRule()
TU161	Back-end::Rules::RulesService::- updateRule()
TU162	Back-end::Rules::RulesService::- updateRule()
TU163	Back-end::Users::UsersService::- addUser()
TU164	Back-end::Users::UsersService::- addUser()
TU165	Back-end::Users::UsersService::- addUser()
TU166	Back-end::Users::UsersService::- getUser()

Test	Metodi
TU167	Back-end::Users::UsersService::-getUser()
TU168	Back-end::Users::UsersService::-getUser()
TU169	Back-end::Users::UsersService::-getUserList()
TU170	Back-end::Users::UsersService::-getUserList()
TU171	Back-end::Users::UsersService::-getUserList()
TU172	Back-end::Users::UsersService::-removeUser()
TU173	Back-end::Users::UsersService::-removeUser()
TU174	Back-end::Users::UsersService::-removeUser()
TU175	Back-end::Users::UsersService::-updateUser()
TU176	Back-end::Users::UsersService::-updateUser()
TU177	Back-end::Users::UsersService::-updateUser()
TU178	Back-end::Events::VAMessageListener::-onMessage()
TU179	Back-end::Events::VAMessageListener::-onMessage()
TU180	Back-end::Events::VAMessageListener::-onMessage()
TU181	Back-end::Events::VAMessageListener::-onMessage()
TU182	Back-end::Events::VAMessageListener::-onMessage()
TU183	Back-end::VirtualAssistant::ApiAiVAAdapter::-query()
TU184	Back-end::VirtualAssistant::VAService::-query()
TU185	Back-end::VirtualAssistant::VAService::-query()
TU186	Back-end::VirtualAssistant::VAService::-query()
TU187	Back-end::VirtualAssistant::VAService::-query()
TU188	Back-end::ObserverAdapter::pause() Back-end::ObserverAdapter::resume()
TU189	Client::ApplicationManager::-ApplicationLocalRegistry::register()
TU190	Client::ApplicationManager::-ApplicationLocalRegistry::query()
TU191	Client::ApplicationManager::-ApplicationLocalRegistry::remove()
TU192	Client::ApplicationManager::-ApplicationRegistryLocalClient::query()
TU193	Client::ApplicationManager::-ApplicationRegistryLocalClient::register()

Test	Metodi
TU194	Client::ConversationApp::ConversationDispatcher::-dispatch()
TU195	Client::ConversationApp::ConversationView::-render()
TU196	Client::ApplicationManager::Manager::-runApplication()
TU197	Client::ApplicationManager::Manager::-runApplication()
TU198	Client::ApplicationManager::Manager::-setFrame()
TU199	Client::ConversationApp::MessageStore::-onCmd()
TU200	Client::ConversationApp::MessageStore::-onCmd()
TU201	Client::ConversationApp::MessageStore::-onCmd()
TU202	Client::ConversationApp::MessageStore::-onCmd()
TU203	Client::ConversationApp::MessageStore::-onCmd()
TU204	Client::ApplicationManager::State::-addApp()
TU205	Client::ApplicationManager::State::-getApp()
TU206	Client::ConversationApp::ConversationApp::-runCmd()
TU207	Client::ConversationApp::ConversationApp::-runCmd()
TU209	Client::Logic::HttpPromise::then()
TU210	Client::Logic::HttpPromise::then()
TU211	Client::Logic::Logic::sendData()
TU212	Client::Logic::Logic::sendData()
TU213	Client::Recorder::Recorder::start() Client::Recorder::Recorder::stop()

Tabella 12: Tracciamento Test di Unità-Metodi

C.13 Tracciamento Test di Validazione-Requisiti

Test	Requisito
TVFO1	RFO1
TVFO1.1.2	RFO1.1.2
TVFO2.1	RFO2.1
TVFO2.1.1.6	RFO2.1.1.6
TVFO2.1.2	RFO2.1.2
TVFO2.1.4	RFO2.1.4
TVFO2.2	RFO2.2
TVFO3.1	RFO3.1
TVFO5	RFO5
TVFO7	RFO7

Tabella 13: Tracciamento Test di Validazione-Requisiti

D Resoconto delle attività di verifica - RR

All'interno di questa sezione sono riportati gli esiti di tutte le attività di verifica effettuate sui documenti consegnati per la **Revisione dei requisiti**. Ove necessario sono state tratte conclusioni sui risultati e su come essi possano essere migliorati.

D.1 Qualità di processo

D.1.1 Miglioramento continuo tramite CMM

Per rendere le performance dei processi costantemente migliorabili e perseguire gli obiettivi quantitativi di miglioramento viene utilizzato il modello Capability Maturity Model (CMM).
All'inizio del periodo i processi si trovavano al livello 1 della scala CMM. In seguito, grazie alla stesura del documento "*Norme di Progetto v1.0.0*" sono state definite regole per ogni tipo di documentazione, strumenti da utilizzare e procedure da seguire. Questo ha permesso un maggiore controllo dei processi, che hanno ottenuto la ripetibilità, proprietà che caratterizza il livello 2 della scala CMM. Si può quindi affermare che i processi hanno raggiunto tale livello. Non si può ancora affermare di aver raggiunto il livello 3 del modello perchè al processo manca ancora la sua caratteristica principale, la proattività.

D.1.1.1 Soddisfacimento obiettivi di qualità

Di seguito sono riportati i valori ottenuti utilizzando le metriche definite sui seguenti obiettivi di qualità:

Obiettivo	Valore	Esito
Rispetto dei tempi - OPC2	10	ottimale
Rispetto dei costi - OPC3	-5%	accettabile

Tabella 14: Esiti del calcolo delle metriche sui processi

Il valore di OPC3 è dovuto al fatto che l'attività degli *Amministratori* ha richiesto più tempo del previsto in quanto è stato necessario modificare alcune funzioni del *software*$_{\text{g}}$ utilizzato per il tracciamento dei requisiti e dei casi d'uso, inoltre l'attività degli *Analisti* ha richiesto più tempo del previsto, in quanto si è dovuta fare un'analisi più approfondita rispetto a quella prefissata per una corretta stesura dei requisiti e dei casi d'uso. Questo è dovuto, in parte, all'interfaccia vocale da progettare, non convenzionale.

D.2 Qualità di prodotto

D.2.1 Documenti

D.2.1.1 Leggibilità e comprensibilità - OPDD1

Di seguito sono riportati i valori ottenuti calcolando l'indice Gulpease sui documenti:

Documento	**Gulpease**	**Esito**
"*Piano di Progetto v1.0.0*"	49	accettabile
"*Norme di Progetto v1.0.0*"	58	accettabile
"*Analisi dei Requisiti v1.0.0*"	66	ottimale
"*Piano di Qualifica v1.0.0*"	54	accettabile
"*Glossario v2.0.0*"	50	accettabile
"*Analisi SDK dei principali Assistenti Virtuali v1.0.0*"	67	ottimale
Verbale esterno 2016-12-17	66	ottimale
Verbale interno 2016-12-10	61	ottimale
Verbale interno 2016-12-19	62	ottimale

Tabella 15: Esiti del calcolo dell'indice Gulpease sui documenti

E Resoconto delle attività di verifica - RP

All'interno di questa sezione sono riportati gli esiti di tutte le attività di verifica effettuate sui documenti da consegnare per la **Revisione di progettazione**. Ove necessario sono state tratte conclusioni sui risultati e su come essi possano essere migliorati.

E.1 Qualità di processo

E.1.1 Miglioramento continuo tramite CMM

All'inizio del periodo i processi si trovavano al livello 2 della scala CMM. In seguito, grazie alla riorganizzazione del documento "*Norme di Progetto v2.0.0*" e alla maggiore esperienza dei membri del gruppo i processi e la loro organizzazione sono migliorati. Questo ci ha permesso di raggiungere il livello 3 della scala CMM.

E.1.1.1 Soddisfacimento obiettivi di qualità

Di seguito sono riportati i valori ottenuti utilizzando le metriche definite sui seguenti obiettivi di qualità:

Obiettivo	**Valore**	**Esito**
Disponibilità PragmaDB - OPC1	X	X
Rispetto dei tempi - OPC2	X	X
Rispetto dei costi - OPC3	X	X
Rischi non preventivati - OPC4	X	X
Structural Fan-In - OPC8	X	X
Structural Fan-Out - OPC9	X	X
Numero di metodi per classe - OPC10	X	X
Numero di parametri per metodo - OPC11	X	X

Tabella 16: Esiti del calcolo delle metriche sui processi

E.2 Qualità di prodotto

E.2.1 Documenti

E.2.1.1 Leggibilità e comprensibilità - OPDD1

Di seguito sono riportati i valori ottenuti calcolando l'indice Gulpease sui documenti:

Documento	**Gulpease**	**Esito**
"*Piano di Progetto v2.0.0*"	X	x
"*Norme di Progetto v2.0.0*"	X	x
"*Analisi dei Requisiti v2.0.0*"	x	x
"*Piano di Qualifica v2.0.0*"	x	x
"*Glossario v2.0.0*"	x	x
Verbale esterno x	x	x
Verbale interno x	x	x
Verbale interno x	x	x

Tabella 17: Esiti del calcolo dell'indice Gulpease sui documenti