Norme di Progetto

ShowRoom3D

SmokingFingertips | smoking.fingertips@gmail.com

Versione	2.0.0
Stato	approvato
Uso	interno
Responsabile	Gabriele Saracco
Redattori	Luca Polese
	Edoardo Gasparini
	Davide Baggio
	Sebastien Biollo
	Alberto Angeloni
Verificatori	Edoardo Gasparini
	Luca Polese
	Alberto Angeloni
	Gabriele Saracco
	Luca Annicchiarico
Destinatari	*Smoking Fingertips*
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
Data di Approvazione	2023-05-20
Anno accademico:	2022/2023

Sommario:

In questo documento verranno inserite tutte le norme atte a regolare lo sviluppo del progetto

Storico delle modifiche

Versione	Data	Nominativo	Ruolo	Descrizione
2.0.0	2023-05-20	Gabriele Saracco	Responsabile	Approvazione del documento
1.3.0	2023-05-10	Luca Polese *Sebastien Biollo*	Responsabile *Verificatore*	Revisione complessiva di coerenza e coesione
1.2.3	2023-04-20	Edoardo Gasparini *Luca Polese*	Responsabile *Verificatore*	Aggiornamenti relativi alla codifica, correzione typo
1.2.2	2023-04-10	Sebastien Biollo *Alberto Angeloni*	Amministratore *Verificatore*	Aggiornamento ed integrazione §4.2
1.2.1	2023-04-06	Davide Baggio *Luca Annicchiarico*	Amministratore *Verificatore*	Incremento capitolo §4.3
1.2.0	2023-03-30	Edoardo Gasparini *Davide Baggio*	Verificatore *Verificatore*	Revisione complessiva di coerenza e coesione
1.1.2	2023-03-25	Luca Polese *Edoardo Gasparini*	Amministratore *Verificatore*	Aggiunti §3.5.6, §3.5.7
1.1.1	2023-03-20	Edoardo Gasparini *Alberto Angeloni*	Amministratore *Verificatore*	Aggiornamento riferimenti §1.4
1.1.0	2023-03-15	Sebastien Biollo *Luca Annicchiarico*	Verificatore *Verificatore*	Revisione complessiva di coerenza e coesione
1.0.2	2023-03-05	Luca Polese *Edoardo Gasparini*	Amministratore *Verificatore*	Aggiunti §2.2.5.5, §2.2.5.8, §3.1.12, §3.1.13, §3.4.7.4 Aggiornati §3.4, §3.1, §3.4.7, §3.4.8
1.0.1	2023-02-28	Luca Polese *Edoardo Gasparini*	Amministratore *Verificatore*	Correzioni tipografiche e riferimenti errati
1.0.0	2023-02-20	Sebastien Biollo	Responsabile	Approvazione del documento
0.7.0	2023-02-16	Luca Polese	Verificatore	Verifica §4.1.4.5, §4.3
0.6.1	2023-02-15	Edoardo Gasparini	Amministratore	Stesura §4.1.4.5, §4.3
0.6.0	2023-02-11	Edoardo Gasparini	Verificatore	Verifica §2.2.6, §4.2

0.5.1	2023-02-10	Luca Polese	Amministratore	Stesura §2.2.6, §4.2
0.5.0	2023-02-08	Gabriele Saracco	Verificatore	Verifica §3.3, §B, §2.2.5
0.4.1	2023-02-07	Alberto Angeloni	Amministratore	Stesura §3.3, §B, §2.2.5
0.4.0	2023-01-26	Edoardo Gasparini	Verificatore	Verifica §4.1.4.1, §4.1.4.4, §A
0.3.1	2022-12-11	Luca Polese	Amministratore	Stesura §4.1.4.1, §4.1.4.4, §A
0.3.0	2022-11-25	Edoardo Gasparini	Verificatore	Verifica §2, §4, §2.1, §3.1.10, §3.4
0.2.2	2022-11-10	Davide Baggio	Amministratore	Stesura §2, §4
0.2.1	2022-11-08	Sebastien Biollo	Amministratore	Stesura §2.1, §3.1.10, §3.4
0.2.0	2022-11-15	Alberto Angeloni	Verificatore	Verifica §3, §3.1.6.1, §3.1.6.2
0.1.1	2022-11-05	Edoardo Gasparini	Amministratore	Stesura §3, §3.1.6.1, §3.1.6.2
0.1.0	2022-11-04	Edoardo Gasparini	Verificatore	Verifica §1
0.0.1	2022-10-27	Luca Polese	Responsabile	Stesura §1

Indice

1 Introduzione **10**
1.1 Scopo del documento . . . 10
1.2 Scopo del progetto . . . 10
1.3 Glossario . . . 10
1.4 Riferimenti . . . 10
1.4.1 Riferimenti normativi . . . 10
1.4.2 Riferimenti informativi . . . 11

2 Processi primari **13**
2.1 Fornitura . . . 13
2.1.1 Scopo . . . 13
2.1.1.1 Fasi della fornitura . . . 13
2.1.2 Descrizione . . . 13
2.1.3 Aspettative . . . 14
2.1.4 Rapporti con il proponente . . . 14
2.1.5 Documentazione fornita . . . 14
2.1.5.1 `Studio di Fattibilità` . . . 14
2.1.5.2 `Analisi dei Requisiti` . . . 15
2.1.5.3 `Piano di Progetto` . . . 15
2.1.5.4 `Piano di Qualifica` . . . 16
2.1.5.5 `Glossario` . . . 17
2.1.5.6 `Lettera di Presentazione` . . . 17
2.1.6 Strumenti . . . 17
2.1.6.1 Google Calendar . . . 17
2.1.6.2 Google Slides . . . 17
2.1.6.3 Google Meet . . . 17
2.1.6.4 GanttProject . . . 17
2.1.6.5 Excel . . . 17
2.2 Sviluppo . . . 18
2.2.1 Scopo . . . 18
2.2.2 Descrizione . . . 18
2.2.3 Aspettative . . . 18
2.2.4 Analisi dei Requisiti . . . 18
2.2.4.1 Scopo . . . 18
2.2.4.2 Descrizione . . . 19
2.2.4.3 Aspettative . . . 19
2.2.4.4 Casi d'uso . . . 19
2.2.4.5 Requisiti . . . 20
2.2.4.6 Metriche . . . 22
2.2.5 Progettazione . . . 22
2.2.5.1 Scopo . . . 22
2.2.5.2 Descrizione . . . 23
2.2.5.3 Aspettative . . . 23

2.2.5.4 Documentazione . . . 23
2.2.5.5 Qualità dell'architettura . . . 24
2.2.5.6 Diagrammi UML . . . 25
2.2.5.7 Design pattern . . . 32
2.2.5.8 Qualità della progettazione . . . 32
2.2.5.9 Test . . . 33
2.2.5.10 Metriche . . . 34
2.2.6 Codifica . . . 34
2.2.6.1 Scopo . . . 34
2.2.6.2 Descrizione . . . 34
2.2.6.3 Aspettative . . . 34
2.2.6.4 Stile di codifica . . . 35
2.2.6.5 Metriche . . . 37

3 Processi di supporto **38**
3.1 Documentazione . . . 38
3.1.1 Scopo . . . 38
3.1.2 Descrizione . . . 38
3.1.3 Aspettative . . . 38
3.1.4 Ciclo di vita dei documenti . . . 38
3.1.5 Template LaTeX . . . 39
3.1.6 Struttura dei documenti . . . 39
3.1.6.1 Intestazione dei documenti . . . 40
3.1.6.2 Storico delle modifiche . . . 40
3.1.6.3 Indice . . . 40
3.1.6.4 Corpo del documento . . . 41
3.1.6.5 Corpo del verbale . . . 41
3.1.7 Documenti del progetto . . . 42
3.1.8 Sigle dei documenti . . . 42
3.1.9 Sigle di progetto . . . 42
3.1.10 Convenzioni stilistiche . . . 42
3.1.10.1 Annotazioni . . . 42
3.1.10.2 Indentazioni . . . 43
3.1.10.3 Nomi assegnati ai file . . . 43
3.1.10.4 Stile del testo . . . 44
3.1.10.5 Elenchi puntati . . . 45
3.1.10.6 Formato delle date . . . 45
3.1.11 Strumenti . . . 45
3.1.12 Controllo ortografico . . . 45
3.1.13 Elementi grafici . . . 46
3.1.13.1 Immagini . . . 46
3.1.13.2 *Diagrammi UML*$_G$ . . . 46
3.1.13.3 Tabelle . . . 46
3.1.14 Metriche . . . 46
3.2 Verifica . . . 46
3.2.1 Scopo . . . 46

3.2.2 Aspettative . . . 47
3.2.3 Descrizione . . . 47
3.2.4 Analisi statica . . . 47
3.2.4.1 Walkthrough . . . 47
3.2.4.2 Inspection . . . 48
3.2.5 Analisi dinamica . . . 48
3.2.5.1 Test di unità . . . 49
3.2.5.2 Test di integrazione . . . 50
3.2.5.3 Test di sistema . . . 50
3.2.5.4 Test di regressione . . . 50
3.2.5.5 Test di accettazione . . . 51
3.2.5.6 Codici relativi ai test . . . 51
3.2.5.7 Stato del test . . . 52
3.2.6 Metriche . . . 52
3.3 Validazione . . . 52
3.3.1 Scopo . . . 52
3.3.2 Aspettative . . . 52
3.3.3 Descrizione . . . 53
3.3.4 Test di accettazione . . . 53
3.3.5 Metriche . . . 53
3.4 Gestione della configurazione . . . 53
3.4.1 Scopo . . . 53
3.4.2 Aspettative . . . 53
3.4.3 Descrizione . . . 54
3.4.4 Codice di versionamento . . . 54
3.4.5 Tecnologie adottate . . . 55
3.4.5.1 Jira . . . 55
3.4.5.2 Git . . . 55
3.4.5.3 Github . . . 55
3.4.6 Lista dei repository . . . 55
3.4.7 Organizzazione dei repository documentali . . . 56
3.4.7.1 Repository `Docs` . . . 56
3.4.7.2 Repository `Docs-LaTeX` . . . 56
3.4.7.3 Gerarchia dei file . . . 56
3.4.7.4 Tipologie di file . . . 57
3.4.8 Organizzazione del repository per il codice . . . 57
3.4.9 Sincronizzazione . . . 58
3.4.9.1 Branch . . . 58
3.4.9.2 Pull Request . . . 58
3.4.10 Comandi base git_G . . . 58
3.4.11 Modifiche al repository . . . 59
3.5 Gestione della qualità . . . 60
3.5.1 Scopo . . . 60
3.5.2 Aspettative . . . 60
3.5.3 Descrizione . . . 60

3.5.4 Controllo della qualità 60
3.5.5 Principi della qualità 60
3.5.6 Istanziazione di un processo 61
3.5.7 Gestione del cambiamento 61
3.5.7.1 Fattori che portano al cambiamento 62
3.5.7.2 Condizioni per un cambiamento efficace 62
3.5.8 PDCA 63
3.5.8.1 Plan 63
3.5.8.2 Do 63
3.5.8.3 Check 63
3.5.8.4 Act 63
3.5.9 Strumenti 63
3.5.10 Struttura delle metriche 63
3.5.11 Struttura degli obiettivi 64
3.5.12 Metriche 64

4 Processi organizzativi **65**
4.1 Gestione dei Processi 65
4.1.1 Scopo 65
4.1.2 Descrizione 65
4.1.3 Aspettative 65
4.1.4 Pianificazione 66
4.1.4.1 Scopo 66
4.1.4.2 Descrizione 66
4.1.4.3 Aspettative 66
4.1.4.4 Assegnazione ruoli 66
4.1.4.5 Ticketing 69
4.1.5 Coordinamento 70
4.1.5.1 Scopo 70
4.1.5.2 Descrizione 70
4.1.5.3 Aspettative 71
4.1.5.4 Comunicazioni 71
4.1.5.5 Riunioni 72
4.1.6 Metriche 74
4.2 Miglioramento 74
4.2.1 Scopo 74
4.2.2 Descrizione 75
4.2.3 Aspettative 75
4.2.3.1 Istituzione del processo 75
4.2.3.2 Valutazione del processo 75
4.2.3.3 Miglioramento 75
4.2.3.4 Metriche 76
4.3 Formazione 76
4.3.1 Scopo 76
4.3.2 Aspettative 76
4.3.3 Formazione dei membri del gruppo 76

A Standard per la qualità **77**
A.1 Funzionalità . . . 77
A.2 Affidabilità . . . 78
A.3 Usabilità . . . 78
A.4 Efficienza . . . 78
A.5 Manutenibilità . . . 79
A.6 Portabilità . . . 79

B Metriche per la qualità **80**
B.1 Metriche interne . . . 80
B.2 Metriche esterne . . . 80
B.3 Metriche della qualità in uso . . . 80
B.4 Metriche per la qualità di processo . . . 81
B.4.1 Miglioramento . . . 81
B.4.2 Fornitura . . . 81
B.4.3 Codifica . . . 82
B.4.4 Documentazione . . . 83
B.5 Metriche per la qualità di prodotto . . . 83
B.5.1 Funzionalità . . . 83
B.5.2 Progettazione . . . 83
B.5.3 Usabilità . . . 84
B.5.4 Manutenibilità . . . 84
B.5.5 Affidabilità . . . 84

Elenco delle figure

1 Diagramma delle classi di una Relazione di Dipendenza 27
2 Diagramma delle classi di una Relazione di Associazione 27
3 Diagramma delle classi di una Relazione di Aggregazione 27
4 Diagramma delle classi di una Relazione di Composizione 28
5 Diagramma delle classi di una Relazione di Ereditarietà 28
6 Diagramma delle classi di una Relazione di Interface Realization 28
7 Rappresentazione di un Attore 29
8 Rappresentazione di un Caso d'uso 30
9 Rappresentazione di un'Inclusione 30
10 Rappresentazione di un'Estensione 31
11 Rappresentazione di una Generalizzazione di un Attore 31
12 Rappresentazione di una Generalizzazione di un Caso d'Uso 32

Elenco delle tabelle

2 Esempio di classificazione di un requisito 22
3 Metriche per l'Analisi dei Requisiti 22
4 Metriche per la Progettazione 24
5 Metriche per la Progettazione 34
6 Metriche per la Codifica 37
7 Metriche per la Documentazione 46
8 Metriche per la verifica 52
9 Metriche per la verifica 53
10 Metriche per la Pianificazione 74
11 Metriche per il Miglioramento 76

1 Introduzione

1.1 Scopo del documento

Questo documento verrà utilizzato dal gruppo *Smoking Fingertips* allo scopo di raccogliere le regole relative al *way of working*$_G$ adottato per lo svolgimento del *progetto*$_G$ didattico.
Tutti i processi (e le relative *attività*$_G$) presenti in questo documento fanno riferimento allo standard ISO/IEC 12207:1995$_G$, da cui il gruppo è partito per definire delle best practices da adottare.
Questa non è la versione finale: è stato intrapreso un approccio di tipo incrementale che richiede un aggiornamento continuo delle *norme*$_G$. Ogni aggiornamento avverrà in funzione degli adeguamenti decisi dal gruppo in corso d'opera.
Il documento dovrà rimanere disponibile per tutti i membri del gruppo per l'intera durata del progetto. Gli stessi si impegnano a visionarlo regolarmente e a rispettarlo così da mantenere coerenza, coesione e uniformità nel progetto.

1.2 Scopo del progetto

Il *capitolato*$_G$ C6 *ShowRoom3D*$_G$ affidato al team si prefigge come scopo quello di realizzare uno *showroom*$_G$ virtuale. L'utente accedendo all'applicazione sarà in grado di muoversi nello spazio visionando gli oggetti esposti. Ognuno degli elementi potrà essere configurato secondo le preferenze dell'utente. Una volta operata la scelta dei parametri, sarà altresì possibile aggiungere l'articolo modificato all'interno del carrello per eventuali acquisti.

1.3 Glossario

Per evitare possibili ambiguità che potrebbero sorgere durante la lettura dei documenti, alcuni termini utilizzati sono stati inseriti nel documento `Glossario` (che attualmente è nella sua versione *2.0.0*).
Il Glossario rappresenta una raccolta delle definizioni dei termini più rilevanti che hanno un significato particolare. Sarà possibile individuare il riferimento al Glossario per mezzo di una *G* a pedice del termine (esempio *way of working*$_G$).

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato d'appalto C6 - ShowRoom3D:**
  https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C6.pdf
- **Standard ISO/IEC 12207:1995 - Processi del ciclo di vita del software:**
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
  https://galileodiscovery.unipd.it/permalink/39UPD_INST/prmo4k/alma990005031230206046

- **Standard ISO 8601 - Formato data e ora:**
  https://www.iso.org/iso-8601-date-and-time-format.html
  https://en.wikipedia.org/wiki/ISO_8601

1.4.2 Riferimenti informativi

- **Documentazione git:**
  https://git-scm.com/docs
- **Libro ProGit:**
  https://git-scm.com/book/it/v2
  - Capitolo 1 - Per Iniziare;
  - Capitolo 2 - Git Basics;
  - Capitolo 3 - Git Branching;
  - Capitolo 4 - Git on the Server;
  - Capitolo 6 - $GitHub_G$;
  - Capitolo 7 - Git Tools, Sezione 7.3 Stashing and Cleaning.
- **Documentazione GitHub:**
  https://help.github.com/en/github
- **Materiale didattico del corso Metodi e Tecnologie per lo Sviluppo Software 2021/2022:**
  https://elearning.unipd.it/math/course/view.php?id=875
  - Lezione 5 - GIT_G;
  - Laboratorio 2 - GitHub.
- **Materiale didattico del corso Ingegneria del Software 2022/2023:**
  - **Processi SW:**
    https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T02.pdf
    * Standard di processo - pagine 9-10;
    * ISO 12207:1995, Processi primari, di supporto e organizzativi - pagine 11-20;
    * Organizzazione di processo - pagine 23-25.
  - **Gestione di progetto:**
    https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T04.pdf
    * Ruoli - pagine 6-9.
  - **Amministrazione di progetto:**
    https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/FC2.pdf
    * Normare il way of working - pagina 3;
    * Supporto a gestione di progetto - pagina 5;

    * Attività di configurazione - pagine 8-12;
    * Gestione delle modifiche - pagina 13.
  - **Progettazione e programmazione: Diagrammi delle classi (UML):** https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf
    * Cosa sono gli Use Case - pagine 8-10;
    * Specifica Use Case - pagine 12-14;
    * Diagrammi dei Casi d'Uso - pagine 16-27.

- **UNI EN ISO 9001:2000 - Qualità, sistema di gestione per la qualità e certificazione:**
  - Capitolo 8 - Misurazioni, analisi e miglioramento
    * Sezione 8.1 - Criteri generali
    * Sezione 8.2 - Le misurazioni
  - Capitolo 11 - La gestione del cambiamento

- **Documentazione LaTeX:** http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf
  - Capitolo 3 - Basi;
  - Capitolo 4 - Testo;
  - Capitolo 5 - Matematica;
  - Capitolo 6 - Tabelle e figure;
  - Appendice A - Norme tipografiche.

- **Documentazione Jira:** https://www.atlassian.com/software/jira/guides

- **Documentazione Unreal Engine 5:** https://docs.unrealengine.com/5.0/en-US/

.

Tutti i riferimenti a siti web sono verificati e aggiornati alla data 2023-05-22

2 Processi primari

2.1 Fornitura

2.1.1 Scopo

Il *processo*$_G$ di fornitura determina, come stabilisce lo *standard*$_G$ ISO/IEC 12207:1995$_G$, l'insieme delle *attività*$_G$, dei *compiti*$_G$ e delle risorse necessarie al *fornitore*$_G$ per svolgere il progetto. Nello specifico, il suo scopo è quello di *tracciare*$_G$ e descrivere le attività eseguite dai componenti del gruppo *Smoking Fingertips*. Sarà così possibile determinare quanto lavoro è stato terminato, quanto ne rimane ancora da completare e fare un confronto con le richieste espresse dal *proponente*$_G$.
Questo processo può essere avviato al completamento della stesura dello `Studio di Fattibilità`. Nello specifico ciò avviene dopo aver compreso quanto richiesto dal proponente.
Il fornitore dovrà stabilire un contratto con il proponente in cui vengono accettati i *requisiti*$_G$ concordati e viene definita una data di consegna del prodotto finale.
Solamente quando l'accordo è stato raggiunto, allora sarà possibile passare alla *fase*$_G$ esecutiva redigendo il `Piano di Progetto` (descritto nella sezione §2.1.5.3).

2.1.1.1 Fasi della fornitura

Secondo lo standard ISO/IEC 12207:1995 il processo di fornitura è costituito dalle seguenti *fasi*$_G$:

1. **Avvio**;
2. **Approntamento di risposte alle richieste**;
3. **Contrattazione**;
4. **Pianificazione**;
5. **Esecuzione e controllo**;
6. **Revisione e valutazione**;
7. **Consegna e completamento**.

2.1.2 Descrizione

In questa sezione sono raccolte le *norme*$_G$ che i membri del gruppo sono tenuti a osservare durante lo svolgimento del progetto didattico al fine di figurare tra i *fornitori*$_G$ del *proponente*$_G$ *Sanmarco Informatica S.p.A* e dei *committenti*$_G$ *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*.

2.1.3 Aspettative

Il gruppo *Smoking Fingertips* instaurerà e manterrà un continuo dialogo con l'azienda *Sanmarco Informatica S.p.A.* (nelle modalità descritte alla sezione §2.1.4) così da ottenere un riscontro sul lavoro svolto fino a quel momento, ma anche per verificare se i vincoli e i requisiti individuati corrispondano a quanto richiesto nel *capitolato*$_G$ e dall'azienda.
Nello specifico i contatti avverranno con il referente aziendale *Alex Beggiato*.

2.1.4 Rapporti con il proponente

Il *proponente*$_G$ mette a disposizione e-mail e *Google Meet*$_G$ come canali di comunicazione tramite i quali chiarire dei dubbi e stabilire nuovi incontri telematici.
La cadenza dei meeting non è regolare, ma viene fissata in base alle necessità del gruppo o dell'azienda.
I meeting sono tenuti da uno o più membri del team che hanno il compito di riportare le informazioni per cui è necessario un confronto con il proponente.
Tra i motivi di discussione figurano:

- chiarimenti relativi a requisiti o vincoli del *capitolato*$_G$;
- dubbi sulla gestione delle tecnologie utilizzate;
- richiesta di feedback di quanto prodotto.

Per ogni colloquio con il proponente verrà steso un resoconto nel `Verbale Esterno` che riferisce alla data in cui tale incontro è avvenuto.
Per ogni *baseline*$_G$, i verbali redatti potranno essere visualizzati nella cartella che ne riporta il nome nel *repository*$_G$ documentale https://github.com/SmokingFingertips/Docs al percorso: `"NomeBaseline"/Documentazione Esterna/Verbali`.

2.1.5 Documentazione fornita

Si elencano i documenti che il gruppo *Smoking Fingertips* consegnerà al *committente*$_G$ *Sanmarco Informatica S.p.A* e ai proponenti Prof. Tullio Vardanega e Prof. Riccardo Cardin .
Quanto segue, vuole assicurare trasparenza per le *attività*$_G$ di Analisi, Pianificazione, Verifica, Validazione e Controllo della qualità per l'intero *ciclo di vita*$_G$ del progetto.

2.1.5.1 `Studio di Fattibilità`

Lo `Studio di Fattibilità` è un documento tecnico utilizzato per valutare la fattibilità di un progetto di sviluppo software. Esso fornisce una panoramica dettagliata del progetto, valutando la sua fattibilità in termini di risorse tecniche, economiche e umane disponibili. In particolare, lo `Studio di Fattibilità` identifica le esigenze del cliente, le possibili soluzioni e le eventuali criticità.
La sua stesura avverrà in seguito alla presentazione dei capitolati di appalto fissata per il giorno 2022-10-18.
Il documento conterrà:

- **Informazioni generali**: tra cui il nome del progetto, del proponente e dei committenti;
- **Descrizione**: ossia una sintesi del prodotto da sviluppare secondo quanto descritto dal capitolato d'appalto;
- **Finalità del progetto**: le finalità richieste dal capitolato d'appalto;
- **Tecnologie**: le tecnologie da utilizzare per lo sviluppo del progetto;
- **Vincoli del progetto**: i limiti e le restrizioni che influenzano la pianificazione, la progettazione e l'implementazione del progetto;
- **Pro e contro**: ossia una lista di fattori emersi durante l'analisi preliminare svolta utilizzando le conoscenze dei singoli membri del gruppo;
- **Criticità**: aspetti critici del prodotto richiesto;
- **Conclusione**: accettazione o rifiuto del capitolato, che tiene in considerazione l'interesse del gruppo verso la realizzazione del prodotto.

2.1.5.2 `Analisi dei Requisiti`

L'`Analisi dei Requisiti` è un documento che descrive in dettaglio i requisiti del progetto, i *casi d'uso*$_G$ del sistema e definisce in modo dettagliato le funzionalità che il prodotto offre.
Il documento ha lo scopo di eliminare eventuali ambiguità che possono insorgere in seguito alla lettura del *capitolato*$_G$.
Maggiori dettagli relativi all'*attività*$_G$ Analisi dei Requisiti e alle *norme*$_G$ per la stesura del documento possono essere trovati nella sezione §2.2.4. Il documento conterrà:

- **Descrizione del prodotto**;
- **Lista dei casi d'uso**: identifica tutti i possibili scenari di utilizzo del sistema da parte degli utenti. Essa elenca tutti i casi d'uso, ovvero le diverse azioni o *attività*$_G$ che gli utenti possono svolgere con il sistema. Ogni caso d'uso è accompagnato da una descrizione dettagliata delle azioni che l'utente compie, consentendo ai progettisti di capire come il sistema deve funzionare in ogni situazione;
- **Lista dei requisiti**: insieme dettagliato di tutte le richieste e i vincoli definiti dal *proponente*$_G$ o estratti dal team per ottenere il sistema software commissionato. La lista dei requisiti conterrà una descrizione delle funzioni del sistema, delle interazioni utente, dei vincoli tecnici, delle prestazioni richieste e di qualsiasi altra specifica necessaria per garantire il successo del progetto.

2.1.5.3 `Piano di Progetto`

Il `Piano di Progetto` è un documento stilato e aggiornato in corso d'opera dal responsabile con il supporto degli amministratori; deve espressamente trattare i seguenti punti:

- **Analisi dei rischi**: vengono analizzati e quantificati eventuali rischi che potrebbero esser riscontrati durante lo sviluppo del progetto.
  Ogni rischio verrà associato ad uno o più compiti. Questo permetterà di mettere a disposizione del team delle soluzioni preventivate con lo scopo di ridurre l'entità del problema;

- **Modello di sviluppo**: viene descritto l'approccio strutturato e metodologico utilizzato nell'ambito dello sviluppo di un prodotto software;

- **Pianificazione**: vengono pianificati in termini di calendario temporale i vari periodi contenenti tutte *attività*$_G$ da svolgere entro lo scadere degli stessi. In essi verranno inserite una stima dell'impegno richiesto da ogni componente del gruppo, l'allocazione delle attività e una presentazione delle risorse, con le rispettive assegnazioni di responsabilità;

- **Preventivo e consuntivo di periodo**: si stimano le durate di ogni singolo periodo per poter completare tutte le attività, da queste stime si ricava il preventivo. Al termine del periodo, verrà redatto il consuntivo che metterà a confronto il preventivo con quanto realmente realizzato per ottenere lo stato di avanzamento del progetto.

2.1.5.4 `Piano di Qualifica`

Il `Piano di Qualifica` è un documento formale che descrive le *attività*$_G$ e le strategie pianificate per garantire la qualità del prodotto software che si intende sviluppare. Esso definisce le metodologie, le tecniche e gli strumenti di verifica e validazione che verranno utilizzati per assicurare che il prodotto finale sia conforme alle specifiche richieste e alle aspettative del *committente*$_G$. Il documento `Piano di Qualifica` è uno strumento essenziale per la gestione del *processo*$_G$ di sviluppo software e consente di monitorare lo stato di avanzamento del progetto rispetto agli obiettivi di qualità prefissati. Ogni membro del gruppo di sviluppo farà riferimento a questo documento, che verrà redatto dagli amministratori, per raggiungere la qualità richiesta. In particolare, il `Piano di Qualifica` contiene le seguenti sezioni:

- **Qualità di processo**: vengono definiti dei parametri e delle *metriche*$_G$ che i membri del gruppo devono rispettare al fine di garantire processi di elevata qualità;

- **Qualità di prodotto**: vengono definiti dei parametri e delle metriche che i membri del gruppo devono rispettare al fine di garantire un prodotto finale di elevata qualità;

- **Testing**: vengono descritti in dettaglio i *test*$_G$ necessari per assicurare che i requisiti vengano soddisfatti nel prodotto;

- **Valutazioni per il miglioramento**: questa sezione presenta un'analisi delle criticità rilevate durante il processo di sviluppo del software, oltre alle azioni messe in atto per ottimizzare il processo stesso.

2.1.5.5 `Glossario`

Il `Glossario` rappresenta un elenco di termini tecnici utilizzati all'interno del progetto, definiti in modo chiaro e preciso, al fine di garantire una comunicazione uniforme tra tutti i membri del team e di prevenire eventuali fraintendimenti. Questo strumento di supporto alla gestione della conoscenza permette di evitare possibili ambiguità e di promuovere la comprensione reciproca tra i membri del team di sviluppo software e gli *stakeholder*$_G$ coinvolti nel progetto, migliorando la qualità della documentazione prodotta e dei risultati ottenuti.

2.1.5.6 `Lettera di Presentazione`

La `Lettera di Presentazione` è un documento che accompagna la consegna delle revisioni del prodotto software con cui il team di sviluppo software si impegna formalmente a completare il *capitolato*$_G$ prescelto. Questo documento elenca la documentazione che verrà messa a disposizione dei committenti e del *proponente*$_G$. Il gruppo si impegna pertanto a rispettare i requisiti minimi consegnando il prodotto finito entro i termini prestabiliti di ogni revisione, come indicato nella lettera stessa.

2.1.6 Strumenti

Gli strumenti software utilizzati per il *processo*$_G$ di fornitura sono i seguenti:

2.1.6.1 Google Calendar

Sistema di calendari utile per programmare le riunioni interne e con il *proponente*$_G$, grazie all'opzione di condivisione degli eventi.

2.1.6.2 Google Slides

Servizio cloud per la creazione di presentazioni del diario di bordo.

2.1.6.3 Google Meet

Servizio di teleconferenze online interfacciabile con gli altri servizi offerti da *Google*$_G$.

2.1.6.4 GanttProject

Software utilizzato dal responsabile di progetto per il processo di Pianificazione. Nello specifico verrà impiegato per l'assegnazione delle risorse, la verifica dei tempi del progetto, la gestione del budget e l'analisi del lavoro svolto/da svolgere. **GanttProject** permetterà di generare i *diagrammi di Gantt*$_G$ presenti nel `Piano di Progetto`.

2.1.6.5 Excel

Software di fogli di calcolo sviluppato da *Microsoft*$_G$ che viene utilizzato principalmente per l'elaborazione di dati e per la creazione di grafici e tabelle. In particolare, Excel verrà utilizzato per la creazione di istogrammi e areogrammi, e per l'effettuazione di calcoli sui dati.

2.2 Sviluppo

2.2.1 Scopo

Secondo lo *standard*$_G$ ISO/IEC 12207:1995$_G$, lo scopo del *processo*$_G$ di sviluppo è quello di definire i compiti e le *attività*$_G$ di analisi, progettazione, codifica, integrazione, testing, installazione e accettazione che rispecchino i requisiti definiti precedentemente nel contratto.
Lo sviluppatore esegue o supporta le attività di questo processo in conformità ad esso.

2.2.2 Descrizione

Segue un elenco delle *attività*$_G$ che caratterizzano il *processo*$_G$ di sviluppo:

- **Analisi dei requisiti** (§2.2.4);
- **Progettazione architetturale** (§2.2.5);
- **Codifica** (§2.2.6).

2.2.3 Aspettative

Per una corretta implementazione del *processo*$_G$ di sviluppo è necessario determinare:

- **Obiettivi di sviluppo**;
- **Vincoli tecnologici**;
- **Vincoli di design**;

Il prodotto finale deve superare con esito positivo i *test*$_G$ e rispettare i requisiti del *proponente*$_G$.

2.2.4 Analisi dei Requisiti

2.2.4.1 Scopo

Gli obiettivi dell'*attività*$_G$ di Analisi dei Requisiti sono:

- Definire con il *proponente*$_G$ lo scopo del prodotto da realizzare rispecchiandone le aspettative;
- Favorire l'attività di progettazione fornendo ai progettisti dei requisiti chiari e di facile comprensione;
- Favorire l'attività di pianificazione fornendo una stima sulle tempistiche necessarie per completare il prodotto, così facendo si può avere una stima dei costi totali;
- Favorire l'attività di verifica fornendo dei riferimenti pratici.

2.2.4.2 Descrizione

È compito degli analisti effettuare l'Analisi dei Requisiti, redigendo un documento con il medesimo nome che deve contenere:

- **Introduzione**: contiene lo scopo del documento stesso;
- **Descrizione**: descrizione delle finalità del prodotto;
- **Attori**: gli utilizzatori del prodotto finale;
- ***Casi d'uso***$_G$: tutte le possibili interazioni che gli attori possono compiere utilizzando il prodotto;
- ***Requisiti***$_G$: le caratteristiche da soddisfare.

2.2.4.3 Aspettative

Dall'attività di Analisi è previsto che venga creata la documentazione formale che includa tutti i requisiti richiesti dal proponente.

2.2.4.4 Casi d'uso

I *casi d'uso*$_G$ sono un insieme di possibili sequenze di interazioni compiute da uno specifico attore per raggiungere un particolare obiettivo all'interno del prodotto. Ogni caso d'uso dev'essere costituito da:

- **identificazione**: espressamente nel formato;

  ```
  UC [numeroPadre].[numeroFiglio] - [titolo]
  ```

  dove:

  - **UC**: Use Case;
  - **[numeroPadre]**: numero identificativo del caso d'uso generico;
  - **[numeroFiglio]**: numero identificativo progressivo relativo ai sotto casi;
  - **[titolo]**: titolo auto esplicativo del caso d'uso.

- **descrizione**: breve descrizione di facile comprensione del caso d'uso;
- **attore**: utente che può compiere quella determinata interazione.

Infine, i casi d'uso devono essere rappresentati graficamente mediante un *diagramma UML*$_G$ per rendere le casistiche più intuitive; si raccomanda l'utilizzo della versione 2.0.

Informazioni sui Casi d'Uso

Nella redazione del documento di `Analisi dei Requisiti`, ogni caso d'uso deve essere accompagnato da una serie di informazioni aggiuntive per garantirne una comprensione completa e precisa. Tra queste, troviamo:

- **Attori**: le entità esterne al sistema che interagiscono con esso per raggiungere un obiettivo specifico; vengono suddivisi in **attore primario** e **attore secondario**;
- **Descrizione**: una breve descrizione che delinea il caso d'uso in modo chiaro e completo;
- **Scenario principale**: un elenco numerato che rappresenta il flusso degli eventi principali che si verificano durante l'esecuzione del caso d'uso;
- **Estensioni**: (opzionale) impiegate per modellare scenari alternativi. Al verificarsi di una determinata condizione, il caso d'uso ad essa collegata viene interrotto;
- **Precondizioni**: le condizioni del sistema che devono essere soddisfatte prima che il caso d'uso possa essere eseguito; Gli input: opzionali, ma utili per specificare con precisione le informazioni che l'attore porta all'interno del sistema;
- **Postcondizioni**: le condizioni del sistema che si verificano al termine dell'esecuzione del caso d'uso; Gli output: opzionali, ma utili per descrivere i valori o gli oggetti che il sistema restituisce come conseguenza dell'esecuzione del caso d'uso.

Di seguito viene riportato un esempio di caso d'uso:

UC 1 - Navigazione nello spazio della showroom tramite tastiera

- **Attore principale**: utente
- **Descrizione**: l'utente deve poter navigare nella showroom usando la tastiera
- **Scenario principale**: l'utente compie un'azione di movimento tramite le freccette della tastiera
- **Estensioni**: l'utente collide con un oggetto (UC3)
- **Precondizioni**: il sistema ha assegnato all'utente una posizione nello spazio
- **Postcondizioni**: il sistema cambia la posizione dell'utente rispetto a quella iniziale

2.2.4.5 Requisiti

I requisiti devono essere identificati da un codice univoco seguendo la convenzione:

`R[Tipologia][Importanza][Codice]`

- **Tipologia**: rappresenta il tipo di *requisito*$_G$ che può assumere uno dei seguenti valori letterali:
  - **V**: requisito di **Vincolo** con cui si descrivono i vincoli relativi ai servizi che il sistema offre;
  - **F**: requisito **Funzionale** con cui si descrivono i servizi o le funzioni che il sistema offre;
  - **Q**: requisito di **Qualità** con cui si descrivono i vincoli di qualità da realizzare (si veda `Piano di Qualifica`);
  - **P**: requisito di **Prestazione** con cui si descrivono i vincoli sulle prestazioni da soddisfare.
- **Importanza**: ossia definisce un indice di importanza che viene associata ad ogni requisito e può assumere uno dei seguenti valori numerici:
  - **1**: requisito **Obbligatorio** che deve essere necessariamente soddisfatto per garantire la presenza delle funzionalità di base all'interno del sistema;
  - **2**: requisito **Desiderabile** che non vincola il sistema nel suo funzionamento. La sua implementazione fornirà al prodotto una maggiore completezza e rappresenta uno dei requisiti che possono essere negoziati con l'azienda *Sanmarco Informatica S.p.A.*;
  - **3**: requisito **Opzionale** che determina ulteriore completezza all'interno del sistema. Rispetto ai precedenti, ha maggiore probabilità di comportare un dispendio di risorse che favoriscono l'aumento dei costi del progetto.
- **Codice**: identificatore univoco espresso tramite una gerarchia CasoBase/SottoCaso, nella forma:

  `[CodiceCasoBase](.[CodiceSottoCaso])*`

  dove
  - **CodiceCasoBase**: codice che insieme alla tipologia ha la funzione di identificare il caso d'uso generico che genera il caso d'uso in esame;
  - **CodiceSottoCaso**: codice identificativo opzionale e progressivo che è relativo ai sotto casi d'uso.

Informazioni sui Requisiti

Nella redazione del documento di `Analisi dei Requisiti`, ogni requisito deve essere accompagnato da una serie di informazioni aggiuntive per garantirne una comprensione completa e precisa. Tra queste, troviamo:

- **Codice**: identificatore univoco del requisito. verrà stabilito secondo la convenzione citata nella sezione precedente;

- **Importanza**: che indica l'importanza del requisito e viene scelto fra obbligatorio, desiderabile e facoltativo. Questa indicazione, pur potendo risultare ridondante, facilita la lettura del documento;
- **Descrizione**: fornisce una descrizione breve, completa così da ridurre le ambiguità sullo scopo del requisito;
- **Fonte**: per garantire la tracciabilità dei requisiti, è necessario indicare la fonte da cui il requisito è stato individuato, che può essere il Capitolato d'Appalto, i Verbali Interni, i Verbali Esterni o i casi d'uso

Di seguito viene riportato un esempio di requisito:

Codice	Importanza	Descrizione	Fonti
RFO1	Obbligatorio	L'utente deve poter navigare nella showroom usando la tastiera.	UC1

Tabella 2: Esempio di classificazione di un requisito

2.2.4.6 Metriche

Metrica	Nome	Riferimento
MROS	Requisiti obbligatori soddisfatti	§B.5.1
MRDS	Requisiti desiderabili soddisfatti	§B.5.1
MROPZS	Requisiti opzionali soddisfatti	§B.5.1

Tabella 3: Metriche per l'Analisi dei Requisiti

2.2.5 Progettazione

2.2.5.1 Scopo

L'*attività*$_G$ di progettazione consiste nel *processo*$_G$ di definizione e documentazione dei requisiti, delle specifiche tecniche e della struttura del prodotto software (ossia le sue caratteristiche), al fine di ottenere un prodotto adeguato alle necessità degli *stakeholder*$_G$. Parte integrante della progettazione è un approccio sistematico ai problemi, con l'obiettivo di assicurare la qualità del prodotto finale. Essa mira anche a suddividere i compiti di implementazione, rendendoli più semplici ed *efficienti*$_G$. La progettazione punta inoltre ad ottimizzare i tempi e le risorse necessarie per il completamento del progetto.
La progettazione è un processo che procede in senso inverso rispetto all'Analisi dei Requisiti: permette di dominare la complessità del prodotto, organizzando e ripartendo le responsabilità di realizzazione.

2.2.5.2 Descrizione

La realizzazione dell'architettura di un sistema inizia con la definizione del *Proof of Concept*$_G$ della *Technology Baseline*$_G$. Questo consente di capire meglio la visione aziendale e l'obiettivo da raggiungere e di determinare le specifiche relative alla progettazione delle componenti del prodotto. Per le specifiche tecniche verranno adottati dei *diagrammi UML*$_G$, grazie ai quali verrà realizzata l'architettura. In questa *baseline*$_G$ si determineranno inoltre i *test*$_G$ da effettuare sul prodotto.
Successivamente, l'architettura sarà approfondita e dettagliata nel documento tecnico allegato alla *Product Baseline*$_G$, che fornisce una descrizione più accurata del sistema. In questo documento dovranno essere incluse anche le linee guida e le best practices per lo sviluppo del sistema, in modo che tutti i sviluppatori possano consultarlo e seguire le indicazioni allo scopo di raggiungere l'obiettivo prefissato.

2.2.5.3 Aspettative

Prima di iniziare a lavorare sull'architettura che comporrà il sistema, il team di progettazione di *Smoking Fingertips* dovrà definire le tecnologie da usare e studiare tutti gli aspetti positivi e le eventuali criticità che comporta la loro scelta.
Una volta effettuata un'attenta analisi, i progettisti dovranno creare una bozza del prodotto (*Proof of Concept*$_G$) che rappresenta quanto studiato e appreso. Il Proof of Concept dovrà essere sviluppato nel rispetto di ogni vincolo stabilito con l'azienda *Sanmarco Informatica S.p.A* , al fine di fornire un prodotto coerente con le richieste formulate.
La progettazione dell'architettura dovrà tener conto delle esigenze dell'azienda e di tutti i requisiti necessari per la realizzazione del sistema.

2.2.5.4 Documentazione

Come accennato nella descrizione di questa sezione, ad ogni *baseline*$_G$ corrisponderà la stesura di documentazione, come elencato in seguito:

Technology Baseline

Baseline$_G$ seguita prevalentemente dai progettisti i quali dovranno fornire:

- ***Proof of Concept***$_G$: raccolta di implementazioni di alcune funzionalità che il prodotto dovrà avere. L'obiettivo è dimostrare la fattibilità di esecuzione delle richieste del *proponente*$_G$;
- **definizione delle componenti**: stabilisce una relazione tra i requisiti software e le componenti che li soddisfano. Racchiude un'analisi che consente di monitorare come le diverse parti del sistema si adattano alle esigenze di un progetto;
- **scelte tecnologiche**: verranno stabilite che tecnologie utilizzare e le motivazioni legate a tali scelte.

Product Baseline

Come nel caso precedente, verranno raccolte dai progettisti le seguenti informazioni:

- ***diagrammi UML***$_G$: come descritto alla sezione §2.2.5.6;
- ***Design Pattern***$_G$: la definizione dell'architettura può essere basata sull'utilizzo di design pattern, che consentono di risolvere problemi ricorrenti in modo rapido ed *efficace*$_G$. I design pattern sono schemi riutilizzabili di progettazione illustrati con diagrammi che ne mostrano la struttura;
- **definizione delle classi**: le classi individuate verranno associate ai requisiti definiti con l'Analisi dei Requisiti;
- **test unitari**: ossia i *test*$_G$ eseguiti per verificare che il funzionamento delle classi e dei metodi che implementano il sistema software siano corretti e conformi ai requisiti. Consistono nel confrontare il comportamento dei moduli software con le specifiche definite.

Metriche

Metrica	Nome	Riferimento
MAC	Accoppiamento tra classi	§B.5.4
MATC	Attributi per classe	§B.5.4
MPM	Parametri per metodo	§B.5.4
MLCM	Linee di codice per metodo	§B.5.4

Tabella 4: Metriche per la Progettazione

2.2.5.5 Qualità dell'architettura

Nel contesto di un progetto software, il compito di definire un'architettura logica del prodotto di qualità è affidato ai progettisti. L'architettura logica del prodotto costituisce la base per lo sviluppo del software. Affinché l'architettura sia di qualità, le diverse componenti del prodotto software devono essere chiaramente identificabili, riusabili e coese, garantendo al tempo stesso il rispetto dei costi fissati per lo sviluppo del progetto.
Per garantire la qualità dell'architettura, i progettisti utilizzano diverse tecniche e metodologie, tra cui l'adozione di pattern architetturali consolidati, l'uso di tool e framework per l'analisi e la valutazione della qualità dell'architettura, la definizione di metriche per la qualità dell'architettura e l'adozione di buone pratiche di programmazione e di documentazione.
Inoltre, l'architettura logica del prodotto deve essere sviluppata tenendo conto dei costi fissati per il progetto. I progettisti devono quindi identificare soluzioni tecniche *efficienti*$_G$ ed *economiche*$_G$, evitando di superare i costi preventivati.
L'architettura dovrà pertanto:

- soddisfare i requisiti presenti nell'`Analisi dei Requisiti` e adeguarsi agli eventuali cambiamenti/aggiunte;

- risultare affidabile: dovrà permettere di svolgere i compiti cui è destinata, in qualsiasi condizione;
- essere sviluppata con componenti semplici, coese e con un basso livello di accoppiamento;
- utilizzare in modo efficiente ed *efficace*$_G$ le risorse;
- adottare dei pattern architetturali consolidati per garantire la scalabilità, la modularità e la manutenibilità del prodotto software;
- definire delle *metriche*$_G$ per la qualità dell'architettura, che consentiranno di valutare la capacità dell'architettura di soddisfare le specifiche esigenze funzionali e non funzionali;
- adottare buone pratiche di programmazione e di documentazione.

2.2.5.6 Diagrammi UML

Per garantire una maggiore comprensibilità delle decisioni progettuali e minimizzare confusione o ambiguità, utilizzeremo dei *diagrammi UML*$_G$ 2.0. In particolare, useremo i seguenti diagrammi:

- **diagrammi delle classi**: illustrano graficamente gli elementi di un modello. Nello specifico permettono di descrivere le classi, i tipi, gli attributi e i metodi utilizzati nel progetto delineando che relazioni intercorrono;
- **diagrammi dei casi d'uso**: illustrano graficamente tutte le funzionalità che il sistema offre, in linea con le richieste del *proponente*$_G$.

Diagrammi delle classi

Ogni diagramma delle classi definisce le caratteristiche statiche e le relazioni che intercorrono fra le componenti del sistema. Questi diagrammi permettono di descrivere ciò che non riguarda l'ambiente a run-time.
Una classe viene rappresentata graficamente come un rettangolo composto da tre righe, ognuna delle quali descrive degli aspetti significativi della classe stessa. A partire dalla prima riga, troviamo:

1. **Nome della classe**: permette di definire univocamente la classe stessa. Il nome viene rappresentato con lettere in grassetto che rispettano il *PascalCase*$_G$ e deve identificare esplicitamente il compito svolto. Qualora la classe fosse *astratta*, il nome utilizzato sarà presentato in *corsivo*;
2. **Attributi**: ognuno degli elementi verrà presentato uno dopo l'altro in una riga separata secondo il formato:

```
visibilità nome:  tipo [molteplicità] = valore/i di default
```

- **Visibilità**: precede obbligatoriamente ogni attributo e rappresenta uno dei seguenti indicatori:
  - $-$: visibilità privata;
  - $+$: visibilità pubblica;
  - $\#$: visibilità protetta;
  - $\sim$: visibilità di package.
- **Nome**: rappresenta univocamente l'attributo. Dev'essere rappresentativo dell'attributo e deve seguire la notazione `nomeAttributo: tipo`.
  Qualora l'attributo fosse di tipo **costante**, allora il nome verrà scritto interamente in maiuscolo `NOMEATTRIBUTO: tipo`;
- **Molteplicità**: nel caso di una sequenza di elementi, come liste o array, la sua lunghezza può essere specificata con la sintassi `tipoAttributo[molteplicità]`.
  Se la sequenza contiene un numero non conosciuto a priori di elementi, verrà adottata la sintassi `tipoAttributo[*]`.
  Nel caso di un singolo elemento, la sua dichiarazione è opzionale;
- **Default**: ogni attributo può essere dichiarato con un valore di default.

3. **Firme dei Metodi**: descrivono il comportamento delle classi individuate. Ogni metodo occuperò una riga e seguiranno il formato:

   ```
   visibilità nome (parametri formali): tipo di ritorno
   ```

   - **Visibilità**: segue la procedura descritta per gli attributi;
   - **Nome**: rappresenta l'identificativo univoco e significativo del metodo. Descrive esplicitamente l'obiettivo del metodo e segue la notazione $PascalCase_G$;
   - **Parametri formali**: possono essere in numero da 0 a n e vengono separati tramite una virgola. Ognuno seguirà la notazione e le regole definite per gli attributi;
   - **Tipo di ritorno**: determina che tipo di oggetto verrà ritornato dal metodo.

   I metodi *getter*, *setter* e i *costruttori* non verranno inclusi fra i metodi.
   I metodi *astratti* verranno scritti in *corsivo*.
   I metodi *statici* verranno sottolineati.

L'assenza di attributi o metodi in una classe, determinerà una visualizzazione di campi vuoti nel diagramma delle classi.

I diagrammi delle classi sono interconnessi da frecce che illustrano le loro relazioni di dipendenza. Le seguenti frecce verranno utilizzate per raffigurare queste relazioni:

- **Dipendenza**: rappresentata con un freccia tratteggiata dalla classe **A** alla classe **B**. Indica il minor grado di accoppiamento fra classi.

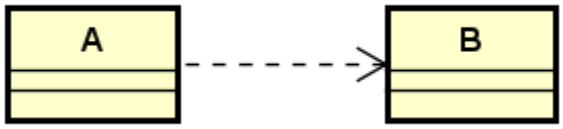

Figura 1: Diagramma delle classi di una Relazione di Dipendenza

- **Associazione**: rappresentata con un freccia dalla classe **A** alla classe **B**. La classe **A** contiene dei campi o delle istanze della classe **B**. È possibile rappresentare le molteplicità di occorrenza tramite un valore posizionato agli estremi della freccia fra i seguenti:
  - **0..1**: **A** possiede 0 o 1 istanza di **B**
  - **0..***: **A** possiede 0 o più istanze di **B**
  - **1**: **A** possiede un'istanza di **B** (non è necessario specificarlo)
  - *: **A** possiede più istanze di **B**
  - **n**: **A** possiede esattamente **n** istanze di B

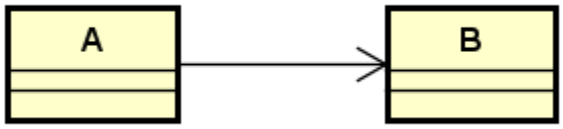

Figura 2: Diagramma delle classi di una Relazione di Associazione

- **Aggregazione**: rappresentata con un freccia a diamante (vuota) dalla classe **A** alla classe **B**. Significa che **B** è parte della classe **A**. Di conseguenza l'esistenza di **A** è condizionata da quella di **B**. Tutto ciò che è aggregato può essere condiviso

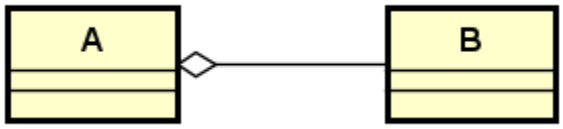

Figura 3: Diagramma delle classi di una Relazione di Aggregazione

- **Composizione**: Una freccia a diamante piena indica una dipendenza che rappresenta un'aggregazione più forte di quella descritta in precedenza. Si tratta di un'associazione che specifica che la classe **A** e la classe **B** devono sempre essere usate insieme, e che gli aggregati possono appartenere solo ad un aggregato (aggregato con cardinalità (1,1)). Inoltre, solo l'oggetto intero può creare e distruggere le sue parti.

Figura 4: Diagramma delle classi di una Relazione di Composizione

- **Generalizzazione**: rappresentata da una freccia vuota continua che va da una classe **B** ad una classe **A** rappresenta il massimo grado di accoppiamento tra le due classi, indicando che ogni oggetto di classe **B** fa anche parte di classe **A**.

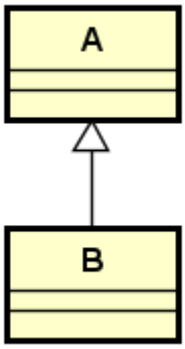

Figura 5: Diagramma delle classi di una Relazione di Ereditarietà

- **Interface Realization**: Un'interfaccia **A** può essere implementata utilizzando una classe **B**, sia concreta che astratta, ed è rappresentato graficamente tramite una freccia da **B** a **A**.

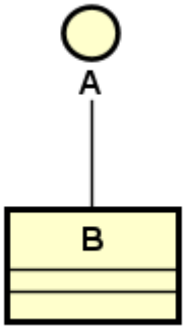

Figura 6: Diagramma delle classi di una Relazione di Interface Realization

Diagrammi dei casi d'uso

Un diagramma dei *casi d'uso*$_G$ è uno strumento di modellazione utilizzato per documentare e descrivere le funzionalità di un sistema. Serve a identificare i flussi di lavoro

attraverso una rappresentazione grafica, descrivendo la modalità in cui un utente interagisce con un sistema.
I casi d'uso sono organizzati in sequenze di azioni, che descrivono gli interventi necessari a un utente per completare una determinata azione e sono collegati tra loro con linee.
Un diagramma dei casi d'uso è uno strumento estremamente utile per progettare un sistema, poiché fornisce una rapida e intuitiva rappresentazione dei flussi di lavoro e delle interazioni tra l'utente e il sistema.
La rappresentazione data dai diagrammi dei casi d'uso non comprende la visione implementativa, in quanto il loro obiettivo è quello di descrivere la funzionalità, vedendola come esterna al sistema.
I diagrammi dei casi d'uso contengono:

- **Attore**: un attore è un agente esterno che interagisce con il sistema.
  Un attore può essere un utente, una persona, una macchina, un altro sistema informatico, un'organizzazione, ecc.
  Un caso d'uso determina una funzionalità che viene messa a disposizione di tale utente, senza fornire i dettagli implementativi.
  A livello di diagramma, un attore è rappresentato come uno *stickyman*$_G$ identificato con un'etichetta contenente il nome identificativo dell'attore.

Figura 7: Rappresentazione di un Attore

- **Caso d'uso**: rappresentano le funzionalità che l'utente può eseguire con il sistema.
  Un caso d'uso è rappresentato da una breve descrizione della funzionalità del sistema a disposizione di uno o più utenti all'interno di un sistema software. Nello specifico rappresenta una descrizione dettagliata del comportamento dell'utente attraverso l'interazione con un sistema software.
  Un caso d'uso è in genere composto da una serie di scenari che descrivono le differenti possibilità che possono verificarsi durante un'interazione tra un utente e un sistema software.
  La sua rappresentazione è data da una numerazione univoca (esempio `UCx.y`) seguito da una breve, ma esaustiva descrizione della funzionalità stessa.
  Ogni caso d'uso verrà collegato tramite una linea continua agli attori che hanno accesso a quella funzionalità.

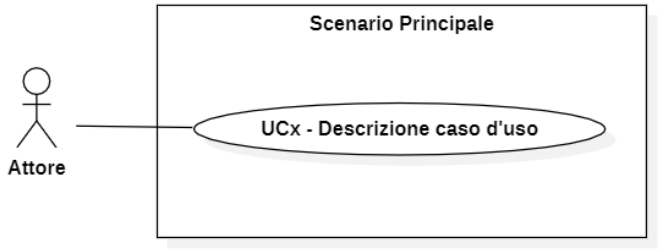

Figura 8: Rappresentazione di un Caso d'uso

Come nel caso dei diagrammi delle classi, è possibile trovare molteplici rappresentazioni delle relazioni che intercorrono fra i casi d'uso:

- **Inclusione**: diciamo che esiste un'inclusione fra un caso d'uso **A** e un caso d'uso **B**, se tutte le istanze del caso d'uso **A** devono eseguire anche le istanze del caso d'uso **B**. Ciò conferisce al caso d'uso **A** la responsabilità dell'esecuzione del caso d'uso **B**, evitando la ripetizione e aumentando il riutilizzo di una medesima struttura.
  L'inclusione viene rappresentata con una freccia tratteggiata, che collega i casi d'uso **A** con tutti i casi d'uso che include (nell'esempio, il caso d'uso **B**). Sopra la freccia verrà scritta la direttiva `«include»`;

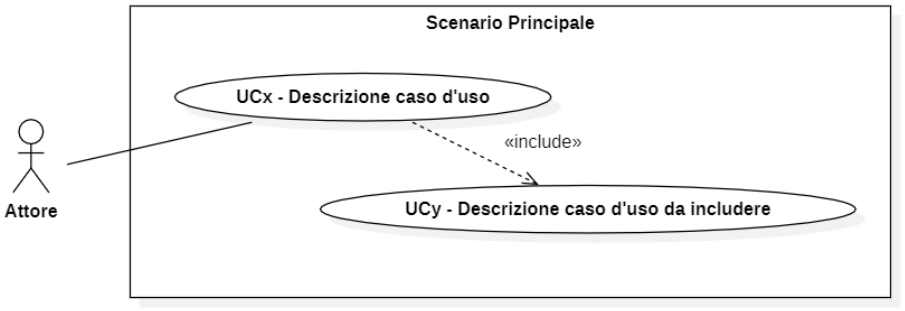

Figura 9: Rappresentazione di un'Inclusione

- **Estensione**: diciamo che esiste un'estensione fra un caso d'uso **A** e un caso d'uso **B**, se tutte le istanze del caso d'uso **A** devono eseguire anche le istanze del caso d'uso **B** incondizionatamente. L'esecuzione del caso d'uso **B** interrompe quella del caso d'uso **A** e la responsabilità di esecuzione dei casi di estensione è di chi estende (nel caso in esempio, il caso d'uso **B**).
  L'estensione viene rappresentata con una freccia tratteggiata, che collega i casi d'uso che estendono **A** (nell'esempio, il caso d'uso **B**) con il caso d'uso da estendere. Sopra la freccia verrà scritta la direttiva `«extend»`;

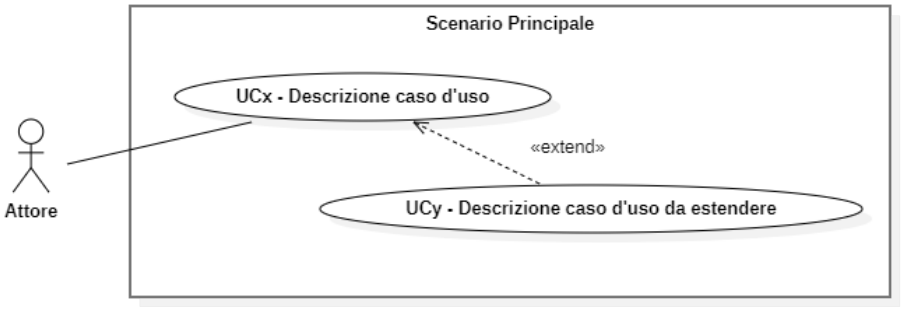

Figura 10: Rappresentazione di un'Estensione

- **Generalizzazione**: La generalizzazione può essere applicata sia agli attori che ai casi d'uso. La generalizzazione di un attore si ha quando un attore padre, dotato di capacità generiche, viene specializzato in comportamenti più specifici negli attori figli. Ogni attore figlio eredita le funzionalità dal padre aggiungendone altre relative al proprio contesto.
  Nei casi d'uso, invece, i figli possono aggiungere o modificare il comportamento dei casi d'uso ereditati dal padre; tutte le funzionalità non ridefinite nei figli mantengono la definizione ereditata. La generalizzazione di attori e casi d'uso viene rappresentata con una freccia continua vuota da un elemento figlio verso un elemento padre.

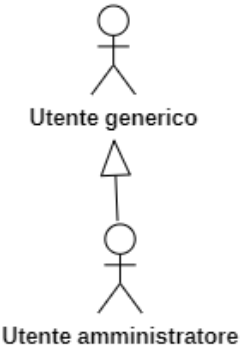

Figura 11: Rappresentazione di una Generalizzazione di un Attore

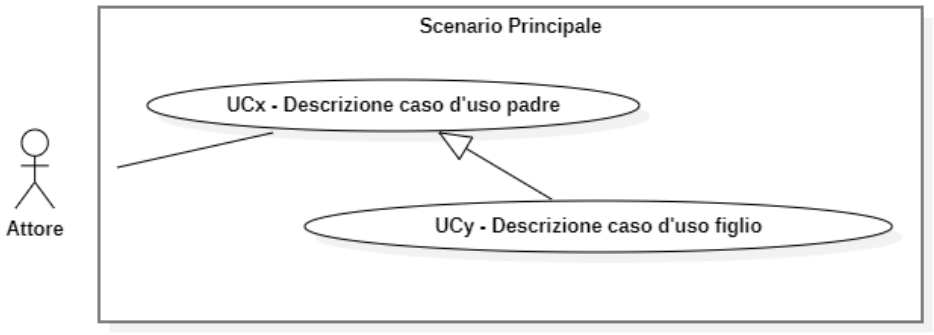

Figura 12: Rappresentazione di una Generalizzazione di un Caso d'Uso

2.2.5.7 Design pattern

Un *design pattern*$_G$ rappresenta una soluzione ad un problema di progettazione ricorrente all'interno di un determinato contesto. Essi forniscono un modello di progettazione che può essere riutilizzato più volte, garantendo allo stesso tempo un'alta qualità della soluzione e una rapida realizzazione.
I design pattern sono adottati in particolare quando si riscontra che una soluzione al problema sia stata utilizzata con successo in un contesto ben definito. Essi sono spesso accompagnati da una guida al loro utilizzo che descrive come utilizzarli al meglio.
Ogni design pattern deve essere corredato da un diagramma che ne illustri il funzionamento, una descrizione testuale della sua logica e una descrizione dell'utilità del design pattern all'interno dell'architettura progettata. Tale documentazione può contribuire a una migliore comprensione di come il design pattern si integri nell'architettura complessiva e aiutare a prevenire errori di progettazione.

2.2.5.8 Qualità della progettazione

Si riferisce alla qualità dell'insieme delle *attività*$_G$ svolte dai progettisti per definire la soluzione tecnica che soddisfi i *requisiti*$_G$ del prodotto software da sviluppare.
I progettisti devono garantire che la progettazione del software sia di alta qualità, sia in termini di funzionalità che di prestazioni, affidabilità, sicurezza, *manutenibilità*$_G$ e usabilità.
Le modalità con cui i progettisti possono garantire la qualità della progettazione del prodotto software sono diverse, e comprendono l'utilizzo di modelli di progettazione ben strutturati, l'adozione di best practice e linee guida di progettazione, la definizione di pattern di progettazione, l'analisi statica e dinamica del codice, la revisione del codice, il testing dell'architettura e la verifica del rispetto dei requisiti di qualità.
Secondo lo *standard*$_G$ ISO/IEC 12207:1995$_G$ questa attività consiste nello stabilire un'architettura di primo livello del sistema. L'architettura deve identificare gli elementi di hardware, software e operazioni manuali. Si deve garantire che tutti i requisiti del sistema siano ripartiti tra gli elementi.
L'architettura del sistema e i requisiti degli elementi devono essere valutati in base ai criteri elencati di seguito:

- I requisiti del sistema devono essere tracciati: prima di iniziare qualsiasi attività di progettazione, è necessario che i progettisti comprendano le funzionalità ed i requisiti che devono essere soddisfatti. Questo significa che è fondamentale effettuare un'analisi dei requisiti dettagliata, al fine di definire in modo preciso le specifiche funzionali del software da sviluppare. Ogni attività di progettazione dovrà essere coerente con quanto descritto all'interno dell'analisi dei requisiti per garantire che il software sviluppato soddisfi le esigenze dell'utente finale e che sia conforme alle specifiche;

- Durante la fase di codifica, verranno:

  - tracciati tutti i problemi che si verranno a manifestare;
  - suddivise le risorse necessarie in modo equo e bilanciato;
  - assegnare i compiti ai membri del team verificando che il carico di lavoro sia simile e proporzionato alle competenze e alla sua disponibilità.

  La pianificazione delle attività di implementazione deve essere realistico e adeguato alle risorse disponibili: il tempo e le risorse dedicate alla codifica devono essere quantificate in modo accurato, tenendo conto delle scadenze prestabilite e dei task di lavoro minimi. In questo modo, è possibile garantire che l'implementazione sia effettuata in modo efficiente e che i problemi possano essere rapidamente individuati e risolti;

- Devono essere utilizzati standard e metodi di progettazione adeguati al progetto che si sta sviluppando: il processo di progettazione deve essere strutturato e organizzato, seguendo procedure e metodologie consolidate nel settore dell'ingegneria del software. L'uso di standard e metodi di progettazione adeguati consente di garantire la qualità del software sviluppato, evitando errori e incongruenze nel processo di sviluppo;

- Nel contesto di sviluppo software, per garantire il successo del modello di sviluppo incrementale, è necessario garantire la correttezza per costruzione della progettazione. Quest'ultima deve essere effettuata in modo preciso e accurato, tenendo conto di tutti i requisiti e le funzionalità richieste dal committente. La progettazione che verrà presentata al committente dovrà essere il più corretta possibile;

- Il livello di dipendenza tra le varie componenti del sistema dovrà essere accettabile. Per fare ciò, è necessario adottare un approccio modulare alla progettazione del sistema, in cui ogni componente abbia una responsabilità singola e ben definita.

2.2.5.9 Test

L'attività di testing permette di garantire la qualità del prodotto finale. In questa fase, vengono identificati i requisiti di testing, i casi di test e i criteri di accettazione che verranno utilizzati per valutare il software. Questa attività ha l'obiettivo di identificare eventuali problemi o errori nel software così da poterli risolti prima del rilascio del prodotto finale. Inoltre, il processo di testing aiuta a garantire che il software sia conforme

alle specifiche e alle aspettative del cliente.
Quest'attività verrà gestita interamente dai progettisti, che avranno inoltre il compito di definire quali test effettuare.
Una descrizione più dettagliata delle tipologie di test e della nomenclatura da utilizzare è disponibile alla sezione §3.2

2.2.5.10 Metriche

Metrica	Nome	Riferimento
MAC	Accoppiamento tra classi	§B.5.4
MATC	Attributi per classe	§B.5.4
MPM	Parametri per metodo	§B.5.4
MLCM	Linee di codice per metodo	§B.5.4

Tabella 5: Metriche per la Progettazione

2.2.6 Codifica

2.2.6.1 Scopo

L'*attività*$_G$ di Codifica, svolta dal *ruolo*$_G$ del programmatore, ha come obiettivo la realizzazione effettiva del prodotto software richiesto dal *proponente*$_G$. Consente di convertire in codice le idee dei progettisti a un livello alto, ottenendo del codice eseguibile dai calcolatori. I programmatori devono rispettare queste regole durante le *fasi*$_G$ di programmazione e implementazione.

2.2.6.2 Descrizione

Nel contesto di un progetto software, è importante che la scrittura del codice segua le linee guida stabilite nella documentazione di prodotto. Questo garantirà che il codice sia di qualità, seguendo le *metriche*$_G$ descritte nel `Piano di Qualifica`.
Per fornire una guida uniforme per tutti i programmatori, la documentazione di prodotto include sia regole di carattere generale, che devono essere seguite per qualsiasi linguaggio di programmazione utilizzato, sia regole più specifiche per il linguaggio principale *C++*$_G$.
Il rispetto di queste linee guida garantirà che il codice sia scritto in modo coerente, *efficiente*$_G$ e facile da mantenere.

2.2.6.3 Aspettative

La codifica mira a creare un prodotto software che risponda alle esigenze del *proponente*$_G$ e che sia conforme a quanto concordato con il medesimo. La corretta adozione delle *norme*$_G$ che seguiranno, permetteranno di assicurare un codice leggibile, uniforme e di qualità.
È importante che le norme e convenzioni vengano seguite in quanto agevolano le *attività*$_G$ di estensione, manutenzione, verifica e validazione con conseguente *miglioramento*$_G$ della qualità del prodotto.

2.2.6.4 Stile di codifica

Il linguaggio che il gruppo ha deciso di adottare per questo progetto è *C++*$_G$. L'assenza di un garbage collector e di un gestore di memoria, la necessità di adottare frequentemente dei puntatori e una sintassi complessa possono rendere più difficile padroneggiare correttamente il linguaggio. Per garantire la qualità e la coerenza del codice scritto, è importante che tutti condividano uno stesso approccio. Per questo motivo, si sono definite delle regole che devono essere seguite quando si scrive codice così da ridurre la probabilità di errore.

Indentazioni

Per garantire una corretta strutturazione del codice, è necessario che i blocchi annidati seguano una corretta indentazione utilizzando un `TAB` per ogni livello di annidamento. Sarà cura di ogni membro del gruppo verificare che l'*IDE*$_G$ adottato associ al tasto `TAB` 4 spazi, così da mantenere consistente il codice steso.

Parentesi

In questo stile di codifica, tutti i blocchi di codice saranno delimitati da parentesi graffe. La convenzione per la posizione delle parentesi graffe aperte sarà la stessa per tutti i metodi, le classi e i costrutti: dovranno essere collocate sulla stessa riga e separati da un solo spazio dal contenuto precedente.

Lunghezza dei metodi

I metodi di un progetto verranno considerati accettabili, solamente se brevi. Risulta essere una buona pratica in quanto porta notevoli vantaggi quali:

- ***Mantenibilità***$_G$: sono più facili da mantenere rispetto a metodi lunghi e complessi in quanto il codice è più leggibile e comprensibile. Si ottiene così un codice robusto e meno suscettibile a errori.
- **Leggibilità**: il codice è reso più accessibile a tutti gli sviluppatori che potrebbero doverlo leggere, comprendere o modificare in futuro.
- **Debugging**: Se un metodo è breve e semplice, è più facile identificare eventuali bug o problemi nel codice. Questo rende il *processo*$_G$ di debugging più *efficiente*$_G$.

Lunghezza delle righe di codice

La lunghezza massima di una riga di codice dovrà essere di 100 caratteri. Questo limite è stato stabilito perché una riga troppo lunga può rendere il codice difficile da leggere e mantenere.
Se una riga supera questa lunghezza in un codice del progetto dovrà essere applicata una delle seguenti soluzioni:

- riformattare la riga, utilizzando più righe per completare la sintassi del metodo (utilizzando un ritorno a capo per dividere una lunga espressione in più righe);
- riformulare il codice in modo che sia più conciso e facile da leggere.

Codice esterno

Quando si lavora con codice proveniente da fonti esterne, è importante adottare lo stile e le convenzioni di codifica utilizzate dagli autori originali. Aderendo alle $norme_G$ di codifica già esistenti, si evitano eventuali problemi di compatibilità e di manutenzione futura del codice.

Univocità dei nomi

Tutte le variabili, i metodi e le classi dovranno avere un nome che li distinguono univocamente, per limitare la possibilità di ambiguità del codice.

Ricorsione

La ricorsione può essere uno strumento utile per lo sviluppo di software, però gli svantaggi che porta possono essere maggiori dei vantaggi. È importante valutare attentamente l'utilizzo della ricorsione perché potrebbe comportare alcune sfide e limitazioni quali:

- **Performance**: La ricorsione può comportare una maggiore complessità computazionale rispetto ad altre soluzioni, poiché ogni chiamata ricorsiva comporta l'aggiunta di un nuovo livello alla pila delle chiamate. Può aumentare la quantità di memoria utilizzata e ridurre la velocità di esecuzione del programma.
- **Debugging**: La ricorsione può essere difficile da testare a causa della sua natura ripetitiva e della profondità della pila delle chiamate. Questo può rendere più complesso capire dove si verifica un errore o dove un problema sta influenzando il comportamento del programma.

Qualora non fosse possibile evitare l'uso di ricorsione, la decisione dovrà essere adeguatamente giustificata tramite commenti.

2.2.6.5 Metriche

Metrica	Nome	Riferimento
MCCM	Complessità Ciclomatica per metodo	§B.4.3
MCC	Code Coverage	§B.4.3
MSC	Statement Coverage	§B.4.3
MBC	Branch Coverage	§B.4.3

Tabella 6: Metriche per la Codifica

3 Processi di supporto

3.1 Documentazione

3.1.1 Scopo

Il *processo*$_G$ di documentazione ha l'obiettivo di registrare delle informazioni prodotte da un processo o da un'*attività*$_G$ del *ciclo di vita*$_G$, le decisioni prese dal gruppo e gli *standard*$_G$ che sono stati adottati per lo svolgimento del progetto didattico.
Tali regole dovranno essere applicate da tutti i membri del team ed i sorgenti che contengono tale documentazione verranno inseriti nel *repository*$_G$ privato all'indirizzo: https://github.com/SmokingFingertips/Docs-LaTeX

3.1.2 Descrizione

La documentazione risulta essere è una parte fondamentale del progetto in quanto permette di tracciare tutto quello che concerne il lavoro svolto e le decisioni prese.
Questa sezione, in particolare, raccoglie tutte le *norme*$_G$ relative alla creazione, all'aggiornamento e al mantenimento della documentazione (interna ed esterna) prodotta dal gruppo *Smoking Fingertips* per ogni *fase*$_G$ del *ciclo di vita del software*$_G$.

3.1.3 Aspettative

- Definire delle procedure ripetibili che permettano di uniformare la documentazione che viene prodotta dal gruppo ed il metodo di lavoro
- Raccogliere e organizzare le norme che i membri del team devono seguire così da semplificare l'operazione di scrittura dei documenti.

3.1.4 Ciclo di vita dei documenti

Il ciclo di vita dei documenti è suddiviso in molteplici *attività*$_G$, che possono essere ripetute:

1. Ogni tipo di documento viene **creato** secondo lo specifico template (si veda `Template` §3.1.5);
2. Una volta creato, è necessaria una **pianificazione** della sua stesura: si effettua una suddivisione del documento in sezioni e si assegna ad ogni sezione un redattore;
3. Viene impostata una **bozza** del documento a partire dai contenuti da inserire;
4. Il team **realizza** il documento redigendone il contenuto;
5. Ad ogni nuova aggiunta il redattore dovrà **controllare**, con una procedura automatica, che il contenuto da lui aggiunto rispetti le norme definite nel documento `Norme di Progetto` e che non si verifichino errori in compilazione;

6. Una volta raggiunto un buon livello di sviluppo della versione attuale, il documento deve essere **revisionato** dal verificatore, il quale dovrà accertarsi della correttezza delle modifiche apportate;
7. Il documento dovrà infine essere **approvato** e la versione stessa potrà essere **rilasciata**.

3.1.5 Template LaTeX

Il team *Smoking Fingertips* ha stabilito di adottare più template per la stesura uniforme dei vari tipi di documenti. Questi possono essere trovati nella cartella `Template` del *repository*$_G$ `Docs-LaTeX`.
I template si suddividono in:

- `Verbali`: per i **verbali** (interni,esterni)
- `Documentazione`: per i restanti **documenti** (interni,esterni)

L'uso di template scritti in linguaggio LaTeX facilita le operazioni di creazione e di mantenimento dei documenti, permettendo ai componenti del gruppo di ottimizzare il tempo impiegato nella stesura della documentazione permettendo a chi sta consentendo di concentrarsi sui suoi contenuti.

3.1.6 Struttura dei documenti

Per ogni documento è definita un cartella che ne riporta il nome e che contiene i file che lo compongono. Il file principale avrà il nome presenta alla sezione §3.1.10.3 e raccoglie le sezioni del documento ed i file di configurazione.
I file di configurazione sono:

- `config.tex`: contiene i pacchetti ed i comandi LaTeXdefiniti dal team che sono necessari alla compilazione. Vengono inserite anche le direttive per l'impaginazione dei documenti;
- `intestazione.tex`: contiene il frontespizio del documento in cui vengono inseriti i dati specifici del documento;
- `StoricoDelleModifiche.tex`: contiene lo storico delle modifiche;
- `Introduzione.tex`: sezione relativa all'introduzione del documento.

Ogni sezione dovrà essere posizionata nella cartella `risorse/sezioni` del `Template` e verrà inserita nel documento finale nella pagina principale del documento tramite il comando:

```
\input{NomeCapitolo.tex}
```

Questa condizione permette di lavorare parallelamente su capitoli disgiunti pianificando in maniera più efficace la distribuzione dei *task*$_G$ ai vari componenti del team e permettendo di effettuare una manutenzione più rapida e settoriale.
Quando il file LaTeXprincipale verrà compilato produrrà in output il file `.pdf` corrispondente al documento intero.
Ogni documento dovrà presentare le seguenti sezioni:

3.1.6.1 Intestazione dei documenti

La prima pagina di ogni documento contiene, in quest'ordine, gli elementi elencati di seguito:

- **Logo del gruppo**: reperibile nella cartella `risorse/immagini` formato `.png`;
- **Nome del gruppo**: *Smoking Fingertips* ;
- **Email**: smoking.fingertips@gmail.com;
- **Nome del progetto**: *ShowRoom3D*;
- **Nome del documento**;
- **Versione**: numero di versione del documento in formato `X.Y.Z`;
- **Stato**: indica se il documento è stato approvato o se è in attesa di approvazione;
- **Responsabile**: nominativo del responsabile che ha approvato il documento;
- **Redattori**: lista dei nominativi che hanno contribuito alla stesura del documento;
- **Verificatori**: lista dei nominativi che hanno contribuito alla verifica del documento;
- **Destinatari**: a chi è rivolto il documento;
- **Data di approvazione**: riporta la data di approvazione del documento;
- **Anno accademico**;
- **Sommario**.

3.1.6.2 Storico delle modifiche

La seconda pagina è dedicata allo **storico delle modifiche**, ovvero una tabella che riporta per ogni modifica del documento i seguenti campi:

- **Versione**: versione raggiunta dal documento dopo la modifica;
- **Data**: la data di tale modifica;
- **Nominativo**: il membro del gruppo che ha effettuato la modifica;
- **Ruolo**: il $ruolo_G$ di tale membro;
- **Descrizione**: una breve descrizione.

3.1.6.3 Indice

Successivamente allo storico delle modifiche si colloca l'indice che elenca le sezioni che compongono il documento. In caso di presenza di tabelle o figure esse saranno elencate nelle pagine seguenti all'indice.

3.1.6.4 Corpo del documento

Il contenuto del documento è suddiviso in capitoli, ognuno dei quali è composto da più sezioni.

3.1.6.5 Corpo del verbale

Il contenuto dei verbali verrà espresso nella seguente modalità:

- **Informazioni sull'incontro**:
  - **Luogo**: che può essere:
    - Il luogo fisico dove si è tenuto l'incontro;
    - "online" se si è svolto online.
  - **Data**: formato `YYYY-MM-DD`;
  - **Ora di inizio**;
  - **Ora di fine**;
  - **Partecipanti**: lista dei partecipanti;
  - **Segretario**: nominativo del redattore del verbale.
- **Ordine del giorno**: un riassunto di quello che verrà discusso durante la riunione;
- **Resoconto**: un elenco delle cose affrontate durante la riunione;
- **Decisioni prese**: una tabella che indica:
  - **Il codice identificativo** della decisione nel formato:

    `V[I/E]_[YYYY-MM-DD].[x]`

    dove:
    - `[I/E]` è:
      - I se il Verbale è interno;
      - E se il Verbale è esterno.
    - `[YYYY-MM-DD]` è la data del Verbale;
    - `[x]` è il numero della decisione presa.
  - **La descrizione delle decisioni**.
- Eventuali **argomenti in sospeso** verranno recuperati alla riunione successiva.

3.1.7 Documenti del progetto

I documenti prodotti sono i seguenti:

- `Studio di Fattibilità`
- `Norme di Progetto`;
- `Piano di Progetto`;
- `Piano di Qualifica`;
- `Analisi dei Requisiti`;
- `Specifica Tecnica`;
- `Manuale d'Uso`;
- `Verbali interni ed esterni`;
- `Glossario`.

3.1.8 Sigle dei documenti

Le sigle relative ai documenti sono:

- `SdF`;
- `NdP`;
- `PdP`;
- `PdQ`;
- `AdR`;
- `ST`;
- `MU`;
- `VI/VE`;
- `G`.

3.1.9 Sigle di progetto

Le seguenti sigle identificano

- Le revisioni di progetto previste:
  - ***Requirements and Technology Baseline***$_G$: RTB;
  - ***Product Baseline***$_G$: PB;
  - ***Customer Acceptance***$_G$: CA.
- I ruoli di progetto:
  - **Responsabile di progetto**: RE;
  - **Amministratore**: AM;
  - **Analista**: AN;
  - **Progettista**: PT;
  - **Programmatore**: PR;
  - **Verificatore**: VE.

3.1.10 Convenzioni stilistiche

3.1.10.1 Annotazioni

Per inserire delle annotazioni all'interno dei documenti, verrà utilizzato il *package*$_G$ `todonotes`. Ogni nota scritta, verrà presentata a margine della sezione cui fa riferimento. Per aggiungere una nota sarà sufficiente digitare il comando:

```
\todo{commento}
```

Queste annotazioni serviranno ad indicare parti del documento incomplete o a suggerire eventuali modifiche da effettuare. Graficamente, la nota apparirà come segue . Per un utilizzo più ad ampio spettro delle annotazioni, si rimanda alla documentazione ufficiale.

questa è una nota

3.1.10.2 Indentazioni

Le indentazioni e i ritorni a capo presenti nei contenuti dei documenti sono deprecati in quanto rendono di difficile lettura i file sorgenti con estensione `.tex` durante le *pull request*$_G$ su *GitHub*$_G$. E' però accettato l'uso delle tabulazioni per l'indentazione degli `\item` degli elenchi puntati.

3.1.10.3 Nomi assegnati ai file

Per nominare i documenti si adotta la convenzione *PascalCase*$_G$, quindi:

- Per ogni parola la lettera iniziale va scritta in **maiuscolo**;
- Le **spaziature** fra le parole vanno **omesse**.

Viene pertanto utilizzato il seguente formato:

`NomeDocumento_vX.Y.Z`

dove:

- `NomeDocumento` corrisponde al nome ufficiale del documento;
- `vX.Y.Z` indica la versione del documento (come definito nella sezione §3.4.4).

Di conseguenza i nomi dei documenti principali saranno:

- **Studio di Fattibilità**: `StudioDiFattibilita_vX.Y.Z`;
- **Norme di Progetto**: `NormeDiProgetto_vX.Y.Z`;
- **Piano di Progetto**: `PianoDiProgetto_vX.Y.Z`;
- **Piano di Qualifica**: `PianoDiQualifica_vX.Y.Z`;
- **Analisi dei Requisiti**: `AnalisiDeiRequisiti_vX.Y.Z`;
- **Specifica Tecnica**: `Specifica Tecnica`;
- **Manuale d'Uso**: `Manuale d'Uso`;
- **Glossario**: `Glossario_vX.Y.Z`.

Per la denominazione dei verbali viene utilizzato il formato:

`VX_YYYY_MM_DD`

dove:

- `VX` indica l'acronico relativo al tipo di verbale. La lettera `V` fa riferimento al termine Verbale e la lettera `X` può essere scelta fra:
  - `I` che corrisponde a Interno;
  - `E` che corrisponde a Esterno.
- `YYYY-MM-DD` indica la data in cui è si è tenuta la riunione o il colloquio (secondo il formato descritto nella sezione §3.1.10.6).

Pertanto i verbali verranno definiti come segue:

- **Verbali Interni**: `VI_YYYY_MM_DD`;
- **Verbali Esterni**: `VE_YYYY_MM_DD`.

3.1.10.4 Stile del testo

Si utilizzano:

- **grassetto** per le voci degli elenchi puntati e per tutte le parole ritenute importanti;
- *corsivo* per:
  - il nome del gruppo (*Smoking Fingertips*);
  - il nome dell'azienda *proponente*$_G$ (*Sanmarco Informatica S.p.A*);
  - le parole che si riferiscono al glossario (seguite da $_G$);
  - i nomi di aziende.
- `Monospace` per indicare:
  - degli esempi di codice LaTeX;
  - delle cartelle o dei *repository*$_G$;
  - dei nomi di documenti;
  - delle estensioni di file;
  - dei riferimenti testuali di altre parti del documento.
- **caratteri maiuscoli** per:
  - le iniziali dei nomi;
  - le lettere che compongono un acronimo;
  - i nomi dei file (§3.1.10.3);
  - le iniziali dei *ruoli*$_G$ svolti dai componenti del gruppo.
- il colore azure per indicare:
  - i link ad una risorsa esterna;
  - un riferimento a capitolo/sezione/sottosezione/paragrafo di un documento;
  - email del gruppo.

3.1.10.5 Elenchi puntati

Viene inserito ";" alla fine delle voci dell'elenco tranne l'ultima che termina con ".". Ogni voce dell'elenco inizia con una lettera maiuscola. La definizione del termine è in grassetto come spiegato nella sezione §3.1.10.4.

3.1.10.6 Formato delle date

Per le date viene adottato lo *standard*$_G$ ISO 8601$_G$: `YYYY-MM-DD`.

- `YYYY`: anno con 4 cifre;
- `MM`: mese con 2 cifre;
- `DD`: giorno con 2 cifre.

3.1.11 Strumenti

Gli strumenti di supporto scelti dal gruppo sono:

- **LaTeX**: linguaggio per la stesura di documenti compilati utilizzando *TexLive*$_G$;
- **Visual Studio Code**: IDE realizzato da *Microsoft*$_G$ che grazie all'estensione *LaTeX Workshop*$_G$ diventa un editor per i file con estensione `.tex`;
- ***GitHub***$_G$: piattaforma per l'*hosting*$_G$ di progetti software.

3.1.12 Controllo ortografico

Per garantire la correttezza ortografica dei documenti scritti in LaTeX, ogni membro del gruppo utilizzerà uno strumento di correzione automatica. Nello specifico, per l'*IDE*$_G$ *Visual Studio Code*$_G$ è disponibile l'estensione *Italian - Code Spell Checker*$_G$ che notifica il redattore degli errori di battitura o di accento presenti nel documento tramite sottolineatura. Questa operazione facilita la correzione dei documenti.
In alternativa, si può utilizzare lo strumento Aspell da terminale, lanciando il comando

```
aspell --mode=tex --lang=it check NomeFile.tex
```

direttamente sul file in formato `.tex`.
Questi strumenti aiutano nella correzione degli errori ortografici, ma non sono in grado di individuare tutti i possibili errori. Redattore e revisore dovranno porre particolare attenzione nella revisione manuale del lavoro, in particolare per quanto riguarda la corretta ortografia dei termini tecnici.

3.1.13 Elementi grafici

La documentazione di un progetto software può contenere diversi elementi grafici, come immagini, grafici *UML*$_G$ e tabelle.

3.1.13.1 Immagini

Tutte le immagini presenti nei documenti dovranno essere centrate rispetto al testo ed essere accompagnate da una didascalia che ne descriva il contenuto. Sarà così possibile comprendere immediatamente il contenuto dell'immagine e il suo contesto all'interno del documento.

3.1.13.2 *Diagrammi UML*$_G$

Vengono comunemente utilizzati per modellare i *casi d'uso*$_G$ e per i diagrammi di progettazione. Ogni diagramma viene inserito nella documentazione sotto forma di immagine. Come già previsto per le immagini, anche i diagrammi seguono lo stesso criterio: devono essere centrati rispetto al testo e accompagnati da una didascalia descrittiva.

3.1.13.3 Tabelle

Ogni tabella verrà contrassegnata con una didascalia descrittiva del contenuto, posta sotto di essa e centrata rispetto alla pagina. Nella didascalia di ogni tabella va indicato l'identificativo `Tabella X`, in cui `X` rappresenta il riferimento numero della tabella all'interno del documento, seguito dal testo della didascalia. Fanno eccezione le tabelle dello storico delle modifiche, che non presentano didascalia né numero.

3.1.14 Metriche

Metrica	Nome	Riferimento
MCO	Correttezza Ortografica	§B.4.4
MIG	Indice di Gulpease	§B.4.4

Tabella 7: Metriche per la Documentazione

3.2 Verifica

3.2.1 Scopo

Lo scopo dell'*attività*$_G$ di verifica è quello di fornire evidenza che gli output di un segmento di software siano conformi alle aspettative e ai *requisiti*$_G$ specificati. Tale attività si baserà su criteri di consistenza, completezza e correttezza dei prodotti ottenuti. La verifica viene eseguita durante tutto il *ciclo di vita del software*$_G$, dalla fase di progettazione alla fase di manutenzione.

3.2.2 Aspettative

L'aspettativa è quella di istanziare per ogni prodotto un *processo*$_G$ di verifica atto a garantire *efficienza*$_G$ e correttezza alle *attività*$_G$. La verifica dovrà pertanto verificare l'output di un processo e restituire in output uno stato conforme alle aspettative. Tale processo si basa sull'utilizzo di tecniche di analisi e test al fine di garantire che i prodotti soddisfino i requisiti specificati.
Per garantire il corretto svolgimento del processo di verifica, è necessario rispettare alcuni punti chiave, tra cui l'applicazione di procedure definite, l'adozione di criteri affidabili per la verifica, l'implementazione di una sequenza di fasi successive e la validazione del prodotto una volta completata la fase di verifica.
Questo processo dovrà garantire che il prodotto si trovi in uno stato stabile, in modo da permettere l'avvio della successiva fase di validazione.

3.2.3 Descrizione

Il *processo*$_G$ di verifica viene eseguito su tutti i processi in esecuzione al raggiungimento di un livello di maturità adeguato o in seguito a modifiche dello stato del processo. Per ogni processo, si analizza la qualità dei prodotti generati e dei processi utilizzati, al fine di garantire che siano conformi agli standard di qualità definiti.
Le *attività*$_G$ relative al processo di verifica vengono assegnate ai dei verificatori che analizzano i prodotti e valutano la loro conformità ai vincoli qualitativi specificati nel `Piano di Qualifica`.
Tali operazioni vengono eseguite nell'ordine specificato nel *modello a* V_G.
Nel `Piano di Qualifica` devono essere documentate tutte quelle che sono le attività che andranno a comporre il processo, descrivendone gli scopi, i risultati sperati e quelli ottenuti.
Dando così delle linee guida al fine di una corretta valutazione della qualità, normando e rendendo ripetibile il processo.
Di seguito vengono specificate le possibili attività che possono essere adottate:

3.2.4 Analisi statica

Il presente tipo di analisi è denominato **statica** in quanto non richiede alcuna esecuzione del prodotto. Tale metodologia prevede la revisione critica del codice e della documentazione allegata al fine di verificare la conformità ai vincoli, l'assenza di difetti e la presenza delle proprietà desiderate nel prodotto in esame. Si sottolinea che il successo di quest'*attività*$_G$ è strettamente legato all'esperienza dei verificatori coinvolti.
Questa tipologia di studio è applicabile a qualsiasi prodotto del progetto e può avvalersi di diversi metodi di lettura, descritti successivamente.

3.2.4.1 Walkthrough

Questa tecnica prevede una lettura a pettine della documentazione e del codice. È prevista una collaborazione fra il verificatore e l'autore del prodotto che si può riassume nei seguenti tre passi:

1. **Pianificazione**: dialogo tra autori e verificatori mirato ad individuare proprietà e vincoli che il prodotto deve soddisfare per essere considerato corretto;
2. **Lettura**: il verificatore legge i documenti e/o il codice, cercando errori e non conformità con i vincoli imposti;
3. **Discussione**: dialogo tra autori e verificatori post lettura per confrontarsi sull'esito della lettura;
4. **Correzione**: *attività*$_G$ demandata agli autori, volta a correggere gli errori trovati ai passi precedenti.

Ciascun *processo*$_G$ sarà documentato e tali documenti costituiranno una garanzia per l'adeguata esecuzione del suddetto processo. Questo compito non è automatizzabile in quanto i criteri possono subire frequenti variazioni.
Questo metodo di verifica sarà impiegato soprattutto nelle prime fasi del progetto, in quanto i documenti sottoposti al processo di verifica saranno relativamente semplici e il costo dell'operazione sarà contenuto. L'intera esecuzione del processo verrà documentata e sarà disponibile per la consultazione nel *repository*$_G$ del progetto.

3.2.4.2 Inspection

L'obiettivo di questa tecnica, è quello di rilevare la presenza di difetti nel prodotto in analisi, eseguendo una lettura mirata, anziché a pettine del codice e della documentazione allegata, specificando preventivamente cosa si andrà a verificare. Tali proprietà andranno a comporre delle liste di controllo che fungeranno da checklist per determinare se l'*attività*$_G$ d'ispezione sia stata eseguita correttamente o meno.
Questo *processo*$_G$ viene articolato nei tre passi descritti in seguito:

1. **Pianificazione**: (vedi 3.2.4.1);
2. **Definizione lista di controllo**: viene determinato cosa vada verificato selettivamente;
3. **Lettura**: (vedi 3.2.4.1);
4. **Correzione**: (vedi 3.2.4.1).

Quest'attività è da preferire al *walkthrough*$_G$ in quanto molto più veloce da eseguire, risultando più efficace della precedente avendo a disposizione delle liste di controllo basate su presupposti ben fondati. Tali liste vengono riportate nel `Piano di Qualifica`.

3.2.5 Analisi dinamica

Nel processo di sviluppo del software, è necessario verificare il codice generato per garantire il suo corretto funzionamento. A tal fine, si utilizza l'**analisi dinamica** che rappresenta una categoria di tecniche di analisi che richiedono che il prodotto sia in esecuzione. L'analisi dinamica prevede l'esecuzione di una serie di test case durante l'esecuzione del codice. Lo scopo dei test è quello di verificare la corretta esecuzione del software e di

rilevare eventuali scostamenti tra i risultati ottenuti e quelli attesi. Per questa ragione, l'analisi dinamica non può essere applicata alla documentazione.
L'analisi dinamica deve essere automatizzata e ripetibile, in modo da garantire una valutazione oggettiva del prodotto.
Nell'ambito dell'ingegneria del software, la tecnica principale di analisi dinamica è il test. Per garantire l'efficacia dei test, ogni prova deve essere decidibile e ripetibile. Ciò significa che, dati gli stessi input, il test deve produrre sempre lo stesso output. Inoltre, ogni test deve presentare dei parametri ben definiti, tra cui la descrizione degli input e degli output, il comportamento atteso del software e le condizioni di esecuzione del test.
L'efficacia dei test dipende dalla qualità del codice scritto e dalla corretta identificazione dei requisiti funzionali e non funzionali del sistema.
Sarà compito del verificatore identificare un dominio dei casi di prova, in modo tale per applicare i test ad un numero finito di casi. Ogni test, poiché deve essere ripetibile, deve specificare:

- **Ambiente**: stato iniziale Hardware e Software.
- **Attese**: ingressi richiesti, uscite ed effetti attesi.
- **Procedure**: le procedure coinvolte nel test e le procedure di analisi dei risultati.

Tali test andranno poi automatizzati, a tal fine verranno usati degli specifici strumenti dal test stesso, tali strumenti sono:

- **Driver**: componente attiva che pilota il test, il suo compito fondamentalmente è quello di attivare l'oggetto del test.
- **Stub**: anche detto calco, ovvero una componente passiva che simula altre parti utili al test ma non direttamente oggetto di test.
- **Logger**: strumento attraverso il quale il test registra i risultati ottenuti durante la sua esecuzione.

Segue un elenco delle varie tipologie di test messe a disposizione del $fornitore_G$, classificate in base alla grandezza dell'oggetto del test.

3.2.5.1 Test di unità

I $test_G$ di unità sono definiti sulle più piccole unità software testabili in maniera individuale, si prestano quindi ad un alto grado di parallelismo, essi vengono pianificati durante la progettazione di dettaglio. Se vengono eseguiti più test sulla stessa unità, ciò viene chiamato "test suite" per quella specifica unità.
Questo tipo di test assume due connotazioni differenti in base a ciò che si vuole effettivamente testare, esistono infatti test funzionali o strutturali che hanno compiti differenti come spiegato in seguito.

Tipo	Descrizione	Vantaggi	Svantaggi
Test Funzionali	L'obiettivo è quello di determinare se dato un certo input l'output effettivo corrisponde con quello atteso, producono "requirement-coverage".	Sono molto veloci da realizzare.	Non assicurano la copertura completa copertura del codice dell'unità.
Test Strutturali	L'obiettivo è appunto quello di testare la completa copertura dei cammini possibili dell'unità, vengono costruiti ad hoc per attivare un determinato cammino, creando così una batteria di test che dovrà coprire completamente il codice dell'unità, producono "structural-coverage".	Assicurano che ogni cammino venga eseguito.	A seconda della complessità ciclomatica del codice dell'unità possono risultare più onerosi o difficili da costruire.

3.2.5.2 Test di integrazione

I $test_G$ di integrazione vengono pianificati durante la $fase_G$ di progettazione architetturale e hanno l'obiettivo specifico di verificare la corretta integrazione di ciò che hanno prodotto i test di unità, ovvero le singole unità architetturali, rilevandone eventuali difetti. Il flusso è incrementale, in quanto, integrando oggetti già testati, si va di volta in volta a creare una baseline di prodotto se il test va a buon fine.
Essi sono inoltre reversibili in modo da poter tornare ad uno stato sicuro in caso di errori durante lo svolgimento di questo tipo di test, l'integrazione può avvenire con un approccio "top-down" oppure "bottom-up".

3.2.5.3 Test di sistema

Questo tipo di $test_G$ viene definito durante la $fase_G$ di Analisi dei Requisiti e ha l'obiettivo specifico di misurare la copertura dei $requisiti_G$ software specificati nel $capitolato_G$ d'appalto, viene svolto dopo i test di integrazione, ovvero quando tutte le componenti del sistema sono state integrate ed è preludio del collaudo, questo tipo di test viene svolto anche durante l'attività di validazione.

3.2.5.4 Test di regressione

I $test_G$ di regressione accertano che correzioni o estensioni effettuate su specifiche unità architetturali, non danneggino il resto del sistema. Questi test consistono nella ripetizione selettiva di test di unità, test di integrazione e test di sistema necessari al fine di poter asserire che tali cambiamenti non intacchino funzionalità precedentemente verificate, causando così una regressione.

Tipo	Descrizione	Vantaggi	Svantaggi
Top-down	L'integrazione parte dalle componenti di sistema con più dipendenze e più valore esterno, rendendo disponibili da subito le funzionalità ad alto livello.	Permette di testare per maggior tempo le funzionalità cardine, rese disponibili per prime	Richiede molti stub, i quali non incrementano le funzionalità
Bottom-up	L'integrazione parte dalle componenti di sistema con meno dipendenze e più valore interno ovvero meno visibili all'utente.	Richiedono pochi stub	Ritarda l'implementazione di funzionalità utente.

Tale test coinvolge i processi di Problem Resolution e Change management, valutando se e come effettuare modifiche al codice.

3.2.5.5 Test di accettazione

Questo tipo di $test_G$ accerta il soddisfacimento completo dei $requisiti_G$ utente, con una condizione aggiuntiva, questo tipo di test infatti, deve essere svolto obbligatoriamente alla presenza del $committente_G$.

3.2.5.6 Codici relativi ai test

Per classificare ogni $test_G$ che il team effettuerà durante l'$attività_G$ di verifica abbiamo deciso di associare un codice identificativo per ciascun test nel formato

`T[tipo][codice]`

dove:

- `[tipo]` è il tipo di test:
  - U per i test di unità.
  - I per i test di integrazione.
  - S per i test di sistema.
  - R per i test di regressione.
  - A per i test di accettazione.
- `[codice]` è un numero associato al test all'interno del suo tipo:
  - se il test non ha padre, è un semplice numero progressivo.

- se il test ha un padre sarà nel formato: `[codice.padre].[codice.figlio]`
  - `[codice.padre]` identifica in maniera univoca il padre del test all'interno della categoria di test relativi al suo tipo.
  - `[codice.figlio]` è un numero progressivo per identificare il test.

3.2.5.7 Stato del test

Ad ogni $test_G$ verrà poi associato uno stato che ne rappresenterà il risultato, l'insieme dei risultati dei test verrà riportato in quello che chiameremo $cruscotto_G$ dei test esso infatti potrà essere:

- **N-I** quando il test non è ancora stato implementato.
- **Passato** quando il test riporta esito positivo.
- **Non passato** quando il test riporta esito negativo.

3.2.6 Metriche

Metrica	Nome	Valore di accettazione	Valore preferibile
M15PTFS	Percentuale test funzionali superati	≥90%	100%

Tabella 8: Metriche per la verifica

Questa tabella è soggetta a un aggiornamento continuo in funzione dell'andamento del progetto. Saranno integrate ulteriori metriche per monitorare lo stato di avanzamento del processo di testing del codice prodotto e per fornire uno standard qualitativo. Pertanto, la tabella non è ancora in uno stato definitivo.

3.3 Validazione

3.3.1 Scopo

Lo scopo di tale $processo_G$ è quello di avere conferma, tramite esame, che:

1. il prodotto software nella sua totalità, sia conforme ai bisogni e funzionante secondo la logica di progettazione.
2. I $requisiti_G$ implementati attraverso il software siano pienamente soddisfatti.

3.3.2 Aspettative

L'aspettativa è quella di arrivare ad un prodotto finale che possa essere rilasciato, previa approvazione da parte del $proponente_G$, arrivando così alla fine del ciclo di vita del progetto.

3.3.3 Descrizione

Il *processo*$_G$ di validazione in ingresso prenderà i *test*$_G$ che verranno svolti durante l'*attività*$_G$ di verifica (normati nella corrispondente sezione delle `Norme di Progetto`), successivamente verrà effettuato il cosiddetto test di accettazione che sara il "core" effettivo del processo che porterà alla validazione del prodotto.

3.3.4 Test di accettazione

Il *test*$_G$ di accettazione coincide con l'operazione di collaudo, l'unica differenza è che deve essere effettuato in presenza del *proponente*$_G$, l'obiettivo è dimostrare che il prodotto soddisfi tutti i *requisiti*$_G$ del *capitolato*$_G$ d'appalto.
La figura principale coinvolta in questo processo è quella del verificatore, che, prima dovrà occuparsi di eseguire test di sistema in un ambiente identico a quello che sarà il contesto in cui il prodotto verrà rilasciato, una volta completati con esito positivo si potrà passare al test di accettazione.

3.3.5 Metriche

Metrica	Nome	Valore di accettazione	Valore preferibile
M15PTFS	Percentuale test funzionali superati	≥90%	100%

Tabella 9: Metriche per la verifica

Questa tabella è soggetta a un continuo processo di aggiornamento e, pertanto, non si trova ancora in uno stato finale. In conformità con il piano di progetto, i Test di Accettazione sono stati pianificati per il periodo CA e, pertanto, non sono previste metriche di avanzamento durante il periodo PB. Di conseguenza, la tabella sarà aggiornata in funzione dello stato di avanzamento del progetto.

3.4 Gestione della configurazione

3.4.1 Scopo

L'obiettivo del presente *processo*$_G$ è quello di organizzare, coordinare e tracciare la procedura di modifica della documentazione e del codice prodotto, al fine di agevolare, in modo indiretto, altre *attività*$_G$ di progetto. Nello specifico, questo processo favorisce quelle attività che richiedono di controllare quali modifiche siano state apportate a un particolare documento in un determinato momento e le ragioni sottese a tali decisioni.

3.4.2 Aspettative

L'implementazione di questo processo può portare a diverse aspettative e vantaggi per il progetto. In questo caso, le attese riguardano tre aspetti principali:

- **rendere la produzione di codice e documentazione più sistematica**: il processo di stesura sarà organizzato e strutturato, garantendo che ogni passo sia documentato in modo completo e coerente. Questo permetterà di avere una visione d'insieme più chiara e di ridurre il rischio di errori o mancanze nella documentazione;
- **uniformare gli strumenti utilizzati** per lo sviluppo del progetto: i membri del team coinvolti nel progetto devono utilizzare gli stessi software, strumenti e metodologie, al fine di rendere più facile la collaborazione e garantire una maggiore coerenza nel risultato finale;
- **classificare i prodotti dei vari processi implementati**: ogni prodotto o artefatto creato durante lo sviluppo viene categorizzato in modo sistematico e coerente, permettendo una facile identificazione e accessibilità. Ciò garantisce una maggiore efficienza e produttività nel processo di sviluppo e facilita la gestione del progetto nel suo complesso.

3.4.3 Descrizione

In questa sezione si presentano le $norme_G$ adottate dal team per favorire la tracciabilità della documentazione e del codice prodotto.
Il processo di gestione della configurazione si occupa di raggruppare e organizzare tutti gli strumenti necessari alla configurazione di tutto ciò viene utilizzato per la produzione di documenti, codice e diagrammi, per il versionamento e peril coordinamento del gruppo.
Questo processo garantisce che ogni elemento prodotto durante lo sviluppo del software sia adeguatamente controllato e versionato, consentendo di monitorare lo stato di avanzamento del progetto e di tenere traccia delle modifiche apportate.
La gestione della configurazione assicura che tutti i membri del team utilizzino gli stessi strumenti e lavorino sulle stesse versioni, evitando possibili problemi di compatibilità e di perdita di dati.
Tra gli strumenti utilizzati per la gestione della configurazione troviamo:

- software per il controllo di versione
- tool per la gestione delle build e delle release
- strumenti per la gestione dei bug e dei problemi
- strumenti per la collaborazione e la comunicazione tra i membri del team.

3.4.4 Codice di versionamento

Per tenere la storia delle variazioni apportate ad un documento, viene automaticamente generata una nuova versione che contempla tali modifiche. Ogni versione è identificata dal suo codice nel formato:

`X.Y.Z`

dove:

- `X`: rappresenta il rilascio pubblico del documento, corrispondente ad una versione approvata dal responsabile. La numerazione di `X` inizia da 0;
- `Y`: rappresenta una revisione complessiva del prodotto per verificarne la coesione e consistenza a fronte di modifiche circoscritte apportate a parti diverse. La numerazione di `Y` inizia da 0 e riparte da questo valore ad ogni incremento di `X`;
- `Z`: rappresenta viene invece incrementato ad ogni modifica circoscritta e relativa verifica del prodotto. Questa fase corrisponde concettualmente alla funzione dei test di unità nel software. La numerazione di `Z` inizia da 0 e riparte da tale valore ad ogni incremento di `X` o `Y`.

Ogni modifica, produce un incremento di versione, che assume un peso diverso a seconda della posizione della cifra incrementata, in quanto, i cambiamenti più importanti vengono riflessi con un incremento della cifra più significativa.

3.4.5 Tecnologie adottate

3.4.5.1 Jira

Per il coordinamento del lavoro del team, viene utilizzato il software *Jira*$_G$ come *Issue Tracking System*$_G$.

3.4.5.2 Git

Il *versionamento*$_G$ del codice sorgente, sia software che documentazione è gestito tramite il software di versionamento distribuito *git*$_G$.

3.4.5.3 Github

Il coordinamento delle operazioni di versionamento è affidato alla piattaforma web *GitHub*$_G$.

3.4.6 Lista dei repository

Vengono utilizzati 3 *repository*$_G$:

- `https://github.com/SmokingFingertips/Docs-LaTeX`: repository privato riservato al team per la condivisione del codice sorgente relativo alla documentazione;
- `https://github.com/SmokingFingertips/Docs`: repository pubblico destinato ai committenti/proponenti dove vengono condivisi i codici sorgente e tutta la documentazione approvata nel repository privato;
- `https://github.com/SmokingFingertips/UnrealShowRoom3D`: repository privato riservato al team per la condivisione del codice sorgente relativo al prodotto da realizzare.

3.4.7 Organizzazione dei repository documentali

3.4.7.1 Repository `Docs`

Il repository denominato `Docs` è organizzato in cartelle, ognuna delle quali rappresenta una revisione di progetto (`RTB` (ossia *Requirements and Technology Baseline*$_G$), `PB` (ossia *Product Baseline*$_G$), `CA` (ossia *Customer Acceptance*$_G$)).
Ogni cartella poi ha due sottocartelle associate, denominate:

- `Documentazione Esterna` che è la *directory*$_G$ utilizzata per contenere il codice sorgente relativo alla documentazione esterna richiesta;
- `Documentazione Interna` che è la directory utilizzata per contenere il codice sorgente relativo alla documentazione interna richiesta.

3.4.7.2 Repository `Docs-LaTeX`

Il repository denominato `Docs-LaTeX` segue la stessa organizzazione del repository precedentemente descritto. Tuttavia, a differenza di quest'ultima, contiene una cartella aggiuntiva chiamata `Template` che include dei file scritti in linguaggio LaTeX. Questi file vengono utilizzati per compilare e creare i documenti.

3.4.7.3 Gerarchia dei file

Le cartelle relative alle revisioni quindi saranno organizzate come segue:

- `RTB` contenente i documenti da consegnare alla revisione Requirements and Technology Baseline suddivisa in
  - **Documenti Esterni**
    - `Piano di Qualifica v1.0.0`;
    - `Piano di Progetto v1.0.0`;
    - `Analisi dei Requisiti v1.0.0`;
    - `Glossario v1.0.0`;
    - una cartella `Verbali` contenente i verbali esterni.
  - **Documenti Interni**
    - `Norme di Progetto v1.0.0`;
    - una cartella `Verbali` contenente i verbali interni.
- `PB` contenente i documenti da consegnare alla revisione Product Baseline suddivisa in
  - **Documenti Esterni**
    - `Piano di Qualifica v2.0.0`;
    - `Piano di Progetto v2.0.0`;
    - `Analisi dei Requisiti v2.0.0`;

        * `Specifica Tecnica v1.0.0`;
        * `Manuale Utente v1.0.0`;
        * `Glossario v2.0.0`;
        * una cartella `Verbali` contenente i verbali esterni.
    - **Documenti Interni**
        * `Norme di Progetto v2.0.0`;
        * una cartella `Verbali` contenente i verbali interni.

- `CA` contenente i documenti da consegnare alla revisione Customer Acceptance suddivisa in
    - **Documenti Esterni**
        * `Piano di Qualifica v3.0.0`;
        * `Piano di Progetto v3.0.0`;
        * `Analisi dei Requisiti v3.0.0`;
        * `Specifica Tecnica v2.0.0`;
        * `Manuale Utente v2.0.0`;
        * `Glossario v3.0.0`;
        * una cartella `Verbali` contenente i verbali esterni.
    - **Documenti Interni**
        * `Norme di Progetto v3.0.0`;
        * una cartella `Verbali` contenente i verbali interni.

3.4.7.4 Tipologie di file

Nelle cartelle sono inclusi:

- file con estensione `.tex` per i sorgenti LaTeX;
- file con estensione `.pdf` che rappresentano la compilazione all'ultima versione dei documenti. Rappresentano il materiale che dovrà essere consegnato;
- immagini da inserire all'interno dei documenti;
- file `.gitignore` che specifica i file che non vengono tracciati intenzionalmente e che git_G deve ignorare. I file già tracciati da git non sono interessati.

3.4.8 Organizzazione del repository per il codice

Il repository `UnrealShowRoom3D` è composta unicamente dalla cartella `ShowRoom`. All'interno della cartella principale, sono presenti diverse sottocartelle, ognuna dedicata a un particolare aspetto del progetto:

- **Binaries**: contiene i file binari generati dalla compilazione del codice sorgente del progetto.

- **Config**: contiene i file di configurazione del progetto come le impostazioni di rendering, input, audio e fisica.
- **Content**: contiene tutti gli *asset*$_G$ del gioco come modelli 3D, texture, animazioni, livelli e script.
- **Intermediate**: contiene i file temporanei generati durante la compilazione e l'elaborazione dei dati del progetto.
- **Saved**: contiene i dati generati dal progetto come i salvataggi del gioco, le immagini di debug e i file di cache.
- **Source**: contiene il codice sorgente del progetto come file `.cpp` e `.h`, che possono essere utilizzati per personalizzare e ampliare il funzionamento del motore di gioco.

La cartella in questione è stata creata utilizzando il motore grafico *Unreal Engine*$_G$. Di conseguenza, la struttura della cartella segue le convenzioni standard previste dal motore grafico.

3.4.9 Sincronizzazione

Il flusso di lavoro adottato prevede l'utilizzo di diramazioni dal *branch*$_G$ principale denominato `main`, il cui scopo è quello di completare un'attività specifica o risolvere una issue segnalata. In entrambi i casi, il nome della diramazione sarà basato sul corrispondente *ticket*$_G$ presente su *Jira*$_G$. Questa metodologia permette al team di sviluppo di gestire le risorse in modo più efficiente, parallelizzando il lavoro e completando più *task*$_G$ contemporaneamente. Una volta che un membro del team avrà completato il proprio *compito*$_G$, si procederà con la verifica e, in seguito, l'approvazione del lavoro svolto.

3.4.9.1 Branch

Ogni *attività*$_G$ verrà assegnata ad un membro del team il quale aprirà un branch specifico a partire dal branch `main`. L'utilizzo di un `feature branch` permetterà di evitare sovrascritture di quanto realizzato da altri membri del gruppo, correggendo eventuali merge-conflicts in un ramo separato. Una volta che sarà completata la verifica per tale attività, verrà eseguito il merge di tale branch all'interno del ramo principale.

3.4.9.2 Pull Request

Quando un compito è stato ultimato ed è pronto per la verifica, sarà responsabilità del membro del gruppo che ha creato il branch aprire una *pull request*$_G$. Il merge nel branch principale avverrà solamente dopo aver ricevuto esito positivo dall'*attività*$_G$ di verifica.

3.4.10 Comandi base *git*$_G$

La sezione seguente presenta un elenco di comandi di base git che i membri del team utilizzeranno durante l'intera durata del progetto. La conoscenza di questi comandi di base è essenziale per lavorare in modo efficace e produttivo durante lo sviluppo del progetto. In questa sezione, verranno illustrati i comandi fondamentali necessari per

gestire il repository del progetto, dalla creazione di una copia locale del repository alla condivisione delle modifiche apportate con gli altri membri del team.
Si noti che i comandi di base git elencati di seguito possono essere utilizzati specificamente a linea di comando, sia attraverso la *CLI*$_G$ di git che tramite il terminale integrato di *Visual Studio Code*$_G$. Alcuni dei comandi base di GitHub includono:

1. `git clone URL_repository`: per clonare il repository git nella propria macchina, a partire dall'URL del repository remoto;
2. `git add -a`: aggiunge tutti i file modificati o nuovi all'area di staging per il commit;
3. `git commit -m "SR3D-xxx messaggio"`: conferma le modifiche apportate ai file nell'area di staging e crea una nuova revisione della storia del repository. Il comando `-m` permette di aggiungere un messaggio al commit che, come in esempio, dovrà contenere il numero del task svolto dal componente del gruppo ed una breve descrizione di quanto è stato realizzato;
4. `git push`: invia le modifiche apportate alla tua copia locale del repository sul server remoto;
5. `git pull`: scarica le ultime modifiche dal repository remoto in locale nel proprio computer;
6. `git branch nome_branch`: crea un nuovo branch nel repository;
7. `git merge nome_branch`: unisce i cambiamenti apportati nel branch `nome_branch` nel branch in cui il repository è posizionato;
8. `git fetch`: scarica i cambiamenti dal repository remoto senza applicarli al repository locale.

3.4.11 Modifiche al repository

Nel gruppo, tutti i membri hanno il permesso di apportare modifiche ai file elaborati, ad eccezione di quelli presenti nel ramo principale (`main`). Per modificare questi file, è necessario fare una richiesta di *pull request*$_G$ (come definito nella sezione §3.4.9.2), che richiede l'approvazione di almeno un altro membro del gruppo. I cambiamenti minori ai codici e ai documenti possono essere apportati autonomamente da qualsiasi membro del gruppo. Tuttavia, se necessario, il membro deve essere in grado di giustificare il suo intervento al resto del team.
Per quanto riguarda le modifiche considerevoli ai documenti già approvati o al codice già verificato, è richiesto un processo più formale. In primo luogo, il membro deve contattare il responsabile che ha precedentemente approvato il prodotto. Successivamente, deve esporre le modifiche che si intendono apportare e le motivazioni che stanno dietro a tali modifiche. Se autorizzato, il membro può effettivamente apportare le modifiche al prodotto. Questo processo è stato progettato per garantire che il codice e i documenti siano modificati solo con l'approvazione adeguata e per prevenire eventuali errori o conflitti.

3.5 Gestione della qualità

3.5.1 Scopo

Il *processo*$_G$ di gestione della qualità ha il fine di garantire che il prodotto software sviluppato sia funzionale, affidabile, sicuro, manutenibile e compatibile con l'ambiente in cui deve essere eseguito nonché dei servizi correlati, come l'assistenza tecnica e la formazione degli utilizzatori, soddisfare i requisiti degli utenti, ed essere conforme agli standard e alle norme in vigore. I processi adottati dal team dovranno pertanto rispettare gli obiettivi di qualità prefissati.

3.5.2 Aspettative

Le aspettative di questo processo comprendono il conseguimento della qualità del prodotto in conformità alle richieste del proponente, la dimostrazione oggettiva della qualità del prodotto, il miglioramento della qualità nell'organizzazione delle attività del gruppo e dei processi, e infine la piena soddisfazione del proponente.

3.5.3 Descrizione

Questa sezione è dedicata alle norme adottate dal team per la gestione della qualità.
Tale *processo*$_G$ si pone l'obiettivo di garantire la qualità dei processi adottati dal *fornitore*$_G$ e dei prodotti sviluppati, al fine di soddisfare quelle che sono le aspettative del cliente e del *proponente*$_G$.
L'aspettativa è quella di arrivare ad ottenere un *miglioramento*$_G$ continuo dei processi perseguendo *standard*$_G$ di qualità.
A tal proposito il gruppo ha adottato la metodologia *PDCA*$_G$ che verrà descritta in seguito.

3.5.4 Controllo della qualità

Si fissano le regole di comportamento, la pianificazione e l'attuazione di procedure e attività finalizzate a garantire il livello di qualità desiderato di un prodotto o servizio attraverso la verifica e la valutazione sistematica di processi, materiali e risultati finali. Questo tipo di controllo di qualità viene incluso nel `Piano di Qualifica`, che definisce le specifiche di qualità per un progetto o un prodotto, identifica le attività di controllo necessarie per soddisfare tali specifiche e stabilisce le responsabilità per l'attuazione di queste attività. Il controllo di qualità nel `Piano di Qualifica` è quindi un'importante componente del processo di assicurazione della qualità e aiuta a garantire che i prodotti o servizi soddisfino gli standard di qualità richiesti.

3.5.5 Principi della qualità

La norma ISO 9000:2000$_G$ (*Sistemi di gestione per la qualità- Fondamenti e terminologia*) riporta un modello che introduce gli otto principi di gestione della qualità:

- l'orientamento al cliente;

- la leadership;
- il coinvolgimento del personale;
- l'approccio per processi;
- l'approccio sistemico alla gestione;
- il miglioramento continuo;
- le decisioni basate sui dati di fatto;
- i rapporti di reciproco beneficio con i fornitori.

Tali principi di gestione per la qualità possono essere usati dal responsabile per guidare l'organizzazione verso il miglioramento delle prestazioni.

3.5.6 Istanziazione di un processo

Il gruppo ha deciso di garantire la qualità attraverso la cura e il monitoraggio sin dall'inizio delle attività di progetto, partendo dalla creazione dei processi. Per raggiungere questo obiettivo, i membri di *Smoking Fingertips* hanno stabilito le seguenti regole per definire ogni processo:

- ogni processo deve avere un obiettivo unico per ridurre la complessità;
- gli obiettivi dei processi non devono sovrapporsi tra loro, ma se c'è un conflitto, il gruppo valuterà la revisione della pianificazione o la modifica del processo;
- quando si inizia un nuovo processo, l'organizzazione delle risorse deve considerare gli altri processi in corso per garantire un utilizzo efficiente e ragionevole delle risorse umane, temporali e materiali;
- all'istanziazione di ogni processo deve corrispondere una nota sulla durata e la data di conclusione;
- ogni processo deve essere analizzato preventivamente per i rischi che potrebbe comportare e pianificare le strategie per contrastarli.

3.5.7 Gestione del cambiamento

Avviare un processo di adozione dei principi della qualità secondo lo standard ISO 9001:2000$_G$ comporta la realizzazione di un cambiamento che influisce sulla cultura dell'organizzazione e quindi sui suoi comportamenti.
Lo standard suggerisce due modi per affrontare il cambiamento:

- **Reattivamente**: quando il cambiamento viene vissuto in termini di reazione, non arreca alcun vantaggio, ma lo si vive come una costrizione e senza convinzione. È la condizione peggiore che si possa realizzare per adeguarsi a nuove situazioni ed esigenze.

- **Proattivamente**: se il cambiamento viene compreso come necessità e per i vantaggi che comporta, allora la condizione migliore è quella di cercare di predisporre le condizioni per accettarlo, gestirlo e per migliorare. Questo richiede un lavoro all'interno da parte della direzione che aiuti a rendere consapevole il personale. E la condizione migliore.

Al termine cambiamento lo standard attribuisce il significato di:

- cambiare un comportamento;
- operare in maniera diversa;
- arrivare a uno stato diverso da quello attuale;
- sostituire un membro con un altro;
- variare.

3.5.7.1 Fattori che portano al cambiamento

Si possono individuare diversi fattori che agiscono, che lo standard riassume in:

- Fattori esterni:
  - l'incremento di competitività;
  - la variazione dei prezzi;
  - la tecnologia;
  - le nuove leggi;
  - le richieste del consumatore.
- Fattori interni:
  - la riduzione dei costi;
  - i nuovi modelli gestionali;
  - lo sviluppo;
  - l'introduzione di nuovi prodotti/servizi.

3.5.7.2 Condizioni per un cambiamento efficace

Il successo del cambiamento dipende dalla posizione che esso assume nei confronti della cultura organizzativa e delle attitudini e credenze personali. Più sarà alto il grado di coinvolgimento delle persone a tutti i livelli, più facilmente si cambierà. Per ottenere ciò è necessario:

- assicurarsi che gli obiettivi siano compresi da tutti;
- analizzare il team per capire dove intensificare gli interventi;
- costruire il cambiamento senza stravolgere le pratiche comuni;

- assicurarsi che l'appoggio del responsabile sia forte;
- curare la comunicazione;
- coinvolgere il proponente ed i committenti.

3.5.8 PDCA

Il *PDCA*$_G$ è una metodologia iterativa che permette il controllo e il miglioramento continuo dei processi e dei prodotti.
Per ottenere un *miglioramento*$_G$ effettivo, è necessario svolgere tutte e quattro le *attività*$_G$ costantemente tali fasi sono:

3.5.8.1 Plan

Consiste nello svolgere le attività di pianificazione atte a stabilire quali processi debbano essere attivati e in quale ordine per arrivare al raggiungimento di obiettivi specifici.

3.5.8.2 Do

Consiste nell'esecuzione materiale di ciò che è stato deciso nella *fase*$_G$ di pianificazione, misurando i risultati raccogliendo dati durante lo svolgimento della stessa.

3.5.8.3 Check

Consiste nello studio e interpretazione dei dati emersi con l'esecuzione (Do), tali dati vengono valutati sulla base di specifiche metriche prefissate e confrontati con i risultati previsti nel Plan.

3.5.8.4 Act

Consiste nel consolidamento di quanto di buono è emerso al passo precedente (Check) e nell'attuazione di strategie correttive al fine di migliorare ciò che non ha portato un risultato coerente con quanto previsto, analizzandone a fondo le cause.
In questo modo il ciclo PDCA verrà via via raffinato e ogni iterazione successiva porterà valore aggiunto.

3.5.9 Strumenti

Gli strumenti utilizzati per la gestione della qualità sono le metriche.
In questa sezione verranno presentate le regole che normano le metriche adottate e i conseguenti obiettivi prefissati.

3.5.10 Struttura delle metriche

Le metriche adottate verranno presentate in seguito, in elenchi strutturati nel seguente modo:

- **Codice**: identificativo della *metrica*$_G$ nel formato:

`M[numero][sigla]`

dove:

- `M` sta per metrica;
- `[numero]` è un numero progressivo univoco per ogni metrica;
- `[sigla]` è una sigla composta dalle iniziali del nome della metrica.

- **Nome**: specifica il nome della metrica;
- **Descrizione**: breve descrizione della funzionalità della metrica adottata;
- **Scopo**: il motivo per cui è importante tale misura al fine del progetto;
- ed eventualmente anche:
  - **Formula**: come viene calcolata;
  - **Strumento**: lo strumento che viene usato per calcolarla.

3.5.11 Struttura degli obiettivi

Gli obiettivi per ciascuna metrica sono definiti dettagliatamente nel `Piano di Qualifica` in delle tabelle strutturate **per colonne** nel seguente modo:

- **Metrica**: il codice della relativa metrica;
- **Nome**: il nome della metrica;
- **Valore accettabile**: il valore considerato accettabile una volta assunto dalla metrica;
- **Valore preferibile**: il valore ideale che dovrebbe essere assunto dalla metrica.

3.5.12 Metriche

Metrica	Nome	Riferimento
MPMS	Percentuale di Metriche Soddisfatte	§B.4.1
MNRNP	Numero di Rischi Non Previsti	§B.4.1
MPRGM	Percentuale di Rischi Gestiti Male	§B.4.1
MDE	Densità Errori	§B.5.5

4 Processi organizzativi

4.1 Gestione dei Processi

4.1.1 Scopo

Come stabilisce lo *standard*$_G$ ISO/IEC 12207:1995$_G$, il *processo*$_G$ di gestione identifica le *attività*$_G$ e i *compiti*$_G$ di progetto che dovranno essere utilizzati per gestire i relativi processi utilizzati dal team.
Sarà compito del responsabile di progetto:

- Assegnare *ruoli*$_G$;
- Assegnare compiti;
- Determinare e amministrare gli strumenti di coordinamento;
- Definire i rischi e loro gestione;
- Calcolare un preventivo contenente il numero di ore ed i costi, suddivisi per ruolo;
- Calcolare un preventivo a finire dato il consuntivo di ogni periodo;
- Gestire le comunicazioni interne ed esterne.

4.1.2 Descrizione

Le *attività*$_G$ di gestione di processo definite dallo *standard*$_G$ sono:

- Inizio e definizione dello scopo;
- Istanziazione i *processi*$_G$;
- Pianificare: stimare i tempi, le risorse ed i costi;
- Assegnare dei ruoli;
- Assegnare dei compiti;
- Esecuzione e controllo;
- Revisione e valutazione periodica delle attività.

4.1.3 Aspettative

Il *processo*$_G$ ha come obiettivo atteso il produrre i seguenti risultati:

- Pianificare precisamente *attività*$_G$ da eseguire;
- Monitorare i progressi effettuati dal gruppo in maniera efficace;
- Assegnare in maniera corretta ed *efficace*$_G$ *compiti*$_G$ e *ruoli*$_G$;

- Facilitare le comunicazioni fra membri del gruppo;
- Facilitare le comunicazioni verso l'esterno;
- Adoperare processi che permettano di regolare le attività perseguendo *economicità*$_G$;
- Calcolare le risorse necessarie a portare a termine il progetto, in base al budget rimanente;
- Monitorare team, processi e prodotti così da effettuando controlli del progetto efficaci.

4.1.4 Pianificazione

4.1.4.1 Scopo

In conformità con lo *standard*$_G$ ISO/IEC 12207:1995$_G$ il responsabile avrà il *compito*$_G$ di predisporre le *attività*$_G$ relative alla pianificazione. Nello specifico sarà necessario verificare che il piano organizzato sia fattibile e che pertanto possa essere correttamente eseguito dai membri del team. I piani associati all'esecuzione del *processo*$_G$ devono contenere le descrizioni delle attività e le risorse necessarie (tempistiche, tecnologie, infrastrutture e personale). L'obiettivo sarà infatti quello di garantire che ogni componente assuma ogni *ruolo*$_G$ (si veda §4.1.4.4), almeno una volta durante lo svolgimento del progetto.
Tale pianificazione verrà stilata dal responsabile e inserita nel documento `Piano di Progetto` che rappresenterà una descrizione delle attività e dei compiti necessari a raggiungere l'obiettivo prefissato per ogni periodo.
Una definizione più dettagliata del contenuto di tale documento potrà essere visionata alla sezione §2.1.5.3.

4.1.4.2 Descrizione

L'attività di pianificazione verrà articolata come segue nella presente sezione:

- Assegnazione ruoli (§4.1.4.4);
- *Ticketing*$_G$ (§4.1.4.5).

4.1.4.3 Aspettative

L'attività di pianificazione ha come obiettivo atteso stabilire come il team *Smoking Fingertips* intende organizzare il proprio lavoro. In questa sezione verranno descritti i passi dalla definizione dei ruoli fino all'assegnazione specifica delle attività ai vari membri del gruppo.

4.1.4.4 Assegnazione ruoli

Al fine di separare le attività per competenza verranno descritti i 6 *ruoli*$_G$ identificati che i componenti del gruppo ricopriranno nel corso del progetto:

- Responsabile;

- Amministratore;
- Analista;
- Progettista;
- Programmatore;
- Verificatore.

Ad ogni membro del gruppo verrà quindi assegnato un ruolo per ogni periodo. Al termine del progetto ogni componente del gruppo dovrà aver ricoperto tutti i ruoli per un periodo di tempo consono. Inoltre, verranno adottate scelte strategiche per evitare eventuali conflitti di interesse, come ad esempio impedire che un membro del team rediga e poi verifichi il proprio lavoro.
Di seguito la descrizione dei ruoli:

Responsabile

Il responsabile è la figura che coordina il gruppo per l'intera durata del progetto. È un *ruolo*$_G$ fondamentale che si occupa di rappresentare il gruppo *Smoking Fingertips* presso il proponente ed i committenti.
In seguito si elencano i principali compiti che questo ruolo comporta:

- Deve gestire la pianificazione dall'inizio alla fine del progetto: determina le attività da svolgere, le loro date di inizio e di fine e le loro priorità;
- Coordina i membri del gruppo: verifica costantemente lo stato delle attività o dei compiti che devono essere portati a termine;
- Controlla i progressi del progetto, è responsabile della stima dei costi e gestisce l'analisi dei rischi;
- Cura le relazioni che verranno intrattenute con i soggetti esterne;
- Approva la documentazione.

Amministratore

L'amministratore definisce, controlla, gestisce e cura l'ambiente di lavoro e gli strumenti che il gruppo utilizzerà per l'intero periodo di svolgimento del progetto. Dovrà pertanto monitorare che i mezzi da utilizzare proposti ai membri del gruppo, siano *efficienti*$_G$ e che permettano di perseguire qualità. Sarà sua cura verificare che il team segua correttamente le regole definite in questo documento e che i servizi a loro disposizione siano funzionali alle attività previste dalla pianificazione, favorendo la produttività.
L'amministratore sarà inoltre portato a:

- Gestire la configurazione di prodotto;
- Gestire il *versionamento*$_G$ della documentazione;

- Redigere, mantenere la documentazione (in particolar modo le norme e le procedure per la gestione dei *processi*$_G$);
- Amministrare delle risorse, delle infrastrutture e dei servizi richiesti per l'esecuzione dei processi di supporto;
- Determinare strumenti necessari ad automatizzare i processi;
- Risolvere problemi legati alla gestione dei processi.

Analista

L'analista è la figura che approfondisce le esigenze espresse nel *capitolato*$_G$. È maggiormente presente soprattutto nelle prime fasi del progetto, in quanto risulta un ruolo fondamentale per la stesura del documento `Analisi dei Requisiti`. Si occupa in particolar modo delle esigenze di business di chi commissiona il software e di effettuare l'analisi dei requisiti. Il prodotto dell'analisi sarà una serie di specifiche tecniche (fra le quali troviamo *casi d'uso*$_G$ e *requisiti*$_G$) che serviranno da input per le fasi successive. Il lavoro dell'analista si colloca per lo più all'inizio del progetto software ed ha lo scopo di fare da mediatore tra i clienti/committenti e gli sviluppatori del software. Risulta una figura di particolare importanza, in quanto un'errata identificazione dei requisiti da soddisfare può compromettere in modo significativo la successiva attività di Progettazione. Si occupa di:

- Studia la realtà di riferimento, definisce il problema e gli obiettivi da raggiungere;
- Raccoglie e studia i bisogni dei clienti/committenti;
- Trasforma i *needs*$_G$ in requisiti soluzione (durante l'Analisi dei Requisiti);
- Categorizza i requisiti raccolti, in modo da poterli suddividere in impliciti/espliciti, obbligatori/opzionali, ecc.;
- Scrivere i documenti `Studio di Fattibilità` e `Analisi dei Requisiti`.

Progettista

Il progettista ha il compito di sviluppare una soluzione con vincoli accettabili che soddisfi i bisogni individuati dall'Analista. Tale *processo*$_G$ richiede di trasformare i *requisiti*$_G$ in un'architettura che modelli il problema, sintetizzando tali requisiti tramite un approccio sintetico alla progettazione: sarà compito del progettista prendere decisioni tecniche e tecnologiche e seguire lo sviluppo ma non la manutenzione.
Si occupa specialmente di:

- Sviluppare un'architettura robusta e sicura dai malfunzionamenti;
- Effettuare scelte per l'ottenimento di soluzioni affidabili, efficienti, sostenibili e che rispettino i requisiti;

- Decomporre il sistema in componenti e organizzarne le interazioni, i ruoli e le responsabilità per favorire la modularizzazione e il riutilizzo del codice;
- Accertare che l'architettura abbia un basso grado di accoppiamento.

Programmatore

Il programmatore si occupa di realizzare la soluzione tramite l'*attività*$_G$ di codifica. Partecipa alla *fase*$_G$ di implementazione e manutenzione del prodotto: verrà implementata l'architettura prodotta dal progettista, assicurandosi che aderisca alle specifiche.
Ogni programmatore ha competenze tecniche e forma la categoria più popolosa all'interno del progetto.
Si occupa di:

- Scrivere codice *mantenibile*$_G$;
- Creare *test*$_G$ ad hoc per la verifica e la validazione del codice;
- Scrivere il manuale utente.

Verificatore

Il verificatore si occupa di controllare il lavoro svolto dagli altri membri del gruppo. Si assicura che le norme di progetto e le attese vengano rispettate. Stabilisce se il *compito*$_G$ è stato svolto in maniera corretta, altrimenti suggerisce correzioni. Rimane presente per l'intera durata del progetto.
Le sue mansioni principali sono:

- verificare se l'esecuzione delle *attività*$_G$ di *processo*$_G$ abbia causato errori;
- redigere la sezione "Retrospettiva" del `Piano di Qualifica`, in cui descriverà che verifiche e che metriche sono state adottate all'interno del progetto.

4.1.4.5 Ticketing

Smoking Fingertips adotta *Jira*$_G$ come *Issue Tracking System*$_G$ (ITS). Jira permette una gestione semplice e chiara dei compiti da svolgere. Il responsabile può creare e assegnare i vari *task*$_G$ in modo che ogni individuo sappia sempre che cosa deve fare in ogni periodo e con che scadenza. Inoltre, ogni membro del gruppo può seguire i progressi svolti durante il periodo vedendo lo stato di avanzamento dei vari task dalla schermata della *Board*$_G$.
Ogni qualvolta serva portare a termine un *compito*$_G$ si segue questa procedura:

1. Il responsabile crea e assegna il task su Jira;
2. L'incaricato apre una *branch*$_G$ su *GitHub*$_G$ seguendo la denominazione suggerita da Jira rendendo in questo modo la branch collegata al task;
3. All'inizio del lavoro di produzione il task viene marcato *in corso* su Jira;

4. Finito il lavoro di produzione viene aperta la *pull request*$_G$ su GitHub assegnando il verificatore. Inoltre vengono aggiornate su Jira le ore di lavoro svolte;
5. Il task viene marcato *in revisione* su Jira;
6. Il verificatore si accerta della bontà del lavoro svolto.
    - Se la verifica ha esito positivo:
        (a) Il verificatore conferma su GitHub la bontà dei cambiamenti effettuati e conferma la pull request;
        (b) Il task viene marcato *in approvazione* su Jira;
        (c) Il responsabile effettua il *merge*$_G$ della branch;
        (d) Il responsabile elimina la branch;
        (e) Il task viene marcato *completato* su Jira.
    - Se la verifica ha esito negativo:
        (a) Il verificatore lascia una lista di cambiamenti suggeriti utilizzando GitHub. In questo modo è sempre verificabile che siano state effettuate tutte le modifiche richieste.
        (b) L'incaricato effettua le modifiche richieste e si ritorna al punto 4;

I task sono creati dal responsabile e hanno:

- **Titolo**: un titolo significativo e coinciso;
- **Descrizione**: una checklist di cose da svolgere o una breve descrizione testuale;
- **Assegnatario**: la persona del gruppo incaricata dello svolgimento del *task*$_G$;
- **Intervallo temporale**: date di inizio e di fine assegnate al task. Rappresenta il periodo in cui il compito deve essere ultimato;
- **Etichetta**: nome della *milestone*$_G$ associata;

4.1.5 Coordinamento

4.1.5.1 Scopo

Il coordinamento è l'*attività*$_G$ che gestisce la comunicazione e la pianificazione degli incontri tra le diverse parti interessate in un progetto di ingegneria del software. Ciò include la gestione della comunicazione interna tra i membri del team del progetto e la comunicazione esterna con *proponente*$_G$ e *committenti*$_G$. Il coordinamento è essenziale per garantire che il progetto progredisca in modo *efficiente*$_G$ e che tutte le parti interessate siano informate e coinvolte in ogni *fase*$_G$ del progetto.

4.1.5.2 Descrizione

Le attività di coordinamento che verranno descritte in questa sezione sono: Comunicazione e Riunioni. Per ogni *compito*$_G$ verranno definite le sottosezioni Interna/e ed Esterna/e.

4.1.5.3 Aspettative

Questa attività vuole determinare un insieme definito di comportamenti che ogni membro del gruppo dovrà rispettare durante lo svolgimento del progetto. La comunicazione risulta essenziale in quanto garantisce che il dialogo tra i membri del gruppo, con il proponente e con i committenti sia chiara e concisa. Di conseguenza, questa attività consente ai team di collaborare in modo più produttivo ed efficiente.

4.1.5.4 Comunicazioni

Comunicazioni interne

Le comunicazioni scritte avverranno tramite i due canali principali: *Telegram*$_G$ e *Discord*$_G$.
Telegram è un servizio di messaggistica istantanea e viene utilizzato per soddisfare i seguenti bisogni:

- Avere un canale di conversazione rapido;
- Avere un canale di conversazione informale;
- Avere un canale di conversazione utilizzabile in maniera efficace tramite smartphone.

Discord fornisce due servizi:

- Un servizio di messaggistica con la possibilità di creare diverse conversazioni dividendole per argomento in modo da facilitare il recupero di informazioni;
- Un servizio di videochiamate utilizzato per lo svolgimento delle riunioni interne in modalità remota.

Per dare una struttura al lavoro di gruppo, le discussioni verranno divise in diversi canali. Per le questioni più importanti e urgenti, che devono essere risolte in fretta, creeremo un canale specifico. Per tutti gli altri argomenti, le discussioni verranno organizzate in base al loro argomento o alla loro natura, con canali comuni creati al fine di facilitare la comunicazione. In questo modo, ogni partecipante avrà una chiara idea di dove pubblicare i messaggi per ottenere aiuto o per discutere su un argomento in particolare.
La piattaforma Discord, permette di notificare un messaggio ad uno o più componenti del gruppo tramite i comandi:

- `@username`: il messaggio verrà notificato all'utente `username`;
- `@everyone`: il messaggio verrà notificato a tutti gli utenti del server.

Nel caso in cui Telegram non fosse disponibile per malfunzionamenti, il gruppo si sposterà temporaneamente su Discord, anche per le comunicazioni più informali.

Comunicazioni esterne

Sarà compito del responsabile, per conto del team, di mantenere un dialogo verso l'esterno tramite l'indirizzo di posta elettronica:

smoking.fingertips@gmail.com

Il responsabile di progetto dovrà assicurarsi che ogni membro del gruppo sia a conoscenza delle corrispondenze pervenute dai committenti e dal proponente, applicando le norme relative alle comunicazioni interne definite in precedenza.

4.1.5.5 Riunioni

Per assicurare che le riunioni vengano condotte nel modo più efficiente possibile, ogni riunione deve avere un segretario. Il segretario è responsabile di mantenere l'ordine durante la riunione, accertando che il programma stabilito venga seguito e che l'argomento di discussione sia trattato in modo chiaro ed esaustivo.
Una volta terminata la riunione, riporterà i punti principali della discussione e l'esito della votazione nel verbale (come descritto in seguito).
Le riunioni possono essere interne o esterne e, indipendentemente dalla loro natura, il verbale svolgerà la stessa funzione: consentire ai partecipanti di rivedere successivamente i dettagli della riunione.

Riunioni interne

Le riunioni hanno cadenza settimanale e sono fissate alle ore 15:00 di ogni venerdì. Il giorno e l'orario è stato concordato tra tutti i componenti gruppo al fine di evitare di doversi accordarsi di volta in volta sulle riunioni (come descritto in `VI_2022_11_15.6`).
In caso risultasse necessario, ogni membro del gruppo può decidere di richiedere una riunione supplementare infrasettimanale. In questo caso la data e l'orario verranno stabilite tramite il canale *Telegram*$_G$ dedicato creando un sondaggio.
Tutte le riunioni interne che avverranno online, saranno tenute nell'apposito canale *Discord*$_G$.

Compiti del responsabile

Sarà cura del responsabile:

- Stabilire preventivamente e in modo preciso i punti dell'ordine del giorno indicando gli argomenti specifici e la pianificazione delle attività da svolgere;
- Comunicare variazioni orarie quali anticipazioni, ritardi o cancellazioni;
- Verificare la presenza della maggioranza dei componenti del gruppo;
- Guidare la discussione e raccogliere i pareri dei membri in maniera ordinata;
- Nominare un segretario per la riunione;

- Aprire e assegnare i $task_G$ di stesura, verifica e approvazione del verbale su $Jira_G$;
- Effettuare l'approvazione del verbale.

Doveri dei partecipanti

I partecipanti si impegnano a:

- Arrivare puntuali alle riunioni;
- Avvisare tempestivamente il responsabile in caso di ritardi o assenze;
- Mantenere un comportamento consono durante lo svolgimento delle riunioni;
- Partecipare attivamente durante la trattazione degli argomenti elencato nell'ordine del giorno.

Verbali interni

Lo svolgimento di una riunione ha come obiettivo la risoluzione dei punti stilati nell'ordine del giorno. Durante lo svolgimento della riunione ogni membro del gruppo prende appunti nel canale testuale apposito del server Discord denominato `#appunti-riunioni`. Al termine di ogni riunione viene creato un $ticket_G$ su Jira che prevede la stesura, la verifica e l'accettazione del verbale. Sarà cura del Segretario redigere il verbale includendo tutte le informazioni utili riportate dagli appunti sopra citati nel formato indicato alla sezione §3.1 di questo documento.
La modalità di stesura del verbale interno è reperibile nella sezione §3.1.6

Approvazione delle decisioni

Una decisione si ritiene approvata quando si verificano le seguenti condizioni:

- L'argomento è stato trattato e discusso durante la riunione;
- La maggioranza del gruppo è d'accordo sulla decisione da prendere;
- Sono presenti alla riunione almeno 5 dei 7 componenti del gruppo.

Riunioni esterne

Le riunioni esterne prevedono la partecipazione dei componenti del gruppo *Smoking Fingertips* in unione a dei soggetti esterni, fra i quali figurano il proponente ed i committenti. Tali riunioni, come già indicato in §4.1.5.5, necessiteranno della nomina di un segretario che in seguito dovrà redigere il verbale.
Per effettuare le riunioni con il proponente viene utilizzato *Google Meet*$_G$ o *Zoom* all'indirizzo che verrà fornito al team di volta in volta.
I membri del gruppo si impegnano ad essere sempre presenti cercando di spostare altri impegni per poter essere presenti a queste riunioni. Qualora gli impegni inderogabili dei membri del gruppo risultassero inderogabili, il responsabile avrà cura di aggiornare il proponente o i committenti, richiedendo di spostare la riunione in un'altra data.

Impegni del gruppo

Il gruppo *Smoking Fingertips* si impegna a:

- Mantenere un dialogo frequente con il proponente durante lo svolgimento del progetto;
- Essere presente alle riunioni con il proponente;
- Arrivare preparato alle riunioni in modo da sfruttare al meglio la possibilità di risolvere dubbi.

Verbali esterni

Come per il caso delle riunioni interne (§4.1.5.5) verrà redatto un *Verbale* con le stesse modalità descritte in precedenza.

Gestione oraria

Al fine di facilitare e rendere più rapida la stesura del consuntivo di periodo il gruppo *Smoking Fingertips* utilizza $Jira_G$. Jira da la possibilità ad ogni membro del gruppo, tramite la funzionalità di $tracciamento_G$ temporale, di tenere traccia del tempo usato per completare i $task_G$.

4.1.6 Metriche

Metrica	Nome	Riferimento
MVP	Valore Pianificato	§B.4.2
MCE	Costo Effettivo	§B.4.2
MVdP	Variazione di Piano	§B.4.2
MVC	Variazione di Costo	§B.4.2
MVR	Variazione dei Requisiti	§B.4.2

Tabella 10: Metriche per la Pianificazione

4.2 Miglioramento

4.2.1 Scopo

Secondo lo $standard_G$ ISO/IEC 12297:1995_G il $processo_G$ di $miglioramento_G$ è un processo che ha lo scopo di stabilire, valutare, misurare, controllare e migliorare un processo del *ciclo di vita del* $software_G$.

4.2.2 Descrizione

Il *processo*$_G$ di miglioramento è composto da due *attività*$_G$:

- Istituzione del processo;
- Valutazione del processo;
- Miglioramento del processo.

4.2.3 Aspettative

Questo processo deve garantire la corretta applicazione dei principi del miglioramento continuo come esposto nel capitolo §3.5

4.2.3.1 Istituzione del processo

Smoking Fingertips dovrà istituire un'istanza dei processi organizzativi per ogni processo afferente al ciclo di vita del software. Inoltre, per ciascuno di questi è appropriato stabilire un meccanismo di controllo mirato a: sviluppare, monitorare e controllare costantemente ogni processo.

4.2.3.2 Valutazione del processo

La valutazione del processo prevede i seguenti compiti:

- Implementazione, documentazione e utilizzo di una procedura di valutazione del processo;
- Conservazione e mantenimento di uno storico delle valutazioni;
- Svolgimento di revisioni ad intervalli di tempo costanti in modo da utilizzarne i risultati ottenuti per accertare l'efficacia e l'efficienza dei processi utilizzati.

Per eseguire quanto esposto fino ad ora il gruppo *Smoking Fingertips* svolgerà alla fine di ogni periodo una procedura di valutazione per ogni processo, rispettando quindi quanto spiegato e previsto dal PDCA (§3.5.8). La valutazione effettuata ad intervalli costanti permette un'individuazione tempestiva dei processi che dovranno essere sottoposti a miglioramento.

4.2.3.3 Miglioramento

Il team applica il miglioramento ai *processi*$_G$ identificati migliorabili dall'attività di analisi sopra descritta. Grazie al miglioramento i processi subiscono delle modifiche migliorative che sono mantenute in uno storico in modo da poter effettuare dei controlli comparativi fra versioni diverse dello stesso processo.

Metrica	Nome	Riferimento
MDE	Densità errori	§B.5.5
MPMS	Percentuale di Metriche Soddisfatte	§B.4.1

Tabella 11: Metriche per il Miglioramento

4.2.3.4 Metriche

4.3 Formazione

4.3.1 Scopo

L'obiettivo di questa *attività*$_G$ è quello di normare il modo in cui il gruppo *Smoking Fingertips* apprende le nozioni richieste per la produzione del progetto.

4.3.2 Aspettative

Ci si attende che il *processo*$_G$ di formazione garantisca che ogni membro del team abbia una competenza sufficiente per poter utilizzare con efficienza le tecnologie scelte dal gruppo per la realizzazione del progetto.

4.3.3 Formazione dei membri del gruppo

Ogni membro del gruppo provvede allo studio individuale delle varie tecnologie necessarie per lo sviluppo del prodotto di seguito elencate:

- **Creazione di documenti**:
  - **LaTeX**: https://www.latex-project.org/help/documentation/.
- **Servizi di terze parti**:
  - **Git**: https://git-scm.com/doc;
  - **GitHub**: https://support.github.com/;
  - **Jira**: https://confluence.atlassian.com/jira.
- **Sviluppo**:
  - **Unreal Engine**: https://docs.unrealengine.com/5.0/en-US/
  - **Automation System**: https://docs.unrealengine.com/5.0/en-US/automation-system-in-unreal-engine/.

Per agevolare l'apprendimento del team, è utile che i membri con maggiore conoscenza condividano nozioni con i membri meno esperti.

A Standard per la qualità

Lo *standard*$_G$ per la valutazione della qualità del software adottato dal gruppo *Smoking Fingertips* è l'ISO/IEC 9126$_G$, ossia una guida che fornisce diverse caratteristiche per garantire che il prodotto soddisfi le esigenze dell'utente e del destinatario finale.
I criteri per la valutazione e la misurazione della qualità si basano sulle seguenti sei caratteristiche chiave:

- **Funzionalità**;
- **Affidabilità**;
- **Usabilità**;
- **Efficienza**;
- **Manutenibilità**;
- **Portabilità**.

Ognuna di queste viene poi suddivisa in sottocaratteristiche per fornire una valutazione più dettagliata della qualità del software.

A.1 Funzionalità

È la caratteristica che fa riferimento alle prestazioni del software rispetto alle specifiche funzionali previste, ovvero della capacità del software di svolgere le attività per cui è stato progettato.
Le sottocaratteristiche di cui è composta sono:

- **Adeguatezza**: è la capacità del software di fornire un set di funzioni che consentano di soddisfare le esigenze dell'utente in modo appropriato;
- **Accuratezza**: è la capacità del software di fornire risultati corretti, graditi ed in linea con le aspettative;
- **Interoperabilità**: è la capacità del software di interagire con gli altri sistemi definiti;
- **Sicurezza**: è la capacità del software di proteggere le informazioni e i dati evitando che si verifichino accessi non autorizzati al sistema;
- **Aderenza alle funzionalità**: è la capacità del software di risultare aderente a *standard*$_G$, convenzioni e regolamenti legali.

A.2 Affidabilità

È la caratteristica che fa riferimento alla capacità del software di funzionare in modo affidabile nel tempo, evitando il più possibile qualsiasi tipo di interruzioni o errori in condizioni specifiche.
Le sottocaratteristiche di cui è composta sono:

- **Maturità**: è la capacità del software di mantenere una certa stabilità durante il suo utilizzo;
- **Tolleranza ai guasti**: è la capacità del software di gestire gli errori in modo appropriato;
- **Recuperabilità**: è la capacità del software di ristabilire un certo livello di prestazioni in caso di malfunzionamenti.

A.3 Usabilità

È la caratteristica che fa riferimento alla facilità con cui gli utenti possono imparare a utilizzare il software, conoscerlo a fondo e che offra un'esperienza lato consumatore positiva.
Le sottocaratteristiche di cui è composta sono:

- **Comprensibilità**: è la capacità del software di essere facilmente intelligibile per gli utenti riguardo all'utilizzo di cui intendono farne;
- **Apprendibilità**: è la capacità del software di consentire all'utente l'apprendimento del prodotto in ogni sua funzionalità, anche mediante l'ausilio della documentazione e dell'aiuto in linea;
- **Operabilità**: è la capacità del software di permettere all'utente di utilizzare e controllare il prodotto in modo intuitivo;
- **Attrattività**: è la capacità del software di essere piacevole da utilizzare per gli utenti;
- **Aderenza all'usabilità**: è la capacità del software di aderire agli *standard*$_G$, convenzioni, guide allo stile grafico e regolamentazioni relative all'usabilità.

A.4 Efficienza

È la caratteristica che fa riferimento alla capacità del software di garantire prestazioni appropriate gestendo in maniera efficiente le risorse del sistema in modo tale che non si causino rallentamenti o altri problemi di performance sotto determinate condizioni.
Le sottocaratteristiche di cui è composta sono:

- **Comportamento rispetto al tempo**: è la capacità del software di garantire una risposta, un tempo di calcolo e quantità di lavoro appropriate;

- **Utilizzo delle risorse**: è la capacità del software di utilizzare un numero e tipo appropriato di risorse in determinate condizioni di utilizzo;
- **Aderenza all'efficienza**: è la capacità del software di aderire agli *standard*$_G$ e convenzioni relative all'efficienza.

A.5 Manutenibilità

È la caratteristica che fa riferimento alla capacità del software di essere facilmente *mantenibile*$_G$ e modificabile per quanto riguarda correzioni, miglioramenti e adattamenti all'ambiente.
Le sottocaratteristiche di cui è composta sono:

- **Analizzabilità**: è la capacità del software di rendere individuabili e note le cause degli errori e malfunzionamenti per poter modificare le parti del prodotto interessate;
- **Modificabilità**: è la capacità del software di favorire l'applicazione di una specifica modifica da implementare;
- **Stabilità**: è la capacità del software di evitare comportamenti ed effetti inaspettati in seguito a modifiche;
- **Provabilità**: è la capacità del software di consentire verifica e validazione;
- **Aderenza alla manutenibilità**: è la capacità del software di aderire agli *standard*$_G$ e convenzioni relative alla manutenibilità.

A.6 Portabilità

È la caratteristica che fa riferimento alla capacità del software di essere trasferito e utilizzato in altri ambienti o sistemi.
Le sottocaratteristiche di cui è composta sono:

- **Adattabilità**: è la capacità del software di essere adattato per diversi ambienti senza particolari azioni specifiche;
- **Installabilità**: è la capacità del software di essere installato in un certo ambiente;
- **Coesistenza**: è la capacità del software di coesistere con altri prodotti indipendenti in ambienti comuni che condividono le medesime risorse;
- **Sostituibilità**: è la capacità del software di sostituire un altro prodotto destinato allo stesso scopo nel medesimo ambiente;
- **Aderenza alla portabilità**: è la capacità del software di aderire agli *standard*$_G$ e convenzioni relative alla portabilità.

B Metriche per la qualità

B.1 Metriche interne

Le metriche interne permettono ad utenti, valutatori e sviluppatori l'individuazione di eventuali problemi che potrebbero influire sulla qualità finale del prodotto, si applicano quindi unicamente al software non eseguibile durante le fasi di progettazione e codifica. È di fondamentale importanza una corretta selezione degli attributi da misurare, in quanto permettono di stimare il livello di qualità esterna ed in uso del software ultimato tramite l'analisi statica dei prodotti intermedi.
Le misurazioni effettuate si basano sui numeri o le frequenze degli elementi che compongono il software e che vengono rilevati.

B.2 Metriche esterne

Le metriche esterne forniscono ad utenti, valutatori e sviluppatori degli elementi per misurare i comportamenti del software rilevati dai $test_G$, dall'operatività e dalla sua esecuzione. Vengono selezionate in base alle caratteristiche che il prodotto finale deve presentare una volta eseguito.

B.3 Metriche della qualità in uso

Le metriche della qualità in uso misurano il grado con cui il prodotto software permette agli utenti di svolgere le proprie attività con efficacia, produttività, sicurezza e soddisfazione nel contesto operativo previsto. Rappresenta quindi la vista esterna che l'utente ha nei confronti del prodotto e viene misurata in base ai risultati ottenuti dal suo utilizzo. Tuttavia è anche l'effetto combinato di alcune caratteristiche interne ed esterne di qualità, la cui relazione dipende dal tipo di utilizzatore:

- **Utente finale**: è influenzato dalle caratteristiche di funzionalità, affidabilità, usabilità ed efficienza;
- **Addetto alla manutenzione**: verifica le caratteristiche di manutenibilità;
- **Responsabile del porting**: verifica le caratteristiche di portabilità.

B.4 Metriche per la qualità di processo

B.4.1 Miglioramento

- **MPMS** - Percentuale di Metriche Soddisfatte: percentuale di metriche che rientrano nell'intervallo di accettabilità

$$MPMS = \frac{metriche\ soddisfatte}{metriche\ totali} \cdot 100;$$

- **MNRNP** - Numero di Rischi Non Previsti: numero rappresentante i rischi non previsti durante la fase di analisi che si sono verificati;
- **MPRGM** - Percentuale di Rischi Gestiti Male: percentuale rappresentante la quantità di rischi verificatisi che sono stati gestiti in malo modo

$$MPRGM = \frac{rischi\ gestiti\ male}{rischi\ totali} \cdot 100.$$

B.4.2 Fornitura

- **MVP** - Valore Pianificato: è il lavoro programmato per essere completato in un determinato tempo

$$MVP = \%lavoro\ pianificato \cdot BAC;$$

- **MG** - Guadagno: valore del lavoro fatto al momento del calcolo

$$MG = \%lavoro\ svolto \cdot EAC;$$

- **MVdP** - Variazione di Piano: numero di giorni di differenza rispetto alla pianificazione

$$MVdP = (FP - IP) - (FC - IC)$$

  Dove:

  - FP : giorno pianificato di fine attività;
  - IP : giorno pianificato di inizio attività;
  - FC : giorno consuntivato di fine attività;
  - IC : giorno consuntivato di inizio attività.

  Il risultato se:

  - > 0: indica un anticipo rispetto alla previsione;
  - 0: indica se si è in linea rispetto alla previsione;
  - < 0: indica se si è in ritardo rispetto alla previsione;

- **MCE** - Costo effettivo: costi sostenuti per il compimento del progetto;

- **MVC** - Variazione di Costo: numero rappresentante lo stato dei costi rispetto al preventivo

$$MVC = CAS - CAP$$

  dove:

  - CAS: costo attività svolte;
  - CAP: costo attività pianificate.

- **MVR** - Variazione dei *Requisiti*$_G$: numero di requisiti in cui c'è stato un cambiamento.

Parametri:

- **BAC** - Budget at Completion: indica il budget complessivo del progetto preventivato;
- **EAC** - Estimated at Completion: indica alla data della misurazione qual è la stima del costo finale che si sta prefigurando al completamento

$$EAC = BAC \cdot \frac{guadagno}{costo\ effettivo}.$$

B.4.3 Codifica

- **MCCM** - Complessità Ciclomatica per Metodo: misura il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso

$$MCMM = e - n + 2$$

  Dove:

  - e: numero di archi del grafo del flusso di esecuzione del metodo;
  - n: numero di vertici del grafo del flusso di esecuzione del metodo.

- **MCC** - *Code Coverage*$_G$: numero di linee di codice convalidate con successo nell'ambito di una procedura di *test*$_G$

$$MCC = \frac{linee\ di\ codice\ percorse}{linee\ di\ codice\ totali} \cdot 100;$$

- **MSC** - Statement Coverage: viene utilizzata per calcolare e misurare il numero di istruzioni che sono state eseguite almeno una volta

$$MSC = \frac{istruzioni\ eseguite}{istruzioni\ totali} \cdot 100;$$

- **MBC** - *Branch*$_G$ Coverage: indice di quante diramazioni del codice vengono eseguite dai *test*$_G$. Un branch è uno dei possibili percorsi di esecuzione che il codice può seguire dopo che un'istruzione decisionale viene valutata

$$MBC = \frac{flussi\ funzionali\ implementati\ e\ testati}{flussi\ condizionali\ riusciti\ e\ non} \cdot 100.$$

B.4.4 Documentazione

- **MCO** - Correttezza Ortografica: rappresenta il numero di errori grammaticali ed ortografici all'interno di un documento;
- **MIG** - Indice Gulpease: indice di leggibilità di un testo tarato sulla lingua italiana, presenta il vantaggio di utilizzare la lunghezza delle parole in lettere anziché in sillabe, semplificandone quindi il calcolo automatico.
  La formula per il calcolo è la seguente:

  $$IG = 89 + \frac{300 \cdot N_f - 10 \cdot N_l}{N_p}$$

  Dove:

  - N_f : numero di frasi;
  - N_l : numero di lettere;
  - N_p : numero di parole.

  I risultati sono compresi tra 0 e 100, dove il valore "100" indica la leggibilità più alta e "0" la leggibilità più bassa. In generale risulta che i testi con un indice:

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

B.5 Metriche per la qualità di prodotto

B.5.1 Funzionalità

- **MROS** - $Requisiti_G$ Obbligatori Soddisfatti: espressa in percentuale

  $$MROS = \frac{requisiti\ obbligatori\ soddisfatti}{requisiti\ obbligatori\ totali} \cdot 100;$$

- **MRDS** - Requisiti Desiderabili Soddisfatti: espressa in percentuale

  $$MRDS = \frac{requisiti\ desiderabili\ soddisfatti}{requisiti\ desiderabili\ totali} \cdot 100;$$

- **MROPZS** - Requisiti OPZionali Soddisfatti: espressa in percentuale

  $$MROPZS = \frac{requisiti\ opzionali\ soddisfatti}{requisiti\ opzionali\ totali} \cdot 100.$$

B.5.2 Progettazione

- **MPRCI** - Percentuale di $Requisiti_G$ di Configurazione Implementati: percentuale rappresentante la quantit'a dei requisiti di configurazione implementati

  $$MPRCI = \frac{requisiti\ di\ configurazione\ implementati}{requisiti\ di\ configurazione\ totali} \cdot 100.$$

B.5.3 Usabilità

- **MFU** - Facilità di Utilizzo: quantità di tempo espressa in minuti necessaria all'utente per assimilare a pieno il funzionamento del prodotto.

B.5.4 Manutenibilità

- **MAC** - Accoppiamento tra Classi: numero intero positivo associato ad una classe che rappresenta da quante altre classi dipende;
- **MATC** - ATtributi per Classe: numero intero positivo che rappresenta il numero di attributi appartenenti ad una classe;
- **MPM** - Parametri per Metodo: numero intero positivo che rappresenta il numero di parametri contenuti in un singolo metodo escludendo il parametro implicito `this`;
- **MLCM** - Linee di Codice per Metodo: numero intero positivo che rappresenta il numero di linee di codice che costituiscono un singolo metodo. Non vengono conteggiate le linee vuote o che contengono unicamente parentesi.

B.5.5 Affidabilità

- **MDE** - Densità degli Errori: percentuale che rappresenta la resistenza a malfunzionamenti del software

$$MDE = \frac{test\ con\ errori}{test\ eseguiti} \cdot 100.$$