AtAVi

Piano di qualifica v1.0.0

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

Versione	1.0.0
Data di redazione	2017-01-08
Redazione	Nicola Tintorri
	Andrea Magnan
Verifica	Pier Paolo Tricomi
Approvazione	Luca Bertolini
Uso	Esterno
Distribuzione	prof. Tullio Vardanega
	prof. Riccardo Cardin
	Zero12

Diario delle modifiche

Versione	Riepilogo	Autore	Ruolo	Data
1.0.0	Approvazione	Luca Bertolini	*Responsabile*	2017-01-08
0.2.2	Inseriti dati resoconto fase 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 fase 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

Introduzione

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 per poter garantire il miglior risultato qualitativo possibile.

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.

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*$_g$).

Riferimenti

Normativi

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

Informativi

- Piano di progetto: "*Piano di Progetto v1.0.0*";
- Slide del corso di Ingegneria del software - 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): https://en.wikipedia.org/wiki/Capability_Maturity_Model;
- Plan-Do-Check-Act (PDCA): https://en.wikipedia.org/wiki/PDCA.

Visione generale della strategia di gestione della qualità

Obiettivi qualitativi

In questa sezione vengono descritti gli obiettivi di qualità che il gruppo Co.Code decide di perseguire durante l'intero progetto. Ogni obiettivo viene definito in modo quantitativo per permettere al team di valutarne il raggiungimento. Vengono quindi fissati dei valori minimi che è obbligatorio superare per soddisfarlo e dei valori ottimali che ne rappresentano il pieno (ma non obbligatorio) conseguimento. A tale scopo vengono utilizzati modelli, metriche e standard.
Viene assegnato un codice identificativo ad ogni obiettivo, al fine di semplificarne il tracciamento con la metrica ad esso associata.
Il metodo di denominazione degli obiettivi è descritto in dettaglio nel documento "*Norme di Progetto v1.0.0*".

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.

Desideriamo che ogni processo possegga le seguenti caratteristiche ottimali:

- dovrebbe essere in grado di migliorarsi continuamente:
  - le sue performance sono costantemente misurabili;
  - deve perseguire sempre gli obiettivi quantitativi di miglioramento.
- dovrebbe rispettare i tempi indicati nel documento "*Piano di Progetto v1.0.0*";
- dovrebbe rispettare i costi dichiarati nel documento "*Piano di Progetto v1.0.0*";

Nelle sezioni successive vengono dichiarati gli obiettivi che il gruppo vuole perseguire. Per ognuno di essi, vengono definiti i criteri con cui si effettuano le misurazioni qualitative, specificando valori minimi e valori ottimali.

Miglioramento costante - OPC1

Per rendere le performance dei processi costantemente migliorabili e perseguire gli obiettivi quantitativi di miglioramento si è deciso di utilizzare il modello Capability Maturity Model (CMM). Si vuole raggiungere come valore minimo il livello 2 di questa scala, mentre, come valore ottimale, il livello 4.
Riassumendo:

- **Modello utilizzato:** CMM;
- **Soglia di accettabilità:** livello 2 previsto da CMM;
- **Soglia di ottimalità:** livello 4 previsto da CMM.

Per una più dettagliata descrizione del modello CMM consultare l'appendice A.
Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.2.1.

Rispetto della pianificazione - OPC2

Per capire se l'attività di un processo rispetta i tempi stabiliti dalla pianificazione all'interno del "*Piano di Progetto v1.0.0*" viene utilizzata la metrica Schedule Variance. Si desidera, come soglia minima accettabile, che un processo sia in ritardo non più del 5% rispetto alla pianificazione. Sarebbe ottimale, invece, non avere ritardi rispetto alla pianificazione o, ancora meglio, essere in anticipo.
Riassumendo:

- **Metrica utilizzata:** Schedule Variance;
- **Soglia di accettabilità:** ritardo al massimo del 5% rispetto alla pianificazione;
- **Soglia di ottimalità:** nessun ritardo (0%) o in anticipo rispetto alla pianificazione.

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.2.2.

Rispetto del budget - OPC3

Per capire se i costi di un processo rientrano nel budget stabilito dalla pianificazione all'interno del "*Piano di Progetto v1.0.0*" viene utilizzata la metrica Cost Variance. Si desidera, come soglia minima accettabile, che un processo non superi il 10% del budget pianificato. Sarebbe ottimale, invece, non superare i costi pianificati o, ancora meglio, avere costi inferiori.
Riassumendo:

- **Metrica utilizzata:** Cost Variance;
- **Soglia di accettabilità:** costi non superiori al 10% rispetto alla pianificazione;
- **Soglia di ottimalità:** costi pianificati (0%) o inferiori.

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.2.3.

Qualità di prodotto

Per garantire la migliore qualità del prodotto è necessario che i processi che lo producono abbiano alta qualità. A tale scopo, il gruppo Co.Code cercherà di seguire lo standard ISO/IEC 9126:2001 (vedi appendice B).
È prevista la realizzazione di due tipologie di prodotto: software e documenti. Nelle sezioni successive vengono dichiarati gli obiettivi di qualità di prodotto che il gruppo vuole perseguire, suddivisi per tipo. Per ognuno di essi, vengono definiti i criteri con cui si effettuano le misurazioni qualitative, specificando valori minimi e valori ottimali.

Qualità dei documenti

Gli obiettivi di qualità riguardanti i documenti prefissati dal gruppo Co.Code sono i seguenti:

- i documenti devono essere corretti a livello ortografico;
- i documenti devono essere corretti a livello concettuale;
- i documenti devono essere comprensibili da individui con licenza superiore.

Verrano ora descritti metriche e criteri utilizzati per garantire le caratteristiche sopra descritte, fissando valori minimi e valori ottimali.

Leggibilità e comprensibilità - OPDD1

Per determinare il grado di leggibilità e comprensibilità del documento, il gruppo ha deciso di utilizzare l'indice Gulpease. Si desidera come soglia minima accettabile un indice maggiore o uguale a 40 e, come soglia ottimale, un indice maggiore di 60.
Riassumendo:

- **Metrica utilizzata:** indice Gulpease;
- **Soglia di accettabilità:** indice maggiore o uguale a 40;
- **Soglia di ottimalità:** indice maggiore di 60.

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.1.1.

Correttezza ortografica - OPDD2

Per determinare il grado di correttezza ortografica del documento, il gruppo ha deciso di utilizzare la seguente metrica: percentuale di errori ortografici rinvenuti e non corretti. Pertanto, la soglia minima accettabile e la soglia ottimale coincidono e corrispondono a una correzione totale degli errori rinvenuti.
Riassumendo:

- **Metrica utilizzata:** percentuale di errori ortografici rinvenuti e non corretti;
- **Soglia di accettabilità:** tutti gli errori ortografici rinvenuti sono stati corretti (0%);
- **Soglia di ottimalità:** tutti gli errori ortografici rinvenuti sono stati corretti (0%).

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.1.2.

Correttezza concettuale - OPDD3

Per determinare il grado di correttezza concettuale del documento, il gruppo ha deciso di utilizzare la seguente metrica: percentuale di errori concettuali rinvenuti e non corretti. Si desidera come soglia minima accettabile che meno del 5% degli errori concettuali rinvenuti non siano stati corretti e, come soglia ottimale, che tutti gli errori concettuali rinvenuti siano stati corretti.
Riassumendo:

- **Metrica utilizzata:** percentuale di errori concettuali rinvenuti e non corretti;
- **Soglia di accettabilità:** non più del 5% degli errori concettuali rinvenuti non sono stati corretti;
- **Soglia di ottimalità:** tutti gli errori concettuali rinvenuti sono stati corretti (0%).

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.1.3.

Qualità del software

Gli obiettivi di qualità riguardanti il software prefissati dal gruppo Co.Code sono un sottoinsieme di quelli definiti nello standard ISO/IEC 9126:2001:

- il prodotto possiede le funzionalità descritte all'interno dei requisiti obbligatori;
- il prodotto è testato in ogni sua parte e in ogni situazione nella quale si può trovare;
- il prodotto è robusto e non interrompe l'esecuzione in seguito a situazioni anomale;

- il prodotto garantisce un funzionamento senza interruzioni;
- il codice risulta manutenibile e facilmente comprensibile.

Funzionalità obbligatorie - OPDS1

Il prodotto deve possedere tutte le funzionalità descritte nei requisiti obbligatori. Per determinare il numero di requisiti obbligatori soddisfatti viene usata come metrica la percentuale di requisiti obbligatori soddisfatti rispetto al totale.
Riassumendo:

- **Metrica utilizzata:** copertura requisiti obbligatori;
- **Soglia di accettabilità:** tutti i requisiti obbligatori sono soddisfatti (100%);
- **Soglia di ottimalità:** tutti i requisiti obbligatori sono soddisfatti (100%).

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.2.1.

Funzionalità desiderabili - OPDS2

Per determinare il numero di requisiti desiderabili soddisfatti viene usata come metrica la percentuale di requisiti desiderabili soddisfatti rispetto al totale. Si desidera come soglia minima accettabile che 70% dei requisiti desiderabili sia soddisfatto mentre, come soglia ottimale, che tutti i requisiti desiderabili siano soddisfatti.
Riassumendo:

- **Metrica utilizzata:** copertura requisiti desiderabili;
- **Soglia di accettabilità:** almeno 70% dei requisiti desiderabili soddisfatti;
- **Soglia di ottimalità:** tutti i requisiti desiderabili sono soddisfatti (100%).

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.2.2.

Funzionalità facoltative - OPDS3

Per determinare il numero di requisiti facoltativi soddisfatti viene usata come metrica la percentuale di requisiti facoltativi soddisfatti rispetto al totale. Si desidera come soglia minima accettabile che 0% dei requisiti facoltativi sia soddisfatto mentre, come soglia ottimale, che tutti i requisiti facoltativi siano soddisfatti.
Riassumendo:

- **Metrica utilizzata:** copertura requisiti facoltativi;
- **Soglia di accettabilità:** 0% dei requisiti facoltativi soddisfatti;
- **Soglia di ottimalità:** tutti i requisiti facoltativi sono soddisfatti (100%).

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.2.3.

Test passati - OPDS4

Il prodotto deve essere testato in ogni sua parte per garantirne il funzionamento. Vengono considerati solo i test riguardanti le funzionalità descritte nei requisiti. Si desidera come soglia minima accettabile che il numero di test passati sia almeno del 90% mentre, come soglia ottimale, 100%.
Riassumendo:

- **Metrica utilizzata:** percentuale di test passati;
- **Soglia di accettabilità:** almeno 90% dei test passati;
- **Soglia di ottimalità:** 100% dei test passati.

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.2.4.

Robustezza - OPDS5

Il prodotto deve essere robusto e non deve interrompere il suo funzionamento in seguito al verificarsi di situazioni anomale. Si desidera come soglia minima accettabile che il numero di situazioni anomale gestite sia almeno del 80% mentre, come soglia ottimale, più del 90%.
Riassumendo:

- **Metrica utilizzata:** Failure Avoidance;
- **Soglia di accettabilità:** gestite almeno 80% delle situazioni anomale;
- **Soglia di ottimalità:** gestite più del 90% delle situazioni anomale.

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.2.5.

Funzionamento senza interruzioni - OPDS6

Il prodotto deve garantire un funzionamento senza interruzioni. Si desidera come soglia minima accettabile che il numero di interruzioni evitate sia almeno del 80% mentre, come soglia ottimale, più del 90%.
Riassumendo:

- **Metrica utilizzata:** Breakdown Avoidance;
- **Soglia di accettabilità:** evitate almeno 80% delle interruzioni;
- **Soglia di ottimalità:** evitate più del 90% delle interruzioni.

Per approfondire la scelta delle soglie di accettabilità e ottimalità consultare la metrica alla sezione 3.2.3.2.6.

Manutenibilità e Comprensibilità del codice - OPDS7

Il grado di manutenibilità e comprensibilità del codice deriva dalla sua complessità e lunghezza. È importante quindi che il prodotto abbia codice manutenibile e privo di incomprensioni al suo interno.
Metriche e soglie verrano definite in dettaglio in versioni successive del documento.

Scadenze temporali

Le scadenze che il gruppo Co.Code ha deciso di rispettare sono riportate nel "*Piano di Progetto v1.0.0*".

La strategia di gestione della qualità nel dettaglio

Risorse

Per garantire un buon funzionamento del processo di verifica verranno impiegati i seguenti tipi di risorse:

- risorse umane;
- risorse hardware;
- risorse software.

Risorse necessarie

Risorse umane

Le risorse umane necessarie al processo di verifica sono i *Verificatori* e il *Responsabile*. Informazioni più dettagliate sui ruoli sono riportate nelle "*Norme di Progetto v1.0.0*".

Risorse hardware

Per eseguire la verifica, il gruppo dovrà avere a disposizione dei computer con un'adeguata potenza di calcolo in grado di sopportare il carico di lavoro.

Risorse software

Le risorse software necessarie alla verifica sono gli strumenti software che eseguono controlli sui documenti e verificano che non violino le "*Norme di Progetto v1.0.0*". Gli strumenti software devono avere le seguenti caratteristiche:

- rilevare eventuali errori ortografici;
- costruire e visualizzare in tempo reale il documento scritto in LaTeX(in modo che sia facile accorgersi di errori nell'utilizzo dei comandi).

Inoltre è necessario disporre di una piattaforma che raccolga i vari errori incontrati e li segnali ai componenti del gruppo che dovranno occuparsene.

Risorse disponibili

Risorse umane

Tutti i membri del gruppo sono a disposizione per eseguire operazioni di verifica. Ognuno dei componenti, a turno, ricoprirà il ruolo di *Responsabile* o di *Verificatore* come definito nel "*Piano di Progetto v1.0.0*".

Risorse hardware

Le risorse hardware disponibili sono i vari computer dei componenti del gruppo incaricati di svolgere il ruolo di *Responsabile* o *Verificatore*.

Risorse software

Le risorse software disponibili comprendono editor LaTeX con controlli integrati e script per controllare la leggibilità e la complessità dei documenti in riferimento all'indice Gulpease. Sarà disponibile anche il sistema di sollevamento delle issue offerto dalla piattaforma GitHub. Per maggiori informazioni sulla procedura di sollevamento e gestione delle issue si veda il documento "*Norme di Progetto v1.0.0*".

Misure e metriche

Misure

Ogni misura effettuata sui processi e sui prodotti deve essere confrontata con una scala. I valori della scala sono:

- **Negativo**: valore non accettabile, bisogna correggere gli errori presenti ed effettuare ulteriori verifiche;
- **Accettabile**: valore accettabile, l'oggetto sottoposto a verifica ha raggiunto una soglia minima.
- **Ottimale**: valore accettabile, l'oggetto sottoposto a verifica ha raggiunto le massime aspettative del team.

Metriche per processi

Viene assegnato un codice identificativo ad ogni metrica, al fine di semplificarne il tracciamento con l'obiettivo ad essa associato.
Il metodo di denominazione delle metriche è descritto in dettaglio nel documento "*Norme di Progetto v1.0.0*".

Capability Maturity Model - MPC1

Per controllare e verificare la qualità dei processi, vengono adottate le metriche fornite dal modello CMM. Per ogni fase di lavoro, viene associato un indice che descriverà la qualità della fase. L'indice, relativo ad una scala appartenente al CMM, può assumere i valori tra 1 (il peggiore) e 5 (il migliore). Il gruppo ha stabilito i seguenti intervalli:

- il valore 1 è considerato negativo;
- i valori 2 e 3 sono considerati accettabili;
- i valori 4 e 5 sono considerati ottimali.

Per approfondimenti sul modello CMM consultare l'appendice A.

Schedule Variance - MPC2

La presente metrica indica se le attività di progetto sono in anticipo o in ritardo rispetto a quelle pianificate nel "*Piano di Progetto v1.0.0*".
Viene calcolata come la differenza fra la data reale di fine di un'attività e la data pianificata di fine dell'attività stessa.
Se la Schedule Variance ha un valore maggiore di 0 allora il processo è in ritardo rispetto alla pianificazione. Viceversa, se il valore è minore di 0, allora il processo è in anticipo. Se invece il valore è uguale a 0 allora è in linea con quanto pianificato.
Il gruppo ha stabilito i seguenti intervalli:

- un valore maggiore del 5% rispetto alla pianificazione è considerato negativo;
- un valore minore o uguale al 5% rispetto alla pianificazione è considerato accettabile;
- valori minori o uguali a 0 sono considerati ottimali.

Cost Variance - MPC3

La presente metrica indica se i costi alla data corrente sono maggiori o minori rispetto a quanto dichiarato nel "*Piano di Progetto v1.0.0*".
Viene calcolata come la differenza fra consuntivo e preventivo.
Il gruppo ha stabilito i seguenti intervalli:

- un valore maggiore del 10% delle risorse preventivate per il processo è considerato negativo;
- un valore minore o uguale al 10% delle risorse preventivate per il processo è accettabile;
- un valore minore o uguale a 0 è considerato ottimale.

Metriche per i prodotti

Viene assegnato un codice identificativo ad ogni metrica, al fine di semplificarne il tracciamento con l'obiettivo ad essa associato.
Il metodo di denominazione delle metriche è descritto in dettaglio nel documento "*Norme di Progetto v1.0.0*".

Metriche per i documenti

La qualità di un documento dipende soprattutto dai suoi contenuti. La loro qualità, tuttavia, è difficilmente quantificabile allo stato attuale del progetto a causa dell'esperienza pressoché nulla del gruppo in quest'ambito. Si è deciso dunque di limitarsi a valutare parametri maggiormente oggettivi e soprattutto misurabili automaticamente attraverso strumenti software.

Indice di leggibilità - MPDD1

Per valutare la qualità del documento si è deciso di utilizzare l'indice di leggibilità. In particolare, è stato considerato l'indice Gulpease, studiato appositamente per la lingua italiana.
Questo particolare indice si basa sulla lunghezza della parola e sulla lunghezza della frase rispetto al numero di lettere. La formula per il suo calcolo è la seguente:

$$Indice\ Gulpease = 89 + \frac{300 * numeroFrasi + 10 * numeroLettere}{numeroParole} \quad (1)$$

Il risultato è compreso tra 0 e 100, dove valori alti indicano leggibilità elevata e viceversa. In generale, risulta che testi con un indice:

- inferiore a 80 risultano difficili da leggere per chi ha la licenza elementare;
- inferiore a 60 risultano difficili da leggere per chi ha la licenza media;
- inferiore a 40 risultano difficili da leggere per chi ha la licenza superiore.

Poichè la documentazione è rivolta a persone istruite, il gruppo ha stabilito i seguenti intervalli per l'indice:

- valori minori di 40 sono considerati negativi;

- valori compresi tra 40 e 60 sono considerati accettabili.
- valori maggiori di 60 sono considerati ottimali.

Errori ortografici rinvenuti e non corretti - MPDD2

Tale metrica è necessaria per capire quanto un documento sia corretto dal punto di vista ortografico. Supponendo che gli strumenti automatici siano in grado di trovare tutti gli errori ortografici all'interno di un testo, allora la correttezza ortografica non può che basarsi sul numero di errori rinvenuti ma non successivamente corretti. Notare che per errori corretti si intende un errore revisionato manualmente da parte di un *Verificatore*. Le correzioni automatiche, infatti, non sono molto attendibili. Il gruppo ha stabilito i seguenti intervalli:

- una percentuale di errori non corretti maggiore allo 0% è ritenuta negativa;
- una percentuale di errori non corretti pari allo 0% è ritenuta accettabile;
- una percentuale di errori non corretti pari allo 0% è ritenuta ottimale.

Errori concettuali rinvenuti e non corretti - MPDD3

Tale metrica è necessaria per capire quanto un documento sia corretto dal punto di vista concettuale. Supponendo che, in seguito alle revisioni, siano stati individuati tutti gli errori concettuali all'interno di un testo, allora la correttezza concettuale non può che basarsi sul numero di errori rinvenuti ma non successivamente corretti. Notare che per errori corretti si intende un errore revisionato manualmente da parte di un *Verificatore*. Il gruppo ha stabilito i seguenti intervalli:

- una percentuale di errori non corretti maggiore al 5% è ritenuta negativa;
- una percentuale di errori non corretti minore o uguale al 5% è ritenuta accettabile;
- una percentuale di errori non corretti pari allo 0% è ritenuta ottimale;

Metriche per il software

Il gruppo Co.Code ha deciso di adottare alcune metriche che hanno il compito di monitorare la qualità interna, la qualità esterna e la qualità in uso del software. Tali metriche sono un sottoinsieme di quelle difinite nello standard ISO/IEC 9126:2001.
Ogni metrica scelta viene associata ad una caratteristica di qualità presente all'interno dello standard:

Metriche scelte	**Caratteristiche di qualità**
MPDS1 - Copertura requisiti obbligatori	Funzionalità
MPDS2 - Copertura requisiti desiderabili	Funzionalità
MPDS3 - Copertura requisiti facoltativi	Funzionalità
MPDS4 - Test passati richiesti	Affidabilità
MPDS5 - Failure Avoidance	Affidabilità
MPDS6 - Breakdown Avoidance	Affidabilità

Tabella 1: Mappa metriche-caratteristiche

Copertura requisiti obbligatori - MPDS1

La seguente metrica controlla quanti requisiti obbligatori sono stati soddisfatti. Viene calcolata come rapporto, espresso in percentuale, tra i requisiti obbligatori soddisfatti e il numero totale dei requisiti obbligatori.

$$\textit{Copertura requisiti obbligatori} = 100 * \frac{\#\textit{requisiti obbligatori soddisfatti}}{\#\textit{requisiti obbligatori totali}} \tag{2}$$

Il gruppo ha stabilito i seguenti intervalli:

- una percentuale minore del 100% è ritenuta negativa;
- una percentuale uguale al 100% è ritenuta accettabile;
- una percentuale uguale al 100% ottimale.

Copertura requisiti desiderabili - MPDS2

La seguente metrica controlla quanti requisiti desiderabili sono stati soddisfatti. Viene calcolata come rapporto, espresso in percentuale, tra i requisiti desiderabili soddisfatti e il numero totale dei requisiti desiderabili.

$$\textit{Copertura requisiti desiderabili} = 100 * \frac{\#\textit{requisiti desiderabili soddisfatti}}{\#\textit{requisiti desiderabili totali}} \tag{3}$$

Il gruppo ha stabilito i seguenti intervalli:

- una percentuale minore del 70% è ritenuta negativa;
- una percentuale maggiore o uguale a 70% è ritenuta accettabile;
- una percentuale uguale al 100% ottimale.

Copertura requisiti facoltativi - MPDS3

La seguente metrica controlla quanti requisiti facoltativi sono stati soddisfatti. Viene calcolata come rapporto, espresso in percentuale, tra i requisiti facoltativi soddisfatti e il numero totale dei requisiti facoltativi.

$$\textit{Copertura requisiti facoltativi} = 100 * \frac{\#\textit{requisiti facoltativi soddisfatti}}{\#\textit{requisiti facoltativi totali}} \tag{4}$$

Il gruppo ha stabilito i seguenti intervalli:

- una percentuale maggiore o uguale a 0% è ritenuta accettabile;
- una percentuale uguale al 100% ottimale.

Percentuale test passati - MPDS4

La metrica controlla che il rapporto, espresso in percentuale, tra il numero di test passati e il numero di test totali rientri tra i valori definiti. Questo ci permette di valutare se il prodotto supera la maggior parte dei test.

$$\textit{Test passati richiesti} = 100 * \frac{\#\textit{ di test passati}}{\#\textit{ di test totali}} \tag{5}$$

Il gruppo ha stabilito i seguenti intervalli:

- una percentuale minore del 90% è ritenuta negativa;
- una percentuale maggiore o uguale a 90% è ritenuta accettabile;

- una percentuale uguale a 100% è ritenuta ottimale;

Failure Avoidance - MPDS5

La metrica controlla che il rapporto, espresso in percentuale, tra il numero di situazioni anomale gestite e il numero di situazioni anomale presentate rientri tra i valori definiti.

$$Failure\ Avoidance = 100 * \frac{\#\ situazioni\ anomale\ gestite}{\#\ situazioni\ anomale\ presentate} \tag{6}$$

Il gruppo ha stabilito i seguenti intervalli:

- una percentuale minore del 80% è ritenuta negativa;
- una percentuale compresa tra 80% e 90% è ritenuta accettabile;
- una percentuale maggiore di 90% è ritenuta ottimale.

Breakdown Avoidance - MPDS6

La metrica controlla che la percentuale di interruzioni evitate dal prodotto rientri tra i valori definiti. Il valore su cui si applicherà la metrica verrà calcolato come il complemento delle interruzioni verificate. Questa metrica ci permette di controllare che il prodotto lavori senza interruzioni.

$$Breakdown\ Avoidance = 100 * \left(1 - \frac{\#\ di\ interruzioni}{\#\ situazioni\ anomale\ presentate}\right) \tag{7}$$

Il gruppo ha stabilito i seguenti intervalli:

- una percentuale minore del 80% è ritenuta negativa;
- una percentuale compresa tra 80% e 90% è ritenuta accettabile;
- una percentuale maggiore di 90% è ritenuta ottimale.

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 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.

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;
- aumento della produttività dell'organizzazione di sviluppo;
- riduzione dei tempi di sviluppo.

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.

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.

Standard ISO/IEC 9126

Lo Standard ISO/IEC 9126 si suddivide in quattro parti:

- modello della qualità del software (9126-1);
- metriche per la qualità esterna (9126-2);
- metriche per la qualità interna (9126-3);
- metriche per la qualità in uso (9126-4).

Lo standard tratta la qualità del software da tre punti di vista:

- **Qualità interna:** è la qualità del software non eseguibile e fa quindi riferimento alle caratteristiche implementative del software quali l'architettura e il codice sorgente;
- **Qualità esterna:** è la qualità del software nel momento in cui esso viene eseguito e testato in un ambiente di prova;
- **Qualità in uso:** è la qualità del software dal punto di vista dell'utente che ne fa uso all'interno di uno specifico sistema e contesto.

Modello della qualità del software

Nella prima parte (9126-1) vengono descritti i modelli per la qualità esterna, interna ed in uso.

Modello della qualità esterna ed interna

Il modello di qualità esterna ed interna stabilito nella prima parte dello standard è classificato in sei caratteristiche generali:

- **Funzionalità:** è la capacità di un software di fornire funzioni in grado di soddisfare specifici bisogni in un determinato contesto;
- **Affidabilità:** è la capacità di un software di mantenere un certo livello di prestazioni in condizioni specifiche di utilizzo in un intervallo di tempo fissato;
- **Usabilità:** è la capacità di un software di essere capito e utilizzato da utenti in un determinato contesto;
- **Efficienza:** è la capacità di un software di fornire un certo livello di prestazioni con la minor quantità di risorse possibile;
- **Manutenibilità:** è la capacità del software di essere modificato, includendo correzioni, miglioramenti o adattamenti;
- **Portabilità:** è la capacità del software di essere trasportato da un ambiente di lavoro ad un altro.

Tali caratteristiche sono misurabili attraverso delle metriche.

Modello della qualità in uso

Gli attributi presenti nel modello relativo alla qualità del software in uso sono classificati in quattro categorie:

- **Efficacia:** è la capacità del software di permettere agli utenti di raggiungere gli obiettivi specificati con accuratezza e completezza;

- **Produttività:** è la capacità di permettere all'utente di utilizzare un numero di risorse in relazione all'efficienza raggiunta in uno specifico contesto di utilizzo;
- **Sicurezza:** è la capacità del software di raggiungere un livello accettabile di rischi per i dati, le persone, il business, la proprietà o gli ambienti in uno specifico contesto di utilizzo;
- **Soddisfazione:** è la capacità del software di soddisfare gli utenti in uno specifico contesto di utilizzo.

Metriche per la qualità del software

Nelle restanti tre parti (9126-2, 9126-3, 9126-4) vengono trattate le metriche per la qualità esterna, interna e in uso.

Metriche per la qualità esterna

Le metriche esterne misurano i comportamenti del software rilevabili dai test, dall'operatività e dall'osservazione durante la sua esecuzione. L'esecuzione del software è fatta in un contesto tecnico rilevante. Le metriche esterne sono scelte in base alle caratteristiche che il prodotto finale dovrà dimostrare durante la sua esecuzione in esercizio.

Metriche per la qualità interna

Le metriche interne si applicano al software non eseguibile (come il codice sorgente) e alla documentazione. Le misure effettuate permettono di prevedere il livello di qualità esterna ed in uso del prodotto finale poiché gli attributi interni influenzano le caratteristiche esterne e quelle in uso.

Metriche per la qualità in uso

Le metriche della qualità in uso rappresentano il punto di vista dell'utente e misurano il grado con cui il software permette agli utenti di svolgere le proprie attività con efficacia, produttività, sicurezza e soddisfazione nel contesto operativo previsto.

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, 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. 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.

Test

Al fine di produrre software di qualità, il gruppo ha strutturato dei test atti a verificare che le funzionalità del software prodotto 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:** test che hanno lo scopo di verificare che tutte le funzionalità richieste dal proponente 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 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;
- **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.

Test di Validazione

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

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

Id Test	Descrizione	Stato
TVFO2.1	L'utente deve verificare che il sistema permetta la creazione di una nuova direttiva. 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*
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*

Id Test	Descrizione	Stato
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 2: Test di Validazione

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 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 le informazioni raccolte durante l'interazione con l'ospite.	*Non Implementato*
TSVO1.1	Vogliamo testare che il software 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 Google Chrome versione 53 o superiore.	*Non Implementato*

Tabella 3: Test di Sistema

Test di Integrazione

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

Id Test	Descrizione	Stato
TI1	Vogliamo verificare che `Recorder`, `Logic`, `Utility`, `Recorder`, `TTS` 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 IBM Watson, api.ai, Slack e `WebAPI`.	*Non Implementato*
TI3	Vogliamo verificare che le seguenti classi, contenute in `Client::ApplicationManager`, interagiscano tra loro correttamente: `ApplicationManagerObserver`, `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*

Id Test	Descrizione	Stato
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*
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*

Tabella 4: Test di Integrazione

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 `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 `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.	*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 `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 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 5: Test di Unità

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 6: Tracciamento Test di Validazione-Requisiti

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::Logic`	TI4
`Client::Recorder`	TI5
`Client::TTS`	TI6
`Client::Utility`	TI7

Tabella 7: Tracciamento Componenti-Test di Integrazione

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::- addFunction()	TU100 TU101
Back-end::Rules::TasksDAODynamoDB::- getFunction()	TU102 TU103
Back-end::Rules::TasksDAODynamoDB::- getFunctionList()	TU104 TU105
Back-end::Rules::TasksDAODynamoDB::- removeFunction()	TU106 TU107
Back-end::Rules::TasksDAODynamoDB::- updateFunction()	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::Utility::ErrorObserver::-next()	TU3
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

Metodo	Test
Back-end::VirtualAssistant::VAService::-query()	TU184 TU185 TU186 TU187
Client::ApplicationLocalRegistry::-query()	TU190
Client::ApplicationLocalRegistry::-register()	TU189
Client::ApplicationLocalRegistry::-remove()	TU191
Client::ApplicationManager::-ApplicationRegistryLocalClient::query()	TU192
Client::ApplicationManager::-ApplicationRegistryLocalClient::register()	TU193
Client::ApplicationManager::ConversationDispatcher::-dispatch()	TU194
Client::ApplicationManager::ConversationView::-render()	TU195
Client::ApplicationManager::Manager::-runApplication()	TU196 TU197
Client::ApplicationManager::Manager::-setFrame()	TU198
Client::ApplicationManager::MessageStore::-onCmd()	TU199 TU200 TU201 TU202 TU203
Client::ApplicationManager::State::-addApp()	TU204
Client::ApplicationManager::State::-getApp()	TU205
Client::ConversationApp::runCmd()	TU206 TU207
Client::ErrorSubject::next()	TU208
Client::Logic::HttpPromise::then()	TU209 TU210
Client::Logic::Logic::sendData()	TU211 TU212
Client::Recorder::Recorder::start()	TU213
Client::Recorder::Recorder::stop()	TU213

Tabella 8: Tracciamento Metodi-Test di Unità

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 9: Tracciamento Requisiti-Test di Sistema

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 10: Tracciamento Requisiti-Test di Validazione

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`

Tabella 11: Tracciamento Test di Integrazione-Componenti

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 12: Tracciamento Test di Sistema-Requisiti

Tracciamento Test di Unità-Metodi

Test	Metodi
TU1	Back-end::AdministrationWebhookService::-webhook()
TU2	Back-end::AdministrationWebhookService::-webhook()
TU3	Back-end::Utility::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()

Test	Metodi
TU26	Back-end::APIGateway::VocalAPI::-queryLambda()
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()

Test	Metodi
TU53	Back-end::Conversations::ConversationsDAODynamoDB::-addMessage()
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()

Test	Metodi
TU81	Back-end::Notifications::NotificationService::-getChannelList()
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::-addFunction()
TU101	Back-end::Rules::TasksDAODynamoDB::-addFunction()
TU102	Back-end::Rules::TasksDAODynamoDB::-getFunction()
TU103	Back-end::Rules::TasksDAODynamoDB::-getFunction()
TU104	Back-end::Rules::TasksDAODynamoDB::-getFunctionList()
TU105	Back-end::Rules::TasksDAODynamoDB::-getFunctionList()
TU106	Back-end::Rules::TasksDAODynamoDB::-removeFunction()
TU107	Back-end::Rules::TasksDAODynamoDB::-removeFunction()
TU108	Back-end::Rules::TasksDAODynamoDB::-updateFunction()
TU109	Back-end::Rules::TasksDAODynamoDB::-updateFunction()

Test	Metodi
TU110	Back-end::STT::STTWatsonAdapter::-speechToText()
TU111	Back-end::Rules::TaskObserver::next()
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()

Test	Metodi
TU139	Back-end::VirtualAssistant::- <<interface>> AgentsDAO::removeAgent()
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()

Test	Metodi
TU166	Back-end::Users::UsersService::-getUser()
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::ApplicationLocalRegistry::-register()
TU190	Client::ApplicationLocalRegistry::-query()
TU191	Client::ApplicationLocalRegistry::-remove()
TU192	Client::ApplicationManager::-ApplicationRegistryLocalClient::query()

Test	Metodi
TU193	`Client::ApplicationManager::-ApplicationRegistryLocalClient::register()`
TU194	`Client::ApplicationManager::ConversationDispatcher::-dispatch()`
TU195	`Client::ApplicationManager::ConversationView::-render()`
TU196	`Client::ApplicationManager::Manager::-runApplication()`
TU197	`Client::ApplicationManager::Manager::-runApplication()`
TU198	`Client::ApplicationManager::Manager::-setFrame()`
TU199	`Client::ApplicationManager::MessageStore::-onCmd()`
TU200	`Client::ApplicationManager::MessageStore::-onCmd()`
TU201	`Client::ApplicationManager::MessageStore::-onCmd()`
TU202	`Client::ApplicationManager::MessageStore::-onCmd()`
TU203	`Client::ApplicationManager::MessageStore::-onCmd()`
TU204	`Client::ApplicationManager::State::-addApp()`
TU205	`Client::ApplicationManager::State::-getApp()`
TU206	`Client::ConversationApp::runCmd()`
TU207	`Client::ConversationApp::runCmd()`
TU208	`Client::ErrorSubject::next()`
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 13: Tracciamento Test di Unità-Metodi

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 14: Tracciamento Test di Validazione-Requisiti

Resoconto delle attività di verifica - fase AR

All'interno di questa sezione sono riportati gli esiti di tutte le attività di verifica effettuate nell'arco della fase AR, come previsto dal documento "*Piano di Progetto v1.0.0*". Ove necessario sono state tratte conclusioni sui risultati e su come essi possano essere migliorati.

Verifica sui processi

Processo di documentazione

Miglioramento costante

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 della fase 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 del processo di documentazione, che ha ottenuto la ripetibilità, proprietà che caratterizza il livello 2 della scala CMM. Si può quindi affermare che il processo di documentazione ha 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.
Secondo le metriche definite il valore raggiunto rappresenta la **soglia minima accettabile**, ma nelle prossime fasi il gruppo si impegnerà a raggiungere la soglia ottimale (sfruttando PDCA).

Rispetto della pianificazione

Per capire se l'attività di un processo rispetta i tempi stabiliti dalla pianificazione all'interno del "*Piano di Progetto v1.0.0*" viene utilizzata la metrica Schedule Variance. Si desidera, come soglia minima accettabile, che un processo sia in ritardo non più del 5% rispetto alla pianificazione. Sarebbe ottimale, invece, non avere ritardi rispetto alla pianificazione o, ancora meglio, essere in anticipo.
Di seguito sono riportati i valori ottenuti calcolando la Schedule Variance sui tempi di stesura di ogni documento nella fase AR:

Documento	**Schedule Variance**	**Esito**
"*Piano di Progetto v1.0.0*"	0%	ottimale
"*Norme di Progetto v1.0.0*"	0%	ottimale
"*Analisi dei Requisiti v1.0.0*"	-10%	ottimale
"*Piano di Qualifica v1.0.0*"	0%	ottimale
"*Glossario v2.0.0*"	0%	ottimale
"*Analisi SDK dei principali Assistenti Virtuali v1.0.0*"	0%	ottimale
"*Studio di Fattibilità v1.0.0*"	0%	ottimale

Tabella 15: Esiti del calcolo della Schedule Variance sul processo di documentazione durante la fase AR

Rispetto del budget

Per capire se i costi di un processo rientrano nel budget stabilito dalla pianificazione all'interno del "*Piano di Progetto v1.0.0*" viene utilizzata la metrica Cost Variance. Si desidera, come soglia minima accettabile, che un processo non superi il 10% del budget pianificato. Sarebbe ottimale, invece, non superare i costi pianificati o, ancora meglio, spendere meno.
Di seguito sono riportati i valori ottenuti calcolando la Cost Variance sui tempi di stesura di ogni documento nella fase AR:

Documento	**Cost Variance**	**Esito**
"*Piano di Progetto v1.0.0*"	0%	ottimale
"*Norme di Progetto v1.0.0*"	+10%	accettabile
"*Analisi dei Requisiti v1.0.0*"	+11%	non accettabile
"*Piano di Qualifica v1.0.0*"	+16%	non accettabile
"*Glossario v2.0.0*"	+6%	accettabile
"*Analisi SDK dei principali Assistenti Virtuali v1.0.0*"	+14%	non accettabile
"*Studio di Fattibilità v1.0.0*"	+10%	accettabile

Tabella 16: Esiti del calcolo della Cost Variance sul processo di documentazione durante la fase AR

Tali valori sono dovuti al fatto che l'attività degli *Amministratori* ha richiesto più tempo del previsto in quanto è stato necessario modificare alcune funzioni del software 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.

Processo di verifica

Miglioramento costante

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 della fase 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, oltre che alla definizione di metriche in questo documento. Questo ha permesso un maggiore controllo del processo di verifica, che ha ottenuto la ripetibilità, proprietà che caratterizza il livello 2 della scala CMM. Si può quindi affermare che il processo di documentazione ha 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.
Secondo le metriche definite il valore raggiunto rappresenta la **soglia minima accettabile**, ma nelle prossime fasi il gruppo si impegnerà a raggiungere la soglia ottimale (sfruttando PDCA).

Rispetto della pianificazione

Per capire se l'attività di un processo rispetta i tempi stabiliti dalla pianificazione all'interno del "*Piano di Progetto v1.0.0*" viene utilizzata la metrica Schedule Variance. Si desidera, come soglia minima accettabile, che un processo sia in ritardo non più del 5% rispetto alla pianificazione.

Sarebbe ottimale, invece, non avere ritardi rispetto alla pianificazione o, ancora meglio, essere in anticipo.
Di seguito è riportato il valore ottenuto calcolando la Schedule Variance sul processo di verifica nella fase AR:

Processo	**Schedule Variance**	**Esito**
Processo di verifica	-6%	ottimale

Tabella 17: Esiti del calcolo della Schedule Variance sul processo di verifica durante la fase AR

Rispetto del budget

Per capire se i costi di un processo rientrano nel budget stabilito dalla pianificazione all'interno del "*Piano di Progetto v1.0.0*" viene utilizzata la metrica Cost Variance. Si desidera, come soglia minima accettabile, che un processo non superi il 10% del budget pianificato. Sarebbe ottimale, invece, non superare i costi pianificati o, ancora meglio, spendere meno.
Di seguito è riportato il valore ottenuto calcolando la Cost Variance sul processo di verifica nella fase AR:

Processo	**Cost Variance**	**Esito**
Processo di verifica	-25%	ottimale

Tabella 18: Esiti del calcolo della Cost Variance sul processo di verifica durante la fase AR

Verifica sui prodotti

Documenti

Leggibilità e comprensibilità

Per determinare il grado di leggibilità e comprensibilità del documento, il gruppo ha deciso di utilizzare l'indice Gulpease. Si desidera come soglia minima accettabile un indice maggiore o uguale a 40 e, come soglia ottimale, un indice maggiore di 60.
Di seguito sono riportati i valori ottenuti calcolando l'indice Gulpease sui documenti della fase AR:

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 19: Esiti del calcolo dell'indice Gulpease sui documenti della fase AR

Correttezza ortografica

Per determinare il grado di correttezza ortografica del documento, il gruppo ha deciso di utilizzare la seguente metrica: percentuale di errori ortografici rinvenuti e non corretti. Pertanto, la soglia minima accettabile e la soglia ottimale coincidono e corrispondono a una correzione totale degli errori rinvenuti.
Di seguito è riportato il numero di errori ortografici trovati:

Errori ortografici	43

Tabella 20: Errori ortografici rinvenuti durante la fase AR

Tutti gli errori ortografici rinvenuti sono stati corretti, quindi è stato raggiunto l'obiettivo **ottimale**.

Correttezza concettuale

Per determinare il grado di correttezza concettuale del documento, il gruppo ha deciso di utilizzare la seguente metrica: percentuale di errori concettuali rinvenuti e non corretti. Si desidera come soglia minima accettabile che non più del 5% degli errori concettuali rinvenuti non siano stati corretti e, come soglia ottimale, che tutti gli errori concettuali rinvenuti siano stati corretti.
Di seguito è riportato il numero di errori concettuali trovati:

Errori concettuali	24

Tabella 21: Errori concettuali rinvenuti durante la fase AR

Tutti gli errori concettuali rinvenuti sono stati corretti, quindi è stato raggiunto l'obiettivo **ottimale**.