HEXADEC

Norme di Progetto

Progetto MegAlexa

hexadec.swe@gmail.com

Informazioni sul documento

Versione	4.0.0
Responsabile	Andrea Chinello
Redattori	Daniele Scialabba, Davide Tognon, Valentin Grigoras
Verificatori	Francesco Barbanti, Giacomo Corrò, Sukhjinder Singh
Uso	Interno
Destinatari	Prof. Tullio Vardanega, Prof. Riccardo Cardin, HexaDec

Registro delle Modifiche

Versione	Data	Autore	Ruolo	Descrizione
4.0.0	2019-06-26	Andrea Chinello	Responsabile	Approvazione documento per rilascio
3.1.0	2019-06-25	Sukhjinder Singh	Verificatore	Verifica documento
3.0.3	2019-06-23	Daniele Scialabba	Redattore	Aggiunta descrizione in §3.1
3.0.2	2019-06-22	Daniele Scialabba	Redattore	Ristrutturata §3
3.0.1	2019-06-21	Davide Tognon	Redattore	Spostate le sigle in §3.1.6.4 nel Glossario
3.0.0	2019-06-02	Sukhjinder Singh	Responsabile	Approvazione documento per rilascio
2.2.0	2019-06-02	Francesco Barbanti	Verificatore	Verifica documento
2.1.1	2019-06-01	Daniele Scialabba	Redattore	Ampliate descrizione frecce in §A.1
2.1.0	2019-06-01	Andrea Chinello	Verificatore	Verifica documento
2.0.3	2019-05-30	Daniele Scialabba	Redattore	Corretto §2.2 secondo indicazioni professore
2.0.2	2019-05-29	Daniele Scialabba	Redattore	Corretti §3.4.4, §2.1 secondo indicazioni professore
2.0.1	2019-05-28	Daniele Scialabba	Redattore	Spostati UML in §A secondo indicazioni professore

Versione	Data	Autore	Ruolo	Descrizione
2.0.0	2019-05-07	Valentin Grigoras	Responsabile	Approvazione documento per rilascio
1.1.0	2019-05-06	Giacomo Corrò	Verificatore	Verifica documento
1.0.4	2019-05-04	Sukhjinder Singh	Redattore	Aggiunte in §A.2 e §A.3
1.0.3	2019-05-04	Francesco Barbanti	Redattore	Modifiche in §2
1.0.2	2019-05-03	Sukhjinder Singh	Redattore	Aggiunte in §3
1.0.1	2019-05-03	Francesco Barbanti	Redattore	Correzione sezioni sbagliate secondo indicazione professore
1.0.0	2019-03-28	Francesco Barbanti	Responsabile	Approvazione documento per rilascio
0.4.0	2019-03-27	Daniele Scialabba	Verificatore	Verifica documento
0.3.1	2019-03-26	Giacomo Corrò	Redattore	Modifica in §A.2
0.3.0	2019-03-26	Valentin Grigoras	Verificatore	Verifica documento
0.2.3	2019-03-21	Andrea Chinello	Redattore	Aggiunta in §2.2.9
0.2.2	2019-03-19	Sukhjinder Singh	Redattore	Stesura di §A
0.2.1	2019-03-17	Sukhjinder Singh	Redattore	Stesura di §2.2

Versione	Data	Autore	Ruolo	Descrizione
0.2.0	2019-03-13	Davide Tognon	Verificatore	Verifica documento
0.1.3	2019-03-13	Giacomo Corrò	Redattore	Aggiunta in §4.6
0.1.2	2019-03-13	Giacomo Corrò	Redattore	Modifica in §4.2.2 e §4.4
0.1.1	2019-03-13	Andrea Chinello	Redattore	Modifica in §3.1.8
0.1.0	2019-03-12	Davide Tognon	Verificatore	Verifica documento
0.0.5	2019-03-11	Andrea Chinello	Redattore	Stesura di §3
0.0.4	2019-03-10	Giacomo Corrò	Redattore	Stesura di §4
0.0.3	2019-03-09	Andrea Chinello	Redattore	Stesura di §2.1
0.0.2	2019-03-08	Andrea Chinello	Redattore	Stesura di §1
0.0.1	2019-03-04	Francesco Barbanti	Responsabile	Creazione del template

Contenuti

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è fornire una linea guida riguardo il metodo di lavoro che il gruppo deve seguire per l'intera durata del progetto. Il documento contiene un insieme di regole per la gestione di processi, attività e task fatti su misura per il progetto. In questo modo ogni componente del gruppo dovrà seguire un metodo di lavoro comune facilitando il coordinamento ed evitando disomogeneità all'interno del progetto.
Questo documento sarà soggetto a modifiche.

1.2 Scopo del prodotto

La principale richiesta del progetto è rendere disponibile, all'utente registrato all'applicazione, delle micro-funzioni chiamate *connettori*G. Tali connettori sono inseriti all'interno di un *workflow*G svolto da un controllo vocale.
Il progetto utilizzerà l'infrastruttura *Amazon Web Services*G, in particolare le tecnologie *Lambda*G, *API Gateway*G,*DynamoDB*G. La comunicazione dei risultati all'utente avviene tramite l'assistente vocale *Amazon Alexa*G, l'applicazione mobile o web. L'applicativo dovrà essere multilingua rendendo disponibile la lingua italiana e quella inglese. Inoltre la creazione di routine dovrà essere univoca per ogni utente, ovvero se l'utente A crea una routine che si chiama "Buongiorno", anche l'utente B può creare una routine che si chiama "Buongiorno", ma le due routine potrebbero avere workflow diversi.

1.3 Glossario

All'interno dei documenti sono presenti termini che possono presentare significati ambigui. Per tale motivo è stato creato il documento *Glossario v4.0.0*D che conterrà tali termini affiancati dal loro significato. Le parole che sono presenti nel *Glossario v4.0.0*D presentano una "G" a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato C4 - MegAlexa**
  https://www.math.unipd.it/~tullio/IS-1/2018/Progetto/C4.pdf
- **VER-E-2019-03-22** ovvero il verbale tenuto il 2019-03-22 tramite conference call con il programma Google Hangouts.

1.4.2 Riferimenti informativi

- **Standard ISO 12207**
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
- **Documentazione Amazon API Gateway**
  https://aws.amazon.com/it/api-gateway/
- **Documentazione AWS Lambda**
  https://aws.amazon.com/it/lambda/
- **Documentazione Amazon DynamoDB**
  https://aws.amazon.com/it/dynamodb/

- **Documentazione *Kotlin*** [G]
  https://kotlinlang.org/docs/reference/

2 Processi Primari

2.1 Processi di fornitura

2.1.1 Scopo

Lo scopo del processo di fornitura è di determinare le procedure e le risorse necessarie allo svolgimento del progetto.Una volta comprese le richieste del proponente e aver stilato uno *Studio di Fattibilità*, il processo può essere avviato con fine di soddisfare ognuna di queste richieste. Inoltre si deve stipulare e concordare con il proponente un contratto per la consegna del prodotto. Si passa dunque a determinare le procedure, le risorse necessarie, e si sviluppa un *Piano di Progetto* che getterà le basi da perseguire fino alla consegna del materiale prodotto.

2.1.2 Descrizione

In questa sezione, vengono trattate le norme che i componenti del tgruppo HexaDec devono rispettare al fine di proporsi e diventare fornitori nei confronti dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin e della proponente zero12 nell'ambito dello sviluppo, progettazione e consegna del prodotto *MegAlexa*.

2.1.3 Ricerca delle tecnologie

Consiste nella ricerca di informazioni riguardante il software, *librerie*G , *framework*G e qualsiasi tecnologia che si ritiene utile per una o più fasi di sviluppo del progetto. Arrivati a delle soluzioni, vengono discusse con tutto il team e la scelta avviene sulla base delle informazioni reperite o sulle esperienze personali. Questa attività è propedeutica ad altre attività di progetto, in particolare all'attività di normazione, pianificazione di qualità e alle attività di processo di sviluppo, descritte nella sezione seguente. Il gruppo si impegnerà a informarsi ed apprendere le tecnologie scelte.
Prodotto attività: scelta di opportuni prodotti di supporto al progetto e padronanza delle tecnologie scelte durante lo sviluppo.

2.1.4 Normazione

L'Amministratore apporta continui incrementi all'insieme di regole che ogni componente del gruppo deve seguire per tutta la durata dello svolgimento dell'attività e dei compiti assegnato ad ognuno di esso. Durante il periodo di svolgimento del progetto, si acquisiscono nuove conoscenze o si riscontrano delle problematiche che potrebbero richiedere delle modifiche di queste regole. Un esempio è la necessità di utilizzare un nuovo strumento di lavoro ritenuto più consono.
Prodotto attività: *Norme di Progetto v4.0.0*D.

2.1.5 Studio di Fattibilità

Gli *analista* conducono una analisi approfondita dei requisiti di ogni capitolato proposti dai committenti al fine di scegliere se accettare o meno il contratto.
Dallo studio di ciascun capitolato vengono ricavate le seguenti informazioni:

- **Informazioni generali**: indica il nome del progetto, l'azienda proponente e il committente;
- **Descrizione del capitolo**: breve introduzione di quello che deve essere il prodotto finale;
- **Finalità di progetto**: descrizione di ciò che è richiesto dal capitolato e delle modalità;

- **Tecnologie interessate**: elenco delle tecnologie consigliate o esplicitamente richieste dal proponente per lo sviluppo del progetto;
- **Conclusioni**: valutazioni riguardo al capitolato e spiegazione dei motivi che hanno portato alla scelta o meno di esso.

Prodotto attività: *Studio di Fattibilità v1.0.0*ᴅ.

2.1.6 Preparazione per la revisione

L'attività consiste nell'approntamento del materiale da presentare in ingresso alla corrispettiva revisione ed eventuali materiali per l'esposizione, quando necessaria. La prima fase comprende la scrittura di una lettera di presentazione da parte del responsabile del team.
Prodotto attività: materiale d'esposizione e lettera di presentazione.

2.1.7 Documentazione fornita

I documenti che verranno stilati e forniti all'azienda zero12 e ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin sono i seguenti:

- ***Piano di Progetto v4.0.0***ᴅ: documento che descrive la pianificazione del progetto, i suoi costi in termini di denaro e tempo;
- ***Analisi dei Requisiti v3.0.0***ᴅ: documento che descrive l'analisi dei requisiti e dei casi d'uso;
- ***Piano di Qualifica v4.0.0***ᴅ: documento che descrive i metodi di verifica e di validazione e le metriche di qualità del prodotto e dei processi;
- ***Glossario v4.0.0***ᴅ: documento che descrive i termini con significato ambiguo presenti negli altri documenti.

2.2 Processo di sviluppo

2.2.1 Scopo

Il processo contiene le attività e i compiti da svolgere, al fine di realizzare il prodotto finale richiesto dal proponente.

2.2.2 Descrizione

Il processo di sviluppo contiene tutte le attività che riguardano la produzione del software richiesto dal proponente, in particolare le attività di analisi,codifica, design, integrazione ed installazione del prodotto da sviluppare. In seguito sono presenti le linee guida che utilizzeremo nelle principali attività di questo processo.

2.2.3 Analisi dei Requisiti

Gli *analisti* hanno il compito di definire nel dettaglio l'utilizzo specifico del prodotto da sviluppare. Il tutto avviene sulla base delle informazioni presenti sulla presentazione del capitolato scelto e di quanto stabilito con il proponente durante le riunioni esterne effettuate.
Lo scopo dell'*Analisi dei Requisiti* è:

- descrivere nel dettaglio l'obiettivo del progetto;
- stabilire e apprendere appieno i requisiti richiesti dall'azienda proponente;

- facilitare le scelte di progettazione;
- facilitare lo sviluppo dei casi d'uso;
- stimare i costi.

I requisiti:

- non devono essere in conflitto tra loro;
- devono essere verificabili;
- devono essere tutti e soli quelli necessari e sufficienti;
- devono garantire soddisfacibilità rispetto ai vincoli di processo.

Prodotto attività: *Analisi dei Requisiti v3.0.0*ᴅ.

2.2.3.1 Classificazione dei requisiti

I requisiti vengono ordinati utilizzando un indice di priorità, uno di tipo e un codice per facilitarne la comprensione, la manutenzione e il tracciamento. Verrà inoltre specificata la fonte del requisito.

R[Priorità][Tipo][Codice]

- **Priorità**: Ogni requisito ha una sua priorità che può essere di tre livelli:
  - **Obbligatorio (0)**: indica un requisito irrinunciabile per il committente e che serve per garantire il funzionamento di base del sistema;
  - **Desiderabile (1)**: indica un requisito che aggiunge valore e completezza al sistema ma non è indispensabile ai fini del suo funzionamento di base;
  - **Opzionale (2)**: indica un requisito la cui utilità è relativa o potrà rivelarsi utile in un secondo momento e la cui implementazione potrebbe portare a un aumento dei costi.
- **Tipo**: Ogni requisito si differenzia in 4 diverse tipologie:
  - **Funzionali (F)**: descrivono come il sistema reagisce a particolari input al fine di soddisfare i servizi forniti;
  - **Prestazionali (P)**: stabiliscono un vincolo sulle prestazioni del sistema in determinate condizioni;
  - **Qualitativi (Q)**: garantiscono la qualità del prodotto includendo requisiti di efficacia ed efficienza;
  - **Dichiarativi o di Vincolo (D)**: rappresentano un vincolo imposto dall'azienda proponente zero12.
- **Codice**: Ogni requisito deve avere un codice identificativo univoco. Questo numero segue la posizione del requisito all'interno della struttura del documento (e.g. 2.5.7).
  Il codice stabilito secondo la convenzione precedente, una volta associato ad un requisito, non potrà cambiare nel tempo.

ID Requisito	Descrizione	Fonte
R0F1	Il sistema deve permettere ad un nuovo utente di registrarsi tramite Amazon	UC1

Tabella 2: Esempio di requisito

2.2.3.2 Classificazione casi d'uso

Gli *analisti* hanno anche il compito di identificare i casi d'uso, dando un grado di importanza e precisione che va dal generico verso il dettaglio.
La convenzione che HexaDec ha fissato riguardante i casi d'uso è la seguente:

UC[codice padre].[codice figlio] - Nome

- **UC:** Use Case;
- **codice padre:** numero che identifica i casi d'uso;
- **codice figlio:** numero progressivo che identifica i sotto casi, esso può includere altri sotto casi;
- **nome:** nome identificativo del caso d'uso.

Ogni caso d'uso è inoltre definito secondo la seguente struttura:

- **Attori:** indica gli *attori*ɢ primari e secondari del caso d'uso;
- **Descrizione:** una breve descrizione generale del caso d'uso;
- **Pre-condizione:** indica le condizioni che sono identificate come vere prima del verificarsi del caso d'uso;
- **Post-condizione:** indica le condizioni che sono vere dopo del verificarsi del caso d'uso;
- **Scenario Principale:** eventi del caso d'uso che vengono numerati;
- **Inclusioni:** eventi comuni a più casi d'uso, situazione per cui un caso d'uso è incluso nell'esecuzione di un altro;
- **Estensioni:** indica l'aumento delle funzionalità di un caso d'uso, ogni istanza di un caso d'uso esegue un altro caso d'uso in modo condizionato;
- **Generalizzazioni:** aggiungono o modificano caratteristiche base.

Per mettere in evidenza attori e servizi, useremo i diagrammi per la rappresentazione grafica di alcuni casi d'uso.

Esempio di caso d'uso:

UC1 - Registrazione

- **Attori primari**: utente non autenticato;
- **Descrizione**: per accedere all'applicazione è necessario che l'attore si registri ad Amazon;
- **Pre-condizione**: l'attore non possiede un account Amazon;
- **Post-condizione**: è stato creato un account Amazon per accedere all'applicazione;
- **Scenario principale**:
  1. l'utente clicca sul pulsante per la registrazione;
  2. l'utente viene reindirizzato alla pagina di registrazione di Amazon;
  3. l'utente completa la registrazione;
  4. l'utente viene reindirizzato all'homepage dell'applicazione.

2.2.3.3 UML 2.0

L'analisi utilizza la seguente tipologia di diagrammi:

- **diagrammi dei casi d'uso:** mette in evidenza gli attori e i servizi del sistema , illustrando nel dettaglio di cadi d'uso.

2.2.4 Progettazione

L'attività di progettazione deve precedere con lo sviluppo del software e consiste nella descrizione della soluzione del problema che sia soddisfacente per gli *stakeholders*G. Essa serve a garantire che il prodotto sviluppato soddisfi le proprietà *Analisi dei requisiti* necessarie per produrre una soluzione delle richieste.

La progettazione permette di:

- garantire la qualità del prodotto;
- costruire un'*architettura*G del prodotto;
- organizzare e ripartire compiti implementativi, così da ridurre la complessità del problema;
- ottimizzare l'uso delle risorse.

Come da *Piano di Progetto v4.0.0*D, la Progettazione si svolgerà nell'arco di due periodi distinti. Nel primo di questi due periodi viene identificata una prima decomposizione logica dell'architettura, invece nel secondo la progettazione avviene in modo atomico e dettagliato.
È compito dei *progettisti* svolgere l'attività di Progettazione, definendo l'architettura logica del prodotto identificando componenti chiare, riusabili e coese, rimanendo nei costi fissati.
L'architettura definita si dovrà attenere alle seguenti qualità:

- **sufficienza:** soddisfare tutti i requisiti definiti in *Analisi dei Requisiti v3.0.0*D;
- **comprensibilità:** comprensibile agli stakeholder;
- **modularità:** suddivisa in parti chiare e distinte;

- **robustezza:** capacità di sopportare ingressi diversi dall'utente e dall'ambiente;
- **flessibilità:** capacità di adattamento al variare dei requisiti con conseguente modifica dei costi;
- **riusabilità:** dovrà essere costruita in modo da poter permettere il riutilizzo di alcune sue parti;
- **efficienza:** dovrà soddisfare tutti i requisiti per ridurre gli sprechi di tempo e risorse;
- **affidabilità:** dovrà garantire il rispetto delle sue specifiche di funzionamento nel tempo;
- **disponibilità:** durante operazioni di manutenzione il sistema non viene totalmente interrotto o solamente per poco tempo;
- **sicurezza:** dovrà essere sicura rispetto a malfunzionamenti e intrusioni;
- **semplicità:** dovrà prediligere la semplicità, contenendo solo il necessario, rispetto ad una inutile complessità;
- **incapsulazione:** l'interno delle componenti non sia visibile dall'esterno;
- **coesione:** dovrà raggruppare le parti secondo l'obiettivo a cui concorrono, in modo da avere maggiore manutenibilità e riusabilità;
- **basso accoppiamento:** non dovrà avere dipendenze indesiderabili.

2.2.4.1 Design Patterns

Una volta finita l'attività di analisi, i *progettisti* hanno il compito di adottare opportune soluzione progettuali sui problemi ricorrenti. Ogni *Design Patterns*G utilizzato, andrà riportato all'interno del documento *Manuale Sviluppatore v1.0.0*D, illustrato tramite un diagramma e una descrizione della sua applicazione all'interno dell'architettura.

2.2.4.2 UML 2.0 Per rendere più chiare e complete le scelte progettuali adottate ed eliminare ambiguità, si utilizzeranno dei diagrammi *UML*G 2.0 e ognuno di essi, avrà una descrizione di ciò che rappresenta.
Tali diagrammi si suddividono in quattro tipi:

- **diagrammi delle classi:** descrivono gli oggetti che fanno parte di un sistema e come interagiscono tra loro;
- **diagrammi di sequenza:** descrivono la collaborazione di più oggetti volti all'implementazione di un determinato comportamento;
- **diagrammi di attività:** descrivono la logica procedurale. Aiuta a descrivere gli aspetti dinamici dei casi d'uso;
- **diagrammi dei *packages*G:** raggruppa elementi UML a un livello più elevato.

2.2.5 Codifica

La codifica è un processo svolto dai *programmatori*. In questa sezione intendiamo offrire una guida alle scelte che abbiamo preso durante questo processo. L'uso di norme e convenzioni per la codifica è fondamentale per la generazione di codice leggibile e uniforme. Questo infatti agevola le fasi di manutenzione, verifica e validazione e migliora la qualità di prodotto.

Prodotto attività: implementazione del prodotto con le sue funzionalità.

2.2.5.1 Intestazione

In ogni file di codifica , ci dovrà essere la seguente intestazione:

```
/*
* Nome file:
* Type:
* Data creazione: AAAA-MM-GG
*
* Descrizione del file:
*
* Autore:
* Versione:
* Registro modifiche:
* Autore, Data ultima modifica, descrizione
*/
```

2.2.5.2 Versionamento

La versione del codice, che verrà riportata all'intestazione del file, deve rispettare il seguente formalismo:

X.Y

- **X**: l'indice di versione principale, un incremento di tale indice rappresenta un avanzamento della versione stabile, che porta il valore dell'indice Y ad essere azzerato;
- **Y**: l'indice di modifica parziale, un incremento di tale indice rappresenta una verifica o una modifica rilevante.

La prima versione di rilascio corrisponde alla 1.0 che indica un prodotto finito stabile. Da questa versione possono avvenire poi la verifica e le modifiche che conducono a un incremento degli indici.

2.2.5.3 Stile di codifica

2.2.5.3.1 Kotlin

In questa sezione vengono indicate le convezioni che sono state prese dal gruppo HexaDec per il linguaggio di programmazione Kotlin. Queste convenzioni dovranno essere rispettate dai *programmatori* durante la scrittura di codice. Il gruppo si è basato sulla documentazione ufficiale del linguaggio di programmazione Kotlin (https://kotlinlang.org/docs/reference/).

Indentazione

1. Utilizzo di parentesi graffe per tutti i blocchi multi-linea. Quando i blocchi sono inline le parentesi non vanno riportate.

Listing 1: Indentazione SI

```
if(condizione){
  return false
 }

if(condizione) return false
```

Listing 2: Indentazione NO

```
if(condizione)
  return false
```

2. Porre l'else sulla stessa riga di chiusura dell'if, separato da uno spazio dalla parentesi graffa di chiusura e apertura.

Listing 3: Indentazione SI

```
if(condizione){
  function1()
  function2()
 } else {
  function3()
 }
```

Listing 4: Indentazione NO

```
if(condizione){
  function1()
  function2()
 }
 else {
  function3()
 }
```

Convenzione nomi

I nomi di classi , metodi e variabili devono essere univoci, esplicative e in inglese al fine di evitare ambiguità di comprensione.

1. **Nome variabili:**
   I nome delle variabili vengono scritte con lettere minuscole compresa la lettera iniziale. In caso il nome sia composto da due parole o più , si adotta la modalità lowerCamelcase, come mostrato in seguito:

Listing 5: Nome variabile SI

```
var s: String = valore
var sonoUnBool: bool = valore
var nome: int = valore
```

Listing 6: Nome variabile NO

```
var s1: String = valore
var sonounbool: bool = valore
var VAR: int = valore
var Var: double = valore
```

Per quanto riguarda le variabili costanti , vengono definite nel seguente modo:

Listing 7: Indentazione Definizione varibili constanti

```
val nomeVariabile: tipoVariabile = valoreVariabile
```

2. **Nome metodi:**
   I nomi dei metodi devono essere scritti utilizzando la modalità lowerCamelcase. La scelta del nome del metodo deve essere sensata in base allo scopo della funzione.

Listing 8: Nome metodi SI

```
fun getNameBlock(): String{
    return "WEATHER"
}
```

Listing 9: Nome metodi NO

```
fun getnameblock(): String{
    return "WEATHER"
}
```

La sintassi per un metodo override è la seguente:

Listing 10: Sintassi metodo override

```
override fun getNameBlock(): String{
    return "WEATHER"
}
```

3. **Nome classe:**
   I nomi delle classi iniziano con la lettera maiuscola.

Listing 11: Nome classe SI

```
class Megalexa{
  ...
}
```

Listing 12: Nome classe NO

```
class megalexa{
  ...
}
```

Commenti
I commenti all'interno di un file di codifica possono essere di due tipi:

- **commenti inline:** sviluppato su una riga;
- **commenti di blocco:** sviluppato su più righe.

1. **Commenti inline:**
   I commenti inline , vengono preceduti da //... :

Listing 13: Commenti SI

```
//example comment inline


//constant variable
val nomeVar: String = valoreVar;
```

Listing 14: Commenti NO

```
val nomeVar: String = valoreVar; //constant variable
```

2. **Commenti di blocco:**
   I commenti su più righe vanno inseriti all'intero di /⋆...⋆/ :

Listing 15: Commenti SI

```
/*
*  returns a new element
*  based on the
*  passed-in tag name
*
*/
```

Listing 16: Commenti NO

```
//returns a new element
//based on the
//passed-in tag name
```

2.2.5.3.2 Node.js

In questa sezione vengono indicate le scelte prese dal gruppo per quanto riguarda il linguaggio di programmazione Node.js. Queste convenzioni dovranno essere rispettate dai *programmatori* durante la scrittura di codice. Il gruppo si è allineato allo stile di codifica descritto dalla guida Airbnb Javascript style guide (https://github.com/airbnb/javascript).

Indentazione

1. Utilizzo di parentesi graffe per tutti i blocchi multi-linea. Quando i blocchi sono inline le parentesi non vanno riportate.

Listing 17: Indentazione SI

```
if(condizione){
  return false;
 }

if(condizione) return false;
```

Listing 18: Indentazione NO

```
if(condizione)
  return false;
```

2. Porre l'else sulla stessa riga di chiusura dell'if, separato da uno spazio dalla parentesi graffa di chiusura e apertura.

Listing 19: Indentazione SI

```
if(condizione){
  function1();
  function2();
 } else {
  function3();
 }
```

Listing 20: Indentazione NO

```
if(condizione){
  function1();
  function2();
 }
 else {
  function3();
 }
```

Convenzione nomi

I nomi di classi , metodi e variabili devono essere univoci, esplicative e in inglese al fine di evitare ambiguità di comprensione.

1. **Nome variabili:**
   I nome delle variabili vengono scritte con lettere minuscole compresa la lettera iniziale. Inoltre le variabili sono dichiarate utilizzando le keyword let che sonoo visibili solo all'interno dove sono state dichiarate e con var che sono variabili globali. In caso il nome sia composto da due parole o più , si adotta la modalità lowerCamelcase, come mostrato in seguito:

Listing 21: Nome variabile SI

```
var s = valore;
var dueParole = valore
var prova= valore
let variabileLocale;
```

Listing 22: Nome variabile NO

```
var s1: String = valore;
var sonounbool: bool = valore;
let VAR: int = valore;
let Var: double = valore;
```

Per quanto riguarda le variabili costanti , vengono definite nel seguente modo:

Listing 23: Indentazione Definizione varibili constanti

```
const nomeVariabile = valoreVariabile;
```

2. **Nome metodi:**
   I nomi dei metodi devono essere scritti utilizzando la modalità lowerCamelcase. La scelta del nome del metodo deve essere sensata in base allo scopo della funzione.

Listing 24: Nome metodi SI

```
function getQuery(){
  ...
}
```

Listing 25: Nome metodi NO

```
function getquery(){
  ...
}
```

3. **Nome classe:**
   I nomi delle classi iniziano con la lettera maiuscola.

Listing 26: Nome classe SI

```
class User {
  constructor(options) {
      this.name = options.name;
  }
}
```

Listing 27: Nome classe NO

```
class user {
  constructor(options) {
    this.name = options.name;
  }
}
```

Commenti

I commenti all'interno di un file di codifica possono essere di due tipi:

- **commenti inline:** sviluppato su una riga;
- **commenti di blocco:** sviluppato su più righe.

1. **Commenti inline:**

   I commenti inline , vengono preceduti da //... :

Listing 28: Commenti SI

```
//example comment inline

//constant variable
const nomeVar = valoreVar;
```

Listing 29: Commenti NO

```
const nomeVar = valoreVarCostante; //constant variable
```

2. **Commenti di blocco:**
   I commenti su più righe vanno inseriti all'intero di /⋆...⋆/ :

Listing 30: Commenti SI

```
/*
*  returns a new element
*  based on the
*  passed-in tag name
*
*/
```

Listing 31: Commenti NO

```
//returns a new element
//based on the
//passed-in tag name
```

In fine, sono state adottate alcune convenzioni per facilitare il riscontro di problemi o incompletezza di funzioni che sono i seguenti:

- //**FIXME:** utilizzato per segnalare problemi. Sorge laddove vi è un errore da risolvere;
- //**TODO:** utilizzato per indicare ciò che deve essere ancora svolto per risolvere un problema.

3 Processi di Supporto

3.1 Documentazione

In questo processo contiene linee guida su come devono essere redatti, modificati, distribuiti e mantenuti tutti i documenti prodotta dal gruppo HexaDec necessari durante il ciclo di vita del software. Lo scopo è quello di fornire una descrizione accurata di tutte le norme, convenzioni e vincoli che devono essere rispettati per ottenere documentazione efficace, coerente e formale.

3.1.1 Implementazione

3.1.1.1 Template LaTeX

Per uniformare la struttura e facilitare la stesura di un documento il gruppo ha creato un template LaTeX riutilizzabile in tutti i documenti ufficiali. Abbiamo definito anche:

- dimensione delle pagine, del testo e relative spaziature;
- comando unico per la creazione della prima pagina, uguale in tutti i documenti;
- comandi volti alla semplificazione e velocizzazione della stesura tramite l'inserimento di nomi di ruoli di progetto, nomi propri di componenti del gruppo,professori, nome del gruppo, del capitolato e dell'azienda proponente ed altri termini ritenuti significativi.

3.1.1.2 Ciclo di vita

Ogni documento segue le seguenti fasi durante il suo ciclo di vita.

1. **Redazione**: In questa fase i redattori producono o modificano il documento;
2. **Verifica**: Il documento entra in questa fase quando i redattori hanno terminato il proprio lavoro. Il *responsabile* incaricherà i *verificatori* di controllare la correttezza, la conformità e la qualità del documento. Se non supera il controllo con esito positivo, i *verificatori* incaricheranno i redattori a correggere gli errori. In caso di esito positivo, il documento passerà alla fase di approvazione;
3. **Approvazione**: In questa fase, il documento è sottoposto al *responsabile* di progetto, che potrà approvarlo o meno. Se approvato, il documento è da considerarsi formale. In caso contrario, il *responsabile* di progetto deve fornire le motivazioni del rifiuto.

3.1.2 Struttura

3.1.2.1 Fontespizio

La prima pagina di ogni documento è strutturata ordinatamente nel seguente modo:

- **Logo del gruppo**: visibile come primo elemento e centrato;
- **Titolo**: nome del documento, centrato;
- **Progetto**: nome del progetto, centrato;
- **Recapito**: indirizzo email del gruppo, centrato;
- **Informazioni sul documento**: una sezione contenente le seguenti informazioni:
  - Versione: numero di versione del documento;
  - Responsabile: nome del *responsabile*;

- Redattori: nomi dei redattori del documento;
- Verificatori: nomi dei *verificatori* del documento;
- Uso: tipo di uso;
- Destinatari: destinatari del documento.

3.1.2.2 Registro delle modifiche

Ogni documento dispone di un registro delle modifiche sotto forma di tabella. Tale tabella contiene le modifiche apportate al documento. Ogni riga della tabella corrisponde a una modifica apportata. Le colonne sono le seguenti:

- **Versione**: versione corrente del documento;
- **Data**: data della modifica;
- **Autore**: nome della persona *responsabile* della modifica;
- **Ruolo**: ruolo dell'autore al momento della modifica;
- **Descrizione**: breve descrizione della modifica apportata.

3.1.2.3 Indice

Tutti i documenti eccetto i verbali devono contenere un indice, il quale consente una visione macroscopica del contenuto del documento, oltre a facilitarne la lettura ipertestuale. La struttura dell'indice è gerarchica e rispetta la numerazione delle sezioni e sottosezioni del documento. Possono essere presenti fino a tre indici differenti:

- per le sezioni del documento;
- per le immagini incluse nel documento;
- per le tabelle, esclusa la tabella dello storico delle versioni.

3.1.2.4 Contenuto principale

La struttura e presentazione delle pagine del contenuto sono comuni a tutti i documenti. Ogni pagina del contenuto è così strutturata:

- **Intestazione**: contiene il logo del gruppo in alto a sinistra e la sezione corrente in alto a destra;
- **Piè di pagina**: contiene il nome del documento con la relativa versione in basso a sinistra e numero di pagina corrente rispetto al totale in basso a destra.

Eventuali note a piè di pagina vanno indicate in basso alla pagina corrente con un numero identificativo e una descrizione.

3.1.3 Design

3.1.3.1 Norme tipografiche

3.1.3.1.1 Stile di testo

- **Grassetto**: viene applicato ai titoli, a termini su cui si vuole mettere enfasi o attirare l'attenzione del lettore e, se necessario, agli elementi di un elenco puntato;
- **Corsivo**: viene applicato nei seguenti casi:
  - parole del glossario;
  - attività del progetto;
  - riferimenti ad altri documenti;
  - ruoli del progetto;
  - parole particolari, solitamente poco conosciute o usate;
  - comandi vocali per Alexa.
- **Maiuscolo**: le uniche parole che è consentito scrivere interamente in maiuscolo sono gli acronimi;
- **Glossario**: ogni parola contenuta nel glossario deve essere marcata, alla sua prima occorrenza in ogni documento, in carattere corsivo e con una G maiuscola a pedice.

3.1.3.1.2 Elenchi puntati

Ogni voce di un elenco comincia per lettera minuscola. Questa regola può non essere applicata per enfatizzare concetti relativamente corti.
Ogni voce di un elenco termina con un "**;**", eccetto l'ultima che termina con un "**.**".
Ogni elemento di un elenco puntato viene rappresentato graficamente da un pallino nel primo livello, da un trattino nel secondo e da un asterisco nel terzo. Se un elenco puntato ha lo scopo di identificare una sequenza ordinata allora deve essere utilizzato un elenco enumerato al primo livello e un elenco alfabetico al secondo.

3.1.3.1.3 Formati comuni

Per le seguenti tipologie di concetto vengono applicati i seguenti formalismi:

- **Data**:

  AAAA-MM-GG

  - **GG**: rappresenta il giorno con due cifre;
  - **MM**: rappresenta il mese con due cifre;
  - **AAAA**: rappresenta l'anno con quattro cifre.
- **Ora**:

  HH:MM

  - **HH**: rappresenta l'ora con due cifre e un valore compreso tra 00 e 23;
  - **MM**: rappresenta i minuti con due cifre e un valore compreso tra 00 e 59.

3.1.3.2 Elementi grafici

- **Tabelle:**
  Ogni tabella deve:
  - essere centrata orizzontalmente nella pagina;
  - avere una breve didascalia;
  - avere un numero progressivo unico in tutto il documento per facilitarne il tracciamento.
- **Immagini:**
  Ogni immagine deve essere centrata orizzontalmente e avere una breve descrizione. Inoltre, per migliorarne la leggibilità, ci deve essere una netta separazione tra paragrafi che precedono e seguono un'immagine.
  I *diagrammi di Gantt*G, UML e i grafici vengono inseriti sotto forma di immagine.

3.1.4 Produzione

3.1.4.1 Classificazione dei documenti

3.1.4.1.1 Documenti informali

Tutte le versioni dei documenti che non sono ancora state approvate dal *responsabile di progetto* sono ritenute informali e l'utilizzo è da considerarsi esclusivamente all'interno del gruppo.

3.1.4.1.2 Documenti formali

Un documento viene definito formale quando è stato approvato dal *responsabile di progetto*. Per raggiungere tale stato il documento deve superare la verifica e la validazione. Solo i documenti formali possono essere distribuiti all'esterno del gruppo.

3.1.4.1.3 Verbali

Documenti destinati a uso interno redatti da un segretario in occasione di incontri interni al gruppo. Non necessitano di versionamento in quanto non necessitano di modifiche successive alla prima stesura. Ogni verbale deve essere approvato dal *responsabile di progetto*.

3.1.4.1.4 Glossario

Documento destinato a uso esterno. Consiste in un elenco di definizioni di tutte le parole possibilmente sconosciute o che possono creare ambiguità e necessitano di una definizione precisa. Tutte le parole del glossario, alla prima occorrenza in un documento, sono scritte in corsivo e marcate con una G a pedice.

3.1.5 Strumenti

3.1.5.1 LaTeX

Per la stesura della documentazione il gruppo utilizza il linguaggio LaTeX in quanto permette:

- di separare il contenuto dalla formattazione tramite un template condiviso da tutti i documenti;
- a individui diversi di lavorare su capitoli diversi di uno stesso documento;
- di generare documenti al massimo grado di professionalità e qualità.

3.1.5.2 Script

Per facilitare e automatizzare il lavoro di verifica e creazione dei documenti, sono stati adottati i seguenti script:

- **gulpease.php** Calcola l'*indice di Gulpease*G per ogni sezione di un documento; questo script viene lanciato durante la verifica di un documento da parte del *verificatore*, in modo da garantire almeno la accettabilità di tale documento;
- **glossarize.php** Questo script crea il glossario, prendendo in input un file che abbia una struttura "termine , descrizione" e produce le sezioni del documento *Glossario*.

3.1.5.3 Visual Studio Code

Per la stesura del codice LaTeX il gruppo utilizza l'editor Visual Studio Code. Questo strumento integra un compilatore e visualizzatore PDF, fornisce suggerimenti per completare i comandi di LaTeX e permette l'integrazione con *Git*G.

3.1.5.4 Astah UML

Per modellare i diagrammi UML 2.0, di casi d'uso e di attività il gruppo utilizza lo strumento Astah UML.

3.1.5.5 Microsoft Project e Instagantt

Per la consegna della *Revisione dei Requisiti* il gruppo ha utilizzato Microsoft Project per la modellazione dei diagrammi di Gantt. Per le prossime consegne si utilizzerà Instagantt.

3.1.6 Mantenimento

3.1.6.1 Versionamento

Ogni documento, ad eccezione dei verbali, deve essere versionato in modo da permettere l'accesso ad ogni singola versione prodotta durante il loro ciclo di vita. Il formalismo da applicare è il seguente:

v[x].[y].[z]

dove:

- **x**:
  - parte da 0;
  - indica il numero di versione principale;
  - viene incremento dal *responsabile di progetto* all'approvazione del documento.
- **y**:
  - parte da 0;
  - indica una modifica parziale;
  - viene incrementato dal *verificatore* ad ogni verifica;
  - viene riportata a 0 a ogni incremento della x.
- **z**:
  - parte da 0;
  - indica una modifica minore, una correzione o aggiunta che non deve essere sottoposto a verifica;
  - viene incrementato dal redattore;
  - viene riportata a 0 a ogni incremento di x e/o y.

3.1.6.2 Repository

Il gruppo ha scelto il software di controllo versione distribuito Git e la piattaforma *GitLab*G per gestire la *repository*G creata su quest'ultima piattaforma.

3.1.6.2.1 Struttura

La repository è organizzata in cartelle:

- **RR**: raccoglie tutti i file necessari per la creazione di documenti da consegnare per la *Revisione dei Requisiti*. Il contenuto della cartella **RR** è suddiviso in:
  - **Esterni**: contiene tutte le cartelle per i documenti esterni con i rispettivi file .tex;
  - **Interni**: contiene tutte le cartelle per i documenti interni con i rispettivi file .tex;
  - **template**: contiene i file del template.
  - **script**: contiene tutti i *script*G realizzati dal gruppo per la gestione dei documenti.
- **RP**: raccoglie tutti i file necessari per la creazione di documenti da consegnare per la *Revisione di Progettazione* e file necessari per fare il *build*G dell'applicazione. La struttura della cartella **RP** è identifica alla *Revisione dei Requisiti* con aggiunta della cartella PoC che contiene i file necessari per la compilazione e il build;
- **RQ**: raccoglie tutti i file necessari per la creazione di documenti da consegnare per la *Revisione di Qualifica* e file nesessari per fare il *build*G dell'applicazione. La struttura della cartella **RQ** è identifica alla *Revisione di Progettazione* con aggiunta della cartella Hexadec App Android e Hexadec App Skill che contengono i file necessari per la compilazione e il build;
- Saranno aggiunte le cartelle **RA** che conterranno file per le successive consegne.

3.1.6.3 Aggiornamento della repository

Per l'aggiornamento della repository è prevista la seguente procedura:

- dare il comando *git pull* per aggiornare la repository locale rispetto alla repository remoto. Nel caso si verifichino dei conflitti:
  - dare il comando *git stash* per accantonare momentaneamente le modifiche apportate;
  - dare il comando *git pull*;
  - dare il comando *git stash apply* per ripristinare le modifiche.
- dare il comando *git add [lista nomi dei file separati da uno spazio]* sul quale sono state apportate le modifiche;
- dare il comando *git commit* e specificare sinteticamente le modifiche apportate;
- dare il comando *git push* per aggiornare la repository.

3.1.6.4 Garanzia della qualità

Si occupa di fornire un'adeguata garanzia che i prodotti e i processi software siano conformi e aderiscano ai piani stabiliti nel ciclo di vita del software.

3.1.6.4.1 Aspettative

L'obiettivo è di ottenere:

- qualità nei processi;
- qualità nei prodotti;
- soddisfazione del proponente;
- qualità del sistema per costruzione.

3.1.6.4.2 Controllo qualità di prodotto

La qualità di prodotto viene garantita dai processi di verifica e validazione e dal rispetto delle norme.

- **Verifica**: accerta che l'esecuzione delle attività di processo siano corrette;
- **Validazione**: accerta che il prodotto rispecchi le aspettative, utilizzando un metodo sistematico, disciplinato e quantificabile;
- **Norme**: il prodotto deve rispettare le norme e gli strumenti descritti all'interno di questo documento.

3.1.6.4.3 Controllo qualità di processo

La qualità di processo viene migliorata continuamente seguendo il modello *PDCA*G che apporta qualità in modo incrementale. Basandosi sullo standard *SPICE*G, si può quantificare la qualità di un processo e studiare le sue caratteristiche di interesse.

3.1.6.4.4 Classificazione metriche

I *verificatori* hanno definito delle metriche per garantire la qualità del lavoro, riportandone gli obiettivi nel *Piano di Qualifica v4.0.0*D.
Tali metriche devono rispettare la seguente nomenclatura:

M[X][Y]

dove:

- **M**: sta per metrica;
- **X**: indica la tipologia della metrica e può assumere i seguenti valori:
  - **PC**: indica una metrica per il processo;
  - **PD**: indica una metrica per il documento;
  - **PS**: indica una metrica per il software.
- **Y**: è un numero intero incrementale che indica il codice univoco della metrica.

3.2 Verifica

3.2.1 Scopo

Si occupa di individuare gli errori introdotti nel prodotto durante la fase di redazione o sviluppo e di realizzare prodotti corretti e coesi.

3.2.2 Aspettative

La corretta esecuzione del processo in esame permette di individuare:

- i criteri per la verifica;
- una procedura di verifica;
- i difetti da correggere.

3.2.3 Descrizione

Per ottenere un prodotto conforme alle aspettative, il processo fa affidamento a due attività:

- **Analisi**: consiste nell'analisi del codice sorgente e la sua successiva esecuzione;
- **Test**: definisce tutti i test che vengono eseguiti sul software.

3.2.4 Metriche

I *verificatori*, per garantire la qualità del lavoro del gruppo, hanno definito delle metriche, riportandone gli obiettivi di qualità nel *Piano di Qualifica v4.0.0*ᴅ. Tali metriche devono rispettare la seguente notazione:

MXY

Dove:

- **X:** Indica se la metrica si riferisce a processi, prodotto documento o prodotto software e può assumere i seguenti valori:
  - **PD:** indica una metrica per il documento;
  - **PS:** indica una metrica per il software;
  - **PC:** indica una metrica per il processo.
- **Y:** indica il codice univoco della metrica,incrementale a partire da uno.

3.2.4.1 Metrice di processo

Metriche di utilità:

- **Earned value** (EV): valore del lavoro completato fino al momento del calcolo;
- **Planned value** (PV): valore del lavoro come da pianificazione al momento del calcolo;
- **Actual cost** (AC): l'importo speso per il progetto fino al momento del calcolo;
- **Estimate to Completion** (ETC): l'importo previsto che verrà speso per completare la parte restante del progetto al momento del calcolo;
- **Budget at Completion** (BAC): è il budget totale assegnato al progetto.

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

3.2.4.1.1 MPC1 Schedule Variance (SV)

Determina l'anticipo o ritardo nello svolgimento dei processi. È ottenibile attraverso la seguente formula:

$$SV = EV - PV.$$

3.2.4.1.2 MPC2 Budget Variance (BV)

Determina se si è sotto budget o oltre il budget. È ottenibile attraverso la seguente formula:

$$BV = EV - AC.$$

3.2.4.1.3 MPC3 Estimated at Completion(EAC)

Determina il costo previsto del progetto, man mano che il progetto procede. È ottenibile attraverso la seguente formula:

$$EAC = AC + ETC.$$

3.2.4.1.4 MPC4 Variance at Completion (VAC)

È una proiezione dell'eccedenza o del deficit di bilancio. È ottenibile attraverso la seguente formula:

$$VAC = BAC - EAC.$$

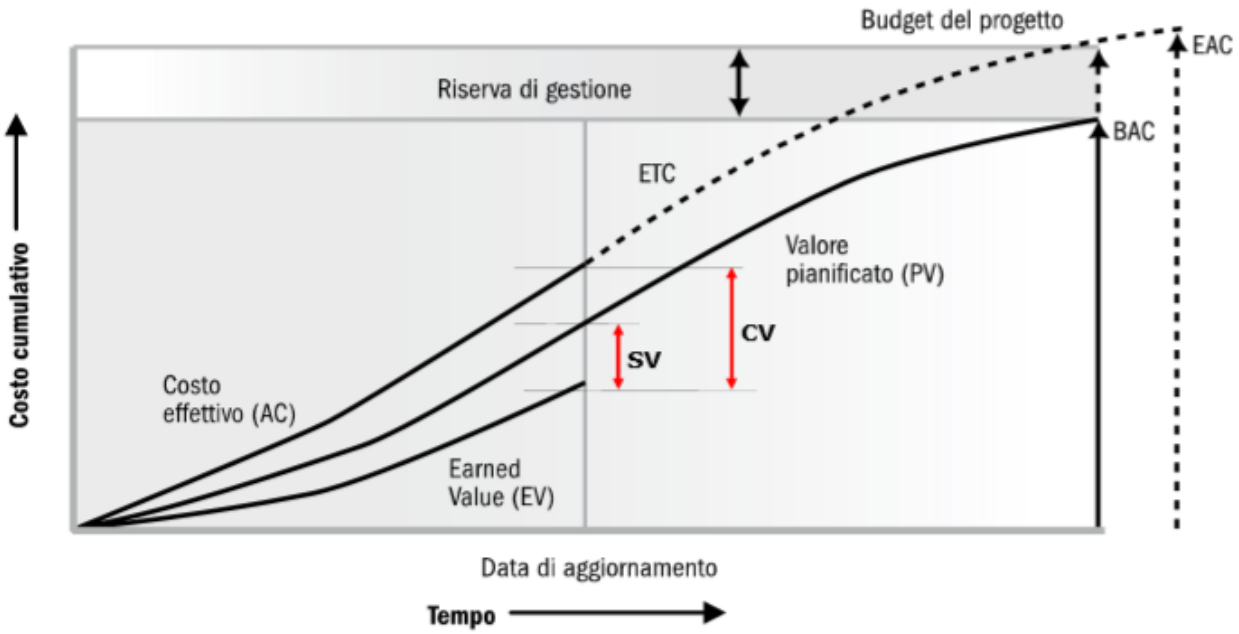

Figura 1: Grafico degli scostamenti
Fonte: https://elearning.dei.unipd.it/mod/resource/view.php?id=112537

3.2.4.1.5 MPC5 Commenti al codice

Percentuale del numero di righe di commento per righe di codice:

$$I = \frac{N_{\text{righeCommneti}}}{N_{\text{righeCodece}}} * 100$$

3.2.4.1.6 MPC6 Numero di commit

Verrà tenuto conto del numero di commit a settimana:

$$I = \frac{N_{\text{commmitSettimana}}}{N_{\text{settimana}}} * 100$$

3.2.4.2 Metriche per i prodotti

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

3.2.4.2.1 MPD1 Indice di Gulpease

Determina l'indice di leggibilità del testo. Esso considera il numero di frasi e lettere rispetto al numero di parole. È ottenibile attraverso la seguente formula:

$$IG = 89 + \frac{300 * num.\,frasi - 10 * num.\,lettere}{num.\,parole}$$

3.2.4.2.2 MPD2 Correttezza ortografica

Gli errori ortografici possono essere verificati tramite Spell Right, lo strumento integrato di Visual Studio Code per il controllo ortografico. Se presenti, i *verificatori* li correggeranno.

3.2.4.3 Metriche per il software

3.2.4.3.1 MPS1 Copertura dei requisiti obbligatori

Indica la percentuale di requisiti coperti dall'implementazione. È ottenibile attraverso la seguente formula:

$$C = (1 - \frac{N_{\text{FOM}}}{N_{\text{FOI}}}) * 100$$

dove N_{FOM} è il numero di funzionalità obbligatorie mancanti all'implementazione e N_{FOI} è il numero di funzionalità obbligatorie individuate nell'attività di analisi.

3.2.4.3.2 MPS2 Copertura dei requisiti desiderabili

Indica la percentuale di requisiti coperti dall'implementazione. È ottenibile attraverso la seguente formula:

$$C = (1 - \frac{N_{\text{FDM}}}{N_{\text{FDI}}}) * 100$$

dove N_{FDM} è il numero di funzionalità desiderabili mancanti all'implementazione e N_{FDI} è il numero di funzionalità desiderabili individuate nell'attività di analisi.

3.2.4.3.3 MPS3 Tolleranza ai guasti

Indica la percentuale di operazioni di testing che si sono concluse in failure. È ottenibile attraverso la seguente formula:

$$F = \frac{N_{\text{FR}}}{N_{\text{TE}}} * 100$$

dove N_{FR} è il numero di failure rilevati durante l'attività di testing e N_{TE} è il numero di test case eseguiti.

3.2.4.3.4 MPS4 Comprensibilità delle funzionalità offerte

Indica la percentuale di operazioni comprese dall'utente senza la consultazione del manuale. È ottenibile attraverso la seguente formula:

$$C = \frac{N_{\text{FC}}}{N_{\text{FO}}} * 100$$

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

3.2.4.3.5 MPS5 Tempo di risposta

Indica la differenza di tempo media trascorsa tra l'esecuzione di una funzionalità e la restituzione dell'eventuale risultato finale. È ottenibile attraverso la seguente formula:

$$T_{\text{RISP}} = \frac{\sum_{i=1}^{n} T_i}{n}$$

dove T_i è il tempo in secondi trascorso tra la richiesta *i* di una funzionalità e il completamento della stessa con eventuale restituzione di un risultato.

3.2.4.3.6 MPS6 Capacità analisi failure

Indica la percentuale di failures incontrate delle quali sono state individuate le cause. È ottenibile attraverso la seguente formula:

$$I = \frac{N_{\text{FI}}}{N_{\text{FR}}} * 100$$

dove N_{FI} è il numero di failures delle quali sono state individuate le cause e N_{FR} è il numero di failures rilevate.

3.2.4.3.7 MPS7 Impatto delle modifiche

Indica la percentuale di modifiche atte a risolvere failures che però hanno introdotto ulteriori failures. È ottenibile attraverso la seguente formula:

$$I = \frac{N_{\text{FRF}}}{N_{\text{FR}}} * 100$$

dove N_{FRF} è il numero di failure risolte introducendo nuove failure e N_{FR} è il numero di failures risolte.

3.2.5 Analisi

L'analisi viene effettuata seguendo due metodi: analisi statica e analisi dinamica.

3.2.5.1 Analisi statica

L'analisi statica permette di rilevare anomalie all'interno dei documenti e del codice sorgente durante il ciclo di vita. È applicabile tramite due tecniche diverse:

- **Walkthrough**: Questa tecnica consiste in una lettura a largo spettro in cerca di anomalie. Si tratta di un metodo dispendioso in termini di efficienza. Verrà utilizzata nella fase iniziale del progetto a causa della non conoscenza e padronanza delle *Norme di Progetto* e del *Piano di Qualifica*;
- **Inspection**: Questa tecnica consiste in una lettura mirata e strutturata. Permette di localizzare errori utilizzando la lista di controllo e le misurazioni effettuate rendendo, con l'acquisizione di esperienza, questa tecnica sempre più efficiente.

Di seguito sono descritte le liste di controllo per la documentazione e il codice che il *verificatore* può utilizzare durante il processo di verifica.

- **Lista di controllo per la documentazione**

Anomalia	Descrizione
Elenchi puntati	Elenchi puntati non conformi alle *Norme di Progetto*
Sintassi	Errori di struttura della frase
Lessico	Uso di vocaboli inappropriati o in modo improprio
Grassetto/Corsivo	Uso non conforme alle *Norme di Progetto*
Formato data	Uso di formati data non conformi *Norme di Progetto*

Tabella 3: Lista anomalie comuni nella documentazione

- **Lista di controllo per il codice**

Anomalia	Descrizione
Clean code	Uso non conforme alle *Norme di Progetto*
Errori front-end	Parti della GUI errate
Output	Risultato delle funzioni Lambda non corretto

Tabella 4: Lista anomalie comuni nel codice

3.2.5.2 Analisi dinamica

L'analisi dinamica è una procedura che prevede l'analisi del software attraverso l'esecuzione di test che ne verificano il corretto funzionamento e segnalano le anomalie. Ogni test deve essere ripetibile, cioè, dato uno stesso input e lo stesso ambiente, si deve ottenere lo stesso output.
Per ogni test devono essere definiti i seguenti parametri:

- **ambiente**: il sistema hardware e software sul quale viene eseguito il test del prodotto;
- **stato iniziale**: lo stato iniziale del prodotto;
- **input**: l'input inserito;
- **output**: l'output atteso;
- **istruzioni aggiuntive**: istruzioni aggiuntive riguardanti le modalità del test e l'interpretazione dell'output.

3.2.6 Test

3.2.6.1 Test di unità

Si eseguono su singole unità di software. Tali test permettono di verificare la correttezza della singola unità. Se le singole unità sono corrette allora è possibile proseguire con i *test di integrazione*G.
Ogni test necessita della scrittura di un *driver*G e di un *stub*G che simulano un'unità chiamante e un'unità chiamata rispettivamente. Tali test vengono identificati nel seguente modo:

TU[codice]

dove **codice** è un numero intero incrementale e identifica l'unità da verificare.

3.2.6.2 Test di integrazione

Sono un'estensione del *test di unità*G. Le singole unità vengono raggruppate e vengono eseguiti test sull'interfaccia tra esse. Man mano che i test vengono superati, si formano raggruppamenti sempre maggiori su cui vengono iterati test di integrazione fino al raggiungimento della dimensione totale del sistema. Tali test vengono identificati nel seguente modo:

TI[codice]

dove **codice** è un numero intero incrementale e identifica la componente da verificare.

3.2.6.3 Test di sistema

Verifica il soddisfacimento completo dei requisiti. Viene fatto alla validazione del prodotto finale, sul sistema nella sua interezza. Tali test vengono identificati nel seguente modo:

TS[codice]

dove **codice** è un numero incrementale e identifica la componente da verificare.

3.2.6.4 Test di regressione

Sono effettuati ogni volta che viene fatta una qualunque modifica al sistema. A seguito di un cambiamento è necessario eseguire nuovamente tutti i test esistenti sul codice modificato per verificare la validità delle modifiche.

3.2.6.5 Test di accettazione

Vengono eseguiti in presenza del richiedente. Se superato, il prodotto è approvato e pronto al rilascio. Tali test vengono identificati nel seguente modo:

TA[tipo][importanza][codice]

dove:

- **tipo** indica il tipo di requisito di cui si tratta e può assumere i seguenti valori:
  - **0**: indica i requisiti obbligatori;
  - **1**: indica i requisiti desiderabili;
  - **2**: indica i requisiti opzionali.
- **importanza** indica l'importanza del requisito e può assumere i seguenti valori:
  - **F**: indica un requisito funzionale;
  - **P**: indica i requisito prestazionale;
  - **Q**: indica i requisito qualitativo;

  - **D**: indica i requisito dichiarativo o di vincolo.
- **Codice** è un numero intero incrementale e identifica la componente da verificare.

I test di accettazione hanno, inoltre, uno Stato che può assumere i seguenti valori:

- **i**: implementato;
- **n.i.**: non implementato;
- **s**: superato;
- **n.s.**: non superato.

3.2.7 Strumenti

3.2.7.1 Verifica ortografica e sintattica

Viene utilizzato l'estensione Spell Right di Visual Studio Code per la verifica dell'ortografia.

3.3 Validazione

3.3.1 Scopo

Il processo di validazione consente di accertare l'efficacia del prodotto finale. L'esito positivo di tale processo garantisce il soddisfacimento di tutti i requisiti e bisogni del committente.

3.3.2 Aspettative

La corretta esecuzione di tale processo consente di individuare:

- una procedura di validazione;
- i criteri per la validazione;
- la conformità del prodotto finito.

3.3.3 Descrizione

Il processo di validazione consiste nell'identificare gli oggetti da validare e valutare che i risultati rispettino le aspettative.

3.3.4 Attività

Le attività per compiere la validazione sono:

- eseguire i test sul prodotto;
- i *verificatori* eseguono i test e stendono un resoconto sui risultati;
- se i risultati non sono soddisfacenti allora si ritorna al primo punto e si rieseguono i test;
- i risultati vengono inviati al proponente.

4 Processi Organizzativi

4.1 Gestione di processo

In questa sezione vengono definite le norme che regolano le comunicazioni tra membri del gruppo con parti esterne.

4.1.1 Comunicazioni interne

Per le comunicazioni interne viene adottato un *workspace*G di *Slack*G, strumento molto versatile grazie anche alle numerose integrazioni con altre piattaforme, tra cui Git.
In questo workspace, sono stati creati vari canali per gestire al meglio le comunicazioni:

- **commit**: contiene tutte le notifiche relative ai commit che il gruppo fa sulla repository;
- **general**: contiene le discussioni di carattere generale sull'organizzazione del progetto, la scelta degli strumenti o per perdere decisioni in modo rapido;
- **incontri**: contiene le discussioni relative ai posti e orari di incontro del gruppo;
- **random**: contiene discussioni non strettamente inerenti al progetto;
- **materiali**: contiene i documenti prodotti e le guide per l'utilizzo degli strumenti necessari allo svolgimento del progetto, per consentire una più facile discussione;
- **divisionelavoro**: contiene le discussioni relative alla divisione del lavoro tra i membri del gruppo.

4.1.2 Comunicazioni esterne

In questa sezione vengono esposte le norme che regolano le comunicazioni con soggetti esterni al gruppo, nello specifico:

- la proponente zero12, con referente Stefano Dindo;
- Prof. Tullio Vardanega, Prof. Riccardo Cardin, ai quali verrà fornita tutta la documentazione richiesta in ciascuna revisione di avanzamento.

Le comunicazioni esterne avvengono unicamente per mezzo scritto. Queste devono avvenire esclusivamente attraverso l'indirizzo mail del gruppo:

hexadec.swe@gmail.com

Ogni membro del gruppo ha accesso all'account di posta elettronica.

4.2 Gestione delle riunioni

Le riunioni possono essere interne o esterne. All'inizio di ogni riunione il *responsabile* nomina a turno un segretario che si occuperà di prendere nota su ciò che viene discusso e successivamente redigere il verbale. Inoltre ha l'onere di far rispettare l'ordine del giorno.

4.2.1 Riunioni interne

La partecipazione alle riunioni interne, che avverranno principalmente di persona, è permessa solamente ai membri del gruppo HexaDec. Al *responsabile* spetta il compito di stilare preventivamente l'ordine del giorno da discutere, fissare una data in accordo con tutti i membri del gruppo e approvare il verbale redatto dal segretario.
Affinché una riunione sia valida, è richiesta la partecipazione di almeno cinque membri del gruppo, i quali sono tenuti a presentarsi in orario, segnalare eventuali ritardi o assenze.

4.2.2 Riunioni esterne

Le riunioni esterne coinvolgono i membri del gruppo HexaDec e uno o più soggetti esterni, appartenenti all'azienda proponente.
Queste si possono svolgere:

- nella sede della proponente;
- attraverso Google Hangouts.

In accordo con il rappresentante dell'azienda proponente, eventuali comunicazioni avverranno via e-mail.
Per quanto riguarda gli incontri, il gruppo si impegna a comunicare con anticipo al *responsabile* Stefano Dindo la volontà di effettuare una conference call.

4.2.3 Verbali delle riunioni

Al termine di ogni riunione il segretario dovrà redigere il relativo verbale, rispettando il seguente schema:

1. **Informazioni generali**:
   - data incontro;
   - luogo;
   - ora di inizio;
   - ora di fine;
   - durata;
   - modalità;
   - oggetto;
   - nominativi dei partecipanti interni;
   - nominativi dei partecipanti esterni;
   - nominativo del segretario.
2. **Riassunto della riunione**: riassunto redatto dal segretario secondo i punti dell'ordine del giorno, precisando le decisioni prese. Potranno essere presenti anche tematiche non inerenti all'ordine del giorno, ma che sono state trattate durante lo svolgimento della riunione.

4.2.3.1 Nomenclatura Al fine di identificare in modo univoco un verbale, la nomenclatura da adottare per ogni verbale è la seguente:

VER-NUMERO-TIPO-DATA

dove

- **VER** sta per verbale;
- **NUMERO** indica il numero del verbale;
- **TIPO** indica la tipologia del verbale, che può essere:
  - **I**: verbale della riunione interna;
  - **E**: verbale della riunione esterna.
- **DATA** è la data del verbale.

4.2.3.2 Tracciamento delle decisioni Per garantire l'attuazione delle decisioni, prese e descritte nei vari verbali, è necessario accompagnare ogni decisione da un codice identificativo del tipo:

DX.YY

dove

- **D** può assumere i valori:
  - **E** se si riferisce ad un verbale esterno;
  - **I** se si riferisce ad un verbale interno.
- **X** indica il numero identificativo del verbale associato;
- **YY** è un numero a due cifre incrementale che indica il numero della decisione all'interno del verbale.

4.3 Processi di pianificazione

In questa sezione sono presentati tutti i sistemi utilizzati dal gruppo per pianificare e coordinare lo svolgimento del progetto.
Ogni membro è tenuto a consultare e utilizzare attivamente questi strumenti, in modo da tener traccia del lavoro svolto.

4.3.1 Ruoli di progetto

I differenti ruoli verranno assegnati ai membri del gruppo a rotazione ad ogni raggiungimento di una scadenza terminale, garantendo la continuità delle attività in corso. Ogni membro dovrà ricoprire tutti i ruoli, per un periodo significativo, durante lo svolgimento del progetto. I ruoli sono i seguenti:

- **Responsabile**: gestisce le risorse umane, i rischi, la pianificazione, il controllo, il coordinamento e le relazioni esterne.
  Le sue responsabilità sono l'approvazione dell'emissione di documenti, l'elaborazione e l'emanazione di piani e scadenze, il coordinamento del gruppo e delle attività.
  Ha il compito di rappresentante del progetto presso l'azienda proponente;
- **Amministratore**: si occupa dell'efficienza e dell'operatività dell'ambiente di sviluppo.
  A lui viene affidato il compito di gestione del controllo della configurazione del prodotto, del versionamento e della documentazione di progetto;
- **Analista**: si occupa dell'attività di analisi, della redazione dello *Studio di Fattibilità* e dell'*Analisi dei Requisiti*.
  Ha il compito di capire a fondo il problema definendo i requisiti espliciti ed impliciti;
- **Progettista**: effettua lo studio di fattibilità del prodotto, costruisce l'architettura in termini di efficienza ed efficacia a partire dal lavoro dell'*analista*. Redige, inoltre, la specifica tecnica del progetto;
- **Programmatore**: gestisce l'attività di codifica del prodotto e le componenti di ausilio necessarie all'esecuzione delle prove di verifica e validazione; deve attenersi alle specifiche fornite dal *responsabile*.
  Punto focale del suo ruolo è produrre un codice manutenibile nel tempo;
- **Verificatore**: si occupa delle attività di verifica e validazione, partecipando all'intero ciclo di vita.
  Redige la parte retrospettiva del *Piano di Qualifica*, ossia illustra l'esito e la completezza delle verifiche e delle prove effettuate.

4.3.2 Pianificazione

Il gruppo ha scelto di appoggiarsi al servizio di gestione delle *issues*G fornito da GitLab, il quale permette di:

- inserire issue;
- attribuire un titolo e una descrizione;
- aggregare issues in una milestone;
- assegnare una issue da risolvere ad uno o più membri del gruppo;
- commentare una issue;
- assegnare delle etichette alle issues.

Il *responsabile* ha il compito di creare le issues e assegnarle ad uno o più membri del gruppo. Ogni issue deve possedere delle etichette, che identificano la priorità e il tipo di task da svolgere e una data ultima per il completamento.
Il gruppo ha scelto di utilizzare unicamente GitLab in modo da avere una situazione più chiara possibile in un unico spazio di lavoro.

4.3.3 Coordinameto

Il gruppo adotta il canale Slack ”divisionelavoro” per coordinare al meglio la divisione del lavoro tra membri del gruppo. Il *responsabile di progetto* si occuperà di dividere il carico di lavoro in modo equo tra i vari componenti del team e comunicherà tali scelte nel canale Slack apposito. Ogni membro dovrà leggere attentamente le istruzioni dettate dal *responsabile* e, in caso di problemi, riferirli.

4.3.3.1 Nomenclatura Al fine di identificare in modo univoco un processo, la nomenclatura da adottare per ogni è la seguente:

PROC[NUMERO]

dove

- **PROC** sta per processo;
- **NUMERO** indica il numero del processo.

4.4 Formazione

4.4.1 Formazione dei membri del gruppo

Ogni membro del gruppo è tenuto a formarsi autonomamente per padroneggiare al meglio le tecnologie che verranno utilizzate nel corso del progetto.
Al fine di facilitare la formazione, i membri del gruppo sono tenuti a condividere le conoscenze già possedute o a realizzare, in piena libertà, delle linee guida informali.

4.4.2 Guide utilizzate

La seguente documentazione dovrà essere consultata e appresa:

- per l'utilizzo di Slack: https://get.slack.help/hc/en-us/categories/200111606-Using-Slack;
- per l'utilizzo di LaTeX: https://www.latex-project.org;
- per l'utilizzo del software Git: https://git-scm.com/docs;
- per l'utilizzo del GitLab: https://docs.gitlab.com;
- per l'utilizzo di Android Studio: https://developer.android.com/studio/intro;
- per l'utilizzo di Kotlin: Kotlin in Action - Autori: Dmitry Jemerov, Svetlana Isakova;
- per l'utilizzo di AWS: https://aws.amazon.com/it/;
- per l'utilizzo di DynamoDB: https://www.dynamodbguide.com;
- per l'utilizzo di AWS Lambda: https://aws.amazon.com/it/lambda/.

A Sintassi UML 2.0

A.1 Diagrammi delle classi

Ogni classe verrà rappresentata tramite un rettangolo tripartito in senso orizzontale che conterrà:

1. **Nome della classe:**
   Sarà univoco , in inglese , evocativo, in grassetto e scritto con la prima lettera in maiuscolo. Nel caso si dovesse trattare di un'interfaccia, il nome sarà preceduto da <<interface>>; nel caso in cui si dovesse trattare di una classe astratta il nome dovrà essere scritto in corsivo;

2. **Variabili**
   Verranno elencati una dopo l'altra, ogni variabile occuperà una riga. Le variabili verranno dichiarate secondo il formato:

   nomevar:tipo

   dove il nome della variabile, dovrà essere univoco, in inglese e scritto con la lettera minuscola. Il tipo potrà essere semplice o definito dall'utente Ogni variabile dovrà essere proceduta obbligatoriamente da uno dei seguenti indicatori:

   - +: visibilità pubblica;
   - #: visibilità protetta;
   - ∼: visibilità package;
   - -: visibilità privata.

3. **Metodi**
   Verranno elencati uno dopo l'altro, ogni metodo occuperà una riga. I metodi verranno dichiarati secondo il seguente formato:

   nomeMetodo(lista-parametri-formali):tipo-ritorno

   dove il nome del metodo sarà univoco, in inglese e scritto con lettere minuscole, invece la lista dei parametri formali, sarà composta da 0 a n elementi separati tra loro da una virgola. Ogni metodo dovrà essere preceduto obbligatoriamente da uno degli indicatori divisibilità precedentemente descritti. Se si tratta di un metodo astratto, esso dovrà essere scritto in corsivo. Se si tratta di un metodo statico, dovrà essere identificato tramite sottolineatura dello stesso.

Nel caso in cui una classe non dovesse possedere variabili e/o metodi, apparirà nel diagramma comunque, seppur vuota, la sezione ad essi dedicata.

I diagrammi delle classi sono collegati fra loro da frecce che esplicitano le dipendenze.
Verranno utilizzati i seguenti tipi di freccia:

- **freccia semplice:** collegamento dalla classe A alla classe B, ciò indica che la classe A ha fra i propri campi dati una o più istanze della classe B. Tale molteplicità verrà esplicitata tramite:
  - **1** - A possiede un'istanza di B;
  - **0...*** - A può possedere 0 o più istanze di B;
  - **0...1** - A può possedere 0 o 1 istanza di B;
  - * - A può possedere più istanze di B;

  - **n** - A può possedere n istanze di B.

- **freccia tratteggiata:** collegamento dalla classe A alla classe B, ciò indica una dipendenza tra A e B secondo una primitiva che va indicata vicina alla freccia. Le primitive possono essere le seguenti:

  1. <<create>> indica che A crea istanza di B;
  2. <<call>> indica che A invoca istanze di B;
  3. <<realize>> indica che A è implementazione di un'interfaccia definita da B.

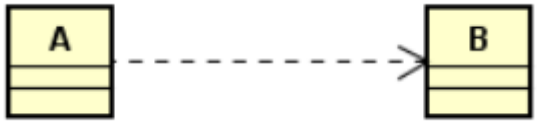

Figura 2: Relazione di dipendenza - diagrammi classi

- **freccia a diamante vuota:** collegamento dalla classe A alla classe B,indica un'aggregazione ovvero B è parte di A, quindi A non avrebbe senso di esistere senza B;

Figura 3: Relazione di aggregazione - diagrammi classi

- **freccia a diamante piena:** collegamento dalla classe A alla classe B,indica la composizione, cioè aggregazione con cardinalità (1,1);

Figura 4: Relazione di composizione - diagrammi classi

- **freccia vuota:** collegamento dalla classe A alla classe B,indica l'ereditarietà, ogni oggetto di A è anche di B;

Figura 5: Relazione di ereditarietà - diagrammi classi

- **freccia vuota tratteggiata:** collegamento dalla classe A alla classe B,indica l'implementazione di un'interfaccia.

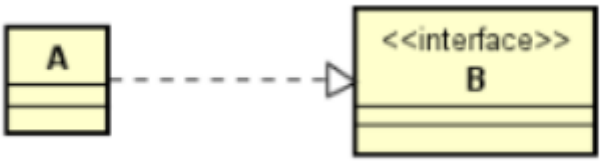

Figura 6: Relazione implementazione interfaccia - diagramma classi

A.2 Diagrammi dei package

Ogni package è rappresentato tramite un rettangolo con un'etichetta per il nome, che dovrà contenere i diagrammi delle classi appartenenti ad essi ed eventuali sotto-package se presenti. Le dipendenze fra i vari package dovranno essere segnalate con una freccia tratteggiata. Tale freccia indica una dipendenza di A ne confronti di B. Si eviteranno le dipendenze cicliche.

Figura 7: Dipendenza package - diagrammi pack

A.3 Diagramma delle attività

I diagrammi delle attività conterranno i seguenti elementi, che verranno indicati come di seguito:

- **Nodo iniziale:** rappresentato da un pallino nero che segnala l'inizio dell'esecuzione;
- **Activity:** rappresentato da un rettangolo che contiene la descrizione. Essa deve essere il più breve possibile e composta per lo più da parole chiave;

Figura 8: Activity - diagrammi attività

- **Subactivity:** rappresentato da un rettangolo che ne contiene il nome, indicato sempre con la lettera maiuscola e un piccolo simbolo in basso a destra;

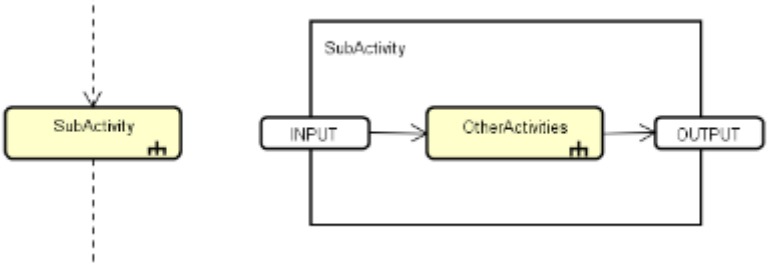

Figura 9: Subactivity - diagrammi attività

- **Branch:** rappresentato da un rombo con n frecce in uscita e una freccia in ingresso. Rappresenta un punto di decisione, dove si può proseguire solo da una delle n uscite;
- **Marge:** il punto di unione degli n rami che genera il branch, rappresentato da un rombo con n frecce in ingresso e una in uscita;

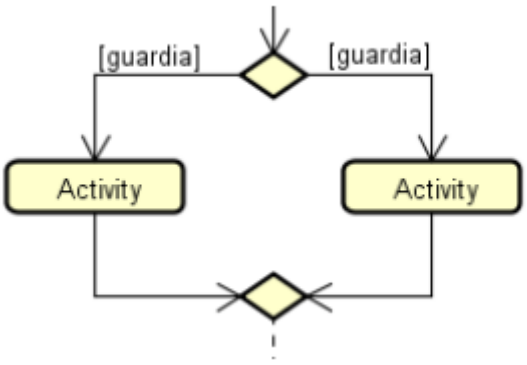

Figura 10: Branch e Merge - diagrammi attività

- **Fork:** appresentato da una linea lunga orizzontale o verticale.E'un punto in cui l'attività si parallelizza senza vincoli di esecuzione temporale, ha una freccia in ingresso e n in uscita;
- **Join:** rappresentato da una linea verticale o orizzontale e prende n frecce in ingresso e una in uscita.

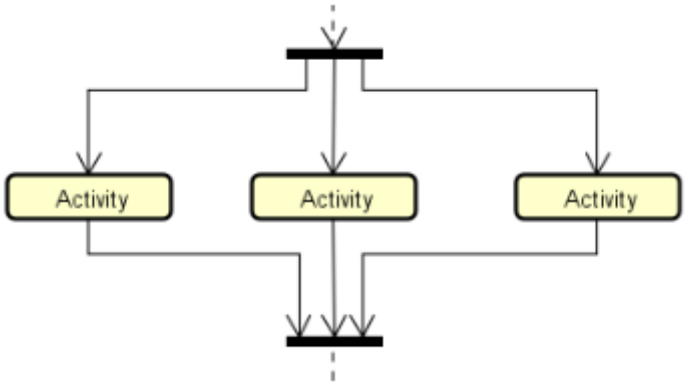

Figura 11: Fork e Join - diagrammi attività

- **Pin:** rappresenta il passaggio di parametro tra un'activity e l'altra. Rappresentato da un piccolo quadrato dove entrano la freccia dell'activity.

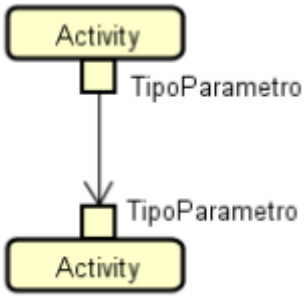

Figura 12: Pin - diagrammi attività

- **Nodo di fine flusso:** punto in cui un ramo di esecuzione va a morire , ma l'esecuzione continua sugli altri rami restanti. Rappresentato da un cerchio vuoto con una X all'interno;
- **Nodo finale:** punto in cui termina l'esecuzione. Rappresentato da due cerchi, quello più esterno vuoto e quello interno pieno.

Figura 13: Nodo fine e nodo fine flusso - diagrammi attività

A.4 Diagrammi di sequenza

I diagrammi di sequenza avranno un senso di lettura verticale, dall'alto al basso. Gli oggetti coinvolti verranno rappresentati tramite un rettangolo, dove al suo interno si troverà il nome per identificarli con il seguente formato: istanza: NomeClasse.
Al di sotto di ogni istanza sarà presente una linea tratteggiata, sormontata a tratti da barre di attivazione che indica che l'oggetto è attivo. Dalle barre di attivazione partiranno delle frecce, che rappresentano un messaggio o segnale verso la linea di oggetti già istanziati. Verranno utilizzati le seguenti frecce:

- **Freccia piena:** indica un messaggio sincrono che corrisponde alla chiamata di un metodo. Al di sopra di essa di dovrà specificare il metodo invocato secondo il seguente metodo: nomeMetodo(lista dei parametri);
- **Freccia semplice:** indica un messaggio asincrono, invocazione del metodo, ma non attende il return;
- **Freccia tratteggiata (con ritorno del metodo chiamato):** . Sopra tale freccia si dovrà indicare il tipo di ritorno del metodo con la seguente sintassi: tipoRitorno;
- **Freccia tratteggiata (con <<create>>):** indica la creazione di un nuovo oggetto e termina in un rettangolo che ne contiene il nome con il seguente formato: nomeClasse;
- **Freccia piene (con <<destroy>>):** indica la distruzione di un oggetto e termina sempre con una **X**, dove muore anche la linea dell'oggetto.

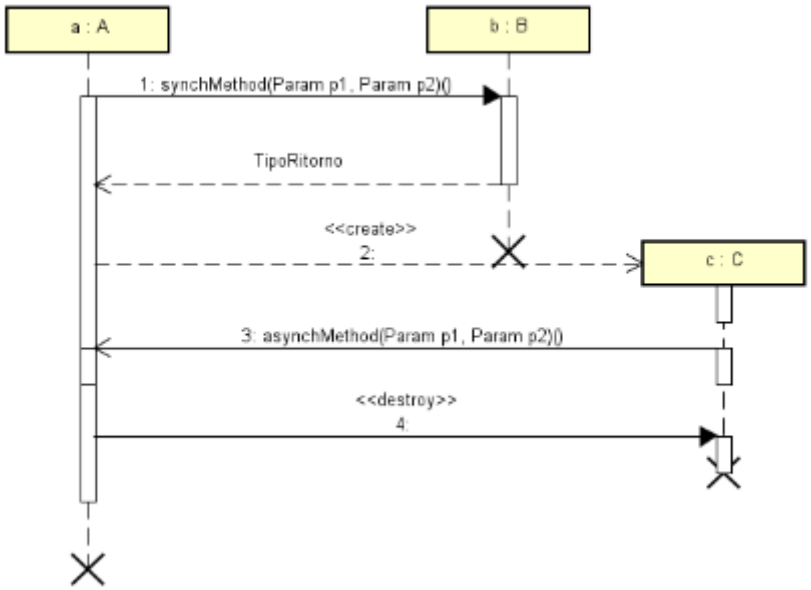

Figura 14: Tipi di segnali - diagrammi sequenza

Inoltre sarà possibile determinare, sui diagrammi di sequenza, dei frame di iterazione associati ad una condizione

- **alt:** frammenti alternativi fra loro e viene eseguito quello con la condizione verificata;
- **opt:** viene eseguito un frammento solo se la condizione è specificata;
- **par:** vengono eseguiti frammenti in parallelo;
- **loop:** viene eseguito più volte più volte un frammento e si arresta con una condizione di guardia;
- **region:** viene eseguita in mutua esclusione un frammento critico.

A.5 Diagrammi dei casi d'uso

I diagrammi dei cadi d'uso sono identificati da un nome in alto a sinistra e dai seguenti componenti:

- **attori:** rappresenta l'utente;
- **caso d'uso:** rappresentato da un cerchio e al suo interno il nome che lo identifica;
- **freccia tratteggiata(<<exstend>>)** indica l'estensione tra due casi d'uso;
- **linea intera:** indica l'associazione tra l'attore e il caso d'uso;
- **linea tratteggiata:** indica un commento;

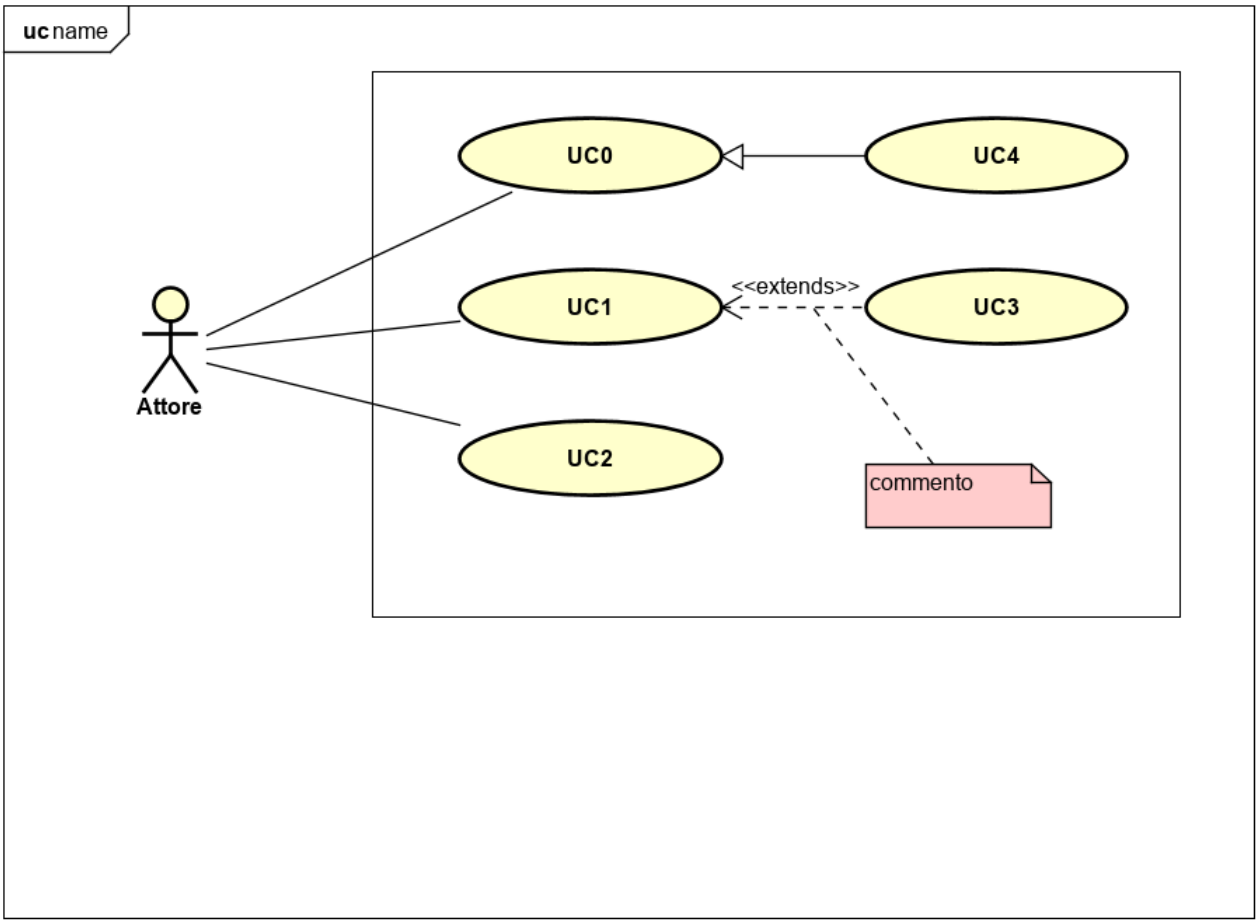

Figura 15: Casi d'uso - diagrammi casi d'uso