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, Pardeep Singh
Redazione	Matteo Pellanda, Luca Stocco Andrea Pavin, Alessandro Pegoraro
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 1.0.0 del
01 Dicembre 2018

Indice

Elenco delle tabelle

Elenco delle figure

Registro delle Modifiche

Ver.	Data	Autore	Ruolo	Descrizione
1.0.0	2018-12-01	Alberto Miola	Responsabile	Approvazione per rilascio documento in RR
0.3.0	2018-12-01	Alberto Miola	Responsabile	Aggiornamento dei *processi di supporto* ed adattamento del documento al nuovo template
0.2.2	2018-11-30	Matteo Pellanda	Amministratore	Stesura delle *metriche*
0.2.1	2018-11-29	Luca Stocco	Amministratore	Correzione di alcuni contenuti
0.2.0	2018-11-29	Alberto Miola	Responsabile	Aggiornamento documento
0.1.0	2018-11-28	Pardeep Singh	Verificatore	Superamento verifica per contenuto e stile
0.0.10	2018-11-28	Matteo Pellanda	Amministratore	Inserimento grafici in *Ciclo di Deming*, *ISO/IEC 15504*
0.0.9	2018-11-27	Luca Stocco	Amministratore	Correzione *Processi organizzativi*
0.0.8	2018-11-26	Pardeep Singh	Verificatore	Segnalazione errori contenuto e ortografici
0.0.7	2018-11-25	Alessandro Pegorarc	Amministratore	Stesura *ISO/IEC 15504*
0.0.6	2018-11-24	Matteo Pellanda	Amministratore	Stesura *Ciclo di Deming*
0.0.5	2018-11-24	Sonia Menon	Verificatore	Segnalazione errori contenuto e ortografici
0.0.4	2018-11-23	Luca Stocco	Amministratore	Stesura *Processi organizzativi*
0.0.3	2018-11-22	Andrea Pavin	Amministratore	Stesura *Processi di supporto*
0.0.2	2018-11-21	Matteo Pellanda	Amministratore	Stesura *Introduzione*, *Processi primari*
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 Scopo del prodotto

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

1.3 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 v1.0.0* che conterrà una lista di termini con la specifica descrizione del suo significato. Un qualsiasi termine presente nel glossario verrà indicato in questo documento scritto in corsivo con una G a pedice che seguirà la parola in questione.

1.4 Riferimenti utili

1.4.1 Riferimenti normativi

- *ISO*$_G$/IEC
  - *ISO*$_G$/IEC 15504
- Indice di Gulpease.

1.4.2 Riferimenti informativi

- Piano di progetto v 1.0.0;
- Piano di Qualifica v 1.0.0;
- Sito del corso di Ingegneria del Software.

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 Studio di fattibilità

In data *16 Novembre 2018* è avvenuta presso la Torre Archimede la presentazione dei capitolati d'appalto da parte di alcune aziende interessate nel coinvolgimento degli studenti in un progetto. Il giorno *20 Novembre 2018* è stata organizzata una riunione del gruppo per discutere di vari argomenti, primo fra tutti la scelta dei capitolati più interessanti da analizzare. Dopo aver scelto il capitolato per il quale proporsi è stata eseguita un'ulteriore analisi relativa ai rischi ed alle opportunità che è stata riportata all'interno dello *Studio di fattibilità 1.0.0*. Si tratta di un documento nel quale viene indicato, per ogni capitolato, le motivazioni per le quali il gruppo **duckware** vorrebbe proporsi come fornitore. In particolare, sono presenti per ogni capitolato:

- Una descrizione del prodotto proposto dall'azienda secondo le regole del capitolato *d'appalto*;
- Una descrizione di quello che sarà l'ambito d'uso del prodotto che ci si propone di creare;
- Le tecnologie richieste dal capitolato per la realizzazione del relativo progetto;
- Le motivazioni fornite dal team *duckware* riguardo l'accettazione o la respinta del capitolato in questione.

2.2 Sviluppo

2.2.1 Analisi dei Requisiti

L'analisi dei requisiti ha lo scopo di individuare i *requisiti*$_G$ 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 1.0.0* 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*$_G$ concordati col cliente;
- Fornire degli esempi di dialoghi tra l'utente e l'assistente vocale, riferiti alle funzionalità che necessitano di un iterazione 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 di *requisiti*$_G$

Ogni *requisito*$_G$ 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*$_G$ che può essere 0 (opzionale), 1 (*requisito*$_G$ 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*$_G$ funzionale, Q se di qualità, V se di vincolo e P se presentazionale.

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 è specificato da una serie di cifre spaziate da un punto. La prima cifra indica il numero del padre, la seconda invece il numero del figlio. Un trattino separera 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 iterazione vocale tra l'utente e l'assistente *Alexa*$_G$;
- **Estensioni**: contiene una lista degli eventuali errori che il *caso d'uso*$_G$ 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. In particolare la logica definita dovrà:

- Soddisfare i *requisiti*$_G$ dell'*Analisi dei requisiti* ed adattarsi ai cambiamenti od alle evoluzioni che potenzialmente avverranno nel tempo;
- Essere comprensibile e modulare;
- Avere la capacità di rispettare le specifiche nel tempo, anche nel caso di malfunzionamenti;
- Ridurre al minimo i tempi di manutenzione.

2.3.2 Diagrammi

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

2.3.3 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 al linguaggio *Java*$_G$.

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 : nome file
Version : versione file
Type : tipo file
Date : data di creazione
Author : nome autore/i
E-mail : email autore/i

License : tipo licenza

Advice : lista avvertenze e limitazioni

Changelog :
Autore || Data || breve descrizione delle modifiche
```

Figura 1: Esempio convenzioni per la documentazione

- 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*$_G$ o una modifica rilevante che verrà sottoposta alla fase di test

2.4.1 *Java*$_G$

In questa sezione vengono indicate delle convenzioni adottate dal team di Google 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 2: 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 Identazione Il corpo di ogni funzione dovrà essere separato dall'inizio della riga con una tabulazione, lasciando le impostazioni standard fornite dall'*IDE*$_G$. 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 3: Esempio identazione

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 4: 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 5: Esempio nomi variabili

La modalità di scrittura delle variabili appena descritto è conosciuto come *lowerCamelCase*$_G$. 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*$_G$).

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 6: 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*$_G$. 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() {
	  //...
	}
	
	voide TestMethod() {
	  //...
	}

Tabella 7: 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*$_G$. Tipocamente 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 8: 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*$_G$.

OK	NO

public Test implements Runnable {
 	
  @Override
  public void run() {
  //...
  }
}
	public Test implements Runnable {
	
	  public void run() {
	  //...
	  }
	}

Tabella 9: Esempio regole pratiche

2. Nel caso di eccezioni da cattuare, 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 10: 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
   ```

3 Processi di supporto

3.1 Documentazione

3.1.1 Descrizione

Verranno qui discusse le scelte intraprese per la scrittura, *verifica*$_G$ 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*$_G$. 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*$_G$ 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*$_G$ 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*$_G$ 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*$_G$ di progetto.

3.1.5 Documenti correnti

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

ESTERNO

- **[AR] - 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*$_G$ 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*$_G$.

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_G 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 $software_G$, 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*$_G$ 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*$_G$. Questi *software*$_G$ rendono la scrittura *LaTeX*$_G$ 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;
- *TexMaker*$_G$.

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*$_G$ 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*$_G$

Per garantire la qualità del lavoro, gli Amministratori hanno suddiviso il lavoro in vari *processi*$_G$ che sono stati poi riportati nel *Piano di Qualifica*. Dovrà essere rispettata la notazione **PRO[value]** dove **value** indica il codice univoco del *processo*$_G$ tramite un numero intero 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][num]** dove:

- **categ**: va ad indicare la categoria della metrica ed assume i seguenti valori:
  - **PRC**: per indicare i *processi*$_G$;
  - **PRD**: per indicare i prodotti.
- **num**: va ad indicare il codice univoco della metrica come numero intero a tre cifre a partire da 1. Esempio **MPRC001**.

3.3 Configurazione

3.3.1 Classificazione delle metriche

3.3.1.1 Descrizione Per le parti versionabili del progetto e per i documenti ufficiali si è scelto l'utilizzo di *GitLab*$_G$, 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*$_G$ 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*$_G$ 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*.

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.

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*$_G$ che assume il nome del documento e la sua versione attuale.

3.3.1.3 Ciclo di vita dei *branch*$_G$ Per migliorare l'*efficienza*$_G$ e l'*efficacia*$_G$ di sviluppo, sono stati creati dei *branch*$_G$, denominati con il nome del rispettivo documento in redazione. Inoltre, è presente un *branch*$_G$ di lavoro, chiamato `develop` nel quale confluiranno tutti i *branch*$_G$, 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*$_G$ `master`.

3.3.1.4 Aggiornamento della *repository*$_G$

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

- "git status": per verificare in che *branch*$_G$ 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*$_G$ indicato;
- "git pull": permette di aggiornare la *repository*$_G$ 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*$_G$;
- "git commit": permette di salvare le modifiche aggiunte in *area di staging*$_G$ al *repository*$_G$ locale;
- "git push": sincronizza il *repository*$_G$ remoto con quello locale.

3.3.2 Strumenti

- Server git: è stato utilizzato *GitLab*$_G$ 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*$_G$

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

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

3.5 Metriche qualità di *processo*$_G$

Verranno utilizzate le seguenti metriche per valutare l'efficienza e l'efficacia dei *processi*$_G$.

3.5.1 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*$_G$, o un valore negativo, indice di un rallentamento.

3.5.2 Budget Variance

Questa formula mostrare 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.

3.6 Metriche qualità di prodotto

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

3.6.1 Indice di Gulpease

L'Indice di Gulpease 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.

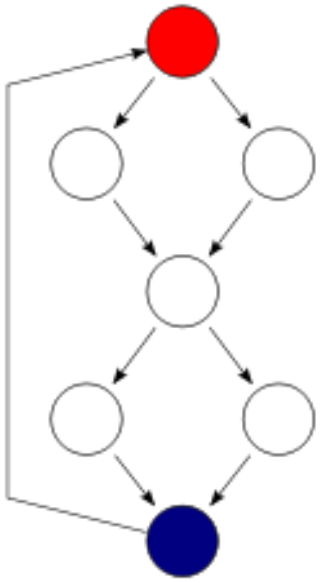
3.6.2 Errori ortografici

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

3.7 Metriche per il *software*$_G$

3.7.1 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.



Tale indice può essere applicato a metodi, moduli e *packages*$_G$ 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].

3.7.2 Numero di metodi

Questa metrica calcola la media di occorrenze di metodi per ciascun *package*$_G$ 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*$_G$ in questione.

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

3.7.3 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* 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.

3.7.4 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].

3.7.5 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.

3.8 Verifica

3.8.1 Descrizione

In questa sezione verranno descritti gli strumenti ed i metodi impiegati per la *verifica*$_G$ 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*$_G$ 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*$_G$

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

3.8.5 Strumenti

- **Software**: per la stesura del codice *Java*$_G$, si utilizzeranno gli *IDE*$_G$ *Intellij IDEA* e *Android Studio*;
- **Documenti**: per il controllo dei documenti verranno utilizzate le funzioni integrate a *TexStudio*$_G$ per controllare che non ci siano errori nei file *LaTeX*$_G$. Verrà utilizzato GlossaryHelper per la gestione del glossario e per controllare che non vi siano presenti duplicati;
- **Gestione di *processi*$_G$ e *feedback*$_G$**: verrà utilizzato il sistema di issues di *GitLab*$_G$;
- **Gestione di tasklist**: si utilizzeranno le funzionalità di Task Management di *Asana*$_G$;
- **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*$_G$ prodotto e consiste in test di unità tramite libreria di testing *JUnit*$_G$;
- **Tracciamento dei requisti**: verrà utilizzato lo un tool chiamato *ProgmaDB* per tracciare i requisti installando lo strumento su un *server*$_G$ remoto condiviso.

3.9 Validazione

3.9.1 Descrizione

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

3.9.2 Procedure

L'attività di *validazione*$_G$ 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*$_G$ e consegnare al proponente i risultati, informandolo sulle modalità di esecuzione della *validazione*$_G$;
   - 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*$_G$, 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*$_G$ monitor:** Grazie all'utilizzo di un *bot*$_G$, ogni modifica fatta da un utente all'interno della *repository*$_G$ 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. 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*$_G$ 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*$_G$ 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*$_G$ 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*$_G$ 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*$_G$ 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 ogni giovedì a partire dalle ore 13.00. 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*$_G$ 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*$_G$ 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*$_G$ 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*$_G$ richiesti;
- Modellare il problema ed effettuare la ripartizione dei *requisiti*$_G$.

4.2.1.4 Progettista Il progettista è il *responsabile*$_G$ delle attività di progettazione, le quali consistono nella definizione di una soluzione accettabile per ogni *stakeholder*$_G$ 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*$_G$ del *processo*$_G$ 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*$_G$ 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*$_G$ e la *validazione*$_G$ 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*$_G$. 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*$_G$ 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*$_G$ 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.

4.3 Strumenti

4.3.1 Pianificazione

Per gestire al meglio la pianificazione del progetto, *duckware* ha deciso di usare *Asana*$_G$. Si tratta di un servizio cloud che crea dei *task*$_G$ ed assegna ad ogni membro un periodo temporale entro il quale svolgere il proprio operato. In questo modo sarà possibile controllare lo stato di avanzamento dei vari *task*$_G$ se essi siano stati completati o se ancora in fase di elaborazione.

4.3.2 Comunicazione

La comunicazione tra i membri di *duckware* avviene tramite l'utilizzo di *Slack*$_G$. Questa è un'applicazione di messaggistica apposita per i gruppi di lavoro che supporta anche l'utilizzo di *bot*$_G$ che possono interagire con sistemi di controllo di versione come *Git*$_G$.

4.3.3 Diagrammi di *Gantt*$_G$

I diagrammi di *Gantt*$_G$ sono stati creati con *Instagantt*$_G$ in quanto quest'ultimo è gratuito e integrabile con *Asana*$_G$.

4.3.4 Calcolo del consuntivo

Gli strumenti utilizzati dal Responsabile di progetto sono stati Microsoft Office *Excel*$_G$ ed *Instagantt*$_G$ per *Asana*$_G$.

4.4 Formazione

4.4.1 Formazione dei membri del gruppo

Tutti i membri di *duckware* devono studiare autonomamente le tecnologie che verranno utilizzate durante il *processo*$_G$ 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.

4.4.2 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***$_G$**:** https://gitlab.com/help
- ***LaTeX***$_G$**:** https://www.latex-project.org

A Ciclo di Deming ($PDCA_G$)

Ogni *processo*$_G$ deve essere organizzato sulla base del metodo di gestione iterativo, composto da quattro fasi, utilizzato per il controllo e il miglioramento continuo dei *processi*$_G$ e dei prodotti.

- **P** - Plan: si stabiliscono gli obiettivi e i *processi*$_G$ 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.

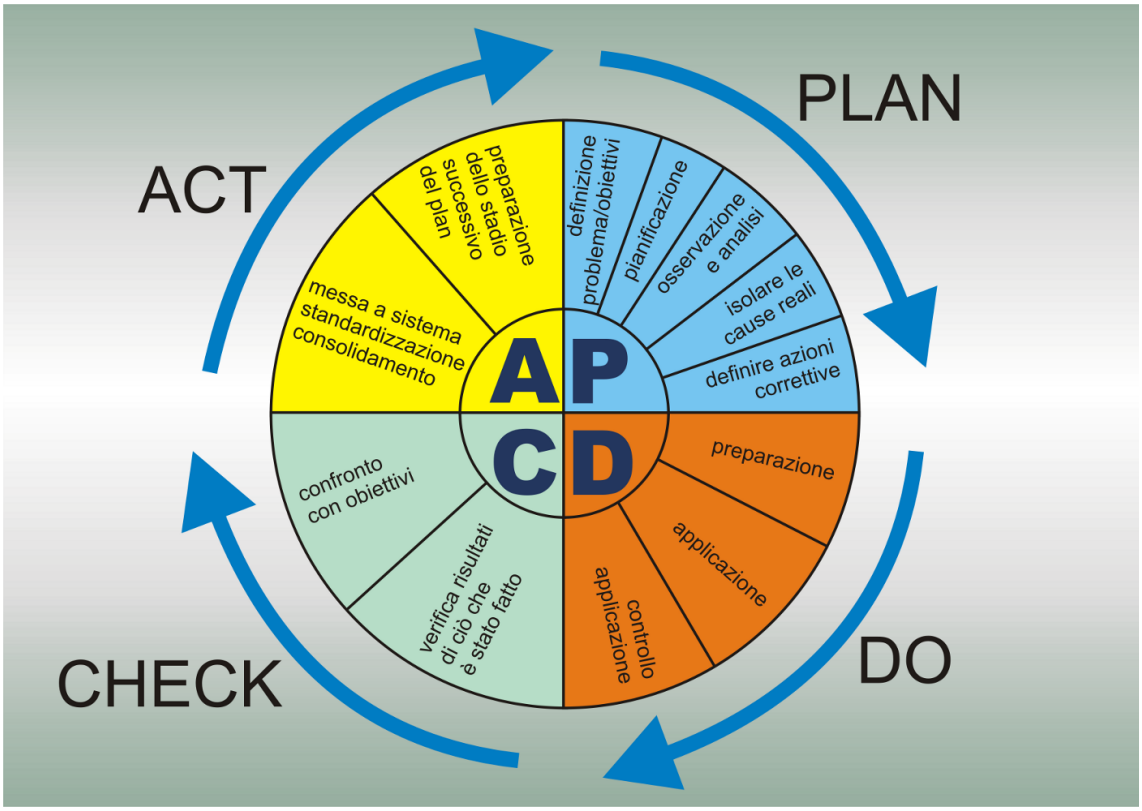


Figura 2: Ciclo di Deming $PDCA_G$

B ISO_G/IEC 15504

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

- Process dimension - dimensione del *processo*$_G$;
- Capability dimension - dimensione della capacità.

La process dimension comprende i seguenti *processi*$_G$:

- 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*$_G$ 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*$_G$ 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*$_G$, quindi i *processi*$_G$ non sono solo funzionanti ma anche monitorati in ogni sua parte;
- **LV 3 - Established process**
  A questo livello il *processo*$_G$ viene definitivamente stabilito;
- **LV 4 - Predictable process**
  A questo livello il *processo*$_G$ 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*$_G$ viene costantemente tenuto sotto controllo e migliorato per adattarsi alle esigenze del progetto in corso.

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

- Livello 1
  - **Process Performance**: capacità di un *processo*$_G$ di raggiungere gli obiettivi trasformando input identificabili in output identificabili.
- Livello 2
  - **Performance Management**: capacita del *processo*$_G$ di elaborare un prodotto coerente con gli obiettivi fissati;
  - **Work Product Management**: capacità del *processo*$_G$ di elaborare un prodotto documentato, controllato e verificato.
- Livello 3
  - **Process Definition**: l'esecuzione del *processo*$_G$ si basa su standard di *processo*$_G$ per raggiungere i propri obiettivi;
  - **Process Deployment**: capacità del *processo*$_G$ 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*$_G$ vengono usati per garantire il raggiungimento dei traguardi definiti in supporto ai target aziendali;
  - **Process Cotrol**: il *processo*$_G$ viene controllato tramite misure di prodotto e *processo*$_G$ per effettuare correzioni migliorative al *processo*$_G$ 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*$_G$ sono identificate e implementate per garantire il miglioramento continuo nella realizzazione degli obiettivi di business dell'organizzazione.

Ciascuna attributo di *processo*$_G$ 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*$_G$ 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%).

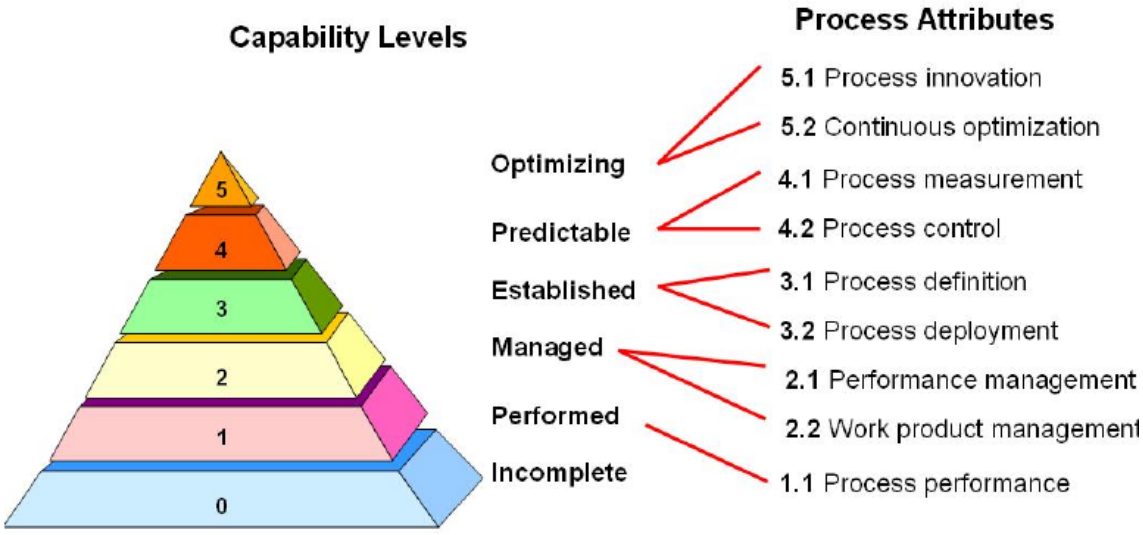


Figura 3: Gerarchia capability dimension ISO_G 15504