INGEGNERIA DEL SOFTWARE
a.a. 2018/2019

Capitolato C4 - MegAlexa

Norme di Progetto

Componenti:

Sonia Menon
Alberto Miola
Andrea Pavin
Alessandro Pegoraro
Matteo Pellanda
Pardeep Singh
Luca Stocco

Destinatari:

Prof. Tullio Vardanega
Prof. Riccardo Cardin

Informazioni sul documento

Responsabile	Alberto Miola
Verifica	Sonia Menon, Alessandro Pegoraro
Redazione	Matteo Pellanda, Luca Stocco
	Andrea Pavin, Pardeep Singh
Uso	Interno
Stato	Approvato
Email	duckware.swe@gmail.com
Riferimento	Capitolato C4 - MegAlexa

Descrizione

Documento interno al Gruppo duckware, contiene linee guida per la gestione di tutti i processi instanziati dal gruppo

Versione 4.0.0 del
11 Aprile 2019

Indice

Registro delle modifiche

Elenco delle tabelle

Elenco delle figure

Registro delle modifiche

Ver.	Data	Autore	Ruolo	Descrizione
3.0.4	2019-05-07	Matteo Pellanda	Amministratore	Modifica in §3.7.14
3.0.3	2019-05-06	Matteo Pellanda	Amministratore	Correzione versione dei documenti citati in §1.4
3.0.2	2019-04-27	Pardeep Singh	Redattore	Modifiche alla §2.3 riguardo alle attività di progettazione
3.0.1	2019-04-26	Pardeep Singh	Redattore	Correzione delle versioni dei documenti riferiti ed altri piccoli errori
3.0.0	2019-04-11	Alberto Miola	Responsabile	Approvazione per rilascio del documento in RQ
2.1.0	2019-04-10	Sonia Menon	Verificatore	Superamento verifica
2.0.4	2019-04-07	Alessandro Pegoraro	Verificatore	Rimozione marcature glossario, lasciata solo alla prima occorrenza
2.0.3	2019-04-02	Pardeep Singh	Amministratore	Inserimento contenuti in §1
2.0.2	2019-03-26	Matteo Pellanda	Amministratore	Inserimento contenuti aggiuntivi in §3.5 e §3.6. Inserito periodo calcolo metriche.
2.0.1	2019-03-24	Alessandro Pegoraro	Verificatore	Correzione changelog secondo correzione RP
2.0.0	2019-03-06	Alberto Miola	Responsabile	Approvazione per rilascio del documento in RP
1.1.0	2019-03-06	Sonia Menon	Verificatore	Superamento verifica
1.0.11	2019-03-05	Pardeep Singh	Verificatore	Correzione errori minori
1.0.10	2019-03-05	Alberto Miola	Amministratore	Inserimento nuovo paragrafo su Node.js
1.0.9	2019-02-22	Matteo Pellanda	Verificatore	Correzione capitolo §3.5 e §3.7
1.0.8	2019-02-19	Alberto Miola	Amministratore	Inserimento nuovi paragrafi in §3.5 e §3.7

1.0.7	2019-02-17	Matteo Pellanda	Verificatore	Rimozione riferimenti su capitolo §1
1.0.6	2019-02-17	Matteo Pellanda	Verificatore	Correzione appendice §A
1.0.5	2019-02-16	Pardeep Singh	Verificatore	Correzione capitolo §1
1.0.4	2019-02-16	Pardeep Singh	Verificatore	Correzione errori minori
1.0.3	2019-02-15	Alberto Miola	Amministratore	Correzione errori di contenuto
1.0.2	2019-02-15	Matteo Pellanda	Amministratore	Correzione dei titoli secondo valutazione RR
1.0.1	2019-02-14	Alberto Miola	Amministratore	Correzione errori di sistassi e di contenuto
1.0.0	2018-12-01	Alberto Miola	Responsabile	Approvazione per rilascio del documento in RR
0.3.0	2018-12-01	Alberto Miola	Amministratore	Aggiornamento del documento
0.2.2	2018-11-30	Matteo Pellanda	Amministratore	Stesura §3.7, §3.5 e §3.6
0.2.1	2018-11-29	Luca Stocco	Amministratore	Correzione errori di sintassi
0.2.0	2018-11-29	Alberto Miola	Amministratore	Aggiornamento del documento
0.1.0	2018-11-28	Pardeep Singh	Verificatore	Superamento verifica
0.0.10	2018-11-28	Matteo Pellanda	Amministratore	Inserimento grafici in §A e §B
0.0.9	2018-11-27	Luca Stocco	Amministratore	Correzione errori §4
0.0.8	2018-11-26	Pardeep Singh	Verificatore	Correzione errori di sintassi §3
0.0.7	2018-11-25	Alessandro Pegoraro	Amministratore	Stesura §B
0.0.6	2018-11-24	Matteo Pellanda	Amministratore	Stesura §A
0.0.5	2018-11-24	Sonia Menon	Verificatore	Correzione errori di sintassi §2 e §4

0.0.4	2018-11-23	Luca Stocco	Amministratore	Stesura §4
0.0.3	2018-11-22	Andrea Pavin	Amministratore	Stesura §3
0.0.2	2018-11-21	Matteo Pellanda	Amministratore	Stesura §1 e §2
0.0.1	2018-11-21	Matteo Pellanda	Amministratore	Creazione scheletro del documento

Tabella 1: Registro delle modifiche

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è quello di fissare tutte le regole e le metodologie da applicare durante la realizzazione del progetto di modo che il gruppo possa lavorare seguendo certi standard ben definiti. In particolare, verranno definiti gli strumenti da utilizzare e le procedure da applicare durante le varie fasi dello sviluppo del *software*$_G$. In tal modo ci sarà una efficiente collaborazione tra i membri.

1.2 Natura del documento

Il presente documento non può essere considerato completo, in quanto sarà revisionato e incrementato nel suo contenuto ad ogni revisione di progettazione nelle rispettive sezioni durante i periodi di lavoro e sviluppo.

1.3 Scopo del prodotto

L'obiettivo del prodotto è la realizzazione di un'applicazione per smartphone, nello specifico per la piattaforma *Android*$_G$, che permetta la creazione di *workflow*$_G$ per l'assistente vocale *Amazon*$_G$ *Alexa*$_G$. Il *back-end*$_G$ sarà realizzato con *Java*$_G$ e *Node.js*$_G$ opportunamente integrati con le *API*$_G$ di *AWS*$_G$, per il *front-end*$_G$ verrà utilizzato *XML*$_G$ per stabilire i layout e Java per gestirne il comportamento. Si parlerà del front-end dell'assistente vocale riferendosi a *VUI*$_G$.

1.4 Glossario

Nel documento ci sono dei termini con un significato ambiguo a seconda del contesto nel quale sono stati utilizzati. Per ovviare a questo problema è presente il documento *Glossario v4.0.0* che conterrà una lista di termini con la specifica descrizione del suo significato. La prima occorrenza di un qualsiasi termine presente nel glossario verrà indicata in questo documento scritta in corsivo con una G a pedice che seguirà la parola in questione.

1.5 Riferimenti utili

1.5.1 Riferimenti informativi

- Sito del corso di Ingegneria del Software[1]
- ISO/IEC 15504[2]

[1] https://www.math.unipd.it/ tullio/IS-1/2018/
[2] http://www.colonese.it/SviluppoSw_Standard_ISO15504.html

2 Processi primari

2.1 Fornitura

Verranno ora trattate le norme che i membri di duckware devono rispettare per poter diventare fornitori nei confronti di *Zero12* e dei committenti, *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*.

2.1.1 Ricerca sulle tecnologie

Questa attività consiste nella ricerca d'informazioni su *framework*$_G$, librerie e tutte quelle tecnologie software ritenute utili in una o più fasi di sviluppo del progetto. Il periodo di ricerca è propedeutico a molte attività, in particolare quelle di *Normazione*, *Pianificazione della qualità* e la maggior parte delle attività nel *processo*$_G$ di sviluppo descritto nella sezione seguente. Viene inclusa anche l'auto-formazione sulle tecnologie scelte.

2.1.2 Normazione

L'amministratore apporta modifiche all'insieme di regole che ciascun membro del team *duckware* deve rispettare durante lo svolgimento dei compiti a lui assegnati. Nel periodo di realizzazione del progetto si riscontrano problemi o si acquisiscono nuove conoscenze che richiedono la modifica delle regole. Un esempio di cambiamento di regole potrebbe essere l'utilizzo di un nuovo strumento ritenuto più appropriato di un altro oppure potrebbero emergere nuovi argomenti per i quali stabilire nuove norme.

2.1.3 Studio di Fattibilità

Gli analisti conducono un approfondimento dei *requisiti*$_G$ per ogni capitolato presentato dai committenti con lo scopo di scegliere se accettare o meno il contratto. In particolare per ciascuna proposta vengono estratte le seguenti informazioni:

- Scopo del prodotto da realizzare;
- Lista delle tecnologie richieste dal capitolato per realizzare il progetto;
- Livello di gradimento verso l'obiettivo del capitolato.

2.1.4 Preparazione alla revisione

Questa attività consiste nell'approntamento del materiale da presentare in ingresso alla revisione ed eventualmente la preparazione del materiale per l'esposizione, quando prevista. La prima fase comprende anche la scrittura di una lettera di presentazione da parte del *responsabile*$_G$ di *duckware*.

2.2 Sviluppo

2.2.1 Analisi dei Requisiti

L'analisi dei requisiti ha lo scopo di individuare i requisiti richiesti dal capitolato preso in questione. Tali informazioni possono essere reperite da verbali, casi d'uso oppure dai capitolati d'appalto stessi. Il risultato finale di questa analisi porterà alla creazione da parte degli analisti del documento *Analisi dei requisiti v4.0.0* che fornirà informazioni riguardo:

- Descrivere l'obiettivo del lavoro del gruppo e fornire riferimenti precisi ai progettisti;
- Fornire una lista dettagliata di funzionalità e requisiti concordati col cliente;
- Fornire degli esempi di dialoghi tra l'utente e l'assistente vocale, riferiti alle funzionalità che necessitano di un'interazione tra i due;
- Dare ai verificatori riferimenti e casi d'uso per poter creare ed eseguire dei test mirati ed affidabili;
- Rendere più semplice il *processo*$_G$ di revisione del codice.

2.2.2 Classificazione dei Requisiti

Ogni requisito è identificato da un codice che verrà rappresentato secondo quanto segue:

R.x.y.z

- **R**: sta per Requisito;
- **x**: indica l'importanza di tale requisito che può essere 0 (opzionale), 1 (requisito non necessario ma può dare maggiore completezza e definizione) e 2 (obbligatorio in quanto necessario per il funzionamento basilare);
- **y**: assume valore F se requisito funzionale, Q se di qualità, V se di vincolo e P se presentazionale.
- **z**: indica un numero progressivo.

2.2.3 Classificazione dei casi d'uso

Gli analisti avranno anche cura di identificare i casi d'uso i quali verranno descritti secondo il seguente schema:

UC[Codice principale].[Codice secondario].[Codice terziario][Codice quaternario] – Identificativo

- **UC** e **Identificativo**: Ogni *caso d'uso*$_G$ è specificato da una serie di cifre spaziate da un punto. Questa serie di cifre identifica una gerarchia, dove il codice principale ha il grado più alto, mentre le altre hanno grado via via più basso. Un trattino separerà il tutto dal nome univoco del caso d'uso;

- **Attori**: gli attori principali e secondari dell'UC (use case);
- **Scopo** e **descrizione**: una descrizione riassuntiva del caso d'uso;
- **Scenario principale**: usando una lista numerata indica il flusso degli eventi specificando per ognuno a quale caso d'uso si riferisce;
- **Voice flow**: descrive l'eventuale interazione vocale tra l'utente e l'assistente Alexa;
- **Estensioni**: contiene una lista degli eventuali errori che il caso d'uso può generare;
- **Pre-condizione**: specifica le condizioni vere prima del verificarsi degli eventi del caso d'uso;
- **Post-condizione**: specifica le condizioni vere dopo il verificarsi degli eventi del caso d'uso.

2.3 Progettazione

2.3.1 Scopo

La progettazione ha il compito di disegnare, attraverso i Progettisti, una soluzione del problema che sia esaustiva per tutti gli *stakeholders*$_G$. Verrà definita la logica del prodotto identificando i componenti e cercando sempre di restare nei costi che sono stati prefissati. L'attività di progettazione viene fatta in due momenti: una prima parte ad alto livello viene fatta durante il periodo di Progettazione della base tecnologica, dove verranno studiati i design pattern che potrebbero essere utilizzati durante il periodo di Progettazione di dettaglio e codifica, durante il quale la progettazione diventa atomica per ogni componente del sistema.
Una volta ottenuta la suddivisione in componenti, è possibile per i programmatori sviluppare il codice per i singoli componenti eseguendo task focalizzate. In particolare, l'architettura definita dovrà rispettare le seguenti proprietà:

- **Sufficienza**: soddisfare i requisiti dell'*Analisi dei requisiti* ed adattarsi ai cambiamenti od alle evoluzioni che potenzialmente avverranno nel tempo;
- **Comprensibilità**: essere comprensibile e modulare;
- **Affidabilità**: avere la capacità di rispettare le specifiche nel tempo, anche nel caso di malfunzionamenti;
- **Riusabilità**: le varie parti dovranno poter essere usate anche in altre applicazioni;
- **Modularità**: deve essere suddivisa in parti chiare e ben distinte, così che possa essere facilmente manutenibile;
- **Robustezza**: deve essere capace di sopportare ingressi diversi, sia da parte di utenti che dall'ambiente;
- **Flessibilità**: deve poter permettere modifiche al variare o all'aggiunta di requisiti senza perdite di performance o senza doverla restaurare profondamente;

- **Disponibilità**: la manutenzione delle sue parti non dovrà inabilitare il funzionamento di tutto il sistema;
- **Efficienza**: deve essere pensata in modo da poter ridurre gli sprechi di tempo e spazio;
- **Semplicità**: ogni parte contiene solo il necessario e nulla di superfluo;
- **Sicurezza**: garantire l'integrità evitando intrusioni da parte di terzi non autorizzati e malfunzionamenti;
- **Incapsulazione**: le componenti dovranno essere progettate in modo che le informazioni interne siano nascoste;
- **Coesione**: in modo che le parti che hanno gli stessi obiettivi stanno insieme;
- **Basso accoppiamento**: le varie parti devono essere poco dipendenti l'una dalle altre, di modo da poter essere facilmente manutenibili.

2.3.2 Obiettivi

Essa serve a garantire che il prodotto possa soddisfare le proprietà specificate nell'attività di analisi ponendo i seguenti obiettivi:

- Rendere chiara e comprensibile ogni parte dell'*architettura*$_G$ ai differenti stakeholders;
- Garantire la qualità di prodotto sviluppato con lo scopo di raggiungere la *correttezza per costruzione*;
- Mantenere nascosti i dettagli implementativi secondo quanto espresso dal principio dell'*information hiding*;
- Mantenere un basso grado di dipendenza fra le varie parti del prodotto;
- Ottimizzare l'uso di risorse.
- Facilitare l'attività di codifica da parte dei singoli Programmatori attraverso una buona organizzazione e ripartizione dei compiti implementativi, riducendo la complessità del problema originale fino alle singole componenti.

2.3.3 Diagrammi

Le scelte adottate dai progettisti verranno adeguatamente descritte attraverso l'uso di diagrammi *UML 2.0*$_G$ per rendere più chiare e meno ambigue le decisioni prese. Verranno in particolare utilizzate le seguenti rappresentazioni:

- **Diagrammi di casi d'uso**: descrivono le funzioni del sistema;
- **Diagrammi delle classi**: descrivono gli oggetti e le dipendenze fra essi;
- **Diagrammi dei package**: descrivono la dipendenza tra i *package*$_G$ (i quali contengono le classi);

- **Diagrammi di sequenza**: descrivono la collaborazione nel tempo degli oggetti;
- **Diagrammi di attività**: descrivono la logica procedurale.

Per poter svolgere nel miglior modo possibile il *processo*$_G$ di progettazione, il team *duckware* ha deciso di utilizzare il software PlantUML[3]. Quest'ultimo consente di realizzare in maniera semplice e veloce diagrammi delle classi, di attività e di sequenza.

2.3.3.1 Diagrammi delle classi

Un diagramma di classe è un grafo orientato che descrive, dal punto di vista statico, l'architettura dell'applicazione mettendo in risalto le relazioni esistenti tra le classi e le interfacce coinvolte nel sistema. Il suo scopo può essere sintetizzato in:

- Descrizione delle responsabilità di un sistema;
- Base per diagrammi dei componenti e di rilascio;
- Analisi e progettazione della visione statica di un'applicazione.

Per creare questo tipo di diagrammi in modo semplice, deve essere creato un file per ogni classe, o gruppo di classi, che abbia estensione di tipo *.iuml*. Per stabilire le relazioni presenti tra le varie componenti interessate verrà creato un altro file con estensione *.puml* in modo da avere un diagramma completo. Nel caso in cui una classe non dovesse avere metodi e/o variabili dovrà comunque apparire all'interno del diagramma. I diagrammi delle classi sono collegati fra loro con frecce che evidenziano le dipendenze; in particolare, verranno usati i seguenti tipi di frecce:

- **Freccia semplice**, da A verso B. Indica che la classe A ha una o più istanze di B tra i suoi campi dati;

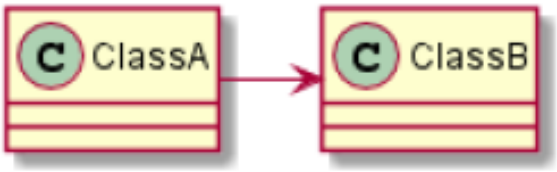

Figura 1: Immagine freccia semplice

- **Freccia tratteggiata**, da A verso B. Indica che la classe A ha una dipendenza da B secondo una primitiva che andrà specificata vicino alla freccia;

[3] http://plantuml.com/

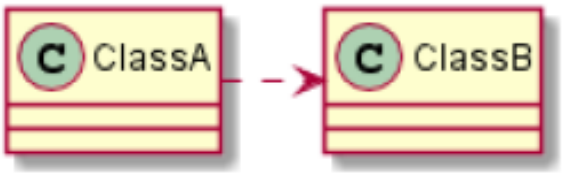

Figura 2: Immagine freccia tratteggiata

- **Freccia a rombo pieno**, da A verso B. Indica una composizione, una relazione nella quale le classi parte hanno significato solo se legate alla classe tutto;

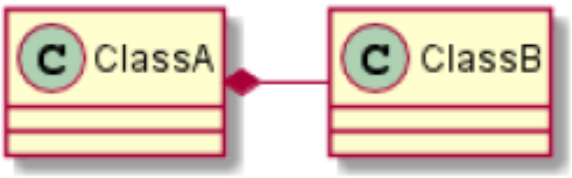

Figura 3: Immagine freccia a rombo pieno

- **Freccia vuota**, da A verso B. Indica il massimo grado di dipendenza fra le classi in quanto stabilisce un rapporto di ereditarietà. Ogni oggetto di A è anche un oggetto di B;

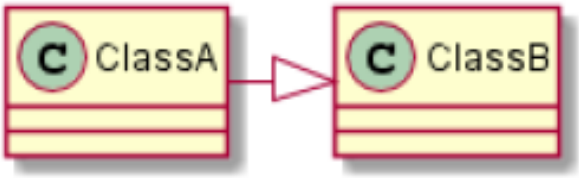

Figura 4: Immagine freccia vuota

2.3.3.2 Diagrammi dei package

Ogni package sarà rappresentato da un rettangolo con un'etichetta contenente il relativo nome. All'interno di quest'area ci saranno tutti i diagrammi delle classi appartenenti al package ed eventuali sotto-package. Una dipendenza tra package si indica con una freccia tratteggiata come mostrato nella figura sottostante.

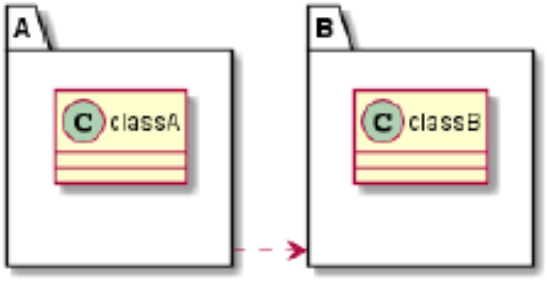

Figura 5: Immagine freccia package

2.3.3.3 Diagrammi di sequenza

I diagrammi di sequenza vanno letti dall'alto verso il basso in quanto tale senso di lettura indica lo scorrere del tempo. Gli oggetti sono rappresentati con dei rettangoli contenenti un nome per identificarli. Sotto ad ogni istanza ci sarà una linea della vita tratteggiata, che sarà sormontata in alcuni tratti da una barra di attivazione (la quale indica i momenti un cui l'oggetto è attivo). Dalle barre di attivazione partiranno frecce rappresentanti il messaggio/segnale verso la linea di vita di oggetti già istanziati o verso una nuova classe. I tipi di frecce sono i seguenti:

- **Freccia piena**: per indicare un messaggio sincrono e corrisponde alla chiamata di un metodo;
- **Freccia**: per indicare un messaggio asincrono che dunque ritorna immediatamente;
- **Freccia tratteggiata**: per indicare il ritorno di un metodo chiamato e sopra tale freccia andrà indicato il tipo di ritorno;
- **Freccia tratteggiata con create**: per indicare la creazione di un nuovo oggetto;
- **Freccia piena con destroy**: per indicare la distruzione di un oggetto e termina sempre in una X;
- **Freccia piena**: per indicare un messaggio sincrono e corrisponde alla chiamata di un metodo.

2.3.3.4 Diagrammi di attività

I diagrammi di attività descrivono il flusso delle attività del software in modo da esplicarne il funzionamento dinamico. Ogni attività viene inserita calcolando che non ci possano essere interferenze durante la sua esecuzione. Un'attività elementare è formata da un nodo iniziale, punto da cui inizia l'esecuzione detto *token*, e da un'activity, contenente la descrizione del token.

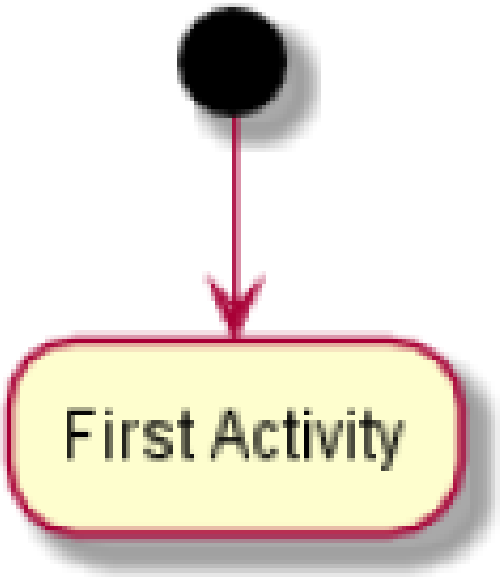

Figura 6: Immagine freccia attività

Il token è rappresentato da un pallino pieno, l'activity è un rettangolo con del testo all'interno contenente la descrizione che dovrà essere molto sintetica. Vi sono anche altre componenti che possono far parte di un diagramma di attività:

- **Subactivity**: rappresentata da un rettangolo con il nome all'interno ed un piccolo tridente in basso a destra. Ciascuna sotto attività ha un input ed un output;
- **Branch**: rappresentato da un rombo con una freccia in ingresso ed n frecce in uscita. Si può prendere solo uno dei rami e ciascuno di questi deve avere scritta la guardia. Consuma e produce token;
- **Merge**: rappresentato da un rombo con n frecce in ingresso ed una in uscita. Esso è il punto in cui gli n rami di un $branch_G$ tornano ad unirsi. Consuma e produce token;
- **Fork**: rappresentato da una linea lunga orizzontale o verticale, è il punto il cui l'attività si parallelizza senza vincoli temporali di esecuzione. Consuma un token e ne genera n;
- **Join**: rappresentato da una linea lunga orizzontale o verticale, è il punto in cui avviene la sincronizzazione fra processi paralleli. Consuma n token e ne genera uno;
- **Segnali**: rappresentati da due figure ad incastro con la prima non bloccante emette il segnale mentre la seconda è bloccante e riceve il segnale;
- **Timeout**: rappresentato da una clessidra, serve a modellare il timeout o eventi ripetuti. Hanno frecce entranti ed uscenti. Un timeout deve cominciare con la descrizione *wait* mentre un ciclo ripetuto con *every*;

- **Nodo fine flusso**: rappresentato da un cerchio vuoto con una X al centro, è il punto in cui un ramo di esecuzione muore;
- **Nodo finale**: rappresentato da due cerchi concentrici, è il punto in cui termina l'esecuzione e consuma un token.

2.3.4 Sviluppo

Lo sviluppo del progetto avviene seguendo il modello iterativo secondo quando spiegato nel documento *Piano di progetto*. Il team *duckware* ha deciso di utilizzare le tecnologie specificate dalla proponente in quanto ha ritenuto che siano adatte per la realizzazione del prodotto e che non ne siano necessarie ulteriori. Il *capitolato d'appalto*$_G$ consiglia le seguenti tecnologie:

- **Java**: per lo sviluppo dell'applicazione e la comunicazione di essa con i servizi di AWS;
- **Node.js**: per l'integrazione di Alexa con l'infrastruttura di AWS;
- **AWS**: servizi di Amazon per la realizzazione del database delle API; nel particolare verranno utilizzati:
  - **AWS DynamoDB**: per la gestione dei dati;
  - **AWS Lambda**: per l'implementazione del servizio *serverless*$_G$.

2.3.5 Obiettivi della progettazione

La progettazione garantisce che il prodotto finale soddisfi tutte le richieste emerse e specificate durante l'analisi. Deve garantire la qualità del prodotto sviluppato e cercare di ottimizzare l'uso delle risorse disponibili. Inoltre, una buona progettazione cerca di suddividere nel migliore dei modi i compiti implementativi al fine di ridurre la complessità del problema e, di conseguenza, facilitare il compito dei programmatori.

2.4 Codifica

I programmatori dovranno rispettare alcune regole che garantiranno uniformità e coesione al codice che verrà prodotto durante la creazione del progetto. Ci sono sia delle norme globali alle quali i programmatori si devono attenere, indipendentemente dal linguaggio di programmazione scelto, sia delle norme specifiche legate a Java e Node.js.

Per chiarezza ogni norma avrà un suo paragrafo composto di titolo, descrizione ed esempio illustrativo di quanto si vuole specificare.

Convenzioni per i nomi

- Ogni nome deve essere unico e soprattutto deve essere appropriato, ovvero deve descrivere al meglio ciò che rappresenta;

- In caso di nome formato dalla composizione di più parole, si potrà utilizzare l'underscore come separatore. In alternativa, si potranno separare i termini con una lettera maiuscola secondo la modalità detta *lowerCamelCase*$_G$. Ad esempio:
  - `variable_name`
  - `ANOTHER_EXAMPLE_FOR_THIS`
  - `yetAnotherExample`
- Usare le lettere maiuscole quando si intende utilizzare una parola come attributo *const* (costante). Altrimenti, usare lettere minuscole. Per esempio:
  - *(costante)* `variable_name`
  - (**non** costante) `ANOTHER_EXAMPLE_FOR_THIS`
  - (**non** costante) `yetAnotherExample`

Convenzioni per la documentazione

- I commenti ed i nomi presenti nel documento devono tassativamente essere scritti in inglese;
- Ogni file dovrà contenere nella sua intestazione, come commento, la seguente struttura:

File : filename.java Version : versione file Date : data di creazione Author : nome autore/i License : tipo di licenza del file Changelog : Autore || Data || breve descrizione delle modifiche

Tabella 2: Esempio commento intestazione file

- La versione va inserita come X.Y dove:
  - **X**: indica la versione principale ed è da incrementare **solo ed esclusivamente** nel caso in cui si passi ad una versione stabile. Un tale incremento comporta l'azzeramento di Y;
  - **Y**: indica la versione parziale ed un incremento di tale indice rappresenta una verifica o una modifica rilevante che verrà sottoposta alla fase di test.

2.4.1 Java

In questa sezione vengono indicate delle convenzioni adottate dal team di *Google*$_G$ che dovranno essere rispettate nel progetto dagli sviluppatori durante la scrittura del codice.

2.4.1.1 Commenti

I commenti su singola riga possono essere inseriti tramite l'uso di un doppio slash *(single line comment block)* oppure da uno slash seguito da un asterisco *(multi-line comment block)*. Quest'ultima tecnica deve essere utilizzata anche per i commenti su più righe dove, per ciascun inizio di riga, deve essere presente un asterisco. È necessario inserire uno spazio dopo ogni asterisco di inizio riga, se presente.

OK	NO
//commento su singola riga	//commento //su più
/* commento su singola riga */	//righe
/* commento * su più * righe */	/*commento *su più *righe*/

Tabella 3: Esempio commenti

Ogni file `.java` dovrà contenere in testa un commento a singola o multipla riga che descriva brevemente il contenuto del file e le funzionalità che offre.

2.4.1.2 Indentazione

Il corpo di ogni funzione dovrà essere separato dall'inizio della riga con una tabulazione, lasciando le impostazioni standard fornite dall'IDE. In caso di istruzioni di controllo annidate, ognuna dovrà essere propriamente incolonnata e spaziata da tabulazioni rispetto all'inizio della riga.

OK	NO
double test(int a) { if (a > 5) { return a * 10; } else { return a * 20; } }	double test(int a) { if (a > 5) { return a * 10; } else { return a * 20; } }

Tabella 4: Esempio indentazione

2.4.1.3 Blocchi di inizio e fine

Ogni blocco dovrà essere contrassegnato dalle parentesi graffe, anche quando esse potrebbero essere omesse, per evitare problemi di comprensione del codice. Nello specifico:

OK	NO
if (true) { return true; } else { return false; }	if (true) return true; else return false;

Tabella 5: Esempio blocchi di inizio e fine

La stessa regola vale anche nel caso di statements quali `while, for` e tutti quelli che consentono l'omissione delle parentesi graffe in caso di istruzione singola.

2.4.1.4 Nomi - Variabili

I nomi delle variabili vanno sempre scritti con una lettera minuscola iniziale ed il resto del corpo deve essere anch'esso minuscolo. In caso di variabili formate da parole composte, è possibile scrivere alcuni caratteri in maiuscolo:

OK	NO
void test() { int a; String qualcosa; String qualcosaAltro; bool ancoraAltro2; //... }	void test() { int A; String Qualcosa; String QualcosaAltro; bool ANCORAALTRO; }

Tabella 6: Esempio nomi variabili

La modalità di scrittura delle variabili appena descritto è conosciuto come lowerCamelCase. Sarà anche possibile separare i nomi delle variabili utilizzando un underscore ma, in questo caso, ogni carattere dovrà essere completamente minuscolo o in maiuscolo (e quindi non adottare il lowerCamelCase).

OK	NO
void test() { int _a; String qualcosa_altro; String qualcosaAltro; bool QUALCOSA_ALTRO_2; //... }	void test() { int _A; String QUALCOSA_altro; String Qualcosa_Altro; bool Ancora_Altro_2; //... }

Tabella 7: Esempio lowerCamelCase

In caso di costanti, sarà necessario scrivere la variabile in maiuscolo e si potranno anche usare gli underscore. `PI_VALUE = 3.1415;` `PIVALUE = 3.1415;`

È fondamentale e necessario che ogni nome di variabile (comprese quelle costanti) debba avere un significato e rappresentare ciò che quella variabile indica. Non sono quindi ammessi nomi di variabili o costanti quali `pippo` o `gigi`.

2.4.1.5 Nomi - Metodi e procedure

I nomi dei metodi devono essere scritti rispettando le regole del lowerCamelCase. Solitamente la nomenclatura deve essere relativa ad un verbo o ad una frase come ad esempio `inviaMessaggio` oppure `stop`.

OK	NO
void test() { //... } void testMethod() { //... }	void TEST() { //... } void TestMethod() { //... }

Tabella 8: Esempio nomi metodi

Non è possibile usare gli underscore per separare il nome del metodo in più unità sintattiche. I nomi dei parametri dei metodi devono anch'essi aderire allo stile di scrittura *CamelCase*$_G$.

2.4.1.6 Nomi – Classi ed interfacce

I nomi delle classi devono essere scritti rispettando le regole dell'*UpperCamelCase*$_G$. Tipicamente i nomi delle classi sono nomi o brevi descrizioni come ad esempio `Character` oppure `ListaImmutabile`. Le stesse regole si applicano per le interfacce ma queste ultime,

in più, possono anche avere un nome che sia un aggettivo come `Leggibile` (dall'inglese `Readable`).

OK	NO
class Dog { //... } class BuildingBlock { //... } interface Drinkable { //... } class Water implements Drinkable { //... }	class dog { //... } class BUILDING_BLOCK { //... } interface Drink { //... } class water implements Drink { //... }

Tabella 9: Esempio nomi classi

2.4.1.7 Regole pratiche

1. Quando si fa l'*override*$_G$ di un metodo è necessario inserire sempre l'annotazione `@Override`. Nel caso il metodo sia stato marcato con `@Deprecated`, allora non è necessario inserire l'annotazione di override.

OK	NO
public Test implements Runnable { @Override public void run() { //... } }	public Test implements Runnable { public void run() { //... } }

Tabella 10: Esempio regole pratiche

2. Nel caso di eccezioni da catturare, queste non devono essere ignorate o catturate senza eseguire alcuna azione. Questa pratica, detta *"exception swallowing"* non è permessa.

OK	NO
try { qualcosa(); qualcosaAltro(); } catch (IOException e){ handle(e); }	try { qualcosa(); qualcosaAltro(); } catch (IOException e){ }

Tabella 11: Esempio regole pratiche

Nell'esempio, il metodo `handle()` è identificativo del fatto che verrà eseguita qualche azione che possa notificare la cattura dell'eccezione.

3. Scrivere metodi che siano il più corti possibile. Quando si eccedono le 40 righe, conviene considerare la possibilità di spezzare il codice e distribuirlo su più metodi.

4. Utilizzare commenti TODO a singola linea che descrivano brevemente cosa deve essere ancora implementato nel metodo oppure se ci sono correzioni da fare.

   ```
   // TODO: aggiungere un flag nel ciclo for che indichi lo stato della variabile x
   ```

2.4.2 Node.js

In questa sezione vengono indicate le convenzioni indicate dal team di sviluppo di Node.js[4] che dovranno essere rispettate nel progetto dagli sviluppatori durante la scrittura del codice.

2.4.2.1 Commenti

I commenti su singola riga possono essere inseriti tramite l'uso di un doppio slash *(single line comment block)* oppure da uno slash seguito da un asterisco *(multi-line comment block)*. Quest'ultima tecnica deve essere utilizzata anche per i commenti su più righe dove, per ciascun inizio di riga, deve essere presente un asterisco. È necessario inserire uno spazio dopo ogni asterisco di inizio riga, se presente.

[4] https://docs.npmjs.com/misc/coding-style.html

OK	NO
//commento su singola riga /* commento su singola riga */ /* commento * su più * righe */	//commento //su più //righe /*commento *su più *righe*/

Tabella 12: Esempio commenti di Node.js

Ogni file `.js` dovrà contenere in testa un commento a singola o multipla riga che descriva brevemente il contenuto del file e le funzionalità che offre.

2.4.2.2 Indentazione

Vanno bene le impostazioni di indentazione fornite dell'editor Visual Studio Code per Node.js. Non ci sono restrizioni sull'uso della tabulazione o del doppio spazio, anche se il primo approccio è preferibile.

2.4.2.3 Blocchi di inizio e fine

Ogni blocco dovrà essere contrassegnato dalle parentesi graffe, anche quando esse potrebbero essere omesse, per evitare problemi di comprensione del codice. La parentesi graffa di apertura deve stare nella stessa linea del costrutto che inizializza.

OK	NO
if (true) { return true; } else { return false; }	if (true) { return true; } else return false;

Tabella 13: Esempio blocchi di inizio e fine

La stessa regola vale anche nel caso di statements quali `while, for` e tutti quelli che consentono l'omissione delle parentesi graffe in caso di istruzione singola.

OK	NO
if (true) { method(); } while(true) { method2(); }	if (true) { method(); } while (true) method2();

Tabella 14: Esempio 2 di blocchi di inizio e fine

2.4.2.4 Variabili

I nomi delle variabili vanno sempre scritti con una lettera minuscola iniziale ed il resto del corpo deve essere anch'esso minuscolo. In caso di variabili formate da parole composte, è possibile scrivere alcuni caratteri in maiuscolo:

OK	NO
void test() { var a; var qualcosa; var qualcosaAltro; var ancoraAltro2; //... }	void test() { var A; var Qualcosa; var QualcosaAltro; var ANCORAALTRO; }

Tabella 15: Esempio nomi variabili

Sarà anche possibile separare i nomi delle variabili utilizzando un underscore ma, in questo caso, ogni carattere dovrà essere completamente minuscolo o in maiuscolo (e quindi non adottare il lowerCamelCase).

OK	NO
void test() { var _a; var qualcosa_altro; var qualcosaAltro; var QUALCOSA_ALTRO_2; //... }	void test() { var _A; var QUALCOSA_altro; var Qualcosa_Altro; var Ancora_Altro_2; //... }

Tabella 16: Esempio lowerCamelCase

In caso di costanti, sarà necessario scrivere la variabile in maiuscolo e si potranno anche usare gli underscore. `PI_VALUE = 3.1415;` `PIVALUE = 3.1415;`

È fondamentale e necessario che ogni nome di variabile (comprese quelle costanti) debba avere un significato e rappresentare ciò che quella variabile indica. Non sono quindi ammessi nomi di variabili o costanti quali `pippo` o `gigi`.

2.4.2.5 Quotes

Utilizzare l'apostrofo (single quote) sempre tranne quando è necessario fare l'escape delle virgolette (double quotes).

OK	NO
var test = 'qualcosa'; var test = 'qualcosa "con" escape';	var test = "qualcosa"; var test = 'qualcosa \' con \' escape';

Tabella 17: Esempio utilizzo di quotes

2.4.2.6 Chiamate sincrone ed asincrone

Usare la versione asincrona (non bloccante) dei metodi ovunque possibile e limitarsi alle chiamate bloccanti solamente nel caso in cui sia strettamente necessario. Il *callback* deve essere sempre l'ultimo argomento della lista mentre il primo deve essere o $Error_G$ oppure *null*.

2.4.2.7 null, undefined, 0

Per quanto riguarda le costanti da utilizzare in Node.js, per quanto in alcuni casi possano essere interscambiabili, vanno usate solamente nel contesto adatto. In particolare:

- Variabili e funzioni booleane devono sempre ritornare *true* oppure *false*; non ritornare 0 o 1;
- Se qualcosa di interno alla funzione presenta problemi o manca di stato, utilizzare il *null*;
- Non impostare un valore a *undefined* per inizializzarlo; utilizzare il valore di default per quel tipo come ad esempio 0 per un intero o "" per una stringa;
- Oggetti booleani sono proibiti.

3 Processi di supporto

3.1 Documentazione

3.1.1 Descrizione

Verranno qui discusse le scelte intraprese per la scrittura, verifica ed approvazione della documentazione ufficiale di *duckware*. Queste norme sono obbligatorie per tutti i documenti, i quali verranno elencati nella sottosezione *Documenti correnti*.

3.1.2 Ciclo di vita della documentazione

Qualsiasi documento dovrà passare per gli stati di "*Sviluppo*", "*Verifica*" ed "*Approvato*". Nel particolare, ciascuno di questi indica una fase precisa:

- **Sviluppo**: Questa fase 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 usando i $ticket_G$;
- **Verifica**: Questa fase inizia dopo l'assegnazione da parte del *responsabile*. I *verificatori* effettueranno i controlli necessari e, in caso di esito positivo, il documento entra automaticamente nella fase "*Approvato*". In caso di esito negativo sarà necessario che il *responsabile* di progetto riassegni il documento ad un *redattore* attraverso una nuova fase di "*Sviluppo*";
- **Approvato**: Questa fase coincide con il completamento della parte di "*Verifica*" avvenuta con successo nella quale il verificatore certifica la correttezza. Il documento verrà dunque consegnato al *responsabile* il quale lo approverà per il rilascio esterno.

3.1.3 Separazione tra documenti interni ed esterni

Ogni documento dovrà avere una specifica classificazione. Vi saranno sia documenti **interni** in lingua italiana che verranno utilizzati da *duckware* internamente, sia documenti **esterni** che verranno condivisi con *la Proponente* ed i committenti. Questi ultimi potrebbero essere scritti in lingua inglese nel caso potesse essere utile al fine di soddisfare le richieste di deploy dell'utente finale.

3.1.4 Nomenclatura dei documenti

Ogni documento, tranne la Lettera di Presentazione ed i Verbali, adotteranno questo schema di nomenclatura:

- **NomeDocumento**: indica il nome del documento e dovrà essere senza spazi con il vincolo di avere una lettera maiuscola all'inizio di ogni parola;
- **vX.Y.Z**: indica il numero versionamento.

Nel particolare, il versionamento sarà composto da tre numeri interi, separati da un punto, con il seguente significato:

- **X**: rappresenta il numero di pubblicazioni ufficiali del documento e ad ogni suo incremento corrisponde un azzeramento di Y e Z;
- **Y**: rappresenta il numero di verifiche terminate con successo e ad ogni suo incremento corrisponde un azzeramento di Z;
- **Z**: rappresenta il numero di modifiche effettuate al documento durante lo sviluppo.

Ogni documento sarà redatto all'interno di un file .tex e solamente dopo lo stato di "Approvato" verrà generato il relativo PDF che conterrà la versione definitiva approvata dal *responsabile* di progetto.

3.1.5 Documenti correnti

Verranno ora esposti i documenti formali, in ordine alfabetico, e divisi per appartenenza (Interno ed Esterno):

ESTERNO

- **[AdR] - Analisi dei Requisiti**: Questo documento viene redatto agli *Analisti* dopo che questi ultimi hanno analizzato il capitolato e interagito con il proponente. All'interno dell'analisi dei requisiti saranno esposti tutti i requisiti del progetto, inclusi i diagrammi di interazione con l'utente ed i casi d'uso;
- **[GL] - Glossario**: Documento che raccoglie tutti i termini che verranno usati nei documenti formali; serve a disambiguare termini o a facilitarne la comprensione;
- **[PdP] - Piano di progetto**: Documento che tratta della pianificazione e dell'analisi della gestione delle risorse di tempo;
- **[PdQ] - Piano di qualifica**: Tale documento descrive gli obiettivi che il gruppo sarà tenuto a soddisfare al fine di garantire la qualità del prodotto e del processo.

INTERNO

- **[NdP] - Norme di progetto**: Documento che descrive gli standard adottati da *duckware* durante lo sviluppo del progetto scelto;
- **[SdF] - Studio di fattibilità**: Documento che analizza i pregi ed i punti a sfavore di ogni capitolato con le relative riflessioni che hanno portato *duckware* alla sua scelta finale.

3.1.6 Norme

3.1.6.1 Struttura dei documenti

Ogni documento presentato è stato realizzato seguendo uno schema generale che dovrà essere rispettato in ogni documento ufficiale, fatta eccezione della lettera di presentazione e dei verbali. I criteri sono i seguenti:

- **Frontespizio**: sezione presente nella prima pagina di ogni documento e conterrà il logo di *duckware*, il titolo del relativo documento, la versione del documenti, il nome del gruppo ed il nome del progetto;
- **Informazioni sul documento**: si compone di una lista di responsabili, verificatori, redattori del documento e della tipologia d'uso del documento, una breve descrizione del contenuto e, a fondo pagina, numero di versione corrente e data di quando è stato modificato per l'ultima volta il documento;
- **Diario delle modifiche**: sarà presente nella seconda pagina di ogni documento e conterrà una tabella, ordinata in modo decrescente per data, delle modifiche apportate al documento. Nello specifico, ogni riga conterrà: versione, data, descrizione delle modifiche, autore e ruolo;
- **Indice delle sezioni**: si tratta di un elenco degli argomenti trattati nel documento che conterrà: titolo, argomento e numero pagina;
- **Indice delle tabelle**: sezione che contiene l'elenco delle tabelle e ha la stessa struttura dell'indice delle sezioni;
- **Introduzione**: contiene lo scopo del documento, le informazioni sul glossario ed i riferimenti utili normativi ed informativi;
- **Contenuto del documento**: tutto ciò che non è stato elencato nei precedenti punti, verrà trattato all'interno del documento.

3.1.6.2 Norme tipografiche

- **Intestazione**: in ogni pagina del documento dopo il frontespizio ci sarà sulla sinistra il nome del capitolo corrente e sulla destra il logo di *duckware*;
- **Parentesi**: le parentesi tonde descrivono esempi e forniscono sinonimi, le quadre rappresentano uno standard ISO o un riferimento ad un codice definito all'interno del documento stesso;
- **Piè di pagina**: a sinistra c'è il nome del documento, a destra il numero di pagina;
- **Stile del testo**
  - Corsivo: dà enfasi ad un termine o concetto;
  - Grassetto: per i titoli, sottotitoli o termini all'interno di elenchi o liste.
- **Formati**
  - *Date*: ogni data verrà formattata secondo il formato dd-mm-yyyy dove dd indica il giorno (a una o due cifre), mm il mese (a due cifre o in forma letterale estesa) e yyyy l'anno (sempre a quattro cifre). Solo il *Registro delle Modifiche* fa eccezione a questa regola, in quanto la data adotterà il formato yyyy-mm-dd, per dare risalto all'andamento cronologico delle iterazioni sul documento;

  - *Grassetto*: va utilizzato per i titoli di paragrafi e per i titoli di elementi in elenco;
  - *URI*: ogni URI sarà in corsivo e di colore blue di modo da essere conformi agli standard degli URI nelle pagine web.

- **Riferimenti informativi**: Ogni riferimento esterno al progetto, come ad esempio guide o software, dovrà essere indicato a piè di pagina;

- **Nomi**: sono stati realizzati comandi personalizzati per richiamare la visualizzazione dei seguenti termini:

  - `\groupName` visualizza il nome del gruppo, "duckware";
  - `\groupEmail` visualizza l'indirizzo email del gruppo, "duckware.swe@gmail.com";
  - `\verif` è un comando che, tramite l'utilizzo di `\renewcommand` posto all'inizio del file .tex principale, permette di visualizzare i nomi dei verificatori assegnati al documento;
  - `\resp` come il comando precedente, visualizza il nome del responsabile del documento;
  - `\editorfrow` e `\editorsrow` inseriscono nel documento i nomi dei redattori del documento, rispettivamente nella prima e nella seconda riga;
  - Sono stati creati dei comandi specifici per i nomi dei singoli componenti dei gruppi, in modo da semplificare e velocizzare la creazione dei documenti. Esempio: `\luca` visualizzerà il nome nel formato "Luca Stocco";
  - Per standardizzare la nomenclatura dei documenti, sono stati aggiunti dei comandi appositi. Esempio: `\pdp` visualizzerà il nome del documento nel formato "Piano di Progetto".

- **Componenti grafiche**: sono ammessi formati PNG e JPG ma sono preferibili immagini informato SVG poiche queste ultime preservano una maggiore qualità anche in caso di ridimensionamento.

È stato creato un template di documentazione che potrà essere impiegato per la realizzazione dei vari documenti ufficiali. Questo template è conforme a tutte le norme di documentazione esposte nelle precedenti sezioni.

3.1.7 Ambiente

La scrittura dei documenti dovrà essere realizzata con *TexStudio*$_G$ o *TexMaker*$_G$, due ambienti di scrittura integrato per la creazione di documenti LaTeX. Questi software rendono la scrittura LaTeX semplice e confortevole oltre che fornire numerose funzionalità come l'evidenziazione della sintassi, il visualizzatore integrato, il controllo dei riferimenti e vari assistenti.
Per maggiori dettagli:

- TeXstudio[5];
- TexMaker[6];

[5] https://www.texstudio.org/
[6] http://www.xm1math.net/texmaker/

3.1.8 Strumenti di supporto

Per facilitare e velocizzare la manutenzione della documentazione è stato realizzato un programma per automatizzare l'esecuzione di certe attività. Sarà possibile utilizzare questo programma in qualsiasi sistema operativo che abbia .NET Framework installato. Il programma `GlossaryHelper.exe` facilita la manutenzione e l'aggiornamento del glossario e di tutti i suoi documenti, nel particolare, si tratta di un eseguibile dotato di GUI in grado di:

- Caricare il file *.tex relativo al glossario per inserire e/o rimuovere termini al suo interno. Il programma verificherà la presenza di duplicati nei termini del glossario e, in caso ce ne siano, impedirà l'inserimento;
- In seguito all'aggiunta o alla rimozione di termini dal glossario, sarà possibile aggiornare tutti i file dei documenti. Sarà sufficiente specificare al programma la cartella root contenente i documenti da analizzare ed il programma avrà cura di selezionare ogni file *.tex per procedere all'aggiornamento.

Per poter eseguire il programma è consigliabile avere installata l'ultima versione di .NET Framework, tuttavia il requisito minimo è quello di avere il supporto per .NET Framework 4.7.2 (per C# 7.1).

3.2 Qualità

3.2.1 Descrizione

Questa sezione descrive le procedure per il calcolo dei parametri descritti nel Piano di qualifica.

3.2.2 Classificazione dei processi

Per garantire la qualità del lavoro, gli Amministratori hanno suddiviso il lavoro in vari processi che sono stati poi riportati nel *Piano di Qualifica*. Dovrà essere rispettata la notazione **PROC[value]** dove **value** indica il codice univoco del processo tramite un numero intero a tre cifre che parte da 1 ed incrementa per ogni unità.

3.2.3 Classificazione delle metriche

Per garantire la qualità del lavoro fatto gli Amministratori hanno definito delle metriche che rispettano la seguente notazione **M[categ][macro categ][num]** oppure **M[categ][macro categ][num][sottonum]**dove:

- **categ**: va ad indicare la categoria della metrica ed assume i seguenti valori:
  - **PRC**: per indicare i processi;
  - **PRD**: per indicare i prodotti;
  - **PDT**: per indicare i test.

- **macrocateg**: indica la macrocategoria della metrica, se esiste altrimenti non compare.
  Per le metriche di prodotto **PRD** può assumere i valori:
  - **D**: per indicare i documenti;
  - **S**: per indicare il software.

- **num**: va ad indicare il codice univoco della metrica come numero intero a tre cifre a partire da 1. Esempio **MPRC001**.

- **sottonum**: va ad indicare il codice univoco di una sotto-metrica appartenente alla metrica padre, con un numero intero ad una cifra a partire da 1. Esempio **MPRC001.1** è una sotto-metrica della metrica **MPRC001**.

3.2.4 Controllo di qualità di processo

La qualità dei processi verrà garantita dal metodo *PDCA*$_G$, descritto nell'appendice A. Grazie a tale metodo si potrà ottenere un miglioramento continuo della qualità di tutti i processi, inclusa la verifica. Per ottenere la qualità dei processi bisogna:

- Definire il processo affinchè esso sia controllabile;
- Controllare il processo in funzione del raggiungimento di un alto livello di efficacia ed efficienza;
- Usare strumenti di valutazione quali PDCA e *SPICE*$_G$.

3.3 Configurazione

3.3.1 Controllo di versione

3.3.1.1 Descrizione

Per le parti versionabili del progetto e per i documenti ufficiali si è scelto l'utilizzo di GitLab, la condivisione per documenti informali ed altro materiale di supporto durante lo sviluppo del progetto avviene per mezzo di *Google Drive*$_G$.

3.3.1.2 Struttura della repository Nella root della *repository*$_G$ sono presenti 4 cartelle:

- **firme:** contiene le immagini delle firme dei componenti del gruppo;
- **Template_latex:** contiene i file `.tex` che gestiscono i comandi personalizzati, i comandi di formattazione delle pagine, i packages da includere per poter compilare con successo i documenti, il logo, le definizioni del glossario, le entries per stilare la bibliografia (se necessaria o richiesta). Inoltre contiene a sua volta le cartelle *esempio_documento* e *esempio_verbale* che descrivono come devono essere strutturate le cartelle dei singoli documenti;

- **Tools:** in questa cartella verranno inseriti tutti gli strumenti di utilità, come ad esempio `GlossaryHelper.exe`;
- **RR:** questa cartella contiene tutti i documenti relativi alla *Revisione dei Requisiti*.
- **RP:** questa cartella contiene tutti i documenti relativi alla *Revisione di Progettazione*.

Nella cartella **RR** vengono distinti i documenti a seconda del loro utilizzo:

- Nella **Cartella Esterni** sono presenti altre sotto cartelle con all'interno i documenti correlati:
  - Analisi dei requisiti;
  - Glossario;
  - Piano di progetto;
  - Piano di qualifica;
  - Verbali.
- Nella **Cartella Interni** invece troviamo le apposite cartelle per i seguenti documenti:
  - Glossario;
  - Studio di fattibilità;
  - Norme di progetto;
  - Verbali interni.

La cartella **RP** avrà la stessa struttura della cartella **RR**. Verranno create successivamente altre directory in base alle necessità che si presenteranno durante lo sviluppo del progetto, come ad esempio la cartella *includes* presente in ogni sotto cartella dei documenti e che racchiude tutti i file inclusi solo in quello specifico documento. All'interno di ogni cartella vi è un file LaTeX che assume il nome del documento e la sua versione attuale.

3.3.1.3 Ciclo di vita dei branch

Per migliorare l'efficienza e l'efficacia di sviluppo, sono stati creati dei branch, denominati con il nome del rispettivo documento in redazione. Inoltre, è presente un branch di lavoro, chiamato `develop` nel quale confluiranno tutti i branch, una volta approvati i singoli documenti. Una volta che tutti i documenti sono stati verificati ed approvati, si raggiunge una fase chiamata *milestone*$_G$, e si può dunque procedere ad effettuare una *release*$_G$. I documenti della saranno contenuti nel branch `master`.

3.3.1.4 Aggiornamento della repository

Per l'aggiornamento della repository verranno usati i seguenti comandi Git:

- "git status": per verificare in che branch ci si trova. Inoltre questo comando permette di tenere sotto controllo le modifiche locali fatte ai file;
- "git checkout *nomeBranch*": permette di spostarsi nel branch indicato;

- "git pull": permette di aggiornare la repository locale;
- "git add *nomeFile*": aggiunge il file specificato all'*area di staging*$_G$;
- "git add .": aggiunge tutti i file che sono stati modificati all'area di staging;
- "git commit": permette di salvare le modifiche aggiunte in area di staging al repository locale;
- "git push": sincronizza il repository remoto con quello locale.

3.3.2 Strumenti

- Server git: è stato utilizzato GitLab per l'affidabilità e il supporto a continuous integration;
- Client git: sono state utilizzate le applicazioni GitKraken (per sistemi operativi basati su Linux) e GitHub Desktop (per sistemi MacOS e Windows), oltre agli strumenti da linea di comando.

3.4 Procedure di controllo di qualità di processo

Per assicurare la qualità del prodotto finale è necessario perseguire la qualità dei processi che lo definiscono. Per adempiere a tale obbiettivo è stato deciso di seguire un'organizzazione interna dei processi incentrata sul principio del miglioramento continuo: PDCA (Plan, Do, Check, Act) e di adottare lo standard ISO/IEC 15504, conosciuto come SPICE (Software Process Improvement and Capability Deter- mination), contenente un modello di riferimento che definisce una dimensione del processo ed una dimensione della capacità. Per ottenere qualità sui processi è necessario:

- **Definire il processo**
  In modo tale che sia controllabile
- **Controllare il processo**
  Con l'obbiettivo di ottenere efficacia ed efficienza
- **Usare strumenti di valutazione**
  PDCA e SPICE

3.5 Metriche qualità per il processo

Verranno utilizzate le seguenti metriche per valutare l'efficienza e l'efficacia dei processi. Saranno quindi fatte delle misurazioni periodiche delle metriche, circa ogni 7 giorni da inizio a fine revisione, sui processi per quantificare l'andamento della qualità

3.5.1 MPRC001 - Schedule Variance

Si tratta di una formala che indica se una pianificazione del progetto è in linea con la schedulazione temporale delle attività. Essa si calcola attraverso la seguente formula:

$$SV = BCWP - BCWS$$

- **BCWP**: valore delle attività completate al momento del calcolo;
- **BCWS**: valore pianificato per realizzare le attività.

Il risultato ne segue un valore positivo, indice di una velocizzazione nello svolgimento dei processi, o un valore negativo, indice di un rallentamento.

3.5.2 MPRC002 - Budget Variance

Questa formula mostra se i costi sono stati previsti correttamente attraverso la seguente formula:

$$BV = BCWS - ACWP$$

- **BCWS**: costo sostenuto fino al momento del calcolo;
- **ACWP**: valore pianificato per la realizzazione delle attività.

Il risultato ne segue un valore positivo, indice di una spesa più bassa rispetto a quanto pianificato in precedenza, o un valore negativo, indice di una spesa fuori dal budget.

3.5.3 MPRC003 - Rischi non previsti

Valore numerico che indica la quantità di rischi esterni presenti ed elencati nel documento Analisi dei Rischi v2.0.0 riscontrati nella corrente fase di progetto. Si tratta di un indice incrementale che parte da 0 per ogni rischio che si presenta senza essere stato individuato in precedenza nel set di rischi.

3.5.4 MPRC004 - Indisponibilità servizi terzi

Numero totale di giorni nei quali i servizi utilizzati non siano disponibili perché offline oppure bloccato. Tale numero parte da 0 ed incrementa di uno per ogni giorno in cui il servizio risulta essere non disponibile. Verrà utilizzato il tool automatico [7].

3.5.5 MPRC005 - Media di commit per settimana

Media dei commit effettuati settimanalmente nelle diverse repository utilizzare dal gruppo. La misurazione avviene grazie ad un bot collegato a GitLab ed integrato su Slack in grado di leggere i log degli eventi sulla repository.

3.5.6 MPRC006 - Misurazione dei test

Verranno ora elencate una serie di misurazioni che hanno lo scopo di tenere traccia delle esecuzioni dei test con i relativi successi o fallimenti.

[7] statusticker

- **MPRC006.1 - Percentuale test passati**: indica la percentuale di test passati ed è utile per capire a che punto ci si trova nella fase di sviluppo del software. La formula per il calcolo del PTP è la seguente:

$$PTP = \frac{TP}{TE} * 100$$

Dove TP indica il numero di test passati mentre TE indica il numero di test eseguiti;

- **MPRC006.2 - Percentuale test falliti**: indica la percentuale di test falliti ed è utile per capire a che punto ci si trova nella fase di sviluppo del software. La formula per il calcolo del PTF è la seguente:

$$PTF = \frac{TF}{TE} * 100$$

Dove TF indica il numero di test falliti mentre TE indica il numero di test eseguiti;

- **MPRC006.3 - Efficienza progettazione test**: indica il tempo medio impiegato per la scrittura di un test. Un valore troppo grande potrebbe indicare un'elevata complessità del test. La formula per il calcolo dell'EPT è la seguente:

$$EPT = \frac{NTT}{TST}$$

Dove NTT indica il numero totale di test progettati mentre TST indica il tempo impiegato per la realizzazione dei test;

- **MPRC006.4 - Contenimento dei difetti**: indica il rapporto in percentuale tra i bug trovati durante i test ed i bug riscontrati durante l'utilizzo del prodotto. Un valore troppo basso dell'indice potrebbe suggerire una scarsa progettazione dei test. La formula per il calcolo del CD è la seguente:

$$CD = \frac{DT}{DTU} * 100$$

Dove DT indica il numero totale di difetti trovati nella fase di test mentre DTU indica la somma dei difetti trovati nei test e quelli trovati durante l'uso del software;

- **MPRC006.5 - Copertura dei test eseguiti**: indica la percentuale di test già eseguiti sul totale di test che verranno eseguiti. Metrica utile per monitorare il livello di avanzamento del lavoro dei verificatori del team. La formula per il calcolo del CTE è la seguente:

$$CTE = \frac{TE}{TT} * 100$$

Dove TE indica il numero totale di test eseguiti mentre TT indica il numero di test totali;

3.5.7 MPRC007 - Copertura requisiti

Verrà ora indicata una categoria di misurazioni utile per tenere traccia dell'esecuzione dei test e della copertura che questi hanno sui requisiti.

- **MPRC007 - Copertura requisiti**: indica la percentuale di requisiti coperti dai test sui requisiti totali. Metrica utile per capire quante parti del prodotto finale hanno un test associato. La formula per il calcolo del *CR* è la seguente:

$$CR = \frac{R}{RTOT} * 100$$

  Dove R indica il numero di requisiti coperti mentre RTOT indica il numero totale di requisiti;

3.6 Metriche qualità per i documenti

Verranno utilizzate le seguenti metriche per valutare l'efficienza e l'efficacia dei prodotti. Saranno quindi fatte delle misurazioni periodiche delle metriche, circa ogni 7 giorni da inizio a fine revisione, sui prodotti per quantificare l'andamento della qualità

3.6.1 MPRDD001 - Indice di Gulpease

L'*Indice di Gulpease*$_G$ è un indice di leggibilità di un testo tarato sulla lingua italiana. L'indice considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero di lettere. La formula per il suo calcolo è la seguente:

$$IG = 89 + \frac{300 * N_{\mathrm{F}} - 10 * N_{\mathrm{L}}}{N_{\mathrm{P}}}$$

Dove le variabili corrispondo:

- $\mathrm{N_F}$ il numero delle frasi;
- $\mathrm{N_L}$ il numero delle lettere;
- $\mathrm{N_P}$ il numero delle parole.

Il risultato finale *IG* è un numero compreso tra 0 e 100. In generale risulta che i testi con indice:

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

3.6.2 MPRDD002 - Errori ortografici

Gli errori ortografici possono essere identificati per mezzo dello strumento di 'Controllo ortografico' presente in TexStudio. Sarà poi compito di Verificatori correggerli e accertarsi che sia grammaticalmente corretto il contenuto del documento.

3.7 Metriche qualità per il software

3.7.1 MPRDS001 - Copertura requisiti obbligatori

Indica la percentuale dei $requisiti_G$ obbligatori coperti dall'implementazione. La formula di misurazione è la seguente:

$$CRO = \frac{ROS}{ROTOT} * 100$$

Dove ROS indica il numero totale di requisiti obbligatori soddisfatti mentre ROTOT indica il numero totale di requisiti obbligatori;

3.7.2 MPRDS002 - Copertura requisiti accettati

Indica la percentuale dei requisiti facoltativi coperti dall'implementazione. La formula di misurazione è la seguente:

$$CRA = \frac{RAS}{RATOT} * 100$$

Dove RAS indica il numero totale di requisiti accettati soddisfatti mentre RATOT indica il numero totale di requisiti accettati;

3.7.3 MPRDS003 - Percentuale di failure

Indica la percentuale dei test che si sono conclusi con esito negativo. La formula di misurazione è la seguente:

$$DF = \frac{NS}{NE} * 100$$

Dove NS indica il numero totale di fallimenti rilevati durante i test mentre NE indica il numero totale di test eseguiti;

3.7.4 MPRDS004 - Blocco operazioni non corrette

Indica la percentuale di funzionalità in grado di gestire in modo opportuno gli errori che potrebbero accadere. La formula di misurazione è la seguente:

$$BONC = \frac{NFE}{NNC} * 100$$

Dove NFE indica il numero totale di errori evitati durante i test effettuati mentre NNC indica il numero totale di test eseguiti che prevedono l'esecuzione di operazioni potenzialmente pericolose, ovvero in grado di generare errori;

3.7.5 MPRDS005 - Comprensibilità funzioni offerte

Indica la percentuale di operazioni comprese immediatamente dall'utente, ovvero quelle azioni che è stato in grado di svolgere subito senza dover consultare il manuale. La formula di misurazione è la seguente:

$$CFO = \frac{NFC}{NFO} * 100$$

Dove nfc indica il numero totale di funzioni comprese immediatamente senza dover leggere il manuale mentre NFO indica il numero totale di funzionalità offerte dal sistema;

3.7.6 MPRDS006 - Facilità di apprendimento di funzionalità

Indica il tempo medio impiegato dall'utente per imparare ad usare correttamente una specifica funzionalità. Si misura tramite indicatore numerico che rappresenta i minuti necessari ad un utente per capire come affrontare una certa funzionalità;

3.7.7 MPRDS007 - Tempo di risposta

Indica il tempo medio che trascorre tra la richiesta da parte del software di una certa risorsa o funzionalità e la restituzione del risultato da parte dell'utente. La formula di misurazione è la seguente:

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

Dove Ti indica il tempo trascorso fra la richiesta i di una funzionalità ed il comportamento delle operazioni necessarie a restituire un risultato di tale richiesta;

3.7.8 MPRDS008 - Impatto delle modifiche

Indica la percentuale di modifiche eseguite in risposta a dei fallimenti nei test che hanno portato alla nascita di nuovi errori all'interno di altre componenti nel sistema. La formula di misurazione è la seguente:

$$IM = \frac{NF}{NFR} * 100$$

Dove NR indica il numero di errori risolti con l'introduzione di altri nuovi errori mentre NFR indica il numero di errori risolti.

3.7.9 MPRDS009 - Complessità ciclomatica

La complessità ciclomatica è una metrica che indica la complessità di un programma ed è interamente basata sulla struttura del grafo rappresentante i vali algoritmi scelti. Nel particolare, i nodi sono le unità atomiche di istruzioni mentre gli archi sono i collegamenti fra tali nodi.

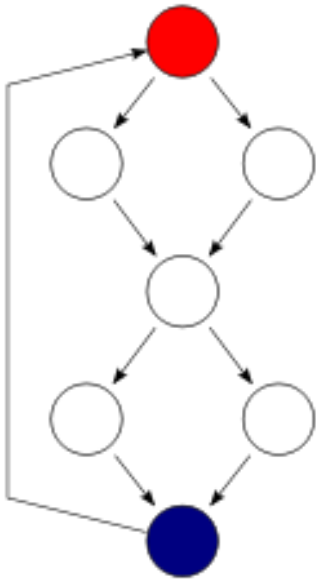

Figura 7: Immagine complessità ciclomatica

Tale indice può essere applicato a metodi, moduli e packages di un programma per limitare la complessità durante lo sviluppo. Durante la fase dei test è possibile utilizzare l'indice per determinare il numero di test necessari in quanto fornisce un limite superiore da non eccedere.

- **Range accettazione**: [0 - 30];

- **Range ottimale**: [0 - 10].

3.7.10 MPRDS010 - Numero di metodi

Questa metrica calcola la media di occorrenze di metodi per ciascun package ed è utile poiché questi ultimi non dovrebbero contenere troppe definizioni di metodi. Un numero superiore alla media potrebbe indicare la necessità di una maggiore suddivisione del package in questione.

- **Range di accettazione**: [2 - 10];
- **Range ottimale**: [3 - 8].

3.7.11 MPRDS011 - Variabili non utilizzate

Il linguaggio *Java*$_G$ consente la creazione di variabili che possono non essere mai effettivamente utilizzare e quindi inutili. Questo tipo di variabili verrà individuato utilizzando un tool apposito chiamato *ProGuard*[8] che si occuperà anche di questo compito, fra i tanti altri oneri di ottimizzazione che gli spettano. Non vi è nessun range di accettazione o alcun range ottimale da definire in quanto il valore di questa metrica deve sempre essere pari a 0.

3.7.12 MPRDS012 - Numero di bug per linea

Questa metrica è utile per quantificare il numero di bug presenti su una certa regione di codice, composta da un certo numero di linee. Rendendo il codice più chiaro e semplice possibile si ridurrà la probabilità di introdurre bug: il rischio di introdurre bug aumenta con il crescere del numero di righe di codice.

- **Range di accettazione**: [0 - 60];
- **Range ottimale**: [0 - 25].

3.7.13 MPRDS013 - Rapporto linee di codice e commento

Questa metrica riguarda le linee di codice prodotte in rapporto con le linee di commento, escludendo le righe vuote. L'indicazione principale riguarda la manutenibilità del codice. Un rapporto troppo basso indica una carenza di informazioni necessarie alla comprensione del codice.

- **Valore accettabile**: rapporto > 0.20;
- **Valore ottimale**: rapporto > 0.30.

[8] https://en.wikipedia.org/wiki/ProGuard_software

3.7.14 MPRDS014 - Code Coverage

Per poter avere una misura di codice verificato e testato si adotterà il Code Coverage che si calcola nel seguente modo:

$$CC = \frac{NMT}{TNMT} * 100$$

Dove NMT indica il numero di metodi sottoposti a test mentre TNMT indica il numero di metodi da sottoporre a test. Verrà utilizzato il tool JaCoCo[9] che è integrabile all'interno dell'IDE_G IntelliJ $IDEA_G$ e Android Studio per il linguaggio Java. Per quanto riguarda Node.js si farà affidamento a [10].

3.8 Verifica

3.8.1 Descrizione

In questa sezione verranno descritti gli strumenti ed i metodi impiegati per la verifica del codice e dei documenti durante la loro realizzazione.

3.8.2 Analisi statica

L'analisi statica è una tecnica applicabile al codice ed alla documentazione che permette la verifica di un prodotto individuandone gli errori. Può essere svolta secondo:

- **Inspection**: si applica quando si ha un'idea delle possibili problematiche che si stanno cercando e si effettua facendo una comparazione tra il prodotto e una tabella di possibili errori creata in precedenza;
- **Walkthrough**: si applica quando non si sa quali sono le tipologie di errori che si stanno cercando. Occorre ispezionare tutto il codice o il documento per trovare qualsiasi tipo di anomalia.

3.8.3 Analisi dinamica

L'analisi dinamica consiste nella creazione ed esecuzione di una serie di test direttamente sul codice, utilizzando anche dei tool già realizzati appositamente per tali scopi. Non è possibile fare analisi dinamica sui documenti.

3.8.4 Verifica diagrammi UML

I verificatori avranno il compito di controllare tutti i diagrammi UML ed assicurarsi che rispettino lo standard UML.

[9] https://www.jetbrains.com/help/idea/code-coverage.html
[10] https://mochajs.org/

3.9 Specifica dei test

3.9.1 Scopo

Per garantire la qualità del prodotto e quindi la rilevazione degli errori durante la fase di sviluppo verrà fatta particolare attenzione sull'analisi dinamica del codice per mezzo di test automatici.

3.9.2 Tipologia dei Test

Sono state create tre macro categorie di test sulla base di tre tipologie di errore per la loro identificazione:

- **Test di modulo** per verificare la logica del software;
- **Test ad alto livello** per verificare le funzionalità del sistema;
- **Test di regressione** per verificare i test precedenti dopo una modifica.

3.9.2.1 Test di modulo

I test di questa tipologia si occupano della verifica logica del software e verranno creati ed eseguiti dai programmatori. Il successo di questi test costituirà il vincolo per poter consegnare/caricare il codice all'interno della repository. Essi si dividono in:

- **Test di unità - TU**
  Si verifica le parti di lavoro prodotte dal programmatore che corrispondo a piccole parti di codice e unità logiche, come ad esempio una classe, un metodo o funzione, oppure un insieme di essi. Si va quindi a isolare dal resto del codice una piccola parte di software testabile, chiamata *unità*. Bisogna dunque stabilire se questa *unità* funziona esattamente come previsto prima di poterla stabilmente integrare nella versione principale del software. L'approccio al test di unità verrà affidato ad un apposito framework di testing chiamato [11].
- **Test di integrazione - TI** Si verifica l'integrazione tra le unità logiche che formano i vari componenti del sistema. Si verifica quindi che i componenti del sistema non contengano errori dovuti all'integrazione tra unità; Per testare l'integrazione si è scelto la tecnica dal basso verso l'alto, ovvero si testano per prime le parti con minore dipendenza funzionale e con maggiore funzionalità e successivamente risalire l'albero delle dipendenze.

3.9.2.2 Test ad alto livello

I test di questa tipologia si occupano di verificare le funzionalità del sistema, ovvero sul comportamento dell'applicazione, e gestire dai verificatori (team di *Quality Assurance*$_G$ - QA). Maggior parte di questi test sono manuali e per garantirne la loro organizzazione verrà gestita tramite degli apposti tool che verranno discussi in sede di *Technology Baseline*$_G$.

[11] JUnit

- **Test funzionali - TF**
  Si verifica l'implementazione delle specifiche del prodotto concentrandosi sulle funzionalità delle suddette specifiche. Tali test sono visti come dei test di unità ad alto livello, difatti l'analisi della struttura è delegata ai Test di unità - TU. Questi test possono essere automatizzati e scritti dai programmatori, ma vengono affiancati ad una revisione ed esecuzione svolta dai verificatori;

- **Test di sistema - TS**
  Si verifica il sistema e l'architettura nella sua interezza. Il test di sistema sancisce quindi la validazione del software finale, giunto ad una release definitiva, e verifica che esso soddisfi tutti i requisiti in modo completo. Tali test sono complessi e pesanti, la loro implementazione verrà discussa in sede di Technology Baseline. Necessario l'utilizzo di componenti software supervisionati dai verificatori, quindi eseguiti dal team di Quality Assurance - QA;

- **Test di validazione - TV**
  Sono dei test finali che hanno il compito di valutare se il sistema sviluppato corrisponde alle richieste del proponente, quindi tali verifiche sono legate ai requisiti. I test sono principalmente manuali ed eseguiti dal team Quality Assurance - QA. Nelle fasi finali dello sviluppo verranno fatti i controlli assieme ai proponenti per determinale la validità del prodotto.

3.9.2.3 Test di regressione

I test di questa tipologia si occupa di rieseguire in modo selettivo i vari test, di modulo e di alto livello esclusi quelli di validazione, elencati e descritti sopra. È quindi necessario l'esecuzione di questi test ogni qual volta che il codice viene modificato: l'inserimento di nuove funzionalità, la correzione di un bug o la modifica di parti del codice sorgente comporta l'esecuzione di tutti i test relativi ad essa. Vengono fatti questi test a fronte del fatto che l'introduzione di modifiche, migliorie e/o correzioni all'interno del codice è fortemente propenso all'inserimento di errori. I test di modulo necessari per la consegna nella repository vengono eseguiti automaticamente, mentre per i test ad alto livello i verificatori pianificheranno l'esecuzione ad ogni superamento di baseline.

3.9.2.4 Tracciamento dei test

Ogni test è strutturato nel seguente modo:

- Codice identificativo
- Descrizione
- Stato

La sintassi scelta per il codice identificativo è la seguente:

T{tipo}{codice}

dove l'attributo **tipo** può essere:

- **U**: Test di unità;
- **I**: Test di integrazione;
- **F**: Test funzionali;
- **S**: Test di sistema;
- **V**: Test di validazione;

mentre l'attributo **codice** è un numero incrementale che parte da 1 ed è associato al test di unità. Lo stato del test può essere uno dei seguenti:

- Implementato;
- Non implementato;
- Superato;
- Non superato;

3.9.3 Strumenti

- **Software**: per la stesura del codice Java, si utilizzeranno gli IDE *Intellij IDEA* e *Android Studio*. Per la stesura del codice javascript per Node.js si utilizzerà *Visual Studio Code*;
- **Documenti**: per il controllo dei documenti verranno utilizzate le funzioni integrate a TexStudio per controllare che non ci siano errori nei file LaTeX. Verrà utilizzato GlossaryHelper per la gestione del glossario e per controllare che non vi siano presenti duplicati;
- **Gestione di processi e feedback**: verrà utilizzato il sistema di issues di GitLab;
- **Gestione di tasklist**: si utilizzeranno le funzionalità di Task Management di *Asana*$_G$, un servizio cloud che crea dei *task*$_G$ ed assegna ad ogni membro un periodo temporale entro il quale svolgere il proprio operato;
- **Gestione del time tracking**: si utilizzeranno le funzionalità di Clockify (applicazione di time tracking free);
- **Analisi dinamica**: L'analisi dinamica sarà estesa a tutto il software prodotto e consiste in test di unità tramite libreria di testing *JUnit*$_G$;
- **Tracciamento dei requisti**: verrà utilizzato un tool chiamato *ProgmaDB* per tracciare i requisti installando lo strumento su un *server*$_G$ remoto condiviso.

3.10 Validazione

3.10.1 Descrizione

Il processo di *validazione*$_G$ ha come scopo quello della verifica del prodotto al fine di assicurarsi che sia conforme a quanto stabilito.

3.10.2 Procedure

L'attività di validazione verrà svolta secondo i seguenti passi:

1. **Tracciamento**: tracciamento dei test con report da parte dei verificatori.
2. **Analisi dei risultati**: in seguito al punto 1, spetterà al Responsabile di progetto l'analisi dei report e la decisione di:
   - Accettare la validazione e consegnare al proponente i risultati, informandolo sulle modalità di esecuzione della validazione;
   - Ripetere i test in toto o in parte aggiungendo indicazioni mirate al miglioramento della fase di test che deve essere nuovamente eseguita.

4 Processi organizzativi

4.1 Processi di coordinamento

4.1.1 Comunicazione

Verranno ora illustrate le norme che regolano la comunicazione sia tra i membri di *duckware* che con entità esterne quali i proponenti ed i committenti.

4.1.1.1 Comunicazioni interne

Le comunicazioni interne di *duckware* avverranno attraverso Slack, il quale consente la creazione di appositi canali tematici in ciascuno dei quali verrà trattato il relativo argomento. In particolare, i canali saranno così suddivisi:

- **Generale:** Verrà discusso di argomenti generali riguardanti l'organizzazione del progetto, la scelta di strumenti di lavoro o per una rapida comunicazione. Verranno anche stabiliti gli argomenti di discussione durante gli incontri;
- **Incontri:** In questo canale verranno scritte dal Responsabile tutti gli argomenti da toccare durante la successiva sessione d'incontro nel programma. Inoltre, verrà stilato un breve riassunto di quanto deciso in merito ai punti da discutere;
- **GitLab monitor:** Grazie all'utilizzo di un bot, ogni modifica fatta da un utente all'interno della repository verrà segnalato all'interno di questo canale;
- **Link utili:** In questo canale non sarà possibile scrivere messaggi ma si potranno solamente visualizzare dei collegamenti a risorse interne che potranno essere utili durante lo sviluppo del progetto;
- **Glossario:** Ogni volta vi sia bisogno di un aggiornamento del glossario verranno qui notificati i termini con le relative definizioni da inserire. Una volta aggiornato il glossario, grazie a GlossaryHelper, i messaggi verranno rimossi per evitare confusione.
- **Sviluppo:** canale dedicato a Java ed allo sviluppo di applicazioni Android nel quale i membri con esperienze passate nella creazione di app possono aiutare i colleghi. Vi sono dei canali, ciascuno avente il nome del documento al quale si riferiscono, che contengono un unico messaggio contente il link al file *.tex di documentazione. È stato creato inoltre un sotto-dominio all'interno del server di uno dei membri di *duckware* per fare il backup di documentazione e file.

4.1.1.2 Comunicazioni esterne

In questa sottosezione vi sono le norme che regolano le comunicazioni con soggetti esterni a *duckware*, come:

- La proponente zero12 rappresentata da Dindo Stefano, con i quali si intende stabilire un rapporto di collaborazione per la definizione dei requisiti necessari al fine di realizzare il prodotto;

- Prof Tullio Vardanega e Prof Riccardo Cardin ai quali verrà fornita la documentazione richiesta in ciascuna revisione di progetto. Con essi si intenderà dialogare per portare un continuo miglioramento al progetto.

Tutte le comunicazioni esterne vengono effettuate attraverso

duckware.swe@gmail.com

e ogni membro del gruppo ha accesso a tale indirizzo email. I proponenti verranno raggiunti attraverso gli indirizzi da loro forniti:

s.dindo@zero12.it

4.1.2 Riunioni

In questa sotto-sezione definiremo le norme relative alle riunioni interne ed esterne. Durante lo svolgimento di ogni riunione ci sarà un responsabile dell'incontro che si prenderà cura di far rispettare tutti i punti di discussione su cui riflettere. Il rapporto finale di quanto discusso verrà poi inserito all'interno del Verbale di Riunione.

4.1.2.1 Verbale di riunione

Il responsabile di riunione redigerà il Verbale di Riunione secondo il seguente schema:

1. **Frontespizio:** Il frontespizio sarà formato dall'indicazione della tipologia del verbale, ovvero Interno o Esterno, e dalla data nella quale tale incontro è avvenuto;
2. **Registro modifiche:** Nell'ultima pagina sarà presente un registro delle modifiche relativo a tutti i cambiamenti apportati al verbale prima di giungere alla sua versione finale;
3. **Informazione sulla riunione:** In questa sezione ci saranno:
   - Lo scopo della riunione: il motivo per il quale si è deciso di fare una riunione;
   - Data e luogo: quando e dove la riunione si è tenuta;
   - Durata: include l'orario di inizio e l'orario di fine della riunione, in formato ventiquattro ore, ad esempio 17.40;
   - Partecipanti alla riunione: nel caso di riunione esterna, verranno prima elencati i committenti e proponenti, altrimenti verrà stilata una lista di partecipanti.
4. **Ordine del giorno:** Elenco puntato degli argomenti da discutere;
5. **Resoconto:** Riassunto redatto dal responsabile dell'incontro secondo quanto stabilito dai punti dell'ordine del giorno. Dovrà essere inserito in grassetto il titolo di ogni punto trattato e poi su una nuova riga la descrizione di quanto è stato deciso.

Ogni file di un verbale verrà chiamato nel seguente modo: verbale_DATA dove DATA è la data nel quale è stato svolto l'incontro. Ad esempio un titolo valido potrebbe essere verbale_12-12-2018.

4.1.2.2 Riunioni interne

La partecipazione alle riunioni interne è rivolta solo ed esclusivamente ai membri di *duckware* ed il Responsabile di progetto ha il compito di stilare l'ordine del giorno. Gli incontri avverranno in data e ora non predefinita, stabilita in maniera collettiva in base alla disponibilità della maggioranza del gruppo. I partecipanti dovranno essere puntuali alle riunioni e comunicare con quanto più anticipo, se possibile, eventuali ritardi o problemi di presenza. Una riunione può avvenire solo se ci sono almeno 5 partecipanti su 7.

4.1.2.3 Riunioni esterne

Le riunioni esterne coinvolgono sia la Proponente che i membri di *duckware*; all'interno del verbale esterno relativo all'incontro verranno inseriti tutti gli argomenti trattati. Le riunioni verranno effettuate sede nella della proponente oppure tramite chiamata di gruppo su programmi di video conferenza come Skype. Quando possibile, gli incontri avverranno presso Torre Tullio Levi-Civita.

4.2 Processi di pianificazione

4.2.1 Ruoli di progetto

La realizzazione del progetto è il prodotto finale di una collaborazione coesa ed organizzata da parte dei membri del gruppo. Ogni membro avrà il suo ruolo all'interno di *duckware* e ciò rifletterà le rispettive figure che ci sono all'interno di una azienda. A rotazione ogni membro del gruppo ricoprirà questi ruoli

- Responsabile di progetto;
- Amministratore;
- Analista;
- Progettista;
- Programmatore;
- Verificatore.

Il cambio del ruolo avverrà in modo tale da poter garantire un costante livello di qualità. Il responsabile avrà l'onere di assegnare i ruoli in modo ragionato al fine di non creare situazioni ambigue. Ad esempio, non sarà possibile che un verificatore debba verificare dei documenti che sono stati creati dallo stesso in quanto non ci sarebbe una sufficiente analisi critica.

4.2.1.1 Responsabile di progetto

Il Responsabile di progetto, o Project manager, partecipa al progetto per tutta la sua durata e su di esso ricadono tutte le responsabilità di scelta ed approvazione. Inoltre è colui che rappresenta *duckware* nei confronti della Proponente e dei committente. Egli dovrà:

- Approvare i documenti;
- Elaborare piani, scadenze e coordinare le attività del gruppo;
- Relazionarsi con il controllo di qualità del progetto;
- Approvare l'offerta.

4.2.1.2 Amministratore

L'amministratore la figura è una figura fondamentale del gruppo poiché deve garantire l'efficienza e l'attività del gruppo stesso. Esso dovrà occuparsi dell'organizzazione dell'ambiente di lavoro redigendo i documenti che regolano le attività di verifica e di lavoro. Infatti le sue mansioni sono:

- Redigere le Norme di Progetto e aiutare nella scrittura del Piano di Progetto;
- Assicurarsi che la documentazione sia corretta ed approvata;
- È responsabile della redazione di piani e procedure relativi alla Gestione per la Qualità.

4.2.1.3 Analista

L'attività degli analisti è necessaria affinché il progetto possa essere realizzato; essi dovranno analizzare il problema e comprenderlo a pieno, al fine di ridurre la probabilità di realizzare gravi errori di progettazione. Dovranno occuparsi nello specifico di:

- Realizzare lo studio di fattibilità, l'analisi dei requisiti e verificare le implicazioni di costi e qualità;
- Studiare e definire al meglio, senza ambiguità, il problema da risolvere per capire cosa deve essere realizzato in base ai requisiti richiesti;
- Modellare il problema ed effettuare la ripartizione dei requisiti.

4.2.1.4 Progettista

Il progettista è il responsabile delle attività di progettazione, le quali consistono nella definizione di una soluzione accettabile per ogni stakeholder coinvolto. I suoi obiettivi sono:

- Definizione della struttura logica del prodotto in modo che sia ottimizzata e quindi facile da mantenere;
- Suddivisione del sistema in problemi di complessità ridotta e trattabili al fine di rendere il lavoro di codifica più facile da realizzare per i programmatori e più facile da verificare per i verificatori.

4.2.1.5 Programmatore

Il programmatore è responsabile del processo di codifica che porta alla realizzazione del prodotto. Il suo compito è quello di implementare attraverso specifici linguaggi quanto è stato definito dal Progettista nell'architettura del progetto. Esso dovrà:

- Scrivere codice propriamente documentato, indentato e versionato;
- Scrivere il manuale utente in modo che contenga il minor numero possibile di ambiguità;
- Creare le componenti per la verifica e la validazione del codice.

4.2.1.6 Verificatore

Il verificatore deve essere presente durante tutta la durata del progetto in quanto si occuperà delle attività di verifica. Esse sono:

- Redazione del piano di qualifica. Al suo interno saranno presenti tutte le prove effettuate ed i risultati ottenuti;
- Accertamento che le attività di processo non abbiano generato degli errori.

4.2.2 Cambio di ruoli

Ciascun membro di *duckware* ricoprirà ognuno dei ruoli sopra riportati e la rotazione avverrà secondo le seguenti regole:

- Sarà obbligatorio tenere in considerazione gli interessi e gli impegni del singolo al fine di poter garantire un'esecuzione ottima;
- Ogni membro non potrà mai effettuare la verifica di un'opera svolta dallo stesso poiché potrebbero sorgere dei conflitti di interesse;
- Il trasferimento di ruolo dovrà avvenire in modo ottimale e per tali motivi sarà necessario redigere un breve documento informale all'interno di *duckware* nel quale ci saranno commenti da parte del precedente assegnatario di quel ruolo.

4.2.3 Stesura del consuntivo

La stesura del consuntivo può essere effettuata solamente dal Responsabile di progetto della fase corrente. Le operazioni che questo deve eseguire sono riportate di seguito:

1. Utilizzare *Instagantt*$_G$ per esportare da Asana un documento di estensione **.xlsx* (compatibile con Microsoft Office Excel) contenente le ore rendicontate nella fase corrente;
2. Scaricare da GitLab il template del foglio di calcolo predisposto;

3. Inserire le ore rendicontate nelle celle ottenendo la differenza di ora tra il preventivo e le ore rendicontate;
4. Aggiungere al *Piano di progetto* una sezione che mostra i risultati ottenuti al punto precedente;
5. Creare dunque una tabella nella sezione precedentemente aggiunta che mostra la differenza tra le ore preventivate e quelle rendicontate;

Per la revisione RR sono stati effettuati solamente i primi tre passaggi in quanto non era possibile effettuare differenze poichè mancavano le ore rendicontate con cui confrontarsi. Gli strumenti utilizzati sono dunque **Instagantt**, tool gratuito per Asana per i *diagrammi di Gantt*$_G$, ed *Excel*$_G$, prodotto della suite Office di Microsoft per i fogli di calcolo.

4.3 Formazione

4.3.1 Formazione dei membri del gruppo

Tutti i membri di *duckware* devono studiare autonomamente le tecnologie che verranno utilizzate durante il processo di creazione del progetto. Verranno realizzate delle guide informali per uso interno, preferibilmente dai membri con maggiore esperienza nel relativo campo, per facilitare la formazione dei membri di *duckware* che presentano lacune.

4.3.2 Guide e materiale usato

La documentazione di riferimento comprende quanto indicato nella seguente lista più tutto ciò che è già stato menzionato all'interno della sottosezione Riferimenti Informativi:

- **GitLab:** https://gitlab.com/help
- **LaTeX:** https://www.latex-project.org
- **Java:** https://docs.oracle.com/javase/10/
- **Android:** https://developer.android.com/guide
- **Node.js:** https://nodejs.org/it/docs/

A Ciclo di Deming (PDCA)

Ogni processo deve essere organizzato sulla base del Ciclo di Deming, un metodo di gestione iterativo, composto da quattro fasi, utilizzato per il controllo e il miglioramento continuo dei processi e dei prodotti. Le quattro fasi previste dal ciclo di Deming sono le seguenti:

- **P** - Plan: si stabiliscono gli obiettivi e i processi necessari per fornire i risultati attesi accordati, si assegnano responsabilità, si analizzano cause di criticità e si definiscono azioni di correzione;
- **D** - Do: si attua il piano implementando le attività secondo le linee definite e si raccolgono dati per la creazione di grafici e analisi da destinare alla fase di "Check" e "Act";
- **C** - Check: studio e raccolta dei risultati e dei riscontri. Viene verificato inoltre l'esito delle azioni di miglioramento;
- **A** - Act: vengono applicate le correzioni necessarie per soddisfare le carenze rilevate e standardizzate le attività.

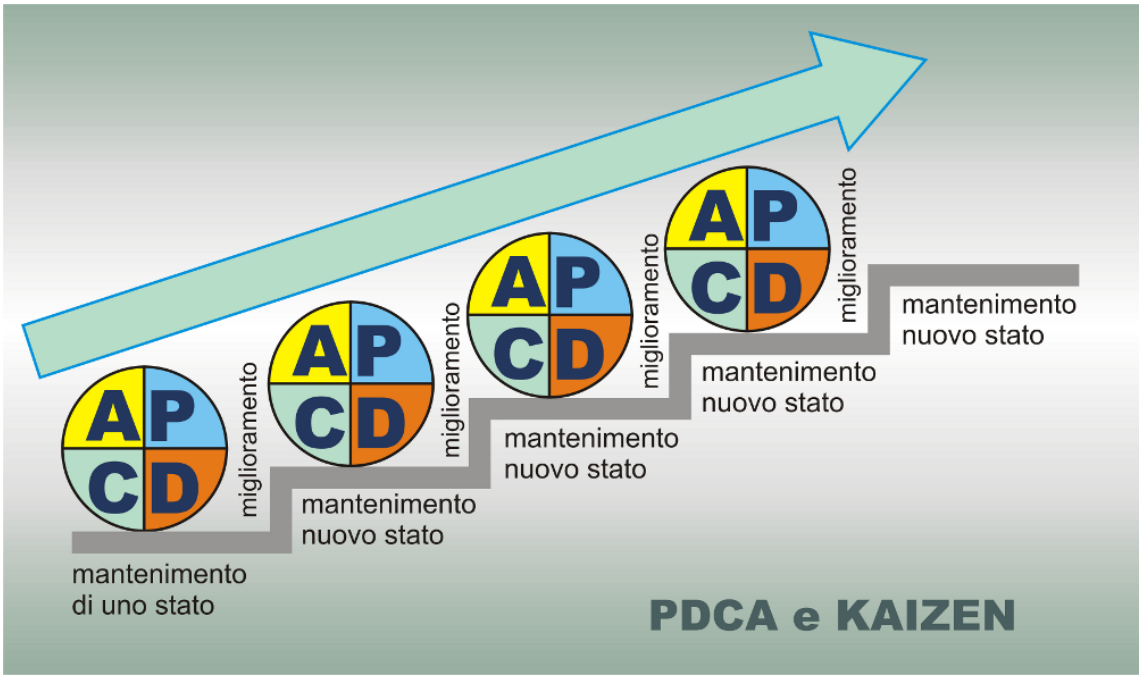

Figura 8: Ciclo di Deming PDCA

Il metodo PDCA può essere utilizzato inoltre per perseguire il miglioramento continuo, attuando tre cicli evolutivi:

- **Ciclo di mantenimento**: alla base delle fasi di *PLAN* e *DO*, ha la finalità di verificare se quanto pianificato ed attuato continua a dare i risultati attesi. Se il risultato della fase di *CHECK* risulta essere:
  - positivo, la fase *ACT* mantiene lo stato attuale;

  - negativo, allora sarà necessario riattivare il ciclo di azione correttiva.

- **Ciclo di azione correttiva**: qualora l'esito della fase di *CHECK* sia negativo si attua il ciclo di azione correttiva per modificare la situazione del processo. Tale ciclo è caratterizzato da due componenti:

  - rimedio, azione immediata finalizzata a correggere;
  - prevenzione, azione pianificata finalizzata a rimuovere le cause.

  Quando il *CHECK* ritorna ad essere positivo, si attiva nuovamente il ciclo di mantenimento.

- **Ciclo di miglioramento**: tale ciclo presuppone una corretta attuazione dei primi due cicli. Il ciclo di miglioramento viene attuato alla necessità di cambiare, per ottenere risultati ancora migliori, sul processo attivo. Sarà quindi necessario ripartire dalla fase *PLAN* e continuare con le fasi *DO*, per poi controllare di nuovo con la fase di *CHECK*.

B ISO/IEC 15504

Nello standard ISO/IEC 15504 specifica come la qualità è strettamente collegata alla maturazione dei processi. Vengono così esposti dei livelli di maturità su cui fare riferimento; il livello più alto corrisponde ad un processo che sarà qualitativamente migliore e maturo. Lo standard contiene un modello di riferimento che definisce:

- Process dimension - dimensione del processo;
- Capability dimension - dimensione della capacità.

La process dimension comprende i seguenti processi:

- Customer/Supplier;
- Engineering;
- Supporting;
- Management;
- Organization.

La capability dimension definisce una scala di maturità a cinque livelli (più il livello base, detto "livello 0") così definiti:

- **LV 0 - Incomplete process**
  A questo livello il processo è allo stato iniziale, nelle prime fasi, pertanto non è ancora in grado di svolgere ed adempiere agli obbiettivi per cui è stato creato;
- **LV 1 - Performed process**
  A questo livello il processo è in grado e ha portato a termine i suoi obbiettivi, ma privo di controllo del lavoro efficace e dettagliato;
- **LV 2 - Managed processs**
  A questo livello vengono implementate delle prime soluzioni che permettono il controllo del processo, quindi i processi non sono solo funzionanti ma anche monitorati in ogni sua parte;
- **LV 3 - Established process**
  A questo livello il processo viene definitivamente stabilito;
- **LV 4 - Predictable process**
  A questo livello il processo viene circoscritto in determinati limiti, in modo tale da essere tenuto ancora più sotto controllo;
- **LV 5 - Optimizing process**
  A questo livello si è raggiunto uno stato ottimale, il processo viene costantemente tenuto sotto controllo e migliorato per adattarsi alle esigenze del progetto in corso.

La capacità (o maturità) dei processi è misurata tramite gli attributi definiti a livello internazionale e che sono:

- Livello 1
  - **Process Performance**: capacità di un processo di raggiungere gli obiettivi trasformando input identificabili in output identificabili.
- Livello 2
  - **Performance Management**: capacita del processo di elaborare un prodotto coerente con gli obiettivi fissati;
  - **Work Product Management**: capacità del processo di elaborare un prodotto documentato, controllato e verificato.
- Livello 3
  - **Process Definition**: l'esecuzione del processo si basa su standard di processo per raggiungere i propri obiettivi;
  - **Process Deployment**: capacità del processo di attingere a risorse tecniche e umane appropriate per essere attuato efficacemente.
- Livello 4
  - **Process Measurement**: gli obiettivi e le misure di prodotto e di processo vengono usati per garantire il raggiungimento dei traguardi definiti in supporto ai target aziendali;
  - **Process Cotrol**: il processo viene controllato tramite misure di prodotto e processo per effettuare correzioni migliorative al processo stesso.
- Livello 5
  - **Process Innovation**: i cambiamenti strutturali, di gestione e di esecuzione vengono gestiti in modo controllato per raggiungere i risultati fissati;
  - **Process Optimization**: le modifiche al processo sono identificate e implementate per garantire il miglioramento continuo nella realizzazione degli obiettivi di business dell'organizzazione.

Ciascuna attributo di processo consiste di una o più pratiche generiche che a loro volta sono elaborate in "Indicatori della pratica" che aiutano nella fase di valutazione delle prestazioni. Ciascun attributo del processo è valutato secondo una scala a quattro valori (N-P-L-F):

- Not achieved (0 - 15%);
- Partially achieved (>15% - 50%);
- Largely achieved (>50% - 85%);
- Fully achieved (>85% - 100%).

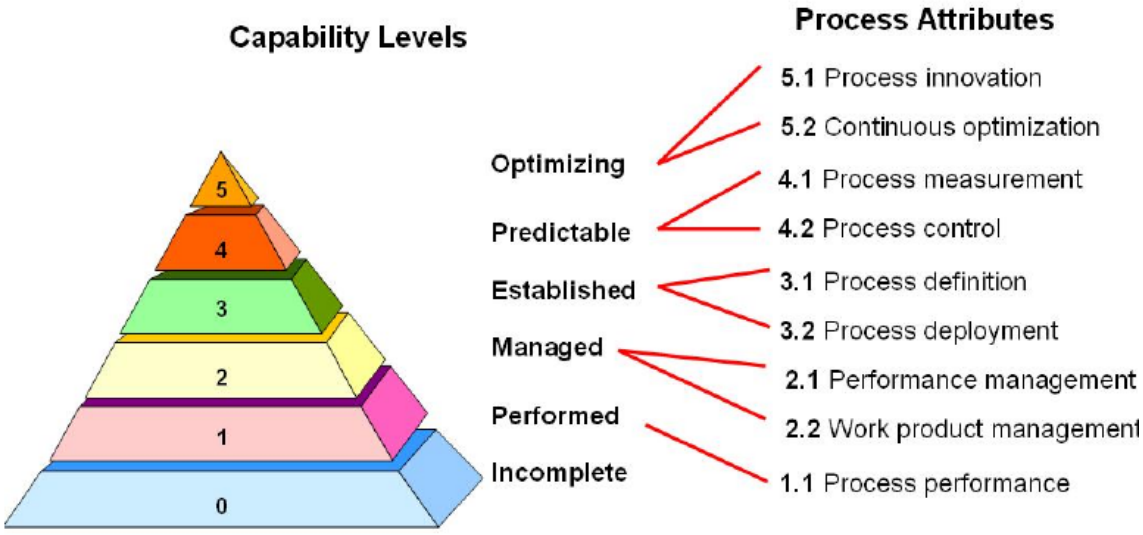

Figura 9: Gerarchia capability dimension ISO 15504