MegAlexa

Arricchitore di skill di Amazon Alexa

Norme di progetto

Gruppo ZeroSeven

Versione	2.0.2
Data Redazione	2018-12-08
Redazione	Gian Marco Bratzu
	Ludovico Brocca
	Matteo Depascale
	Andrea Deidda
	Bianca Ciuche
Verifica	Mirko Franco
	Stefano Zanatta
Approvazione	Stefano Zanatta
Uso	Interno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo ZeroSeven
Email di contatto	zerosevenswe@gmail.com

Registro delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
2.0.2	2019-04-01	Stesura sezioni §2.2.3, §2.2.4, §2.2.5 e §2.2.6	Mirko Franco	??
2.0.1	2019-03-23	Modifica 2.1.3 Corretti numeri di sezione	Mirko Franco	??
2.0.0	2019-03-07	Approvazione per il rilascio	Gian Marco Bratzu	Responsabile
1.2.0	2019-03-02	Verifica documento	Andrea Deidda	Verificatore
1.1.2	2019-02-13	Stesura §4.2	Ludovico Brocca	Amministratore
1.1.1	2019-02-13	Modifica§3.4.2.2	Ludovico Brocca	Amministratore
1.1.0	2019-02-07	Verifica §B.1, B.2, §2.2.2	Gian Marco Bratzu	Verificatore
1.0.4	2019-02-04	Stesura §B.1	Ludovico Brocca	Analista
1.0.3	2019-02-03	Modifica §B.2	Stefano Zanatta	Amministratore
1.0.2	2019-02-02	Stesura §B.2	Bianca Andreea Ciuche	Amministratore
1.0.1	2018-01-12	Stesura §2.2.2	Mirko Franco	Amministratore
1.0.0	2018-01-09	Approvazione per il rilascio	Stefano Zanatta	Responsabile
0.2.0	2018-12-29	Verifica documento	Stefano Zanatta	Verificatore
0.1.0	2018-12-18	Verifica §3	Mirko Franco	Verificatore
0.0.6	2018-12-21	Modifica §1	Andrea Deidda	Amministratore

0.0.5	2018-12-17	Stesura §4	Ludovico Brocca	Amministratore
0.0.4	2018-12-16	Stesura §2	Matteo Depascale	Amministratore
0.0.3	2018-12-16	Stesura §1	Bianca Ciuche	Amministratore
0.0.2	2018-12-10	Stesura §3	Gian Marco Bratzu	Amministratore
0.0.1	2018-12-08	Struttura documento	Ludovico Brocca	Amministratore

Indice

1 Introduzione **6**
- 1.1 Scopo del documento . . . 6
- 1.2 Scopo del prodotto . . . 6
- 1.3 Glossario . . . 7
- 1.4 Riferimenti . . . 7
  - 1.4.1 Normativi . . . 7
  - 1.4.2 Informativi . . . 7

2 Processi primari **8**
- 2.1 Fornitura . . . 8
  - 2.1.1 Studio di fattibilità . . . 8
  - 2.1.2 Rapporti di fornitura con la Proponente . . . 9
  - 2.1.3 Documentazione fornita . . . 9
- 2.2 Sviluppo . . . 9
  - 2.2.1 Analisi dei requisiti . . . 9
    - 2.2.1.1 Fonti dei requisiti . . . 10
    - 2.2.1.2 Classificazione dei requisiti . . . 10
    - 2.2.1.3 Casi d'uso . . . 11
    - 2.2.1.4 Codice identificativo CU . . . 11
  - 2.2.2 Progettazione . . . 11
    - 2.2.2.1 Scopo . . . 11
    - 2.2.2.2 Diagrammi . . . 13
  - 2.2.3 Diagrammi delle classi . . . 13
  - 2.2.4 Diagrammi dei package . . . 16
  - 2.2.5 Diagrammi di sequenza . . . 17
  - 2.2.6 Diagrammi di attività . . . 18
  - 2.2.7 Codifica . . . 19
    - 2.2.7.1 Suddivisione dei package dell'applicazione . . . 19
    - 2.2.7.2 Suddivisione dei package della skill . . . 20
    - 2.2.7.3 Stile di codifica: regole generali . . . 20
    - 2.2.7.4 Utilizzo delle librerie . . . 21

2.2.7.5 Indentazione 21
2.2.7.6 Linee vuote 21
2.2.7.7 Dichiarazione di variabili 22
2.2.7.7.1 Kotlin 22
2.2.7.7.2 Node.JS 22
2.2.7.8 Parentesizzazione 24
2.2.7.8.1 NodeJS 25
2.2.7.9 Commenti 25
2.2.7.10 Codifica di XML 26
2.2.7.10.1 Apertura e chiusura dei tag 26
2.2.7.10.2 Dichiarazione di id e stringhe univoche 27
2.2.7.11 Intestazione 27
2.2.7.12 Versionamento 28

3 Processi di Supporto 29
3.1 Gestione della configurazione 29
3.1.1 Versionamento 29
3.1.1.1 Comandi basilari 29
3.1.1.2 Creazione della copia locale 29
3.1.1.3 Comandi principali 29
3.1.1.4 Git Flow 30
3.1.1.5 Gestione dei rilasci 30
3.2 Documentazione 30
3.2.1 Nomenclatura 31
3.2.1.1 Ciclo di vita della documentazione 31
3.2.2 Struttura dei documenti 32
3.2.2.1 Frontespizio 32
3.2.2.2 Registro delle modifiche 33
3.2.2.3 Indice 33
3.2.2.4 Elenco delle immagini e tabelle 33
3.2.2.5 Struttura delle pagine interne 33
3.2.3 Norme tipografiche 34
3.2.3.1 Punteggiatura 34
3.2.3.2 Formati 34
3.2.3.3 Stile del testo 35
3.2.3.4 Norme redazionali 35
3.2.3.5 Componenti grafiche 36
3.2.4 Strumenti a supporto della documentazione 36
3.2.4.1 TeXstudio 36
3.2.4.2 Script automatici 36
3.3 Garanzia di qualità 37

3.3.1 Classificazione metriche e obiettivi 37
3.3.1.1 Classificazioni degli obiettivi 37
3.3.1.2 Classificazione delle metriche 37
3.4 Verifica 38
3.4.1 Verifica di processi 38
3.4.2 Verifica dei prodotti 38
3.4.2.1 Verifica dei documenti 38
3.4.2.1.1 Analisi statica 38
3.4.2.1.2 Procedura di controllo dei documenti 39
3.4.2.2 Verifica del prodotti software 39
3.4.2.2.1 Analisi dinamica 39
3.4.2.2.2 Tecnologie adottate 39
3.4.2.2.2.1 JUnit 39
3.4.2.2.2.2 Mocha e Chai Assert 39
3.4.2.2.3 Test di Unità 40
3.4.2.2.4 Test di Integrazione 40
3.4.2.2.5 Test di Sistema 40
3.4.2.2.6 Test di accettazione 40
3.4.2.2.7 Tracciamento dei test 40
3.4.3 Procedure di verifica del software 41
3.4.3.1 Codifica e esecuzione 41
3.4.3.1.1 JUnit: 41
3.4.3.1.1.1 Esecuzione: 42
3.4.3.1.2 Mocha e Chai Assert: 42
3.4.3.1.2.1 Esecuzione: 42
3.4.3.2 Test per AWS Lambda e Api Gateway 42
3.4.3.3 Test sulla skill MegAlexa 42
3.5 Validazione 43
3.5.1 Procedura 43

4 Processi organizzativi 44
4.1 Gestione di processo 44
4.1.1 Comunicazioni interne 44
4.1.2 Comunicazione esterne 44
4.1.3 Gestione delle riunioni 44
4.1.3.1 Verbali di riunione 44
4.1.4 Riunioni interne 45
4.1.5 Riunioni esterne 45
4.1.6 Identificazione delle decisioni 45
4.1.7 Ruoli di progetto 46
4.1.7.1 Responsabile 46

4.1.7.2 Amministratore 46
4.1.7.3 Analista 46
4.1.7.4 Progettista 46
4.1.7.5 Programmatore 47
4.1.7.6 Verificatore 47
4.1.8 Ticketing 47
4.2 Calcolo delle ore 47
4.2.1 Avvio del timer 47
4.3 Ambiente di lavoro 47
4.3.1 Coordinamento 48
4.3.1.1 Versionamento 48
4.3.1.2 Pianificazione e Ticketing 48
4.3.2 Documentazione 48
4.3.2.1 LaTeX 48
4.3.2.2 Editor 48
4.3.2.3 Script 48
4.3.2.4 Diagrammi UML 48
4.3.2.5 Creazione diagrammi di Gantt 49
4.3.3 Ambiente di sviluppo 49
4.3.3.1 Sistemi operativi 49
4.3.4 Ambiente di verifica 49
4.3.4.1 Documenti 49
4.4 Formazione del personale 49

A Lista di controllo **51**

B Metriche **52**
B.1 Metriche per la qualità di processo 52
B.1.1 MP001 - Schedule Variance 52
B.1.2 MP002 - Cost Variance 52
B.1.3 MP003 - SPICE capability level 53
B.1.4 MP004 - SPICE process attributes 53
B.1.5 MP005 - Occorrenza rischi non previsti 53
B.1.6 MP006 - Indisponibilità dei servizi 53
B.1.7 MP007 - Complessità ciclomatica 54
B.1.8 MP08 - Numero di parametri per metodo 54
B.1.9 MP009 - Numero di livelli di annidamento 54
B.1.10 MP010 - Attributi per classe 54
B.1.11 MP011 - Tempo medio del team di sviluppo per la risoluzione di errori 55
B.1.12 MP012 - Efficienza della progettazione dei test 55

B.1.13 MP013 - Percentuale build superate 55
B.1.14 MP014 - Media commit giornaliero 55
B.1.15 MP015 - Percentuale requisiti obbligatori soddisfatti . 55
B.1.16 MP016 - Percentuale requisiti desiderabili soddisfatti . 56
B.2 Metriche per la qualità di prodotto 56
B.2.1 MPR001 - Errori ortografici 56
B.2.2 MPR002 - Indice di Gulpease 56
B.2.3 MPR003 - Errori inerenti alla correttezza dei documenti 56
B.2.4 MPR004 - Errori inerenti alle Norme di Progetto . . . 56
B.2.5 MPR005 - Completezza dell'implementazione funzionale 57
B.2.6 MPR006 - Correttezza rispetto alle attese 57
B.2.7 MPR007 - Totalità di failure 57
B.2.8 MPR008 - Tempo di risposta 57
B.2.9 MPR009 - Comprensibilità delle funzionalità offerte . . 58
B.2.10 MPR010 - Facilità apprendimento 58
B.2.11 MPR011 - Capacità di analisi failure 58
B.2.12 MPR012 - Impatto delle modifiche 58

Capitolo 1

Introduzione

1.1 Scopo del documento

Il seguente documento espone le norme che i membri del gruppo $ZeroSeven_G$ dovranno obbligatoriamente adottare nello svolgimento del progetto "$MegAlexa_G$". Ogni membro ha il dovere di leggere il documento e di seguire le regole in esso esposte per garantire un modo di lavorare comune, massimizzare l'efficenza e l'efficacia riducendo il numero di errori. Nel documento verranno definite le norme riguardanti:

- L'identificazione dei ruoli e delle relative mansioni da svolgere;
- L'interazione tra i membri del gruppo e con le entità esterne;
- Le modalità di lavoro durante le fasi di $progetto_G$;
- La stesura dei documenti;
- L'ambiente di sviluppo.

Le norme verranno prodotte incrementalmente, al progressivo maturare delle esigenze di progetto, trattando prima quelle più impellenti e ricorrenti e successivamente quelle che interverranno più avanti, sempre garantendo che ogni attività da svolgere sia stata precedentemente normata.

1.2 Scopo del prodotto

Lo scopo del progetto è quello di sviluppare un applicativo Web e Mobile in grado di creare delle routine personalizzate per gli utenti gestibili tramite $Alexa_G$ di $Amazon_G$. L'obiettivo è quello di creare $skill_G$ in grado di avviare $workflow_G$ creati dagli utenti fornendogli dei $connettori_G$.

1.3 Glossario

I termini tecnici, gli acronimi e le abbreviazioni che necessitano di un chiarimento o di una spiegazione verranno riportati in modo chiaro e conciso nel documento *Glossario v2.0.0*, al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti. Le occorrenze di vocaboli presenti nel Glossario è marcata da una "G" maiuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- *Capitolato*$_G$ C4 [1];
- ISO/IEC 12207 [2];
- ISO 8601 [3].

1.4.2 Informativi

- Software Engineering - Ian Sommerville - 9th Edition (2010);
- Slide del corso "Ingegneria del Software" - Amministrazione di progetto[4];
- Sito di *GitHub*$_G$ [5];
- https://guides.codepath.com/android/Organizing-your-Source-Files per la suddivisione dei package;
- Sito ufficiale di Mocha[6].

[1] https://www.math.unipd.it/~tullio/IS-1/2018/Progetto/C4.pdf
[2] https://www.iso.org/standard/43447.html
[3] https://www.iso.org/iso-8601-date-and-time-format.html
[4] https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/FC1.pdf
[5] https://github.com/
[6] https://mochajs.org/

Capitolo 2

Processi primari

2.1 Fornitura

Questo $processo_G$ ha lo scopo di trattare le norme e i termini che i membri del gruppo $ZeroSeven_G$ sono tenuti a rispettare per diventare fornitori della proponente Zero12 s.r.l e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.
L'aspettativa del gruppo è instaurare e mantenere un rapporto di collaborazione costante con Zero12 s.r.l. per poter soddisfare al meglio i bisogni della Proponente e rispettare i vincoli richiesti.

2.1.1 Studio di fattibilità

Dopo la presentazione dei $capitolati_G$, è compito del *Responsabile* convocare le riunioni necessarie per consentire al gruppo di esprimere idee e opinioni riguardo ai diversi capitolati. Le informazioni raccolte verranno utilizzate dagli *Analisti* per redigere lo *Studio di Fattibilità*. L'analisi si basa sui seguenti punti:

- **Descrizione generale**;
- **Finalità del progetto**;
- **Tecnologie utilizzate**;
- **Conclusione**.

2.1.2 Rapporti di fornitura con la Proponente

Durante l'intero progetto si intende instaurare un costante rapporto collaborativo con la Proponente, nella persona del Sig. Stefano Dindo, con il fine di:

- determinare bisogni in modo più accurato possibile;
- scegliere in modo collaborativo i requisiti del prodotto;
- accordare insieme vincoli di qualità del prodotto;
- stimare tempi e costi.

A seguito della consegna del prodotto, salvo ulteriori accordi, non seguirà l'attività di manutenzione.

2.1.3 Documentazione fornita

Di seguito viene indicata la documentazione che verrà fornita alla *Proponente*$_G$ e al Committente con lo scopo di rendere trasparenti le scelte di:

- **pianificazione**: *Piano di Progetto*;
- **processi**: *Norme di Progetto*;
- **verifica e validazione**: *Piano di Qualifica*;
- **obiettivi di qualità**: *Piano di Qualifica*;
- **requisiti**: *Analisi dei Requisiti*.

2.2 Sviluppo

Il *processo*$_G$ di sviluppo comprende tutte le attività e i compiti svolti dal gruppo per produrre il software richiesto dal *proponente*$_G$.
Dallo sviluppo del *prodotto*$_G$ ci si aspetta che questo soddisfi i test di verifica e di validazioni, gli obiettivi imposti e i bisogni della Proponente. Il processo di sviluppo si svolge secondo lo standard *ISO/IEC 12207*.

2.2.1 Analisi dei requisiti

Gli *Analisti* si occupano di analizzare nel dettaglio i *requisiti*$_G$ che il prodotto finale dovrà soddisfare, il risultato viene poi documentato nell' *Analisi dei Requisiti*. Vengono utilizzati i casi d'uso per l'analisi e la ricerca dei requisiti. Il documento redatto dovrà rispettare le specifiche di seguito riportate.

2.2.1.1 Fonti dei requisiti

Compito di analisti è quello di redigere una lista di *requisiti*$_G$. Tali requisiti possono essere ricavati da varie fonti, le quali sono:

- **Capitolato**: i requisiti emersi dall'analisi del documento fornito dalla *Proponente*$_G$;
- **Verbali Esterni**: i requisiti emersi durante incontri con la Proponente;
- **Interni**: i requisiti emersi tramiti analisi e discussione interna del gruppo *ZeroSeven*$_G$;
- **Casi d'uso**: i requisiti emersi dall'analisi di uno o più casi d'uso.

2.2.1.2 Classificazione dei requisiti

Ogni *requisito*$_G$ deve avere la seguente nomenclatura:

$$\mathbf{R\{A\}\{B\}\{XX\}.\{YY\}}$$

dove:

- **A:** corrisponde a uno dei seguenti requisiti:
  - F: funzionale;
  - Q: di qualità;
  - P: di prestazione;
  - V: di vincolo.
- **B:** corrisponde a uno dei seguenti requisiti:
  - O: obbligatorio;
  - F: facoltativo;
  - D: desiderabile.
- **XX:** numero che identifica i *requisiti*$_G$;
- **YY:** numero progressivo che identifica i sottocasi, esso può, a sua volta, includere altri sottocasi.

2.2.1.3 Casi d'uso

Ogni caso d'uso deve presentare i seguenti campi:

- Codice identificativo;
- Titolo
- Attori primari;
- Attori secondari (se presenti);
- Descrizione;
- Precondizione;
- Postcondizione;
- Flusso principale degli eventi.

Se un caso d'uso risulta essere particolarmente complesso allora viene corredato da un diagramma dei casi d'uso in linguaggio UML.

2.2.1.4 Codice identificativo CU

Ogni caso d'uso deve avere la seguente nomenclatura:

$$\mathbf{UC}\{\mathbf{XX}\}.\{\mathbf{YY}\}$$

dove:

- **UC:** Use Case;
- **XX:** numero che identifica i casi d'uso;
- **YY:** numero progressivo che identifica i sottocasi, esso può, a sua volta, includere altri sottocasi.

2.2.2 Progettazione

2.2.2.1 Scopo

L'attività di Progettazione precede obbligatoriamente la produzione del software e consiste nel descrivere e fornire una soluzione al problema che sia soddisfacente per tutti gli *stakeholders*$_G$. Essa serve a garantire che il prodotto sviluppato sia adeguato rispetto ai bisogni emersi nell'attività di Analisi. La progettazione permette di:

- costruire l'architettura logica del prodotto;
- facilitare la codifica da parte dei *Programmatori*, riducendo la complessità del problema originale, permettendo di organizzare i compiti e possibilmente facilitando la parallelizzazione;
- perseguire la correttezza per costruzione.

Come da *Piano di Progetto v3.0.0* la Progettazione si svolgerà nell'arco di due periodi distinti. Nel primo di questi due periodi viene identificata una prima decomposizione logica dell'architettura, dopo aver opportunamente studiato i vari design patterns. Nel secondo la progettazione avviene in modo atomico e dettagliato, in modo che i *Programmatori* possano implementare l'intero sistema nel modo più parallelo possibile. L'architettura dovrà soddisfare le seguenti qualità:

- **Sufficienza**: deve soddisfare i requisiti descritti nell'*Analisi dei Requisiti v2.0.0*;
- **Comprensibilità**: può essere compresa facilmente da tutti gli *stakeholder*$_G$;
- **Modularità**: deve essere suddivisa in parti chiare e ben distinte, ognuna con una singola responsabilità, perseguendo information hiding;
- **Robustezza**: è in grado di sopportare ingressi di tipi diversi, sia da parte dell'utente che dall'ambiente;
- **Flessibilità**: deve permettere modifiche a costi contenuti in caso di variazione dei requisiti;
- **Riusabilità**: deve permettere il riuso delle sue parti in altre applicazioni;
- **Efficienza**: deve utilizzare il minor numero di risorse possibili, in termini di tempo e spazio;
- **Affidabilità**: deve garantire di rispettare le sue specifiche di funzionamento nel tempo;
- **Disponibilità**: deve essere possibile effettuare manutenzione in tempi ridotti, possibilmente senza rendere indisponibile l'intero sistema;
- **Sicurezza**: deve essere esente da gravi malfunzionamenti e non deve essere vulnerabile a intrusioni;

- **Semplicità**: ogni parte contiene solo il necessario;
- **Incapsulazione**: le componente devono essere progettate in modo da nascondere dettagli implementativi;
- **Coesione**: le parti con un obiettivo comune devono stare insieme;
- **Basso accopiamento**: parti distinte devono dipendere poco o niente tra di loro.

2.2.2.2 Diagrammi

Al fine di rendere chiare e comprensibili le scelte progettuali effettuate e ridurre le possibili ambiguità, sarà necessario fare largo uso di vari tipi di diagrammi UML_G 2.0, quali:

- **Diagrammi delle classi**: descrivono i tipi di oggetti che fanno parte di un sistema mettendone in evidenza le relazioni statiche;
- **Diagrammi di sequenza**: descrivono la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento;
- **Diagrammi di attività**: descrivono il flusso di operazioni di un'attività attraverso la sua logica procedurale;
- **Diagrammi dei package**: descrivono un raggruppamento di un numero arbitrario di elementi (ad esempio classi) in un'unica unità ad alto livello.

2.2.3 Diagrammi delle classi

Lo scopo del diagrammi delle classi può essere riassunto in:

- Descrizione e definizione del tipo degli oggetti che fanno parte di un sistema;
- Descrizione e definizione delle relazioni statiche fra i tipi degli oggetti.

Le norme di seguito riportare valgono sia per *Kotlin*, sia per *NodeJS*.
Ogni classe viene rappresentato come un rettangolo diviso in tre sezioni:

- **Nome classe**: deve essere univoco, in notazione $CamelCase_G$, in inglese;

- **Attributi**: elenco campi dati della classe, secondo la notazione `visibility name : type`. Il modificatore di accesso deve essere inserito obbligatoriamente e deve essere uno dei seguenti:
  - +: visibilità pubblica;
  - -: visibilità privata;
  - #: visibilità protetta.
- **Operazioni**: elenco dei costruttori, modificatori e metodi di quel tipo, utilizzando la sintassi `visibility operationName(value: type) : type` .

Nel caso in cui una classe non dovesse contenere campi dati e/o metodi, apparirà nel diagramma comunque, seppur vuota, la sezione ad essi dedicata. Nel caso di classi astratte il nome della stessa sarà scritto in corsivo. Aderen-

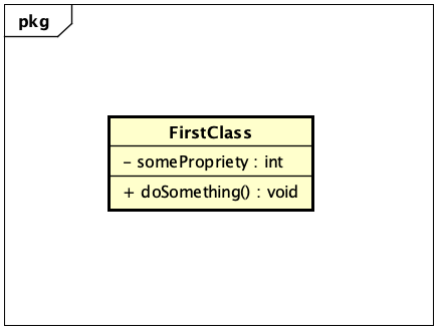


Figura 2.1: Esempio diagramma di una classe

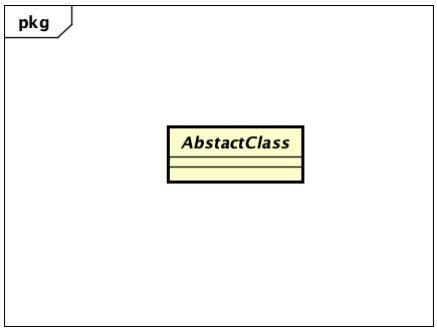


Figura 2.2: Esempio diagramma di una classe astratta

do alla sintassi UML2.x un'interfaccia viene rappresentata con cerchio vuoto

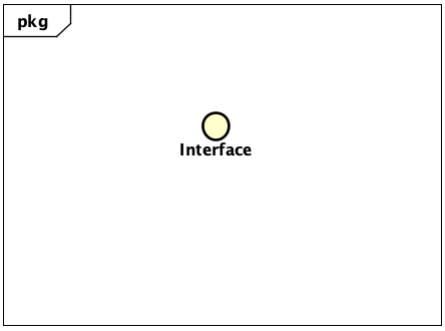


Figura 2.3: Esempio diagramma di un'interfaccia

con al di sotto il nome dell'interfaccia e l'elenco di operazioni. I diagrammi delle classi sono collegati fra loro da frecce che esplicitano le dipendenze. In particolare, verranno considerati i seguenti tipi di freccia:

- Freccia semplice: da classe A a classe B: indica che la classe A ha fra i propri campi dati una o più istanze della classe B;

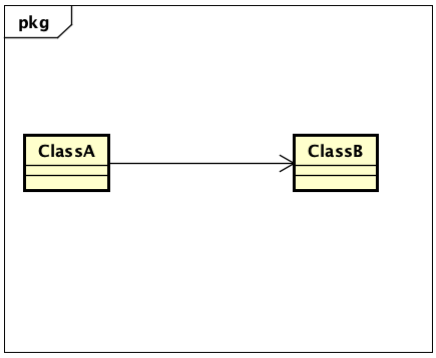


Figura 2.4: Relazione di dipendenza forte in un diagramma delle classi

- Freccia tratteggiata, da classe A a classe B: indica una dipendenza, significa che A dipende da B secondo una primitiva che viene indicata al di sopra della freccia;

- Freccia a diamante vuota, da classe A a classe B: indica un'aggregazione, una relazione non forte ovvero una relazione nella quale le classi che che ne prendono parte hanno entrambe significato anche prese in modo singolo;

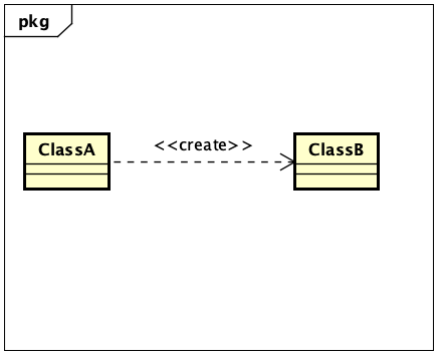


Figura 2.5: Relazione di dipendenza debole in un diagramma delle classi

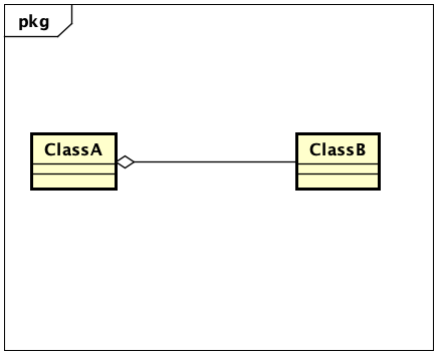


Figura 2.6: Relazione di aggregazione in un diagramma delle classi

- Freccia a diamante piena, da classe A a classe B: indica la composizione, una relazione forte dove un'istanza della classe contenuta ha senso solo se esiste un'istanza della classe che contiene;

- Freccia vuota, da classe A a classe B: indica ereditarietà ed è il grado massimo di dipendenza fra classi. Indica che un oggetto A è anche un oggetto di tipo B.

2.2.4 Diagrammi dei package

Ogni package viene rappresentato attraverso un rettangolo con un'etichetta per il nome, che può contenere eventuali sotto-package. Le dipendenze tra i vari package vengono segnalate attraverso una freccia tratteggiata. Tale freccia, disegnata dal package A al package B, indica una dipendenza di A nei confronti di B.

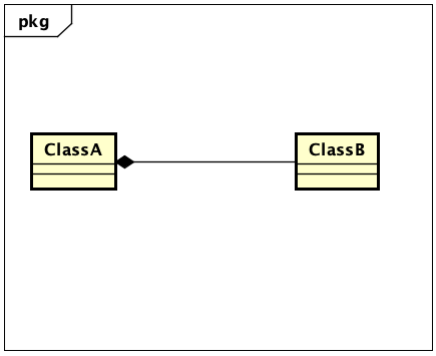


Figura 2.7: Relazione di composizione in un diagramma delle classi

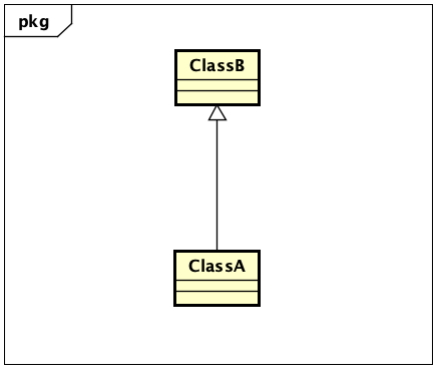


Figura 2.8: Relazione di ereditarietà in un diagramma delle classi

2.2.5 Diagrammi di sequenza

I diagrammi di sequenza descrivono la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento. Ogni diagramma di sequenza avrà un senso di lettura verticale dall'alto al basso, verso che indica lo scorrere del tempo. I partecipanti vengono rappresentati tramite un rettangolo, al cui interno si trova un nome per identificarli rispettando la sintassi `name : Class`. Al di sotto di ogni istanza si trova la linea della vita. In ogni linea della vita ci sarò una barra di attivazione che indica in quale momento un partecipante è attivo. Da una barra di attivazione partono delle frecce, che rappresentano un messaggio, verso la linea della vita di oggetti già istanziati o, in alternativa, verso un nuovo partecipante per crearlo. In particolare , vengono utilizzati i seguenti tipi di frecce:

- Freccia piena, per rappresentare un messaggio sincrono: corrisponde alla classica chiamata di un metodo, dove il chiamante rimane in atte-

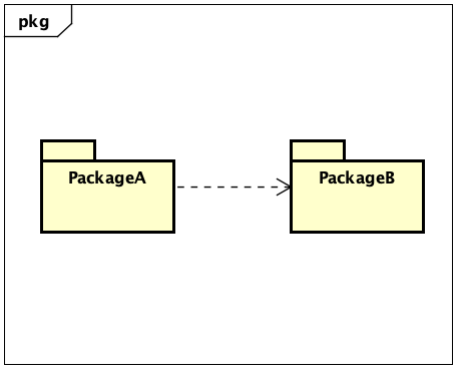


Figura 2.9: Esempio di diagramma di package con dipendenza

sa di una risposta. Sopra questa freccia si deve specificare il metodo chiamato rispettando la sintassi `method(param)`;

- Freccia, per indicare un messaggio asincrono: il chiamante non resta in attesa di risposta;
- Freccia tratteggiata, per indicare il messaggio di ritorno. Sopra questa freccia va indicato il tipo che viene ritornato;
- Freccia tratteggiata sormontata da « create »: indica la creazione di un nuovo oggetto e termina sempre in un partecipante (rettangolo);
- Freccia piena sormontata da « destroy »: indica la distruzione di un oggetto e termina sempre in una X, nella quale termina anche la linea della vita di un oggetto.

2.2.6 Diagrammi di attività

I diagrammi di attività descrivono la logica procedurale decomponendo un'attività in un insieme di azioni. Essi contengono i seguenti elementi collegati sempre da frecce, che vengono indicati come mostrato di seguito:

- Nodo iniziale: rappresentato da un pallino pieno. Indica il punto da cui inizia l'attività;
- Activity: raffigurata come un rettangolo che ne contiene una breve descrizione;

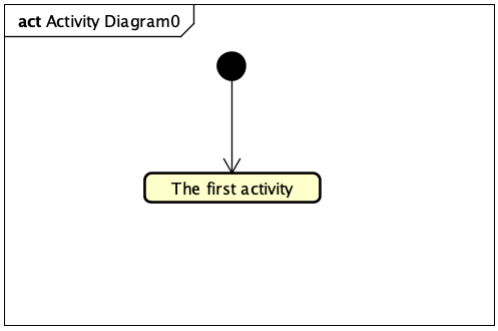


Figura 2.10: Rappresentazione di un nodo iniziale e attività in un diagramma

2.2.7 Codifica

Questa attività segue la Progettazione e consiste nell'implementazione effettiva di quanto precedentemente progettato. Per lo sviluppo dell'applicazione Android è stato deciso di utilizzare il software *Android Studio*$_G$.

2.2.7.1 Suddivisione dei package dell'applicazione

Con riferimento al sito elencato nella sezione 1.4.2, la suddivisione dei file sorgente per package avverrà nel modo seguente:

- **com.megalexa.activities:** contiene tutti file contenenti le *activities*$_G$ dell'applicazione.
- **com.megalexa.adapters:** contiene tutti i file contenti gli adapters

  view: conterrà tutti gli *adapters*$_G$ che alcuni widget particolari di android richiedono (come le RecyclerView e le ListView);

  connectors: contiene tutti i file a supporto dei blocchi contenuti nel model: nello specifico, un connettore si rende necessario se il blocco a cui fa riferimento necessita di controlli aggiuntivi in rete per risultare corretto.
- **com.megalexa.fragments:** contiene i file riguardanti i *fragments*$_G$ prodotti dalle activities;
- **com.megalexa.viewModel:** contiene tutti i file che compongono il *ViewModel*$_G$, che mette in comunicazione la parte grafica con quella logica;
- **com.megalexa.models:** contiene tutti i file riguardanti la logica dell'applicazione;

- **com.megalexa.util:** contiene tutte la classi di utilità definite nel corso dello sviluppo.

Sarà permesso definire ulteriori package all'interno di quelli sopra elencati, nel caso in cui ci sia bisogno di un maggiore dettaglio.

2.2.7.2 Suddivisione dei package della skill

- **MegAlexaSkill::lambda**: contiene il codice della skill MegAlexa che viene eseguito da AWS Lambda;
- **MegAlexaSkill::lambda::blocks**: contiene la parte di business della skill;
- **MegAlexaSkill::lambda::connection**: contiene le classi utili alla connessione e interfacciamento con il database;
- **MegAlexaSkill::node_modules**: contiene tutte le dipendenze utili alla skill.

2.2.7.3 Stile di codifica: regole generali

Vengono elencate le norme alle quali i *Programmatori* devono attenersi durante l'attività di programmazione e implementazione.

- **Parentesi:** devono essere aperte utilizzando il metodo *inline*, ovvero iniziano nella stessa riga del costrutto saltando uno spazio, inoltre blocchi di codice vanno racchiusi tra parentesi graffe;
- **Univocità dei nomi:** il nome delle classi, dei metodi e delle variabili deve essere il più esplicativo possibile al fine di garantire una maggior comprensione;
- **Nomi delle costanti:** la definizione delle costanti deve essere fatta utilizzando solo lettere maiuscole.
- **Nomi delle classi:** il nome delle classi deve iniziare con la lettera maiuscola;
- **Nomi dei metodi:** il nome dei metodi deve iniziare con la lettera minuscola, se sono composti da più parole allora le parole successive iniziano con la lettera maiuscola;
- **Lingua:** la lingua utilizzata è l'inglese, sia per i nomi delle classi, dei metodi e delle variabili, sia per i commenti del codice scritto.

2.2.7.4 Utilizzo delle librerie

- Assicurarsi che le librerie utilizzate non presentino funzioni deprecate;
- assicurarsi che le versioni delle librerie utilizzate siano stabili (versioni 1.0.0 e successive);
- minimizzare il più possibile l'inclusione di librerie aggiuntive.

2.2.7.5 Indentazione

Viene definito il modo corretto di indentare il codice: l'apertura delle parentesi deve essere posta sulla stessa linea della dichiarazione a cui fa riferimento e separata da uno spazio.

Corretto

```
private fun Foo() {
\\contenuto
}
```

Errato

```
private fun Foo()
{
\\contenuto
}
```

2.2.7.6 Linee vuote

Lasciare righe vuote dopo i blocchi e prima di un nuovo statement.

Corretto

```
\\contenuto
   function a():Any {
   return bar
   }

   function b(): Any {
   return foo
   }
```

Errato

```
\\contenuto
   function a():Any {
   return bar
   }
   function b(): Any {
   return foo
   }
```

2.2.7.7 Dichiarazione di variabili

2.2.7.7.1 Kotlin In Kotlin vi è una distinzione fra variabili in sola lettura (`val`) e in lettura e scrittura(`var`). É opportuno scegliere accuratamente il tipo di variabile da dichiarare, e tutte devono seguire la seguente sintassi.

```
val foo = Foo()
```

Se risulta necessario dichiarare una variabile come `lateinit`, specificare il tipo come segue

```
lateinit val foo: Type
```

Per dichiarare i campi dati di una classe, Kotlin fornisce una sintassi compatta che è opportuno seguire, l'inizializzazione dei campi dati deve avvenire mediante il costrutto `init`:

```
class myClass(private val param1: Type,private val param2: Type) {
   init {
   this.param1= param1
   this.param2= param2
   }
}
```

Le dichiarazioni di import vanno poste obbligatoriamente all'inizio del file:

Corretto

```
import java.util.*

class example() {
\\contenuto
}
```

Errato

```
class example {
\\contenuto
}

import java.util.*
```

2.2.7.7.2 Node.JS Le variabili globali e le dichiarazioni di `require` devono essere poste all'inizio del file:

Corretto

```
var axios =
    require("axios")

class Example {
\\contenuto
}

class SecondExample {
\\contenuto
}
```

Errato

```
class Example{
\\contenuto
}

var axios =
    require("axios")

class SecondExample {
\\contenuto
}
```

La dichiarazione di `export` viene posta obbligatoriamente alla fine del file:

Corretto

```
\\contenuto
class Example {
\\contenuto
}

modules.export = Example;
```

Errato

```
\\contenuto
modules.export = Example;
\\contenuto
class Example {
\\contenuto
}
```

2.2.7.8 Parentesizzazione

I blocchi di codice aventi una sola riga devono omettere le parentesi graffe:

Corretto

```
if(foo.isValid())
   return true
```

Errato

```
if(foo.isValid()) {
   return true
}
```

In presenza di blocchi annidati l'indentazione deve essere la seguente:

Corretto

```
class a() {
   \\contenuto
   function b() {
   \\contenuto
   }

   function c() {
   \\contenuto
   }
}
```

Errato

```
class a() {
\\contenuto
function b() {
\\contenuto
}
function c() {
\\contenuto
}
}
```

2.2.7.8.1 NodeJS La parentesizzazione di blocchi annidati deve seguire il seguente schema:

Corretto

```
class Example {
   constructor {
   \\contenuto
   }

   data() {
      fetchData() {
      \\contenuto
      }
   }
}
```

Errato

```
class Example {
constructor {
\\contenuto
}

data() {
   fetchData() {
   \\contenuto
   }
}
}
```

2.2.7.9 Commenti

Sfruttare il comando fornito da *JavaDoc*$_G$ per fornire una descrzione accurata delle funzioni che vengono implementate.

Corretto

```
/** fun()
* description
* description
*/
fun a():Unit {
\\contenuto
}
```

Errato

```
\\description
\\description
\\description
\\description
fun a(): Unit {
\\contenuto
}
```

JavaDoc inoltre mette a disposizione dei comandi aggiuntivi che permettono di identificare automaticamente i parametri e il tipo di ritorno delle funzioni:

- **@param:** per fornire la descrizione di un parametro;
- **@return:** per fornire un dettaglio maggiore sul risultato della funzione;

- **@exception \@throws:** descrive le eccezioni lanciate dai metodi;
- **@link:** per collegare un package o una classe a cui si fa riferimento.

2.2.7.10 Codifica di XML

XML_G viene utilizzato da kotlin per definire la disposizione degli oggetti grafici sullo schermo, vengono quindi definite le norme sull'indentazione del codice e l'annidamento dei tag.

2.2.7.10.1 Apertura e chiusura dei tag Nel caso in cui un tag non abbia figli, esso dovrà essere chiuso come segue:

```
<exampletag />
```

In presenza di molteplici attributi lo stile di scrittura deve essere il seguente:

```
<exampleTag
      attribute1= "1"
      attribute2= "2"
      attribute3= "3"
      attribute4= "4"
      attribute5= "5"
/>
```

La presenza di tag annidati deve essere indentata nel seguente modo:

```
<parentTag
      attribute1= "1"
      attribute2= "2"
      attribute3= "3"
      attribute4= "4"
      attribute5= "5"
>
      <childTag1
            attribute1= "1"
            attribute2= "2"
      />
```

```
    <childTag2
        attribute3= "3"
        attribute4= "4"
        attribute5= "5"
    />
</parentTag>
```

2.2.7.10.2 Dichiarazione di id e stringhe univoche

L'*XML schema*$_G$ adottato da Kotlin per definire i layout in XML permette di definire id univoci nel seguente modo:

```
<TextView
    android:id="@+id/idView"
/> s
```

e di definire le stringhe collegandole in un file separato chiamato `strings.xml`, in modo da permettere una facile conversione a una seconda lingua:

```
<TextView
    android:text="@string/todo"
/>
```

è quindi obbligatorio definire id univoci e inserire componenti testuali come appena descritto.

2.2.7.11 Intestazione

L'intestazione di ogni file deve essere scritta nella seguente maniera:

```
/*
* File: nome del file
* Version: versione del file
* Date: data di creazione del file
* Author: nome di crea e successivamente di chi modifica il file
*
* License: tipo di licenza
*
* History: registro delle modifiche
* Author || Date || Description
```

*
*\

2.2.7.12 Versionamento

Viene inserito all'interno dell'intestazione, descritta precedentemente, utilizzando la seguente nomenclatura:

X.Y.Z

dove:

- **X:** rappresenta l'indice primario, l'avanzamento di questo corrisponde una maggior stabilità del file, inoltre comporta l'azzeramento dell'indice Y;
- **Y:** rappresenta l'indice di verifica, il suo incremento corrisponde a una verifica del file;
- **Z:**: rappresenta l'indice di modifica, il suo incremento corrisponde a una modifica del file.

La versione *1.0.0* rappresenta un file completo e stabile in grado di essere testato, questo significa che le funzionalità obbligatorie sono state implementate e pertanto sono disponibili per essere testate.

Capitolo 3

Processi di Supporto

3.1 Gestione della configurazione

3.1.1 Versionamento

Per un progetto di tali dimensioni risulta necessario mantenere uno storico completo di tutte le modifiche apportate a qualunque file o documento non generato automaticamente da IDE o simili. Inoltre è necessario uno strumento che permetta il lavoro collaborativo. Per questo scopo si è scelto l'utilizzo del sistema di versionamento Git_G, attraverso il servizio $GitHub_G$.

3.1.1.1 Comandi basilari

Si riportano di seguito i comandi fondamentali del software Git per fornire un punto di appoggio a tutti i membri del gruppo. Tutti i comandi, tranne quello per la creazione della copia locale, sono da eseguire all'interno della cartella contenente il repository.

3.1.1.2 Creazione della copia locale

La creazione della copia locale viene effettuata attraverso il comando:
`git clone https://github.com/nomeaccount/nomerepostory`.

3.1.1.3 Comandi principali

- `git status`: mostra lo stato del repository locale con i file modificati, i file in area di staging, i file tracciati e non tracciati;
- `git checkout`: permette di cambiare branch attivo nel repository locale;

- `git add nomeFile`: permette di aggiungere un file all'area di staging;
- `git commit -m "Descrizione commit"`: fa il commit dei file presenti in area di staging nel repository locale;
- `git push`: aggiorna sovrascrivendo il repository remoto con quello locale;
- `git pull`: aggiorna sovrascrivendo il repository locale con quello remoto.

3.1.1.4 Git Flow

Per facilitare la collaborazione si sceglie usare il git flow[1] *workflow*$_G$. L'Amministratore si assicurerà che tutte le macchine siano configurate correttamente per il suo utilizzo.

3.1.1.5 Gestione dei rilasci

L'amministratore si occuperà di creare i rami di feature necessari per una proficua collaborazione tra i membri nel repository. Ogni documento (o parte di esso) e ogni funzionalità software verrà implementata in un ramo di feature. Quando la funzionalità diventa stabile e corretta verrà rilasciata in un ramo di integrazione chiamato "develop". Prima di ogni revisione verrà effettuato un rilascio nel ramo master a partire dall'ultimo commit. Tale rilascio costituirà *baseline*$_G$ e sarà punto di partenza verso la prossima *milestone*$_G$. Tale attività verrà effettuata nei tempi descritti nel *Piano di Progetto v2.0.0*.

3.2 Documentazione

Il *processo*$_G$ di Documentazione stabilisce una serie di norme e convenzioni per la stesura dei documenti così da renderli formali, validi e coerenti. Un documento viene definito formale quando viene approvato dal *Responsabile*. Ogni documento formale verrà classificato come:

- **Interno:** documenti ad uso esclusivo del gruppo *ZeroSeven*$_G$, quali *Norme di progetto* e *Studio di fattibilità*;
- **Esterno:** documenti consultabili anche da attori esterni al gruppo ZeroSeven.

[1] https://it.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

3.2.1 Nomenclatura

Tutti i documenti formali, esclusi i verbali, seguiranno questo sistema di nomenclatura:

- **Nome del documento:** Il nome del documento deve essere scritto in *Upper Camel Case*$_G$. NormeDiProgetto per il documento corrente;
- **Versione:** La documentazione prodotta deve essere corredata del numero di versione secondo la seguente codifica:

  v.X.Y.Z

  dove:

  - **X:** indica il numero crescente di uscite formali. Sarà compito del Responsabile azzerare gli indici Y e Z ad ogni rilascio;
  - **Y:** indica lo stato del documento secondo la seguente numerazione:
    0. per la documentazione in fase di sviluppo;
    1. per la documentazione in fase di *verifica*$_G$;
    2. per la documentazione in fase di approvazione;
    3. per la documentazione approvata e formale.
  - **Z:** indica il numero crescente di modifiche apportate al documento. Ogni modifica deve essere riscontrabile con il diario delle modifiche. Deve essere azzerato quando il responsabile approva il documento.

I documenti verranno citati secondo il formato NomeDocumento v.X.Y.Z mentre i file saranno rinominati `NomeDocumento_v.X.Y.Z.pdf`
I verbali sia interni che esterni seguiranno la nomenclatura VerbaleUsoData.

3.2.1.1 Ciclo di vita della documentazione

Ogni documento formale deve passare gli stadi di "Sviluppo", "Verifica" e "Approvato".

- **Sviluppo:** inizia con la creazione del documento e termina con la conclusione della stesura di tutte le sue parti. In questa fase i Redattori aggiungono le parti assegnate tramite *ticket*$_G$;
- **Verifica:** il documento entra nella fase di verifica dopo l'assegnazione di un ticket a un *Verificatore* da parte del *Responsabile*. I *Verificatori* effettueranno le procedure di controllo dello stesso. Al termine del

controllo in caso positivo il documento entra automaticamente in fase di "Approvazione", altrimenti i loro riscontri vengono consegnati al *Responsabile*, che provvederà ad assegnare nuovamente il documento ad un Redattore attraverso una nuova fase di Sviluppo;

- **Approvazione:** l'approvazione di un documento coincide con il superamento positivo della verifica. Sarà onere del Responsabile decidere, dopo un attenta lettura, se approvare il documento per il rilascio esterno o se è necessario modificare il documento;
- **Approvato:** il documento è pronto per il rilascio esterno.

3.2.2 Struttura dei documenti

Al fine di uniformare la struttura grafica e di permettere ai membri del gruppo di concentrarsi solo sulla stesura del contenuto è stato creato un template LaTeX. Ogni documento sarà composto da:

- Frontespizio;
- Diario delle modifiche;
- Indice;
- Elenco delle immagini e tabelle (se presenti);
- Contenuto delle pagine interne.

3.2.2.1 Frontespizio

La prima pagina di tutti i documenti dovrà essere così composta:

- Titolo del *progetto*$_G$;
- Titolo del documento;
- Logo e nome del gruppo;
- Descrizione in forma tabellare contenente informazioni importanti quali:
  - Versione del documento;
  - Data di Redazione;
  - Redattore;

- *Verifica*$_G$;
- Approvazione;
- Uso;
- Distribuzione;
- Email di contatto.

3.2.2.2 Registro delle modifiche

Successivo al frontespizio deve essere sempre presente un registro riassuntivo delle modifiche del documento in forma tabellare contenente:

- Versione dopo la modifica;
- Data della pubblicazione della modifica;
- Breve descrizione della modifica effettuata;
- Autore della modifica;
- Ruolo.

3.2.2.3 Indice

Tutti i documenti, eccezion fatta per i verbali, devono contenere un indice il quale permette una visione macroscopica del contenuto del documento, permettendo una lettura ipertestuale e non necessariamente sequenziale.

3.2.2.4 Elenco delle immagini e tabelle

Dopo il diario delle modifiche di ogni documento dovrà essere presente un *Elenco delle figure* ed un *Elenco delle tabelle*.

3.2.2.5 Struttura delle pagine interne

Le pagine interne dei documenti rispettano i canoni previsti dal template LaTeXe oltre al contenuto interno sono composte da:

- **intestazione:**
  - logo del gruppo posto a sinistra;
  - nome del capitolo corrente a destra.

- **Piè di pagina:**
  - titolo del documento completo di versione, posto a sinistra;
  - numero di pagina posto a destra.

3.2.3 Norme tipografiche

Questa sezione racchiude le convenzioni riguardanti tipografia, ortografia e uno stile uniforme e disciplinato per tutti i documenti.

3.2.3.1 Punteggiatura

Ogni segno della punteggiatura va sempre unito all'ultima lettera della parola che lo precede e separato con uno spazio dalla lettera iniziale della parola che lo segue. Le lettere maiuscole vanno poste solo dopo il punto, il punto di domanda, il punto esclamativo e all'inizio di ogni elemento di un elenco puntato, oltre che dove previsto dalla lingua italiana. È inoltre utilizzata l'iniziale maiuscola nel nome del team, del *progetto*$_G$, dei documenti, dei ruoli di progetto e delle fasi di lavoro.

3.2.3.2 Formati

- Date: le date presenti nei documenti devono seguire la notazione definita dallo standard ISO 8601:2004 YYYY-MM-DD dove:
  - YYYY: rappresenta l'anno scritto utilizzando quattro cifre;
  - MM: rappresenta il mese scritto utilizzando due cifre;
  - DD: rappresenta il giorno scritto utilizzando due cifre.
- Monospace: sarà utilizzato il carattere monospace per formattare il testo contenente parti di codice, comandi, nomi di classi;
- Percorsi: per gli indirizzi email e web deve essere utilizzato il comando LaTeX, mentre per gli indirizzi relativi va usato il formato monospace;
- Maiuscolo: l'utilizzo del carattere maiuscolo per l'intera parola è riservato esclusivamente agli acronimi.

3.2.3.3 Stile del testo

- **Corsivo:** il corsivo deve essere utilizzato esclusivamente nei seguenti casi:
  - Ruoli: Dovrà essere utilizzato il corsivo quando si parla di ruoli di progetto (es. *Analista*);
  - Documenti: Il nome di un documento andrà scritto in corsivo (es. *Norme di progetto*).
- **Grassetto:** il grassetto può essere utilizzato esclusivamente negli elenchi puntati per dare risalto al concetto sviluppato;
- **Sottolineato:** Non è previsto l'uso del testo sottolineato.

3.2.3.4 Norme redazionali

- **Elenchi:** Al termine di ogni punto di un elenco verrà utilizzato il carattere (;) eccetto per l'ultimo punto per il quale verrà utilizzato il carattere (.);
- **Riferimenti informativi:** Ogni riferimento a *prodotti*$_G$, guide, software, libri esterno al progetto dovrà essere indicato tramite un'annotazione a piè di pagina;
- **LaTeX:** Ogni riferimento a LaTeXverrà scritto utilizzando il comando `\LaTeX`;
- **Comandi LaTeX:** sono stati realizzati dei comandi personalizzati al fine di evitare errori di battitura e unificare tutti i documenti facilitando il lavoro del gruppo.
  - **\intestazioni**: inserisce un'intestazione personalizzata con il nome del documento e il logo *ZeroSeven*$_G$;
  - **\mailzeroseven**: inserisce l'indirizzo email del gruppo per un eventuale contatto;
  - **\progetto**: inserisce il nome del *progetto*$_G$;
  - **\logo**: inserisce il logo del gruppo ZeroSeven;
  - **\glossario**: indica un termine da inserire nel glossario(marcato da una G maiuscola a pedice).
- **Sigle:** Nonostante sarà preferito l'utilizzo delle parole per intero potranno essere utilizzate le seguenti sigle:

- ADR = Analisi dei Requisiti;
- NDP = Norme di Progetto;
- PDP = Piano di Progetto;
- PDQ = Piano di Qualifica;
- SDF = Studio di Fattibilità;
- RR = Revisione dei Requisiti;
- RQ = Revisione di Qualifica;
- RP = Revisione di Progetto;
- RA = Revisione di Accettazione.

3.2.3.5 Componenti grafiche

- **Tabelle:** Ogni tabella presente all'interno dei documenti deve essere accompagnata da una didascalia, in cui deve comparire un numero identificativo incrementale per la tracciatura della stessa all'interno del documento;
- **Immagini:** Le immagini presenti all'interno dei documenti devono essere nel formato Scalable Vector Graphics (SVG_G). In questo modo si garantisce una maggior qualità dell'immagine in caso di ridimensionamento. Per consentire l'inclusione delle immagini nei documenti, le immagini dovranno essere convertite nel formato PDF. Qualora non sia possibile salvare le immagine in formato vettoriale è preferito il formato Portable Network Graphics (PNG).

3.2.4 Strumenti a supporto della documentazione

Per la scrittura della documentazione in modo coerente e uniforme si deve usare il linguaggio di markup LaTeX.

3.2.4.1 TeXstudio

E' utilizzato per la scrittura e il controllo ortografico dei documenti.

3.2.4.2 Script automatici

Viene implementato uno script per permettere il calcolo automatico dell'*indice di Gulpease*$_G$, esso verrà eseguito ad ogni commit tramite *Travis CI*$_G$, nel caso in cui la build presenti degli errori grammaticali, essi dovranno essere immediatamente corretti.

Aggiungere, se necessario, i termini mancanti nel file `.dictionary`, contenente i termini che lo script deve ignorare.

3.3 Garanzia di qualità

3.3.1 Classificazione metriche e obiettivi

Questa *processo*$_G$ definisce norme e struttura delle metriche, obiettivi di qualità, metodologie e strumenti per perseguire qualità di processo e di prodotto.

3.3.1.1 Classificazioni degli obiettivi

Gli obiettivi di qualità inclusi nel *Piano di Qualifica v2.0.0* devono rispettare la seguente notazione:

Q[Tipo][ID] : [Nome].

- **Tipo**: stabilisce se l'obiettivo si riferisce a *prodotti*$_G$ o processi e può assumere i seguenti valori:
  - **P**: per indicare i processi;
  - **PR**: per indicare i prodotti.
- **ID**: identifica univocamente l'obiettivo attraverso un numero progressivo;
- **Descrizione**: breve descrizione dell'obiettivo di qualità.

3.3.1.2 Classificazione delle metriche

Risulta importante fissare delle metriche per permettere di monitorare costantemente la qualità di prodotto e processo.Tali metriche dovranno rispettare la seguente notazione: M[Tipo][ID] : [Nome].

- **Tipo**: stabilisce se la metrica si riferisce a prodotti o processi e può assumere i seguenti valori:
  - **P**: per indicare i processi;
  - **PR**: per indicare i prodotti.

- **ID**: identifica univocamente la metrica attraverso un numero progressivo;
- **Descrizione**: breve descrizione della metrica.

3.4 Verifica

La $verifica_G$ è un processo atto a evidenziare ed eliminare la possibile presenza di errori. Di seguito verranno descritti gli strumenti e le pratiche utilizzate per la verifica del codice e dei documenti durante la loro realizzazione. Durante questa prima fase di progetto la verifica si è focalizzata principalmente su documenti.

3.4.1 Verifica di processi

Ciascun $processo_G$ verrà costantemente monitorato in tutta la sua esecuzione e documentato alla fine di ogni periodo nell'appendice "Resoconto della verifica" nel *Piano di Qualifica v2.0.0*, secondo le metodologie ISO/IEC 15504.

3.4.2 Verifica dei prodotti

La verifica della qualità di prodotto si divide in attività e procedure differenti a seconda del $prodotto_G$ che si sta verificando. Per i prodotto software si procede con analisi statica e dinamica, mentre per i documenti si applica soltanto analisi statica.

3.4.2.1 Verifica dei documenti

3.4.2.1.1 Analisi statica Al fine di verificare i documenti verranno utilizzati $inspection_G$ e $walkthrough_G$:

- **Walkthrough:** questa metodologia prevede una lettura profonda e attenta del documento. Gli errori riscontrati verranno corretti e aggiunti all'appendice "Lista di controllo" delle *Norme di Progetto v2.0.0* per permettere di utilizzare inspection le volte successive;
- **Inspection:** al contrario della metodologia precedente, questa risulta essere molto più veloce perché attraverso una lista di controllo degli errori permette un'analisi più efficace delle criticità, omettendo le parti che non presentano problematiche.

3.4.2.1.2 Procedura di controllo dei documenti

1. Viene assegnato un *ticket*$_G$ a un *Verificatore*;
2. Vengono controllati gli errori comuni attraverso la lista di controllo;
3. Viene controllato il rispetto delle *Norme di Progetto*;
4. Viene effettuata una lettura profonda e corrette eventuali nuove anomalie;
5. Calcolo dell'*indice di Gulpease*$_G$;
6. Vengono aggiunte eventuali nuove anomalie alla lista di controllo;
7. Viene chiuso il ticket.

3.4.2.2 Verifica del prodotti software

3.4.2.2.1 Analisi dinamica É compito dei *Progettisti* definire i test nel periodo di progettazione, ad eccezione dei test d'unità(vedi sezione 3.4.2.2.3); vengono fornite indicazioni specifiche sulla corretta codifica dei test e la loro natura, suddividendo per tecnologie adottate qualora le differenze richiedano un dettaglio maggiore.
L'esecuzione dei test automatici tramite *Travis CI*$_G$ viene avviene unicamente per i test di unità, in quanto il costo computazionale di test di integrazione e di sistema risulta essere troppo elevato. Per i test riguardanti il back-end (*AWS Lambda*$_G$ e *API Gateway*$_G$), si ricorda che la copertura del codice deve essere totale.

3.4.2.2.2 Tecnologie adottate

3.4.2.2.2.1 JUnit Viene adottato per la codifica dei test su *Kotlin*$_G$.

3.4.2.2.2.2 Mocha e Chai Assert Vengono adottati per la codifica dei test su *Node.Js*$_G$: il primo è un *framework*$_G$ che fornisce funzioni specifiche e log sul risultato dei test, mentre il secondo ne amplia le funzionalità tramite l'inclusione di una *assertion-library*$_G$ che rende la sintassi molto più leggibile e facile da codificare.

3.4.2.2.3 Test di Unità I test di unità sono del codice prodotto, prodotto dai *Programmatori*, che esercitano un'unità del programma. Per unità si intende una funzionalità atomica che può essere verificata in modo isolato, in modo da assicurare che il risultato del test non sia influenzato da altre unità. Vengono sviluppate del programmatore che sviluppa le unità, per verificare l'assenza di alcuni errori, e documentare il comportamento dell'unità prodotta. Devono essere veloci da eseguire, indipendenti tra loro e nell'ordine di esecuzione. Inoltre dovrebbero essere eseguiti ad ogni modifica del codice sorgente.

3.4.2.2.4 Test di Integrazione I test di integrazione hanno il compito di verificare se sono rispettati i contratti di interfaccia tra più moduli o sottosistemi. I sottosistemi possono essere interni, quindi già verificati singolarmente dai test di unità, oppure esterni, come database, file system, etc. Risultano più lenti da configurare ed eseguire rispetto ai test di unità.

3.4.2.2.5 Test di Sistema I test di sistema verificano il comportamento del sistema nella sua interezza rispetto alle specifiche.

3.4.2.2.6 Test di accettazione I test di accettazione vengono eseguiti su tutto il sistema e verificano la conformità rispetto agli use cases e ai requisiti concordati con il committente.

3.4.2.2.7 Tracciamento dei test Ogni test è strutturato come segue:

- Codice identificativo;
- Descrizione;
- Stato.

Il codice identificativo segue la sintassi

T{Tipo}{Codice}

dove

- **Tipo**: può essere
  - U: se di unità;
  - I: se di integrazione;
  - S: se di sistema;

  - A: se di accettazione.

- **Codice**: può essere
  - **Numerico**: incrementale a partire da 1, associato a test di unità o di integrazione;
  - **Di Requisito**: riportando il numero del requisito che testa secondo il tracciamento dei requisiti requisiti presente in `Analisi dei Requisiti v2.0.0`.

Lo stato di un test può essere:

- implementato;
- non implementato;
- non eseguito;
- superato;
- non superato.

3.4.3 Procedure di verifica del software

3.4.3.1 Codifica e esecuzione

3.4.3.1.1 JUnit: la struttura di un progetto su *Android-studio*$_G$ prevede una suddivisione dei test a seconda se richiedano o meno l'ambiente di sviluppo Android per essere eseguiti: i primi sono posizionati nel percorso *app/src/androidTest*, mentre i secondi in *app/src/test*.
Alternativamente, è possibile definire un test che utilizzi dei *Mock*$_G$ degli oggetti creati dall'ambiente di sviluppo, tuttavia è altamente sconsigliato perché richiederebbe una quantità di lavoro eccessivo e letture successive del codice sarebbero poco intuitive.
I test che richiedono l'ambiente di sviluppo *Android*$_G$ sono:

- test che fanno riferimento a metodi che utilizzano le librerie Android;
- test che richiedono l'utilizzo della rete per essere effettuati;
- test che fanno riferimento a parti dell'interfaccia grafica.

I test che non rientrano nelle seguenti categorie (riguardanti, ad esempio, la logica dell'applicazione) possono essere scritti come normali test JUnit nella cartella sopracitata.

3.4.3.1.1.1 Esecuzione: L'esecuzione dei test avviene mediante l'invocazione del test con il comando *gradle* [*myTask*] fornito da *gradle*$_G$.

3.4.3.1.2 Mocha e Chai Assert: Mocha permette l'esecuzione seriale di test asincroni mediante l'invocazione di un singolo comando, e rende la loro codifica semplice ed efficace.
Segue una lista di direttive da perseguire nella codifica dei test con Mocha:

- Evitare l'utilizzo dell'operatore lambda (=>) in tutti i casi, poichè esso non permette al compilatore di Mocha di accedere al contesto delle funzioni a cui fa riferimento, se ciò non risulta necessario, una soluzione che prevede delle lambda porta a difficoltà riguardanti la manutenibilità del codice;
- Studiare con cura gli Hooks per la concatenazione dell'esecuzione dei test;
- Segnare con la dicitura *Pending* i test non ancora implementati, in modo che i *Progettisti* possano immediatamente identificare i test da codificare;
- Evitare di commentare i test da non eseguire, usare invece il comando *skip()*, che da dalla versione 3.0 è supportato anche per i test asincroni;
- Evitare l'utilizzo del comando *retries()* del tutto.

3.4.3.1.2.1 Esecuzione: L'esecuzione avviene mediante l'invocazione del comando *npm test* da terminale.

3.4.3.2 Test per AWS Lambda e Api Gateway

L'esecuzione dei test sul comportamento delle lambda integrate ad *API Gateway*$_G$ utilizzate dalla skill e dall'applicazione avviene mediante la console fornita da Amazon[2], che fornisce tutte le informazioni necessarie per verificare il corretto funzionamento delle stesse. La console, inoltre, viene utilizzata per testare la lambda comunicante con la skill.

3.4.3.3 Test sulla skill MegAlexa

La verifica del corretto funzionamento della skill consiste in un processo comprendente due passi:

[2] http://console.aws.amazon.com/

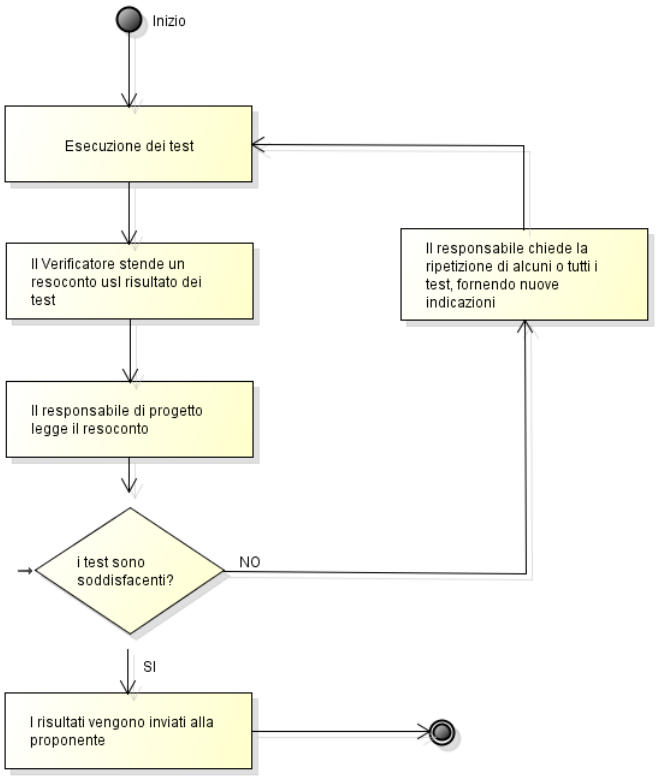
1. **Test della Lambda:** descritto nella sezione 3.4.3.2
2. **Test dell'interazione:** tramite la console per sviluppatori fornita da Amazon[3], è possibile verificare il comportamento della skill prima della sua distribuzione, e simulare un'interazione fra l'utente e Alexa.

3.5 Validazione

Il processo di validazione permette di accertare la conformità del prodotto rispetto a quanto è stato pianificato.

3.5.1 Procedura

Viene descritta la procedura per la validazione delle unità di codice che compongono il prodotto:



[3] https://developer.amazon.com/alexa/console/ask?

Capitolo 4

Processi organizzativi

4.1 Gestione di processo

4.1.1 Comunicazioni interne

Per le comunicazioni interne è stato adottato un gruppo *Telegram*$_G$, un'applicazione di messaggistica istantanea dove risulta possibile anche aggiungere Bot per automatizzare compiti ripetitivi, inviare file anche di grosse dimensioni, ricercare messaggi anche marcandoli con un hashtag.

4.1.2 Comunicazione esterne

Le comunicazioni esterne avvengono unicamente per mezzo scritto. Queste devono avvenire esclusivamente attraverso l'indirizzo mail del gruppo: *zerosevenswe@gmail.com*. Ogni mail ricevuta a questo indirizzo viene inoltrata automaticamente alle caselle mail di ogni componente del gruppo. L'accesso all'account di gruppo è possibile solo da parte del *Responsabile*.

4.1.3 Gestione delle riunioni

Le riunioni possono essere interne o esterne. All'inizio di ogni riunione il Responsabile nomina a turno un segretario che si occuperà di prendere nota di tutto ciò di cui viene discusso oltre ad avere l'onere di far rispettare l'ordine del giorno.

4.1.3.1 Verbali di riunione

Al termine di ogni riunione il segretario ha il compito di redigere il relativo verbale secondo il seguente schema:

- **Verbale [Tipo] [Data]**: nel frontespizio, con il tipo che indica se il verbale è interno o esterno e per data si intende la data nella quale si è svolta la riunione;
- **Informazioni sulla riunione:**
  - **Motivo della riunione**;
  - **Luogo e data**;
  - **Ora di inizio e fine**;
  - **Partecipanti**.
- **Ordine del giorno**;
- **Resoconto**.

4.1.4 Riunioni interne

La partecipazione alle riunioni interne è permessa solamente ai membri del gruppo *ZeroSeven*$_G$. Queste si possono svolgere:

- **In presenza**: incontri di persona;
- **Hangouts**: video chiamate di gruppo attraverso la piattaforma *Google Hangouts*$_G$.

4.1.5 Riunioni esterne

Le riunioni esterne vedono coinvolti oltre ai membri del gruppo ZeroSeven anche uno o più soggetti esterni. Queste si possono svolgere in:

- **Sede della Proponente**$_G$;
- **Torre Archimede**;
- **Google Hangouts**.

4.1.6 Identificazione delle decisioni

Le decisioni prese durante gli incontri devono essere identificate con un codice univoco per permettere di riferirsi ad esse se necessario, perciò nei verbali sarà presente il tracciamento delle decisioni nel quale verrà associato ad ogni decisione un identificativo che segue questa codifica:

`[NumeroDecisione]-[VER-DATA]`

dove:

- **NumeroDecisione:** è il numero della decisione presa in uno specifico incontro;
- **VER-DATA:** è il nome del verbale dell'incontro in cui è stata presa tale decisione.

4.1.7 Ruoli di progetto

I componenti del gruppo di progetto sono tenuti a ripartire tra loro in modo equo e omogeneo i differenti ruoli. A tal fine i ruoli verranno assegnati a rotazione dal *Responsabile*, il quale si assicurerà di non creare conflitti di interesse.

4.1.7.1 Responsabile

E' il responsabile ultimo dei risultati del progetto. Sue responsabilità sono l'approvazione dell'emissione di documenti, l'elaborazione e l'emanazione di piani e scadenze. Coordina il gruppo e le attività. E' suo compito gestire le relazioni con il controllo di qualità interno al *progetto*$_G$. Infine approva l'offerta e i suoi allegati.

4.1.7.2 Amministratore

L'amministratore è responsabile dell'efficienza e dell'operatività dell'ambiente di sviluppo. Si occupa della redazione e dell'attuazione dei piani e delle procedure di Gestione per la qualità. Redige le norme di *progetto*$_G$ per conto del responsabile e collabora alla stesura del piano di progetto. Gestisce l'archivio della documentazione di progetto, controlla versioni e configurazioni del *prodotto*$_G$.

4.1.7.3 Analista

Si occupa dell'attività di analisi, della redazione dello *Studio di Fattibilità* e dell'*Analisi dei requisiti*$_G$.

4.1.7.4 Progettista

Suo compito è l'attività di progettazione, inoltre redige la specifica tecnica, la *Definizione di prodotto*$_G$ e la parte programmatica del *Piano di qualifica*.

4.1.7.5 Programmatore

Si occupa delle attività di codifica che hanno lo scopo di realizzare il *prodotto*$_G$ e le componenti di ausilio necessarie all'esecuzione delle prove di verifica e di validazione.

4.1.7.6 Verificatore

Si occupa dell'attività di *verifica*$_G$. Redige la parte retrospettiva del piano di qualifica, che illustra l'esito e la completezza delle verifiche e delle prove effettuate secondo il piano. Si occupa inoltre dell'attività di controllo, in particolare verifica se i documenti e i prodotti dei processi rispettano le attese e sono conformi alle *Norme di Progetto*.

4.1.8 Ticketing

Per la gestione di attività e compiti viene utilizzato *Asana*$_G$ [1], che permette di gestire facilmente progetti di grandi dimensioni. L'assegnazione di compiti tramite questo strumento è affidata al *Responsabile*, seguendo le scadenze precedentemente stabilite nel *Piano di Progetto*.

4.2 Calcolo delle ore

Viene integrato ad Asana *Harvest*$_G$, che permette il time tracking delle ore di lavoro suddivise per ruolo, producendo dei grafici utili ai membri del gruppo per avere un quadro generale dell'andamento del lavoro.

4.2.1 Avvio del timer

Sebbene sia possibile inserire le ore manualmente, è consigliato avviare il timer al momento dell'inizio del lavoro, allo scopo di ottenere misurazioni più precise.
Per avviare il timer è sufficiente cliccare l'icona in alto a destra una volta aperta la task assegnata su Asana.

4.3 Ambiente di lavoro

Gli strumenti indicati di seguito possono subire variazione durante lo svolgersi dei lavori.

[1] Guida e Documentazione di Asana - https://asana.com/guide

4.3.1 Coordinamento

4.3.1.1 Versionamento

Per il versionamento si sceglie di utilizzare il software Git, attraverso la piattaforma *GitHub*$_G$. Questa scelta deriva dal fatto che questo sistema era già conosciuto quasi da tutti i membri del gruppo. Inoltre essendo Git un sistema di versionamento distribuito si riduce il rischio di perdita di dati.

4.3.1.2 Pianificazione e Ticketing

Per quanto riguarda la gestione delle attività si è scelto di utilizzare *Asana*$_G$, in quanto il servizio premium risulta essere gratuito per gli studenti. Asana viene usato in concomitanza con *Instagantt*$_G$, in quanto quest'ultimo permette una migliore gestione dei diagrammi.

4.3.2 Documentazione

4.3.2.1 LaTeX

Per quanto riguarda la stesura della documentazione, si è deciso di utilizzare LaTeX. Nonostante non sia subito di facile compressione, è stato scelto in quanto garantisce una migliore qualità tipografica dei documenti rispetto a un normale word processor.

4.3.2.2 Editor

L'editor consigliato per scrivere in LaTeXè *TeXstudio*$_G$ in quanto si presenta come un software molto completo, supporta la codifica UTF-8, implementa il completamento automatico dei comandi e supporta il dizionario italiano. Questo semplifica il lavoro di stesura dei documenti.

4.3.2.3 Script

Al fine di automatizzare la verifica dei documenti è stato creato il seguente script:

- *gulpease.py*: uno script per calcolare l'*indice di Gulpease*$_G$.

4.3.2.4 Diagrammi UML

Per modellare con *UML*$_G$ il gruppo ZeroSeven ha scelto di utilizzare Astah Professional, disponibile in versione gratuita grazie alla licenza accademica. Astah Professional supporta la sintassi UML 2.x e permette di modellare i

diagrammi dei casi d'uso, i diagrammi delle classi, degli oggetti, delle attività e di sequenza.

4.3.2.5 Creazione diagrammi di Gantt

Lo strumento scelto per la realizzazione dei diagrammi di $Gantt_G$ è $GanttProject_G$, in quanto è gratuito, open source e multi piattaforma.

4.3.3 Ambiente di sviluppo

4.3.3.1 Sistemi operativi

I membri del gruppo $ZeroSeven_G$ potranno lavorare indifferentemente su Windows, Linux o MacOS dal momento che tutti gli strumenti scelti ai fini del progetto sono disponibili per tutti e tre i sistemi operativi citati.

4.3.4 Ambiente di verifica

4.3.4.1 Documenti

Per quanto riguarda la verifica dei documenti, $TeXstudio_G$ non ha di default installato il dizionario per il controllo ortografico per la lingua italiana. E' stato quindi preparato un pacchetto da scompattare all'interno della cartella "dictionaries" dentro la directory di installazione di TeXstudio in modo da poter selezionare come lingua preferita l'italiano nella sezione "Controllo Linguistico" nelle impostazioni di TeXstudio. In caso di difficoltà sarà compito dell'*Amministratore* aiutare i membri del gruppo a effettuare correttamente questa configurazione.

4.4 Formazione del personale

Per quanto riguarda la formazione, i componenti del gruppo ZeroSeven devono provvedere autonomamente con lo studio delle tecnologie e degli strumenti necessari nel corso del progetto, prendendo come riferimento la seguente documentazione:

- Per l'utilizzo di LaTeX: https://www.latex-project.org;
- Per l'utilizzo del software Git: https://git-scm.com/docs;
- Per l'utilizzo del Gitflow Workflow: https://it.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

I membri del gruppo dovranno essere formati in modo da essere pronti per le attività pianificate nel *Piano di Progetto v2.0.0*. Qualora alcuni membri presentino lacune, essi dovranno informare preventivamente il *Responsabile* che provvederà a ridistribuire il carico di lavoro.

Appendice A

Lista di controllo

Durante l'applicazione del walkthrought ai documenti, sono state riportate le tipologie di errori più frequenti. La lista di controllo risultante è la seguente:

- **Norme stilistiche**:
  - **ordine righe nella tabella delle modifiche:** errore riscontrato nei primi documenti, la tabella deve iniziare con la modifica più recente e terminare con quella meno recente;
  - **nomi dei documenti in grassetto:** devono essere in corsivo;
- **Italiano**:
  - **evitare il futuro:** quando possibile, il presente aiuta a rendere più chiara la frase;
- **LaTeX**:
  - **Gestione cartelle in modo ricorsivo:** errore grave. Un file non può essere ripetuto più volte in cartelle diverse.

Appendice B

Metriche

B.1 Metriche per la qualità di processo

Verranno utilizzate le seguenti metriche per valutare l'efficienza e l'efficacia processi.

B.1.1 MP001 - Schedule Variance

Fornisce una misura di quanto lo stato del progetto è in ritardo o in anticipo rispetto alla pianificazione delle attività. Essa è il risultato della seguente formula:

$$SV = \text{BCWP} - \text{BCWS}$$

Dove:

- **BCWP (Budget Cost of Work Performed)**: è il valore (in giorni o euro) delle attività realizzate alla data corrente. Rappresenta il valore prodotto dal progetto ossia la somma di tutte le parti completate e di tutte le porzioni completate delle parti ancora da terminare;
- **BCWS (Budget Cost of Work Scheduled)**: è il costo pianificato (in giorni o Euro) per realizzare le attività di progetto alla data corrente.

B.1.2 MP002 - Cost Variance

Indica se si è speso più o meno di quanto è stato previsto. Il Cost Variance è dato dalla seguente formula:

$$CV = \text{BCWS} - \text{ACWP}$$

Dove:

- **BCWS (Budget Cost of Work Scheduled)**: è il costo pianificato (in giorni o Euro) delle attività svolte ad una certa data;
- **ACWP (Actual Cost of Work Performed)**: è il costo effettivamente sostenuto (in giorni o Euro) dalle attività svolte a tale data.

B.1.3 MP003 - SPICE capability level

Per ogni processo, lo standard 15504 definisce 6 livelli di maturità (da 0 a 5) determinati da un processo di *Process Assessment* che ha lo scopo di determinare l'effettiva qualità dei processi in uso, un processo raggiunge la sua massima efficacia quando raggiunge il livello 5 (raggiunge quindi l'ottimalità).

B.1.4 MP004 - SPICE process attributes

Per ogni attributo di processo, lo SPICE definisce 4 livelli (N-P-L-F) di ottimalità riferiti all'attributo stesso, il valore viene dedotto in base ai risultati delle metriche adottate per ciascun processo, un processo raggiunge il livello massimo quando tutte le metriche a lui riferite presentano risultati circoscritti al range di ottimalità definito.

B.1.5 MP005 - Occorrenza rischi non previsti

Contatore che viene incrementato all'occorrenza di un rischio non elencato nell'analisi dei rischi del *Piano di Progetto*, un alto valore indica l'eccessiva occorrenza dello stesso e la necessità di un'analisi al fine di mitigare il suo impatto nell'attività di progetto.
Viene resettato ad ogni inizio di una nuova fase del progetto.

B.1.6 MP006 - Indisponibilità dei servizi

Contatore che viene incrementato ogni qualvolta uno strumento esterno risulta inutilizzabile a causa di errori non gestibili dai membri del gruppo.
Viene resettato ad ogni inizio di una nuova fase del progetto.

B.1.7 MP007 - Complessità ciclomatica

Metrica software che indica la complessità di un programma tenendo in considerazione moduli, funzioni, metodi e classi. Nello specifico, essa è calcolata tramite il grafo di controllo di flusso del programma, dove i nodi sono gruppi indivisibili di istruzioni e gli archi connettono due nodi se il secondo gruppo di istruzioni può essere eseguito immediatamente dopo il primo, e il suo valore è determinato dal numero di cammini linearmente indipendenti all'interno del codice sorgente.
É quindi opportuno definire un valore di complessità ciclomatica preciso: valori alti sono indice di scarsa manutenibilità del codice mentre valori bassi potrebbero indicare scarsa efficienza dei metodi. Esso fornisce, inoltre, un indice del carico di lavoro richiesto per la fase di testing (un valore alto richiede più test per una copertura completa).
Il range di ottimalità stabilito varia da 0 a 10, come suggerito dall'ideatore della metrica Thomas J. McCabe.

B.1.8 MP08 - Numero di parametri per metodo

Definire un range relativo al numero di parametri permette di individuare possibili errori nella progettazione (nel caso in cui un metodo abbia un numero di parametri eccessivo).

B.1.9 MP009 - Numero di livelli di annidamento

Metrica per indicare il numero di chiamate annidate di procedure controllate all'interno dei metodi.
Un valore elevato è indice di un basso livello di astrazione del codice e una complessità eccessivamente elevata.

B.1.10 MP010 - Attributi per classe

Un numero elevato di attributi in una classe potrebbe essere indice di un errore di progettazione. Viene quindi definita una metrica che identifichi range accettabili e ottimali per questo parametro. Nel caso in cui una classe abbia un numero eccessivo di parametri, valutare la possibilità di suddividere la stessa in più classi, suddividendo quindi le funzioni ad essa assegnate.

B.1.11 MP011 - Tempo medio del team di sviluppo per la risoluzione di errori

Indica la quantità di tempo medio utilizzato per risolvere un bug dal team di sviluppo, utile per capire l'impatto medio dell'introduzione di un bug sui tempi di sviluppo. Si misura applicando la seguente formula:

$$\frac{\text{tempo totale speso per la correzione dei difetti}}{\text{numero totale di bug trovati}}$$

B.1.12 MP012 - Efficienza della progettazione dei test

Indica il tempo medio per la scrittura di un test, un numero troppo elevato potrebbe indicare che si stanno progettando test troppo complessi o che si sta cercando di testare parti del codice superflue. Si calcola secondo la seguente regola:

$$\frac{\text{numero totale di test progettati}}{\text{tempo per la loro stesura}}$$

B.1.13 MP013 - Percentuale build superate

Ogni build viene controllata tramite script automatici con *Travis CI*$_G$ e la metrica è il risultato del seguente calcolo:

$$\frac{\text{build superate con esito positivo}}{\text{build totali}}$$

B.1.14 MP014 - Media commit giornaliero

Tale metrica è risultato del seguente calcolo:

$$\frac{\text{commit totali settimanali}}{\text{numero giorni settimana}}$$

B.1.15 MP015 - Percentuale requisiti obbligatori soddisfatti

Tale metrica è risultato del seguente calcolo:

$$\frac{\text{requisiti obbligatori soddisfatti}}{\text{requisiti obbligatori totali}}$$

B.1.16 MP016 - Percentuale requisiti desiderabili soddisfatti

Tale metrica è risultato del seguente calcolo:

$$\frac{\text{requisiti desiderabili soddisfatti}}{\text{requisiti desiderabili totali}}$$

B.2 Metriche per la qualità di prodotto

Verranno utilizzate le seguenti metriche per valutare l'efficienza e l'efficacia dei prodotti.

B.2.1 MPR001 - Errori ortografici

Misura il numero di errori ortografici presenti nel documento. La misura viene fatta attraverso script automatici e la verifica da parte dei *Verificatori*.

B.2.2 MPR002 - Indice di Gulpease

L'*indice di Gulpease*$_G$ è un indice di leggibilità di un testo tarato sulla lingua italiana, per il suo calcolo vengono considerate due variabili linguistiche: la lunghezza delle parole e la lunghezza delle frasi rispetto al numero delle lettere.

$$G = 89 + \frac{300 \times N_F - 10 \times N_L}{N_P}$$

N_F indica il numero delle frasi, N_L indica il numero di lettere e N_P indica il numero di parole nel testo.

B.2.3 MPR003 - Errori inerenti alla correttezza dei documenti

Misura il numero di errori inerenti alla correttezza del documento. I *Verificatori* hanno il compito di valutare se un documento rispetta le scelte prese dal gruppo.

B.2.4 MPR004 - Errori inerenti alle Norme di Progetto

Misura il numero di errori inerenti alle *Norme di Progetto*. I *Verificatori* hanno il compito di valutare se un documento rispetta le *Norme di Progetto*.

B.2.5 MPR005 - Completezza dell'implementazione funzionale

Misura la quantità in percentuale di requisiti funzionali soddisfatti dalla corrente implementazione. Viene utilizzata la seguente formula:

$$CO = \frac{N_{RS}}{N_{RT}} \times 100$$

N_{RS} indica il numero di requisiti soddistatti, N_{RT} indica il numero totale di requisiti.

B.2.6 MPR006 - Correttezza rispetto alle attese

Misura la percentuale di risultati affini rispetto alle attese. Viene utilizzata la seguente formula:

$$CRA = (1 - \frac{N_{TD}}{N_{TE}}) \times 100$$

N_{TD} indica il numero di test che producono risultati discordi rispetto alle attese, N_{TE} indica il numero totale di test-case eseguiti.

B.2.7 MPR007 - Totalità di failure

Misura la percentuale di test conclusi con una failure. Viene utilizzata la seguente formula:

$$TF = \frac{N_{FR}}{N_{TE}} \times 100$$

N_{FR} indica il numero di failure rilevati durante l'attività di testing, N_{TE} indica il numero totale di test-case eseguiti.

B.2.8 MPR008 - Tempo di risposta

Misura la differenza media di tempo trascorsa dall'esecuzione di una funzionalità e la restituzione dell'eventuale risultato. Viene utilizzata la seguente formula:

$$TR = \frac{\sum_{i=1}^{n} T_i}{n}$$

T_i indica il tempo (in secondi) trascorso dalla richiesta di una funzionalità ed il completamento di questa con un eventuale restituzione del risultato.

B.2.9 MPR009 - Comprensibilità delle funzionalità offerte

Misura la quantità in percentuale di operazioni comprese dall'utente che non richiedono la consultazione del manuale. Viene utilizzata la seguente formula:

$$C = \frac{N_{FC}}{N_{FO}} \times 100$$

N_{FC} indica il numero di funzionalità comprese in modo immediato dall'utente, N_{FO} indica il numero di funzionalità totali offerte dal sistema.

B.2.10 MPR010 - Facilità apprendimento

Misura il tempo medio che occorre ad un utente per imparare ad usare in maniera corretta una certa funzionalità. Si misura tramite un indicatore numerico che indica i minuti impiegati da un utente per apprendere il funzionamento di una certa funzionalità;

B.2.11 MPR011 - Capacità di analisi failure

Misura la quantità in percentuale di failures incontrate di cui sono state tracciate le cause. Viene utilizzata la seguente formula:

$$CAF = \frac{N_{FI}}{N_{FR}} \times 100$$

N_{FI} indica il numero di failure delle quali sono state individuate le cause, N_{FR} indica il numero di failures rilevate.

B.2.12 MPR012 - Impatto delle modifiche

Misura la quantità in percentuale di modifiche introdotte per risolvere failures che hanno introdotto nuove failures nel prodotto. Viene utilizzata la seguente formula:

$$IM = \frac{N_{FRE}}{N_{FR}} \times 100$$

N_{FRF} indica il numero di failure risolte introducentro nuove failure, N_{FR} indica il numero di failures risolte.