



NORME DI PROGETTO

Gruppo Cyber13 - Progetto P2PCS

Informazioni sul documento

Versione	3.0.0
Data Redazione	07/06/2019
Responsabile Redazione	Ilaria Rizzo Elena Pontecchiani Andrea Casagrande
Verifica	Matteo Squeri Fabio Garavello
Approvazione	Ilaria Rizzo
Uso	Interno
Destinatari	Cyber13 Prof. Tullio Vardanega Prof. Riccardo Cardin
Mail di contatto	swe.cyber13@gmail.com

Diario delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
2.0.0	07/06/2019	Approvazione del documento per il rilascio in RQ	Ilaria Rizzo	Responsabile
2.1.2	29/05/2019	Verifica con esito positivo	Fabio Garavello	Verificatore
2.1.1	28/05/2019	Correzione §2.1, §3.3.3.1	Matteo Squeri	Verificatore
2.1.0	28/05/2019	Verifica con esito negativo	Matteo Squeri	Verificatore
2.0.2	26/05/2019	Ampliamento: §2.1, §3.1.6, §3.3.3.1.2, §3.3.3.1.3, §3.4.4.2, §3.5.3, §4.1.3.1, §4.1.3.2, §4.4.2 Spostamento di §3.6 in §4.3. Correzione: §2.1, §3.4.1.	Elena Pontecchiani	Amministratore

2.0.1	22/05/2019	Riorganizzazione metriche per la qualità dei processi	Andrea Casagrande	Redattore
2.0.0	07/05/2019	Approvazione del documento per il rilascio in RP	Daniel Mirel Bira	Responsabile
1.1.0	06/05/2019	Verifica generale con esito positivo per il rilascio del documento in RP	Elena Pontecchiani	Verificatore
1.0.2	05/05/2019	Estensione §2.2	Ilaria Rizzo	Amministratore
1.0.1	04/05/2019	Correzione e estensione §1.4.2; correzione formale in §2 e §4; spostato §4.5 in processi di supporto	Ilaria Rizzo	Amministratore
1.0.0	04/04/2019	Approvazione del documento per il rilascio in RR	Andrea Casagrande	Responsabile

0.3.0	04/04/2019	Verifica generale con esito positivo per il rilascio del documento in RR	Daniel Mirel Bira	Verificatore
0.2.1	09/03/2019	Approvazione delle norme per l'inizio della stesura degli altri documenti	Matteo Squeri	Responsabile
0.2.0	09/03/2019	Verifica con esito positivo per il rilascio del documento in RR	Fabio Garavello	Verificatore
0.1.2	08/03/2019	Contenuti rielaborati seguendo le direttive riguardanti la documentazione esposti nelle norme	Andrea Casagrande	Analista
0.1.1	07/03/2019	Correzione errori di ortografia e di forma	Ilaria Rizzo	Analista

0.1.0	07/03/2019	Verifica con esito positivo per quanto riguarda i contenuti del documento	Fabio Garavello, Matteo Squeri	Verificatori
0.05	06/03/2019	Stesura §4	Ilaria Rizzo, Andrea Casagrande,	Amministratori
0.04	05/03/2019	Stesura § 3	Elena Pontecchiani, Ilaria Rizzo, Andrea Casagrande,	Amministratori
0.03	04/03/2019	Stesura § 2	Elena Pontecchiani, Ilaria Rizzo, Andrea Casagrande,	Amministratori
0.02	04/03/2019	Stesura §1 e parziale stesura §2	Elena Pontecchiani	Amministratore
0.01	04/03/2019	Creazione scheletro documento	Elena Pontecchiani	Amministratore

Indice

1	Introduzione	12
1.1	Scopo del documento	12
1.2	Scopo del prodotto	12
1.3	Glossario	12
1.4	Riferimenti	13
1.4.1	Riferimenti normativi	13
1.4.2	Riferimenti informativi	13
2	Processi primari	14
2.1	Processo di fornitura	14
2.1.1	Scopo	14
2.1.2	Aspettative	14
2.1.3	Attività	14
2.1.3.1	Studio di fattibilità	14
2.1.3.2	Approntamento di risposte e richieste	15
2.1.3.3	Contrattazione	15
2.1.3.4	Pianificazione	16
2.1.3.5	Garanzia della qualità	16
2.1.3.6	Esecuzione e controllo	17
2.1.3.7	Revisione e valutazione	17
2.1.3.8	Consegna e completamento	17
2.2	Processo di sviluppo	18
2.2.1	Scopo	18
2.2.2	Aspettative	18
2.2.3	Attività	18
2.2.3.1	Analisi dei requisiti	18
2.2.3.1.1	Scopo	18
2.2.3.1.2	Aspettative	19
2.2.3.1.3	Strategia di individuazione dei requisiti	19
2.2.3.1.4	Classificazione dei requisiti	19
2.2.3.1.5	Esposizione dei requisiti	20
2.2.3.1.6	Classificazione dei casi d'uso	20
2.2.3.1.7	Tracciamento componenti-requisiti	21
2.2.3.2	Progettazione	21
2.2.3.2.1	Scopo	21
2.2.3.2.2	Aspettative	22
2.2.3.2.3	Caratteristiche dell'architettura logica	22
2.2.3.2.4	Norme per i diagrammi delle classi	23
2.2.3.2.5	Norme per i diagrammi dei package	26
2.2.3.2.6	Norme per i diagrammi di sequenza	27
2.2.3.2.7	Norme per i diagrammi di attività	29
2.2.3.2.8	Attività	33
2.2.3.3	Codifica	34
2.2.3.3.1	Scopo	34
2.2.3.3.2	Aspettative	35

2.2.3.3.3	Stile di codifica	35
2.2.3.3.4	Convenzioni per la documentazione	41
2.2.3.4	Metriche	41
2.2.3.4.1	M[PROC][0001]: Numero campi dati per classe	41
2.2.3.4.2	M[PROC][0002] - Parametri per metodo	41
2.2.3.4.3	M[PROC][0003] - Metodi per classe	42
2.2.3.4.4	M[PROC][0004]: Grado di instabilità	42
2.2.3.4.5	M[PROC][0005] - Complessità Ciclomatica	42
2.2.3.4.6	M[PROC][0006] - Linee di commento per linee di codice	42
2.2.3.4.7	M[PROC][0007] - Indice di manutenibilità	42
2.2.3.5	Strumenti utilizzati	42
3	Processi di supporto	44
3.1	Processo di documentazione	44
3.1.1	Scopo	44
3.1.2	Aspettative	44
3.1.3	Ciclo di vita della documentazione	44
3.1.4	Separazione tra documenti interni e esterni	46
3.1.5	Nomenclatura dei documenti	46
3.1.5.1	Versioni di un documento	46
3.1.5.2	Formato dei file	46
3.1.6	Documenti correnti	47
3.1.7	Formattazione dei documenti	47
3.1.7.1	Strutturazione dei file	47
3.1.7.2	Norme tipografiche	49
3.1.8	Strumenti utilizzati	50
3.2	Processo di garanzia della qualità	50
3.2.1	Scopo	50
3.2.2	Aspettative	50
3.2.3	Classificazione dei processi	50
3.2.4	Classificazione delle metriche	51
3.2.5	Procedure	51
3.2.6	Metriche per la qualità di processo	52
3.2.6.0.1	M[PROC][0010] - Halstead Difficulty per-function	52
3.2.6.0.2	M[PROC][0011] - Halstead Volume per-function	52
3.2.6.0.3	M[PROC][0012] - Halstead Effort per-function	52
3.2.7	Metriche per la qualità di prodotto	52
3.2.7.1	Metriche per i documenti	52
3.2.7.1.1	M[PROD][D][0001]: Indice di <i>Gulpease</i> _g	52
3.2.7.1.2	M[PROD][D][0002]: Errori ortografici	53
3.2.7.2	Metriche per il Software	53
3.2.7.2.1	M[PROD][S][0001]: Copertura requisiti obbligatori	53

3.2.7.2.2	M[PROD][S][0002]: Copertura requisiti accettati	53
3.2.7.2.3	M[PROD][S][0003]: Accuratezza rispetto alle attese	54
3.2.7.2.4	M[PROD][S][0004]: Densità di <i>failure</i> _g . . .	54
3.2.7.2.5	M[PROD][S][0005]: Blocco di operazioni non corrette	54
3.2.7.2.6	M[PROD][S][0006]: Tempo di risposta . . .	54
3.2.7.2.7	M[PROD][S][0007]: Comprensibilità delle funzioni offerte	55
3.2.7.2.8	M[PROD][S][0008]: Facilità di apprendimento delle funzionalità	55
3.2.7.2.9	M[PROD][S][0009]: Utilizzo effettivo delle funzionalità	55
3.2.7.2.10	M[PROD][S][0010]: Capacità di analisi di failure	55
3.2.7.2.11	M[PROD][S][0011]: Impatto delle modifiche	55
3.2.7.2.12	M[PROD][S][0012]: Rapporto linee di commento su linee di codice	56
3.2.7.2.13	M[PROD][S][0013]: Versioni di Android supportate	56
3.2.7.2.14	M[PROD][S][0014]: Copertura del framework <i>Octalysis</i> _g	56
3.3	Processo di configurazione	56
3.3.1	Scopo	56
3.3.2	Aspettative	56
3.3.3	Attività	56
3.3.3.1	Versionamento	56
3.3.3.1.1	Descrizione	56
3.3.3.1.2	Struttura delle repository	57
3.3.3.1.3	Branch	58
3.3.3.1.4	Aggiornamento della repository	58
3.3.3.2	Strumenti	59
3.4	Processo di verifica	59
3.4.1	Scopo	59
3.4.2	Aspettative	59
3.4.3	Descrizione	59
3.4.4	Attività	60
3.4.4.1	Analisi statica	60
3.4.4.2	Analisi dinamica	60
3.4.4.2.1	Test	60
3.4.5	Metriche	61
3.4.5.1	M[PROC][0008] - Percentuale di codice coperto da test	61
3.4.5.2	M[PROC][0009] - Percentuale di test passati	61
3.4.5.3	M[PROC][0010] - Percentuale di test non passati . . .	61

3.4.6	Strumenti	61
3.5	Processo di validazione	62
3.5.1	Scopo	62
3.5.2	Aspettative	62
3.5.3	Attività	62
4	Processi organizzativi	63
4.1	Processo di coordinamento	63
4.1.1	Scopo	63
4.1.2	Aspettative	63
4.1.3	Comunicazione	63
4.1.3.1	Comunicazioni interne	63
4.1.3.2	Comunicazioni esterne	63
4.1.4	Riunioni	64
4.1.4.1	Riunioni interne	64
4.1.4.2	Riunioni esterne	65
4.1.4.3	Verbale di riunione	65
4.2	Processo di pianificazione	66
4.2.1	Scopo	66
4.2.2	Aspettative	66
4.2.3	Ruoli di progetto	66
4.2.3.1	Responsabile di progetto	66
4.2.3.2	Amministratore	67
4.2.3.3	Analista	67
4.2.3.4	Progettista	67
4.2.3.5	Programmatore	67
4.2.3.6	Verificatore	67
4.2.3.7	Rotazione dei ruoli	68
4.2.4	Ticketing	68
4.2.4.1	Task list	68
4.2.4.2	Task	68
4.2.4.3	Ticket	68
4.2.5	Metriche	69
4.2.5.1	M[PROC][0011] - Schedule Variance	69
4.2.5.2	M[PROC][0012] - Budget Variance	69
4.2.5.3	M[PROC][0013] - Rischi non individuati	69
4.3	Processo di formazione	70
4.4	Procedure	70
4.4.1	Creazione e gestione dei task	70
4.4.2	Gestione dei ticket	70
4.4.3	Stesura del consuntivo	72
4.5	Strumenti	72
4.5.1	Pianificazione	72
4.5.2	Comunicazione	72
4.5.3	Creazione diagrammi di Gantt	73
4.5.4	Calcolo del consuntivo	73

A	Ciclo di Deming	74
B	SOLID Principles	76
C	Stati di progresso per SEMAT	78

Elenco delle figure

1	Notazione per la descrizione delle classi	23
2	Freccia per la relazione di dipendenza	25
3	Relazione di dipendenza tra le classi A e B	25
4	Freccia per la relazione di associazione	25
5	Relazione di associazione tra le classi A e B	25
6	Freccia per la relazione di aggregazione	25
7	Relazione di aggregazione tra le classi A e B	25
8	Freccia per la relazione di composizione	26
9	Relazione di composizione tra le classi A e B	26
10	Freccia per la relazione di ereditarietà/generalizzazione	26
11	Relazione di ereditarietà/generalizzazione tra le classi A e B	26
12	Relazione di dipendenza tra i package	27
13	Formalismo per gli oggetti coinvolti dei diagrammi di sequenza	27
14	Formalismo per per rappresentare la barra di attivazione di un oggetto dei diagrammi di sequenza	28
15	Messaggio sincrono dei diagrammi di sequenza	28
16	Messaggio asincrono dei diagrammi di sequenza	28
17	Ritorno di un metodo chiamato dei diagrammi di sequenza	28
18	Formalismo per la creazione di oggetti nei diagrammi di sequenza	28
19	Formalismo per la distruzione di oggetti nei diagrammi di sequenza	29
20	Nodo iniziale di un diagramma di attività	30
21	Activity di un diagramma di attività	30
22	Costrutto fork di un diagramma di attività	31
23	Costrutto join di un diagramma di attività	31
24	Costrutto branch di un diagramma di attività	31
25	Nodo finale di un diagramma di attività	31
26	Costrutto puntamento di un diagramma di attività	31
27	Costrutto pin di un diagramma di attività	32
28	Costrutto partizione di un diagramma di attività	32
29	Costrutto di attesa di un segnale di un diagramma di attività	33
30	Costrutto di generazione di un segnale di un diagramma di attività	33
31	Costrutto timeout di un diagramma di attività	33
32	Indentazione1 corretta	35
33	Indentazione1 scorretta	35
34	Indentazione2 corretta	36
35	Indentazione2 scorretta	36
36	Indentazione4 corretta	36
37	Indentazione4 scorretta	37
38	Parentesizzazione 1 corretta	37
39	Parentesizzazione 1 scorretta	37
40	Eccezione per parentesizzazione 1 corretta	37
41	Parentesizzazione 2 corretta	38
42	Parentesizzazione 2 scorretta	38
43	Sintassi commenti TODO	39

44	Sintassi @SuppressWarnings	39
45	Inizializzazione corretta	40
46	Inizializzazione scorretta	40
47	Inizializzazione corretta	40
48	Diagramma del ciclo di vita della documentazione	45
49	Diagramma del ciclo di vita di un ticket	71
50	Ciclo PCDA	74

1 Introduzione

1.1 Scopo del documento

Il documento ha lo scopo di fissare precise norme che regoleranno l'intero svolgimento del progetto.

Tutti i membri del team sono obbligati a visionare tale documento e a sottostare alle norme ivi descritte. Le norme contenute nel documento si occupano di regolamentare i diversi processi all'interno del progetto. La regolamentazione tramite norme permette di sviluppare prodotti coerenti e consistenti tra i vari componenti del gruppo e facilitare l'esecuzione delle operazioni di verifica. In particolare le norme si occuperanno di regolamentare:

- Le modalità di interazione tra i membri del team e le persone esterne al team;
- L'organizzazione del team: verranno definiti e assegnati ruoli ai membri del team. Per ogni ruolo verranno identificate determinate mansioni;
- Le convenzioni tipografiche e le modalità di stesura delle documentazioni;
- Gli ambienti di sviluppo, il *repository*_[g] e il *ticketing*_[g];

Nel caso in cui ve ne sia necessità, ogni membro del team potrà contattare il *Project Manager*_[g] per suggerire eventuali cambiamenti

1.2 Scopo del prodotto

Lo scopo del prodotto è quella di realizzare un'applicazione *Android*_[g] che implementi un servizio di car sharing *peer-to-peer*_[g].

1.3 Glossario

Onde evitare ambiguità o incomprensioni di natura lessicale, si allega il *Glossario v1.0.0*. All'interno del documento saranno presenti parole di ambito specifico, uso raro che potrebbero creare incomprensioni. Per una maggiore leggibilità tali parole sono riconoscibili all'interno dei vari documenti in quanto scritte in corsivo e con un 'g' a pedice tra barre orizzontali (per esempio *Glossario*_[g]). Nel caso esistano più ripetizioni di una stessa parola del glossario all'interno di uno stesso paragrafo, la dicitura con la lettera 'g' a pedice sarà inserita solo la prima volta che la parola comparirà. Il comando LaTeX da utilizzare per contrassegnare un termine da glossario all'interno dei documenti è `\cigl{..}`.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- *Norme di Progetto v2.0.0*;

1.4.2 Riferimenti informativi

- Sito del corso di Ingegneria del Software:
 - Per quanto riguarda le nozioni di ruoli di progetto:
<https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/L06.pdf>
 - Per quanto riguarda gli strumenti di condivisione:
<https://www.math.unipd.it/~tullio/IS-1//2010/Approfondimenti/A12.pdf>
- Software Engineering - Ian Sommerville - 10th Edition (in particolare §2, §4, §5);
- Materiale suggerito dalla proponente:
 - Materiale $MVP_{|g|}$:
<https://model-view-presenter-android-guidelines>
 - Materiale Testing e $Refactoring_{|g|}$
https://www.youtube.com/watch?v=_NnElP05BU0

2 Processi primari

2.1 Processo di fornitura

2.1.1 Scopo

In questa sezione vengono sancite precise norme per quanto riguarda il rapporto di fornitura con la proponente *GaiaGo_[g]* e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.

Le norme ivi trattate sono tassative e i membri del gruppo Cyber13 sono tenuti a rispettarle al fine di proporsi e diventare fornitori.

Nello specifico le attività che la sezione intende normare, in conformità allo standard ISO/IEC 12207:1995, sono lo studio di fattibilità e la pianificazione, descritti nel dettaglio in §2.1.3.

2.1.2 Aspettative

Il gruppo intende mantenere un costante dialogo con il proponente per avere un riscontro efficace sul lavoro svolto.

Nel corso dell'intero progetto il team Cyber13 si auspica di instaurare con la committente GaiaGo, nel particolare con il referente Filippo Pretto, un rapporto di collaborazione al fine di:

- Determinare come adempire appieno ai requisiti fissati dal proponente;
- Determinare vincoli sui processi e i requisiti;
- Eseguire una stima dei costi;
- Concordare la qualifica del prodotto.

2.1.3 Attività

2.1.3.1 Studio di fattibilità Lo studio di fattibilità è un'attività di analisi che ha lo scopo di analizzare i diversi progetti proposti dalle aziende al fine di scegliere quello più conforme agli interessi e capacità del gruppo.

Il team si auspica di proporsi e diventare fornitori nei confronti della Proponente Red Babel e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin nell'ambito della progettazione, sviluppo e consegna del prodotto P2PCS.

Lo studio di fattibilità si compone delle seguenti attività, eseguite sequenzialmente in quest'ordine:

- Studio dei capitolati proposti dove per ognuno sono stati analizzati:
 - Il problema da risolvere;
 - Il prodotto da realizzare;

- Le tecnologie da impiegare;
- Analisi dei rischi associati al progetto;
- Implicazioni di costo e qualità;
- Scelta del capitolato per il quale proporsi come fornitori;
- Redazione del documento *Studio di Fattibilità v1.0.0*.

Tutte le attività svolte all'interno del processo di studio di fattibilità sono state riassunte, tracciate e rendicontate (con relative decisioni annesse) nel documento *Studio di Fattibilità v1.0.0*. Tale documento include le motivazioni che hanno portato il gruppo Cyber13 a proporsi come fornitore per il prodotto indicato e riporta per ciascun capitolato:

- **Informazioni sul capitolato:** In cui vengono riportati nome del progetto, azienda proponente e i committenti.
- **Descrizione:** All'interno della quale viene descritto brevemente cosa il progetto richiede.
- **Studio del dominio:** Nel quale vengono indicati i domini applicativi e tecnologici del capitolato.
- **Esito finale:** Dove sono riportati aspetti positivi, fattori di rischio e conclusione finale del gruppo in merito al capitolato.

2.1.3.2 Approntamento di risposte e richieste

L'attività ha lo scopo di contattare l'azienda al fine di presentarsi e proporsi come fornitori per il progetto P2PCS.

Il team si auspica di presentare in modo chiaro e tracciabile la proposta di esecuzione del progetto.

Per adempiere allo scopo preposto, il team durante questa fase svolge le seguenti attività:

- Redarre un discorso, che verrà poi sottoposto all'azienda proponente, nel quale il team si presenta come fornitore;
- Contattare l'azienda proponente al fine di fissare un colloquio remoto (via videochiamata, con strumenti individuati nella sezione 4.1.4.2);
- Eseguire un colloquio remoto conoscitivo con il referente Filippo De Pretto per l'azienda proponente GaiaGo. Durante il colloquio i membri del team si sono presentati e hanno esposto la proposta preventivamente elaborata.

2.1.3.3 Contrattazione

L'attività ha lo scopo di negoziare e stipulare un contratto con il committente per fornire un prodotto o servizio software relativo al progetto P2PCS.

Il team si auspica di negoziare in modo chiaro e tracciabile un contratto per l'esecuzione del progetto.

Il team, ogni avanzamento di revisione, redige dei documenti formali chiamate "Lettere di presentazione", dove all'interno vengono esposti il preventivo a finire e i prodotti consegnati per ogni fase.

2.1.3.4 Pianificazione

L'attività si prefigge di pianificare e sviluppare un piano di gestione del progetto in base ai requisiti.

Il team si attende di sviluppare una pianificazione che possa essere adeguata allo svolgimento del prodotto nel modo più conforme possibile ai vincoli contrattuali precedentemente individuati.

Il responsabile, aiutato dagli amministratori avrà il compito di pianificare il lavoro all'interno dei membri del team. L'attività culmina nella redazione del documento *Piano di Progetto*. Tale documento formale descrive:

- Gli obiettivi del progetto;
- Le attività da svolgere per adempire agli obiettivi rispettando i requisiti fissati. Attraverso l'attività di pianificazione si organizzano le diverse attività da svolgere con precise tempistiche da rispettare;
- Le risorse coinvolte (in termini di personale e di tecnologie impiegate);
- I costi previsti, che vengono stabiliti attraverso il preventivo e il consuntivo. Sulla base della pianificazione si stima la quantità di lavoro necessaria per ogni fase, proponendo così un preventivo per il costo totale del progetto. Alla fine di ogni attività si redige inoltre un consuntivo di periodo per tracciare l'andamento rispetto a quanto preventivato;
- I rischi potenziali attraverso l'analisi dei rischi. Si analizzano nel dettaglio i rischi che potrebbero insorgere nel corso del progetto e i modi per affrontarli, capendo la probabilità che essi accadano e il livello di gravità ad essi associato.

2.1.3.5 Garanzia della qualità L'attività ha lo scopo di individuare le strategie da adottare per garantire la qualità del materiale consegnato e dei processi attuati.

Il team si attende di sviluppare un prodotto di qualità che possa rispondere al meglio alle richieste degli *stakeholder*_[g].

I verificatori dovranno scegliere una strategia da adottare per la *verifica*_[g] e la *validazione*_[g] del materiale prodotto dal gruppo. Le decisioni che vengono prese a tal proposito sono tracciate nel documento formale "Piano di Qualifica", che dovrà contenere:

- **Visione generale delle strategie di verifica:** Si stabiliscono le procedure di controllo sulla qualità di processo e di prodotto, tenendo in considerazione le risorse a disposizione.
- **Misure e metriche:** Si devono stabilire delle metriche oggettive per i documenti, i processi e il software.
- **Gestione della revisione:** Si devono stabilire le modalità di comunicazione delle anomalie e le procedure di controllo per la qualità di processo.
- **Pianificazione del collaudo:** Si devono definire nel dettaglio le metodologie di collaudo del prodotto realizzato.
- **Resoconto delle attività di verifica:** Alla fine di ogni attività si devono riportare le metriche calcolate e un resoconto sulla verifica di tale attività.

2.1.3.6 Esecuzione e controllo

Il fornitore deve attuare ed eseguire i piani sviluppati nella fase di pianificazione e sviluppare il prodotto software monitorando il progresso e la qualità.

Il team si attende di sviluppare un prodotto conforme alle richieste degli *stakeholder*_[g], lavorando in maniera conforme con i piani sviluppati nella fase di pianificazione. Le attività che comprendono questa fase sono:

- Sviluppo del prodotto;
- Redazione di documenti, che tengano traccia dello svolgimento del lavoro del team.

2.1.3.7 Revisione e valutazione

Il fornitore deve eseguire la verifica e la validazione al fine di dimostrare che il prodotto software consegnato risponde ai requisiti richiesti dagli stakeholder.

2.1.3.8 Consegna e completamento

Il fornitore consegna il prodotto software nelle tempistiche e modalità sancite dal contratto con la proponente.

2.2 Processo di sviluppo

2.2.1 Scopo

La presente sezione del documento contiene tutte le attività di analisi, design, codifica, integrazione ed installazione relative al prodotto da sviluppare. Di seguito sono raccolte le linee guida che verranno utilizzate dai membri del gruppo nelle principali attività di questo processo.

Pertanto il processo si compone delle seguenti attività:

- Analisi dei requisiti;
- Progettazione;
- Codifica.

2.2.2 Aspettative

Gli obiettivi perseguiti in fase di sviluppo sono i seguenti:

- Realizzare un prodotto finale conforme alle richieste del proponente;
- Realizzare un prodotto finale soddisfacente i test di verifica e di validazione.

2.2.3 Attività

2.2.3.1 Analisi dei requisiti

2.2.3.1.1 Scopo

L'analisi dei requisiti è un documento ad uso esterno, che assolve i seguenti compiti:

- Descrivere lo scopo del lavoro;
- Fornire ai *Progettisti*_[g] riferimenti precisi e affidabili;
- Fissare i requisiti e le funzionalità del prodotto concordate con il cliente;
- Fornire una descrizione completa di tutti i *requisiti*_[g] individuati in fase di analisi dagli analisti e dei casi d'uso riguardanti il progetto *P2PCS*_[g];
- Fornire una base per raffinamenti successivi al fine di garantire un miglioramento continuo del prodotto e del processo di sviluppo;
- Facilitare le revisioni del codice;
- Stimare i costi.

Le informazioni estrapolate sono state ottenute dall'esposizione del capitolato e soprattutto attraverso incontri con il referente dell'azienda proponente.

In seguito sono elencate le norme che riguardano i contenuti esposti nel documento

Analisi Dei Requisiti v1.0.0, al fine di evitare ambiguità o difformità di natura formale.

La tecnica utilizzata per l'analisi e la ricerca dei requisiti è quella dei casi d'uso.

2.2.3.1.2 Aspettative

L'obiettivo dell'attività è la creazione del documento formale *Analisi Dei Requisiti v1.0.0*.

2.2.3.1.3 Strategia di individuazione dei requisiti

I requisiti sono stati estrapolati dalle varie fonti attraverso una strategia *top-down*_[g], ovvero da un contesto generale sempre più specializzandosi verso il particolare. Contestualizzando alla tecnica dei casi d'uso, sono stati dapprima individuati i casi d'uso principali, e poi in seguito eventuali sottocasi, estensioni o inclusioni.

2.2.3.1.4 Classificazione dei requisiti

Ad ogni requisito individuato in fase di analisi viene identificato un codice univoco, espresso nel seguente formato:

$$R[\text{Priorità}][\text{Tipo}][\text{Codice}]$$

Dove:

- **R**: abbreviazione di requisito.
- **Priorità**: Indica l'importanza di un requisito, e può assumere i seguenti valori (mutuamente esclusivi ed espressi in ordine decrescente di importanza):
 - C: Dall'inglese 'compulsory', che significa obbligatorio, indica un requisito irrinunciabile per il committente.
 - D: Dall'inglese 'desirable', desiderabile. Requisito auspicabile ma non strettamente necessario.
- **Tipo**: Indica la classificazione del vincolo. Può assumere i seguenti valori:
 - F: Indica un requisito funzionale, ovvero la definizione di una funzione/caratteristica che deve essere implementata in un sistema. Questa tipologia di requisiti descrivono quindi come il software reagisce a situazioni ed input particolari;
 - Q: Indica un requisito di qualità. Includono i requisiti di *efficacia*_[g], *efficienza*_[g] e i requisiti per garantire la qualità del prodotto;
 - V: Indica requisiti di vincolo imposti dalla proponente GaiaGo.
- **Codice**: Numero intero univoco e incrementale di tre cifre associato al particolare requisito sulla base dell'ordine in cui compare nella struttura del documento.

2.2.3.1.5 Esposizione dei requisiti

Per facilitare la lettura dei requisiti individuati, vengono introdotte delle norme riguardanti la loro esposizione all'interno del documento *Analisi Dei Requisiti v1.0.0*.

- I requisiti individuati sono raggruppati in tabelle per appartenenza a una certa tipologia (funzionali, prestazionali, di qualità e di vincolo);
- Ogni riga di tabella contenente la descrizione dei requisiti si comporrà dei seguenti campi:
 - Codice identificativo del requisito (secondo la nomenclatura §2.2.3.1.3);
 - Breve descrizione testuale del requisito;
 - Fonti del requisito individuato (casi d'uso che seguono la nomenclatura in §2.2.3.1.5, Documenti o decisioni interne).

2.2.3.1.6 Classificazione dei casi d'uso

Gli Analisti hanno inoltre il compito di individuare i diversi *Casi d'uso*_{|g|}, elencandoli attraverso una strategia *Top down*_{|g|}, ovvero partendo dal generale e in seguito scendendo nel particolare.

I casi d'uso hanno lo scopo di descrivere scenari di interazione tra utenti e un sistema. Ciascun caso d'uso presente nel documento di Analisi dei Requisiti sarà corredato delle seguenti informazioni:

- **Codice identificativo e nome:** Ogni caso d'uso è identificato da un codice univoco strutturato nel seguente modo:

UC [Codice Padre].[Codice Figlio]-Nome

Dove:

- UC: Indica "User case", caso d'uso in inglese;
 - Codice Padre: Numero intero;
 - Codice Figlio: Uno o più numeri interi che specificano l'annidamento del caso;
 - Nome del caso d'uso.
- **Attori:** Indica gli attori principali (obbligatoriamente) e secondari (opzionalmente, se esistono) del caso d'uso.
 - **Scopo e descrizione:** Riporta una breve descrizione del caso d'uso.
 - **Scenario principale:** Descrizione di ciò che il caso d'uso vuole modellare, indicando ogni azione che ne fa parte.
 - **Precondizione:** Specifica le condizioni che sono identificate come vere prima del verificarsi degli eventi del caso d'uso.
 - **Postcondizione:** Specifica le condizioni che sono identificate come vere dopo il verificarsi degli eventi del caso d'uso.

- **Scenari alternativi (opzionale):** Descrizione di una possibilità alternativa allo scenario principale.
- **Inclusioni (opzionale):** Usate per non descrivere più volte lo stesso flusso di eventi, inserendo il comportamento comune in un caso d'uso a parte.
- **Estensioni (opzionale):** Descrivono i casi d'uso che non fanno parte del flusso principale degli eventi, allo stesso modo di quanto descritto in "Scenario principale".

Faremo inoltre uso dei *diagrammi dei casi d'uso*_[g], descritti nel linguaggio *UML 2.0*_[g] per i casi d'uso più complessi, per mettere in evidenza gli attori ed i servizi del sistema.

2.2.3.1.7 Tracciamento componenti-requisiti

Al fine di verificare la *necessità*_[g] e la *sufficienza*_[g] dei requisiti individuati, si provvede con il loro tracciamento. L'attività si concretizza in una tabella all'interno del documento Analisi dei Requisiti contenente, per ogni riga, i seguenti campi:

- Fonte;
- Requisiti che la fonte individua.

2.2.3.2 Progettazione

2.2.3.2.1 Scopo

L'attività precede la fase di realizzazione ed è svolta dai progettisti. Essi hanno il compito di sviluppare e documentare una visione architeturale ad alto livello del prodotto che abbia le caratteristiche indicate in §2.2.3.2.3 rimanendo nei costi fissati al fine di sviluppare un prodotto che entri in uno stato "Demonstrable" in riferimento al ciclo SEMAT dell'appendice C. Nel processo di descrizione dell'*architettura*_[g] saranno fissate le componenti che andranno a costituire il sistema, con relative dipendenze e interazioni tra le loro istanze. I progettisti definiranno inoltre le interfacce necessarie a consentire l'interazione tra componenti e i *design pattern*_[g] da utilizzare.

Durante questa attività è molto importante che i progettisti si confrontino con l'azienda proponente, allo scopo di ottenere feedback e consigli. L'attività di progettazione è documentata nella *Technology Baseline*_[g] e nella *Product Baseline*_[g] che saranno consegnati rispettivamente alla Revisione di Progettazione e alla Revisione di Qualifica.

L'attività di progettazione deve rispettare i requisiti ed i vincoli stabiliti tra il gruppo e il proponente. In questa fase i progettisti pongono i seguenti obiettivi:

- Garantire la correttezza del prodotto sviluppato, perseguendo la correttezza per costruzione;

- Organizzare e ripartire compiti implementativi, riducendo la complessità del problema originale fino alle singole componenti facilitandone la codifica da parte dei singoli programmatori;
- Rendere chiara e comprensibile ogni parte dell'architettura ai differenti *stakeholders*_[g];
- Mantenere nascosti i dettagli implementativi, seguendo il principio dell' *information hiding*_[g];
- Ottimizzare l'uso delle risorse disponibili.

2.2.3.2.2 Aspettative

I Progettisti in fase di progettazione hanno il compito di descrivere una soluzione al problema che sia soddisfacente per tutti gli *stakeholders*_[g]. La progettazione viene fatta in due momenti: una prima parte ad alto livello viene fatta durante il periodo di Progettazione della base tecnologica, dove verranno studiati i design pattern che potrebbero essere utilizzati durante il periodo seguente di Progettazione di dettaglio e codifica, durante il quale la progettazione diventa atomica per ogni componente del sistema, in modo che i Programmatori possano sviluppare il codice eseguendo task focalizzate e singolari.

L'intero processo ha come risultato la realizzazione dell'architettura del sistema, così da fornire le istruzioni necessarie ai programmatori per sviluppare il prodotto finale richiesto.

2.2.3.2.3 Caratteristiche dell'architettura logica

L'architettura definita dovrà avere le seguenti qualità:

- Sufficienza: deve soddisfare i requisiti individuati nel documento *Analisi Dei Requisiti v1.0.0*;
- Modularità: deve essere suddivisa in parti chiare e ben distinte, così che possa essere facilmente manuttenibile;
- Robustezza: deve essere capace di sopportare ingressi diversi, sia da parte di utenti che dall'ambiente;
- Flessibilità: deve poter permettere modifiche al variare o all'aggiunta di requisiti senza perdite di performance o senza doverla restaurare profondamente;
- Efficienza: deve essere pensata in modo da poter ridurre gli sprechi di tempo e spazio;
- Affidabilità: deve avere una elevata capacità di rispettare le specifiche nel tempo;
- Disponibilità: la manutenzione delle sue parti non dovrà disabilitare il funzionamento di tutto il sistema;
- Sicurezza: rispetto ad intrusioni e malfunzionamenti;

- Semplicità: ogni parte contiene solo il necessario e nulla di superfluo;
- Incapsulazione: le componenti dovranno essere progettate in modo che le informazioni interne siano nascoste;
- Coesione: in modo che le parti che hanno gli stessi obiettivi stanno insieme;
- Basso accoppiamento: parti distinte devono essere poco dipendenti l'una dalle altre, in modo da poter essere facilmente manutenibili.

2.2.3.2.4 Norme per i diagrammi delle classi

Al fine di rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, durante la fase di progettazione si farà uso dei diagrammi delle classi, dedicati alla descrizione degli oggetti che fanno parte di un sistema e delle loro dipendenze.

Lo scopo dei diagrammi delle classi può pertanto essere sintetizzato in:

- Descrizione delle classi, dei tipi e delle loro relative relazioni;
- Analisi e progettazione della visione statica di un'applicazione;
- Descrizione delle responsabilità di un sistema;
- Base per i diagrammi dei componenti e di rilascio;

Per creare i diagrammi delle classi, deve essere creato un file per ogni classe o per ogni raggruppamento di classi. Il suddetto file viene prodotto attraverso lo strumento integrato nella suite di Google, detto *draw.io*_[g]. Il file viene quindi salvato nel Drive condiviso dai membri del team, in modo che possa essere accessibile in maniera semplice da ogni membro del team.

Per i diagrammi delle classi il team segue una notazione comune sia alle classi appartenenti al backend dell'applicazione (scritte in Kotlin) sia alle classi appartenenti alla parte server-side in PHP ospitata sul server web. La notazione per indicare una classe è la seguente: Ogni classe è identificata da un nome univoco e, se esistono, da

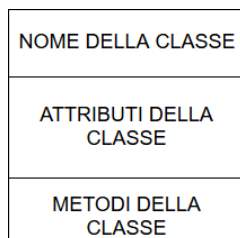


Figura 1: Notazione per la descrizione delle classi

attributi e metodi. Al fine di rendere i diagrammi delle classi più esaustivi possibili, il team segue una precisa notazione per la rappresentazione degli attributi e dei metodi di una classe.

Nello specifico, ogni attributo di una classe viene dichiarato nel seguente modo:


```
visibilità nomeAttributo:tipo[molteplicità]=default{proprietàAggiuntive}
```

Dove

- **visibilità**: indica lo scope degli attributi. Esso può essere:
 - Pubblico: indicato con il carattere "+";
 - Privato: indicato con il carattere "-";
 - Protetto: indicato con il carattere ".".
- **nomeAttributo**: indica il nome univoco dell'attributo;
- **tipo**: indica il tipo dell'attributo;
- **molteplicità**: caratteristica opzionale, propria per le collezioni di più elementi;
- **proprietàAggiuntive**: caratteristica opzionale che indica particolari proprietà dell'attributo (per esempio utilizzato per indicare array ordinati).

Per quanto riguarda invece i metodi di una classe, il team segue la seguente dichiarazione:

```
visibilità nomeMetodo(listaParametri):tipoRitorno
```

Dove

- **visibilità**: indica lo scope del metodo. Esso può essere:
 - Pubblico: indicato con il carattere "+";
 - Privato: indicato con il carattere "-";
 - Protetto: indicato con il carattere ".".
- **nomeMetodo**: indica il nome univoco del metodo;
- **listaParametri**: indica la lista dei parametri passati al metodo. Ogni parametro passato viene scritto nel seguente modo:

```
tipo_parametro:nome_parametro
```

Nel caso esistano più parametri, essi sono separati da una virgola. Nel caso invece in cui non si passi alcun parametro alla funzione le parentesi rimangono vuote;

- **tipoRitorno**: tipo di ritorno del metodo;

Per quanto riguarda le relazioni di dipendenza tra le classi, il team fa uso di notazioni grafiche che collegano le due classi in relazione tra loro. Per ogni tipo di relazione esistono frecce diverse. Le notazioni sono le seguenti:

- **Dipendenza:** la freccia è tratteggiata, con la parte finale vuota; parte dall'oggetto della classe che richiede l'uso dell'oggetto di un'altra classe, e si conclude in quest'ultima.



Figura 2: Freccia per la relazione di dipendenza

Per esempio per indicare che la classe A è dipendente dalla classe B utilizzeremo la notazione:

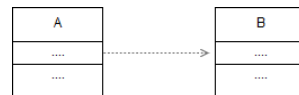


Figura 3: Relazione di dipendenza tra le classi A e B

- **Associazione:** la freccia è piena, con la parte finale inesistente, pertanto non esiste un vero e proprio verso di percorrenza.



Figura 4: Freccia per la relazione di associazione

Per esempio per indicare che la classe A contiene un riferimento a un oggetto della classe destinazione B utilizzeremo la notazione:

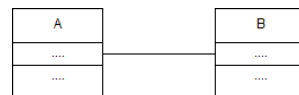


Figura 5: Relazione di associazione tra le classi A e B

- **Aggregazione:** la freccia è piena, con la parte iniziale a forma di rombo vuoto; la freccia in questo caso non è propriamente direzionale, ma il rombo ne indica la sorgente.



Figura 6: Freccia per la relazione di aggregazione

Per esempio per indicare che la classe A è parte della classe B utilizzeremo la notazione:

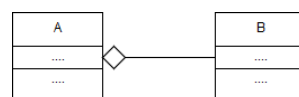


Figura 7: Relazione di aggregazione tra le classi A e B

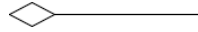


Figura 8: Freccia per la relazione di composizione

- **Composizione:** la freccia è piena, con la parte iniziale a forma di rombo pieno; la freccia in questo caso non è propriamente direzionale, ma il rombo ne indica la sorgente.

Per esempio per indicare che la classe A è composizione della classe B utilizzeremo la notazione:

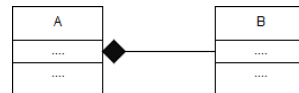


Figura 9: Relazione di composizione tra le classi A e B

- **Ereditarietà/Generalizzazione:** la freccia è piena, con la parte finale vuota; la freccia è direzionale, parte dalla classe che viene generalizzata e si conclude nella classe genitore che generalizza la figlia.



Figura 10: Freccia per la relazione di ereditarietà/generalizzazione

Per esempio per indicare che la classe A generalizza la classe B utilizzeremo la notazione:



Figura 11: Relazione di ereditarietà/generalizzazione tra le classi A e B

2.2.3.2.5 Norme per i diagrammi dei package

Al fine di rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, durante la fase di progettazione si farà uso dei diagrammi dei package, dedicati alla descrizione della dipendenza tra classi raggruppate in package.

I diagrammi dei package hanno il compito di raggruppare elementi e fornire un namespace per gli elementi raggruppati. Praticamente tutti gli elementi UML possono essere raggruppati in package (quindi classi, oggetti, use case, componenti).

Ogni package verrà rappresentato tramite un rettangolo con un'etichetta per il nome, che dovrà contenere i diagrammi delle classi appartenenti al package ed eventuali sotto-package. Le dipendenze fra i vari package dovranno essere segnalate con una freccia tratteggiata. Tale freccia, disegnata dal package A al package B, indica una dipendenza di A nei confronti di B.

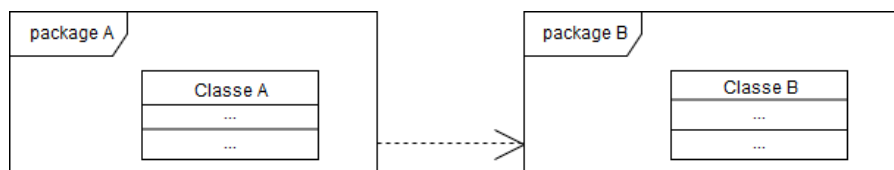


Figura 12: Relazione di dipendenza tra i package

2.2.3.2.6 Norme per i diagrammi di sequenza

Al fine di rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, durante la fase di progettazione si farà uso dei diagrammi di sequenza, dedicati a descrivere la collaborazione nel tempo tra un gruppo di oggetti.

I diagrammi di sequenza hanno il compito di mostrare l'esecuzione di un determinato caso d'uso a runtime. Descrivono il comportamento di un gruppo di oggetti che devono implementare collaborativamente un comportamento.

Per creare i diagrammi di sequenza, deve essere creato un file apposito. Il suddetto file viene prodotto attraverso lo strumento integrato nella suite di Google, detto *draw.io*_[g]. Il file viene quindi salvato nel Drive condiviso dai membri del team, in modo che possa essere accessibile in maniera semplice da ogni membro del team.

Ogni diagramma di sequenza avrà un senso di lettura verticale dall'alto al basso (verso che indica lo scorrere del tempo). La notazione per indicare una sequenza segue i seguenti formalismi:

- **Oggetti coinvolti:** rappresentati tramite un rettangolo, al cui interno si troverà un nome per identificarli nel formato istanza `:nomeClasse`.

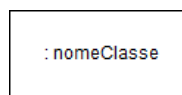


Figura 13: Formalismo per gli oggetti coinvolti dei diagrammi di sequenza

- **Linea della vita:** è una linea tratteggiata che si trova sotto ad ogni istanza. Talvolta, quando l'oggetto è attivo, sarà sormontato da una barra di attivazione;
- **Barra di attivazione:** rettangolo posto al di sopra della linea della vita che sta a indicare i momenti in cui l'oggetto è attivo. Da una barra di attivazione partiranno delle frecce, che rappresentano un messaggio/segnale, verso la linea della vita di oggetti già istanziati o, in alternativa, verso una nuova istanza di classe per crearla.

In particolare, verranno utilizzati i seguenti tipi di frecce:

- Freccia piena: per indicare un messaggio sincrono: corrisponde alla chiamata di un metodo. Sopra tale freccia si dovrà specificare il metodo invocato secondo il formato `nomeMetodo(lista parametri formali)`:

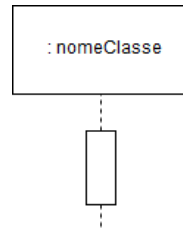


Figura 14: Formalismo per per rappresentare la barra di attivazione di un oggetto dei diagrammi di sequenza

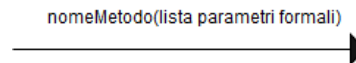


Figura 15: Messaggio sincrono dei diagrammi di sequenza

- Freccia: per indicare un messaggio asincrono (chi invoca il metodo non attende il return). Sopra tale freccia si dovrà specificare il metodo invocato secondo il formato `nomeMetodo(lista parametri formali)`:

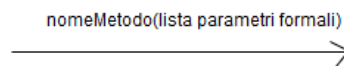


Figura 16: Messaggio asincrono dei diagrammi di sequenza

- Freccia tratteggiata : per indicare il ritorno di un metodo chiamato. Sopra tale freccia si dovrà indicare il tipo di ritorno secondo il formato `tipoRitorno`:

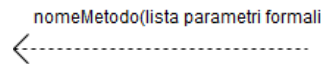


Figura 17: Ritorno di un metodo chiamato dei diagrammi di sequenza

- Freccia tratteggiata sormontata da «create»: per indicare la creazione di un nuovo oggetto. Termina sempre in un rettangolo che ne contiene il nome nel formato `NomeClasse`:

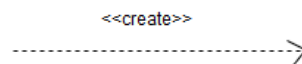


Figura 18: Formalismo per la creazione di oggetti nei diagrammi di sequenza

- Freccia tratteggiata sormontata da «destroy»: per indicare la distruzione di un nuovo oggetto. Termina sempre in una X, nella quale muore anche la linea della vita dell'oggetto:

Sarà inoltre possibile determinare sui diagrammi di sequenza dei frame di interazione associati ad una guardia (o condizione):

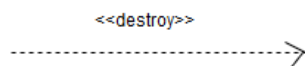


Figura 19: Formalismo per la distruzione di oggetti nei diagrammi di sequenza

- alt: indica dei frammenti in alternativa fra loro, verrà eseguito quello per cui la guardia è verificata;
- opt: indica un frammento che viene eseguito solo se la guardia è specificata;
- par: indica frammenti eseguiti in parallelo;
- loop: indica un frammento che viene eseguito più volte, la condizione di arresto del ciclo è la guardia;
- region: indica un frammento critico che deve essere eseguito in mutua esclusione.

Potranno essere inseriti dei commenti nelle sezioni del diagramma che necessitano di ulteriori informazioni o per indicare particolari stati del diagramma.

2.2.3.2.7 Norme per i diagrammi di attività

Al fine di rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, durante la fase di progettazione si farà uso dei diagrammi di attività, dedicati a descrivere la logica procedurale; questo tipo di diagrammi pertanto modellano un processo. Organizza più entità in un insieme di azioni secondo un determinato flusso.

Essi modellano un'attività relativa ad un qualsiasi elemento di modellazione, ad esempio:

- Classi;
- Casi d'uso;
- Interfacce;
- Componenti;
- Interfacce;
- Operazioni di classe.

In fase di progettazione questo tipo di diagrammi vengono utilizzati per:

- Modellare il funzionamento di un'operazione;
- Modellare un algoritmo.

Per creare i diagrammi delle attività, deve essere creato un file per ogni attività. Il suddetto file viene prodotto attraverso lo strumento integrato nella suite di Google, detto *draw.io*_[g]. Il file viene quindi salvato nel Drive condiviso dai membri del team,

in modo che possa essere accessibile in maniera semplice da ogni membro del team. La notazione per indicare un'attività segue i seguenti formalismi:

- **Nodo iniziale:** nodo obbligatorio, dal quale inizia l'esecuzione di un'attività;



Figura 20: Nodo iniziale di un diagramma di attività

- **Attività:** ogni attività svolta all'interno del diagramma di sequenza viene contenuta in un rettangolo che ne contiene una breve descrizione testuale

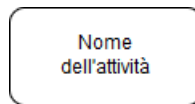


Figura 21: Activity di un diagramma di attività

- **Fork:** indica un'elaborazione parallela;

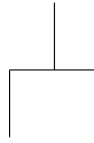


Figura 22: Costrutto fork di un diagramma di attività

- **Join:** indica una sincronizzazione tra processi paralleli;

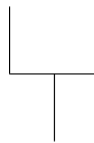


Figura 23: Costrutto join di un diagramma di attività

- **Branch:** indica la decisione di uno dei nodi da cui continuare l'attività;

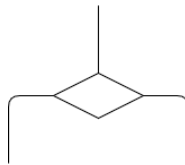


Figura 24: Costrutto branch di un diagramma di attività

- **Nodo finale:** nodo obbligatorio, nel quale si conclude l'esecuzione di un'attività.



Figura 25: Nodo finale di un diagramma di attività

- **Puntamento:** formalismo utile nel caso vi sia esigenza di modellare molteplici azioni. Permette a un'azione di puntare una sottoazione formalizzata altrove

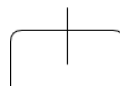


Figura 26: Costrutto puntamento di un diagramma di attività

- **Pin:** formalismo che permette di passare parametri tra le azioni. I parametri passati possono essere sia in input che in output, e rispettivamente saranno posizionati in alto o in basso rispetto l'activity



Figura 27: Costrutto pin di un diagramma di attività

- **Partizioni:** formalismo utile per attribuire responsabilità diverse alle azioni

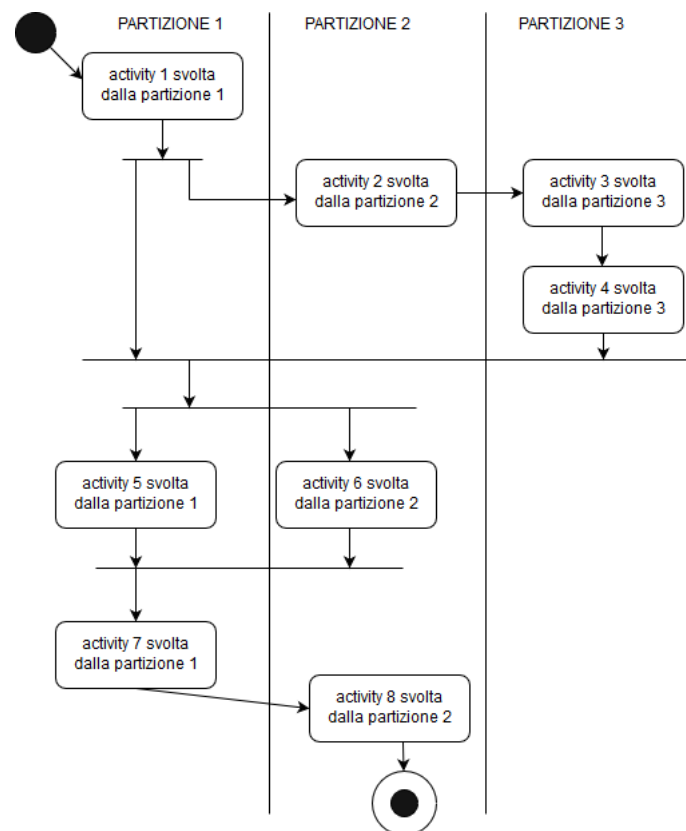


Figura 28: Costrutto partizione di un diagramma di attività

- **Ricezione segnali ext:** formalismo che indica che l'azione sta aspettando di ricevere un segnale esterno per partire



Figura 29: Costrutto di attesa di un segnale di un diagramma di attività

- **Generazione segnali:** formalismo che indica che l'azione, alla sua conclusione, genera un token/segnale

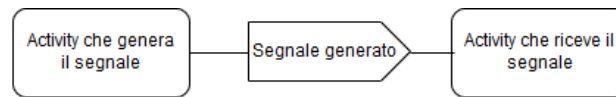


Figura 30: Costrutto di generazione di un segnale di un diagramma di attività

- **Timeout:** formalismo che serve per modellare timeout o eventi ripetuti. I timeout hanno frecce entranti e frecce uscenti, al contrario degli eventi ripetuti che possono avere solo archi uscenti. Di fianco devono avere una descrizione così composta:
 - I timeout devono cominciare con "wait", con di seguito un'indicazione temporale o una descrizione in linguaggio naturale breve e concisa;
 - Le azioni ripetute con "every", con di seguito un'indicazione temporale o una descrizione in linguaggio naturale breve e concisa.



Figura 31: Costrutto timeout di un diagramma di attività

2.2.3.2.8 Attività Technology Baseline

L'attività consiste nell'individuare l'architettura logica ad alto livello del prodotto e gli strumenti (tool, framework, libreria) per il supporto alla fase di sviluppo che ne seguirà.

Sarà eseguita pertanto un'analisi comparativa di varie tecnologie esistenti al fine di capire quali siano le più consone allo sviluppo del progetto P2PCS. Per avere ulteriori riscontri in merito alle metodologie di sviluppo da utilizzare verranno fissati incontri (anche per vie telematiche) con la proponente GaiaGo.

- **Diagrammi UML:** durante questa attività si devono produrre diagrammi delle classi (e conseguentemente diagrammi di package) che descrivano l'organizzazione logica delle diverse componenti del sistema;
- **Design pattern architetturale:** durante questa attività i Progettisti devono individuare il design pattern architetturale più consono alla realizzazione del prodotto. Al fine di presentare al Professor Riccardo Cardin (in veste di committente) nel modo più esaustivo possibile le scelte fatte in questo senso, occorre produrre un diagramma che descriva l'applicazione del design pattern architetturale scelto. Tale diagramma sarà pertanto descritto in fase di colloquio di technology baseline;
- **Test di integrazione:** i Progettisti devono definire delle classi di verifica. Tali classi sono necessarie per verificare che i componenti del sistema funzionino nella maniera prevista.

Product Baseline

L'attività, che segue alla Technology baseline, consiste nella progettazione di dettaglio del prodotto, ovvero nell'implementazione effettiva, attraverso gli strumenti individuati in fase di Technology baseline, del prodotto. La progettazione di dettaglio di P2PCS avviene seguendo il modello incrementale, spiegato nel dettaglio nel documento *Piano di Progetto v1.0.0*. L'analisi è comprensiva di design pattern contestualizzati all'architettura.

- **Diagrammi UML:** durante l'attività vengono utilizzati i diagrammi delle classi, di attività e di sequenza per meglio esplicitare lo svolgimento del lavoro. Tali diagrammi verranno poi forniti al Professor Riccardo Cardin (in veste di committente);
- **Definizione di classe:** durante il colloquio con il professor Riccardo Cardin, inoltre, verranno fornite spiegazioni in merito allo scopo delle classi e delle funzionalità che esse modellano (attraverso una breve panoramica dei metodi e degli attributi propri della classe).
- **Test di unità e Continuous integration :** I Progettisti devono definire i test di unità necessari per verificare che i componenti del sistema funzionino nel modo previsto. L'attività di integrazione sarà effettuata utilizzando il servizio, compatibile con GitHub, *Bitrise_[g]*. Questo servizio implementa il modello di integrazione continua: ad ogni commit su repository GitHub, Bitrise crea una build ed esegue automaticamente i test di unità. In questo modo sarà possibile identificare immediatamente modifiche o errori che inficiano sul comportamento del prodotto.

2.2.3.3 Codifica

2.2.3.3.1 Scopo

La fase ha come scopo l'effettiva realizzazione del prodotto software richiesto, ri-

spettando le metriche stabilite nel documento *Piano di Qualifica v1.0.0*. In questa fase si concretizza la soluzione attraverso la programmazione, in modo da ottenere il prodotto software finale.

Durante l'attività di codifica i programmatori devono seguire le linee-guida ivi indicate, al fine di rendere il codice più uniforme e leggibile per favorire le fasi manutenzione, verifica e validazione, e di conseguenza migliorare la qualità del prodotto. All'inizio verranno elencate delle norme generali a cui i programmatori devono attenersi con qualsiasi linguaggio di programmazione utilizzato, mentre di seguito verranno elencate delle norme via via più specifiche. Le norme alle quali i programmatori si devono attenere sono tratte trattate nella sezione §2.2.3.3.3.

2.2.3.3.2 Aspettative

Obiettivo dell'attività è la creazione di un prodotto software conforme alle richieste prefissate con il proponente.

2.2.3.3.3 Stile di codifica

Lo stile di codifica imposto viene tratto dall' *Android Code Style for Contributors*_[9]. Ogni norma è rappresentata da un paragrafo. Ciascuna ha un titolo, una breve descrizione e, se necessario, un esempio che illustra i modi accettati o meno. Alcune di esse includono inoltre una lista di possibili eccezioni d'uso.

Indentazione 1: I blocchi innestati devono essere correttamente indentati, usando per ciascun livello di indentazione quattro spazi.

Esempi di utilizzo:

```
1  for(int i=0; i<10; i++){
2  |    x++;|
3  }
```

Figura 32: Indentazione1 corretta

```
1  for(int i=0; i<10; i++){
2  |x++;
3  }
```

Figura 33: Indentazione1 scorretta

Indentazione 2: Per andare a capo di una linea di codice usare otto spazi.

Esempi di utilizzo:

```

1 Instrument i=
2     someLongExpression(
3         with, a, lot, of,
4         variables);
5

```

Figura 34: Indentazione2 corretta

```

1 Instrument i=
2     someLongExpression(
3         with, a, lot, of,
4         variables);
5

```

Figura 35: Indentazione2 scorretta

Indentazione 3: È vietato l'utilizzo di tabulazioni, che devono essere necessariamente sostituite da spazi. Al fine di assicurare il rispetto di questa regola si consiglia di configurare adeguatamente il proprio editor o *IDE*_[g].

Indentazione 4: Il codice che appartiene ad un blocco, deve essere innestato allo stesso livello di quel blocco.

Esempi di utilizzo:

```

1 int x;
2 int y;
3 for(int i=0; i<10; i++){
4     y=0;
5     for(int j=0;j<5;j++){
6         x++;
7         x=x+y;
8         if (x>4){
9             x--;
10        }
11    }
12 }
13
14

```

Figura 36: Indentazione4 corretta

```

1  int x;
2  int y;
3  for(int i=0; i<10; i++){
4  y=0;
5  for(int j=0;j<5;j++){
6  x++;
7  x=x+y;
8  if (x>4){
9  x--;
10 }
11 }
12 }

```

Figura 37: Indentazione scorretta

Parentesizzazione 1: I blocchi di codice vanno racchiusi tra parentesi graffe. Esempi di utilizzo:

```

1  if(x<10){
2  |    x++|
3  }

```

Figura 38: Parentesizzazione 1 corretta

```

1  if(x<10)
2  |    x++;|

```

Figura 39: Parentesizzazione 1 scorretta

Eccezioni: è possibile evitare le parentesi graffe se e solo se il corpo del blocco si riduce ad una sola riga breve, ad esempio:

```

1  if(x<10) x++;
2

```

Figura 40: Eccezione per parentesizzazione 1 corretta

Parentesizzazione 2: Le parentesi graffe iniziano nella stessa riga del codice, non in quella sottostante.

Esempi di utilizzo:

Scrivere metodi brevi: Quando possibile mantenere i metodi brevi e concentrati sullo scopo. A volte metodi “lunghi” sono appropriati, quindi questa regola non è una regola obbligatoria ma consigliata. Se il corpo del metodo si protrae per

```
1  class MyClass{
2      int value1;
3      int value2;
4      public int f(i){
5          i++;
6      }
7  }
8
```

Figura 41: Parentesizzazione 2 corretta

```
1  class MyClass
2  {
3      int value1;
4      int value2;
5      public int f(i)
6      {
7          i++;
8      }
9  }
10
```

Figura 42: Parentesizzazione 2 scorretta

più di quaranta righe di codice conviene pensare come suddividerlo senza minare la struttura del programma.

Scrivere brevi linee di codice: Una riga di codice deve essere lunga al massimo 100 caratteri, in caso si superi questo limite occorre spezzare la riga in due.

Nomi 1 - Univocità dei nomi: Classi, metodi, variabili devono avere un nome univoco ed esplicativo al fine di evitare quanto più possibile ambiguità di comprensione.

Nomi 2 - Classi: Le classi iniziano sempre con una lettera maiuscola.

Nomi 3 - Costanti: I nomi delle costanti definite come *public*_[g] *static*_[g] *final*_[g] vanno scritte usando solo maiuscole.

Nomi 4 - Metodi: nomi dei metodi iniziano con una lettera minuscola, se sono composti da più parole le successive devono iniziare con una lettera maiuscola. Le sigle e gli acronimi, all'interno dei nomi dei metodi, vengono trattate come parole.

Nomi 5: Tutto il resto (variabili di istanza, variabili locali, parametri, nomi dei metodi) deve iniziare con una lettera minuscola.

Documentazione 1 - Lingua utilizzata: I nomi di variabili, costruttori, metodi, classi e i commenti per documentare il codice vanno scritti in inglese.

Documentazione 2 - TODO: Usare un commento “TODO” per codice temporaneo, soluzioni a breve termine o evidentemente migliorabili. Un commento TODO va preceduto dalla parola TODO e da due punti “:”. In caso i commenti TODO si riferiscano ad eventi futuri, occorre specificare l’evento o la data il più precisamente possibile.

```
1 //TODO: Remove this code after the UriTable2 has been checked in.
2 //TODO: Change this to use a flag instead of a constant.
3 //TODO: Fix by April 2019.
4 //| TODO: Remove this code after all prod. mixers understand protocol V7.
5
```

Figura 43: Sintassi commenti TODO

Documentazione 3 - Annotazioni: vanno usate obbligatoriamente le seguenti annotazioni:

- **@Override:** Da usare ogni qual volta si sovrascriva la dichiarazione o l’implementazione da una super classe;
- **@Deprecated:** Per sconsigliare l’uso di codice funzionante;
- **@SuppressWarnings:** Per eliminare avvertimenti che altrimenti non sarebbero rimovibili (ad esempio con l’uso di *Generics_[9]*), va sempre preceduta da un commento TODO che specifichi la condizione per la quale l’avvertimento è impossibile da eliminare.

```
1 //TODO: the third-party class
2 //com.third.useful.Utility.rotate() needs generics
3 @SuppressWarnings ("generic-cast")
4 List<String> blix = Utility.rotate (blax);
5
```

Figura 44: Sintassi @SuppressWarnings

Consistenza 1 - Visibilità: Le variabili vanno dichiarate nel blocco più interno

che comprenda completamente l'uso di quella variabile. Vanno assolutamente evitate variabili globali. Le variabili dei cicli vanno dichiarate all'interno dello statement a meno che non ci sia una buona ragione per fare altrimenti.

Consistenza 2 - Inizializzazione: Le variabili devono sempre essere inizializzate il prima possibile. Vicino ad ogni dichiarazione dovrebbe esserci un'inizializzazione. Nel caso in cui non si abbiano sufficienti informazioni per inizializzare una variabile sensibile occorre posporre la dichiarazione a quando si saranno ottenute.
Esempio corretto: Esempio scorretto: Eccezioni: Variabili usate all'interno di ecce-

```
1 String s="hi";|
2
```

Figura 45: Inizializzazione corretta

```
1 String s;
2 s="hi";
3
```

Figura 46: Inizializzazione scorretta

zioni nei blocchi try/catch, per esempio:

```
1 int n;
2 try{
3     n=Integer.parseInt(someString);
4 } catch (NumberFormatException nfe) {
5     n=10;|
6 }
```

Figura 47: Inizializzazione corretta

Importazione 1:Indicare la classe esatta quando si importa un pacchetto, evitando l'uso di *.

Eccezioni: è permesso l'uso di * nel caso di pacchetti java o javax.

Importazione 2 - Ordine di importazione:Usando le clausole di importazione dei *package_{|g|}*, questi vanno raggruppati in tre gruppi divisi da una linea vuota tra loro, secondo questo ordine:

- Package di Android;
- Package di terze parti;
- Package java o javax.

Inoltre per ogni gruppo i package vanno ordinati in ordine alfabetico con le maiuscole che precedono le minuscole quindi Z viene prima di a.

2.2.3.3.4 Convenzioni per la documentazione

- **Intestazione:** Ogni file contenente codice deve avere la seguente intestazione contenuta in un commento e posta all'inizio del file stesso:

```
File:nome del file;  
Version: versione del file nella forma X.Y descritta  
        in seguito;  
Type: tipo del file;  
Date: data di creazione del file;  
Author: autore del file;  
  
License: tipo licenza del file;  
  
Advice: lista avvertenze e limitazioni legate al file;  
  
Changelog: registro modifiche strutturato come:  
           Author || Data || Description
```

Per quanto riguarda la versione del file, essa sarà rappresentata nella seguente forma: X.Y, dove X, Y sono numeri interi che, rispettivamente, rappresentano l'indice di una versione principale e l'indice di una modifica parziale. L'incremento del valore X rappresenta un avanzamento della versione stabile e implica l'azzeramento dell'indice Y. L'incremento dell'indice Y rappresenta una verifica o una modifica rilevante all'interno del documento (per esempio l'aggiunta o la rimozione di una o più istruzioni).

La versione 1.0 deve rappresentare la prima versione del file completo e stabile, cioè quando le sue funzionalità obbligatorie sono state definite e si considerano funzionanti. Solo dalla versione 1.0 è possibile testare il file, con degli appositi test predefiniti, per validarne la qualità.

2.2.3.4 Metriche

2.2.3.4.1 M[PROC][0001]: Numero campi dati per classe

Indice del numero di campi definiti in una classe. Un numero eccessivo di campi dati rischia di rendere la classe poco specializzata e indica una cattiva progettazione.

2.2.3.4.2 M[PROC][0002] - Parametri per metodo

Indice del numero di parametri definiti in un metodo. Un numero eccessivo di

parametri per metodo potrebbe sovraccaricare lo *stack*_{|g|} e comprometterne la funzionalità.

2.2.3.4.3 M[PROC][0003] - Metodi per classe

Indice del numero di metodi definiti in una classe.

2.2.3.4.4 M[PROC][0004]: Grado di instabilità

È un modo per misurare l'instabilità delle componenti di un sistema, in particolare la possibilità di effettuare modifiche ad un elemento del sistema senza influenzarne altri all'interno dell'applicazione. Questo risultato dipende dall'indice afferente (numero di classi esterne dipendenti da quelle interne) ed efferente (numero di classi interne dipendenti da quelle esterne).

$$I = \left(\frac{Ce}{Ca + Ce} \right) * 100$$

dove Ce è indice efferente e Ca è indice afferente.

2.2.3.4.5 M[PROC][0005] - Complessità Ciclomatica

Indica la complessità di funzioni, moduli, metodi o classi di un programma contando il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso.

2.2.3.4.6 M[PROC][0006] - Linee di commento per linee di codice

Indica la percentuale di linee di commento presenti all'interno del codice sorgente.

$$P = \left(\frac{Nc}{Nsloc} \right) * 100$$

dove Nc è il numero di linee di commento e Nsloc è il numero di linee di codice prodotte.

2.2.3.4.7 M[PROC][0007] - Indice di manutenibilità

Permette di stabilire quanto sarà semplice mantenere il codice prodotto.

$$MI = 171 - 3.42 * \ln(\text{aveE}) - 0.23 * \ln(\text{aveV}) - 1616.2 * \ln(\text{aveLOC})$$

dove aveE è l'Halstead Effort medio per modulo, aveV è la complessità ciclomatica media per modulo, e aveLOC è il numero medio di linee di codice per modulo.

2.2.3.5 Strumenti utilizzati

Di seguito sono elencati gli strumenti utilizzati dal team Cyber13 in fase di sviluppo.

Gli elementi indicati con * sono stati identificati a seguito di un'analisi preliminare. Tali strumenti potrebbero cambiare in caso di necessità differenti da quanto pianificato.

- **Creazione diagrammi UML:** Per la produzione dei diagrammi UML viene utilizzato *Draw.io*_[g], idoneo dato il funzionamento sul cloud e la condivisione diretta su *Google Drive*_[g];
- **IDE:** Si utilizza *Android Studio*_[g] per la codifica in Java e Kotlin.
- **Realizzazione Slide di presentazione:** per preparare le slide di presentazione del progetto il team utilizza lo strumento Presentazioni Google, che realizza il file di presentazione direttamente all'interno del Google Drive associato alla mail del gruppo e permette ad ogni membro del team di apportare modifiche e aggiornamenti anche da remoto.
- **Continuos Integration:** Implementata attraverso la piattaforma *Bitrise*_[g].

3 Processi di supporto

3.1 Processo di documentazione

3.1.1 Scopo

In questa sezione sono indicate le norme per quanto riguarda per la stesura, *verifica*_[g] e approvazione riguardante la documentazione ufficiale. Le norme ivi indicate sono tassative per tutti i documenti formali presentati, esplicitamente riportati nella sezione §3.1.6.

3.1.2 Aspettative

Il gruppo desidera consegnare documentazione formale coerente. Per conseguire tale fine vengono seguite pedissequamente le norme relative alla documentazione ivi indicate.

3.1.3 Ciclo di vita della documentazione

Ogni documento formale presentato dovrà superare con esito positivo le seguenti fasi:

- **Sviluppo:** Comprende le decisioni in merito alla strutturazione dei contenuti da comprendere nel documento e la stesura degli stessi. Il documento concluso in questo stadio è non formale.
- **Verifica:** Quando si ritiene conclusa la stesura di un documento non formale, viene chiamato in causa il Responsabile di progetto. È sua responsabilità assegnare il documento ai Verificatori, la cui incombenza è svolgere l'attività di verifica. Quest'ultima consiste nel controllare la correttezza formale del documento. Al termine del controllo gli esiti possibili sono due:
 - Esito negativo da parte dei verificatori: Nel caso in cui i Verificatori individuino degli errori o delle difformità nel documento; sarà loro dovere comunicare le proprie valutazioni al Responsabile di progetto. Quest'ultimo riassegnerà il documento ai redattori, che ripeteranno la stesura del documento o delle sue parti ritenute incorrette. Il ciclo si ripete finché i Verificatori non hanno più segnalazioni.
 - Esito positivo da parte dei verificatori: I verificatori non individuano difformità. Il documento entra in fase di approvazione da parte del Responsabile di progetto.
- **Approvazione:** Segue l'esito positivo della fase di verifica. Successivamente il documento verrà quindi passato al Project Manager, che sarà responsabile dell'approvazione o meno del documento. Gli esiti possibili sono due:

- Se il documento non viene considerato adeguato al rilascio il Responsabile di progetto comunicherà ai redattori le modifiche da apportare. In casi estremi, potrà imporre che la stesura del documento dovrà essere rifatta nella sua totalità.
- Se il documento verrà approvato lo si riterrà un documento formale, e potrà essere distribuito alle persone nominate nella lista di distribuzione.

A seguire è presente un diagramma rappresentante una visione schematica del ciclo di vita sopra descritto:

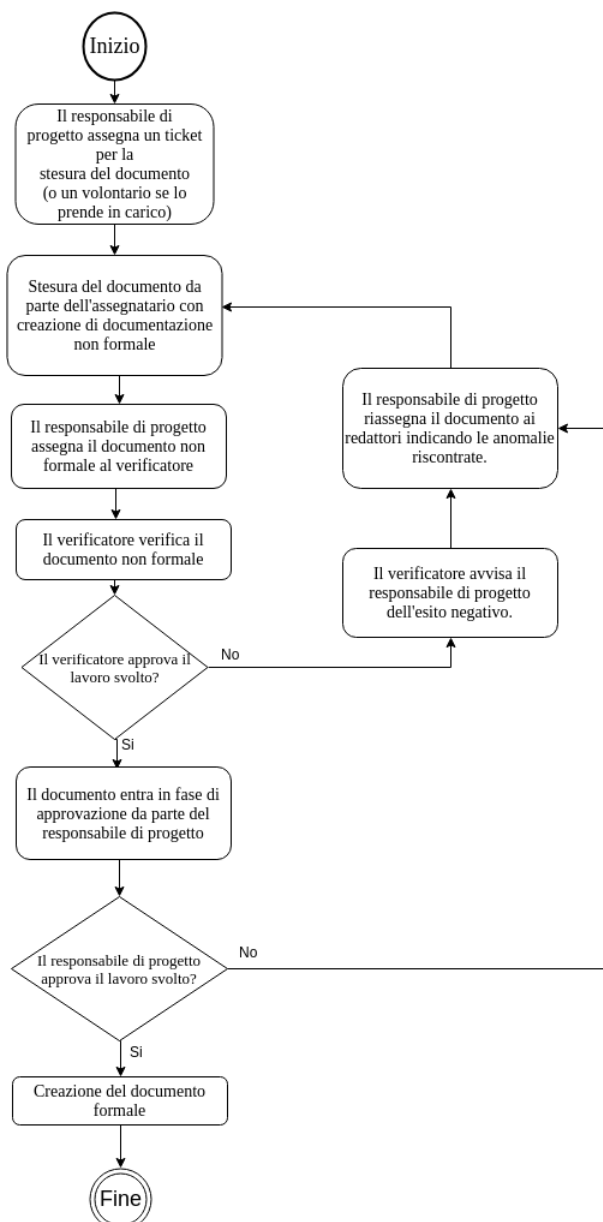


Figura 48: Diagramma del ciclo di vita della documentazione

3.1.4 Separazione tra documenti interni e esterni

I documenti formali redatti dal gruppo Cyber13 rientreranno tassativamente in una delle seguenti categorie (mutuamente esclusive):

- **Documenti interni** : Documenti ad uso esclusivo del team Cyber13, redatti in lingua Italiana.
- **Documenti esterni**: Documenti che verranno condivisi anche con il proponente e i committenti. Nel caso di documenti utili per il *deploy_[g]* del software o per il loro utilizzo da parte degli utenti finali, dovranno essere redatti in lingua inglese.

3.1.5 Nomenclatura dei documenti

Per quanto riguarda il nome dei documenti formali essi verranno nominati attraverso la dicitura "NomeDocumento", senza spazi e con lettere maiuscole all'inizio di ogni parola, seguita dalla versione, nella forma: "NomeDocumento_vX.Y.Z.estensione". Per quanto riguarda i verbali (interni ed esterni) si seguirà una particolare nomenclatura, descritta nella sezione 4.1.4.3.

3.1.5.1 Versioni di un documento

Il numero di versione è indicato tramite il carattere 'v' seguito da un numero, da un punto, da un numero, da un punto e un numero: vX.Y.Z. Ogni volta che si inserisce una modifica nell'omonimo diario, va assegnato un nuovo numero di versione, in modo coerente con le seguenti regole:

- Quando verrà creato un documento avrà necessariamente il numero di versione 'v0.0.1'.
- **X**: Rappresenta il numero di pubblicazioni ufficiali del documento (quindi ogni qualvolta il documento superi con esito positivo la fase di approvazione). Ad ogni nuova pubblicazione del documento i valori Y e Z vengono azzerati e quello di X incrementato di uno.
- **Y**: Rappresenta il numero di verifiche eseguite sul documento. L'incremento dell'indice Y implica l'azzeramento del valore dell'indice Z;
- **Z**: Rappresenta il numero di modifiche effettuate al documento durante il suo sviluppo.

3.1.5.2 Formato dei file

I documenti sono redatti attraverso lo strumento Latex, pertanto ogni documento si trova nel formato .tex durante il suo sviluppo. Dopo il superamento della fase di approvazione del documento da parte del Responsabile viene il documento viene esportato in formato PDF.

3.1.6 Documenti correnti

Di seguito si presentano i documenti formali, classificati per appartenenza (interno o esterno) consegnati:

- **Analisi dei Requisiti:** Uso esterno, sigla (AdR).
Documento per esporre e scomporre i requisiti del progetto contenente i *casì d'uso_{|g|}* relativi al prodotto e diagrammi di interazione con l'utente. Viene scritto dagli Analisti dopo aver analizzato il capitolato e interagendo con il Proponente in riunioni esterne.
- **Glossario:** Uso esterno, sigla (GL).
Documento per raccogliere le definizioni dei termini o concetti che saranno usati nei documenti formali per facilitarne la comprensione.
- **Piano di Progetto:** Uso esterno, sigla (PdP).
Documento per l'analisi e la pianificazione della gestione delle risorse di tempo e umane.
- **Piano di Qualifica:** Uso esterno, sigla (PdQ).
Documento per descrivere standard e obiettivi che il gruppo dovrà raggiungere per garantire la qualità di prodotto e processo.
- **Studio di Fattibilità:** Uso interno, sigla (SdF).
Documento per indicare le riflessioni, punti di forza e caratteristiche sfavorevoli per ogni capitolato proposto sulla base dei quali il gruppo ha fatto la sua scelte.
- **Norme di Progetto:** Uso interno, sigla (NdP).
Documento per mostrare le direttive e gli standard utilizzati all'interno del gruppo di lavoro Cyber13 per lo sviluppo del progetto;
- **Developer manual:** Uso esterno, sigla (DM).
Manuale in lingua inglese indirizzato agli sviluppatori che avranno il compito di mantenere o estendere il prodotto P2PCS in futuro. A suo interno vengono pertanto elencate le tecnologie impiegate, i design pattern scelti e le varie scelte architettureali che hanno caratterizzato lo sviluppo del prodotto;
- **User manual:** Uso esterno, sigla (UM).
Manuale in lingua inglese indirizzato ai futuri utenti che utilizzeranno il prodotto P2PCS in futuro. A suo interno vengono pertanto elencate le funzionalità che il prodotto offre e in che modo l'utente può usufruirne.

3.1.7 Formattazione dei documenti

3.1.7.1 Strutturazione dei file

Ogni documento formale si deve attenere alla seguente strutturazione dei contenuti:

- **Frontespizio:** Si tratta della prima pagina dei documenti formali, conterrà le seguenti informazioni (in ordine dall'alto verso il basso):
 - Intestazione: "Università degli Studi di Padova";

- Logo del gruppo: Centrato;
- Titolo del documento: Centrato;
- Nome del gruppo - Nome progetto: In corsivo.
- **Sezione di informazione del documento:** Si trova sempre nella prima pagina del documento e contiene le seguenti informazioni:
 - Versione: Attenendosi alle norme della sezione 3.1.4.2;
 - Data Redazione: Seguendo il formato indicato al punto 3.1.6.2 "Formati ricorrenti";
 - Responsabile: Nome del responsabile che ha supervisionato il documento;
 - Redazione: Nomi dei redattori del documento;
 - Verifica: Nomi dei verificatori del documento;
 - Approvazione: Nome del responsabile che ha approvato il documento;
 - Uso: Interno o esterno;
 - Destinatari: A cui è indirizzato il documento;
 - Mail di contatto: Mail del gruppo Cyber13.
- **Diario delle modifiche:** Tabella inserita nella seconda pagina del documento. Ha lo scopo di riepilogare l'elenco delle modifiche che sono state apportate al documento nel corso del processo di redazione. L'assegnamento delle versioni si attiene alle norme indicate nella sezione 3.1.4.2 "Versioni del documento". Il diario è una tabella ordinata in modo decrescente secondo la data della modifica e, di conseguenza, il numero di versione. Si adotta questo stile per focalizzare gli ultimi cambiamenti poiché trattati nelle prime righe della tabella. Ogni riga del diario dovrà contenere i seguenti elementi nell'esatto ordine in cui sono indicati:
 - Versione del documento;
 - Data della versione;
 - Descrizione delle modifiche apportate;
 - Autore delle modifiche apportate;
 - Ruolo dell'autore delle modifiche;
- **Indice delle sezioni:** Gli indici hanno lo scopo di riepilogare e dare una visione macroscopica della struttura del documento. Permettono quindi di rintracciare i contenuti tramite una gerarchia. Ogni documento, esclusi i verbali, dovrà essere corredato dall'indice dei contenuti. Esso sarà posizionato dopo il diario delle modifiche. Se sono presenti tabelle o immagini all'interno del documento, l'indice dei contenuti sarà seguito dalla lista delle tabelle e poi dalla lista delle figure.

- **Elenco delle tabelle:** Vengono indicate tutte le tabelle inserite all'interno del documento.
- **Elenco delle figure:** Vengono indicate tutte le figure inserite all'interno del documento.
- **Introduzione:** Sezione sempre presente nei documenti che ne chiarifica:
 - Scopo del documento;
 - Scopo del prodotto;
 - Riferimenti: normativi e informativi.
- **Contenuto del documento:** Le restanti pagine del documento sono interamente occupate dal contenuto dello stesso. In ogni pagina, esclusa il frontespizio, devono comparire i seguenti elementi:
 - Logo del team: Dovrà comparire, nell'intestazione della pagina, il logo del gruppo Cyber13. Tale logo sarà posizionato a sinistra.
 - Sezione corrente: Il numero e il nome della sezione in cui ci si trova dovranno essere presenti nell'intestazione. Tali elementi saranno posizionati a destra;
 - Nome del documento: A piè di pagina, a sinistra, deve apparire il nome del documento completo di versione.
 - Numero di pagina: A piè di pagina, a destra, deve comparire il numero della pagina espresso nella seguente notazione: "Pagina x di n", dove x è la pagina corrente e n il numero di pagine totali.

3.1.7.2 Norme tipografiche

- **Virgolette:** Alte singole ' ' per singolo carattere, alte doppie " " per racchiudere stringhe mentre parentesi angolari « » per racchiudere citazioni.
- **Parentesi:** Tonde per descrivere esempi e fornire sinonimi o precisazioni, quadre per rappresentare uno *standard*_[g] *ISO*_[g], uno stato relativo a un *ticket*_[g] o un riferimento ad un codice definito all'interno del documento stesso.
- **Punteggiatura:** Ogni segno di punteggiatura deve essere seguito da uno spazio e non avere spazi precedenti al segno stesso;
- **Stile del testo:**
 - Corsivo: Per dare enfasi ad una parola, un concetto o per indicare il nome di un termini di glossario e nomi di documenti.
 - Grassetto: Per i titoli, sottotitoli ed elementi di elenchi e liste di definizione di primo livello.
 - Azzurro: Per indicare dei collegamenti ipertestuali.

- **Elenchi:** La prima parola di ogni punto appartenete a un elenco inizierà con la lettera maiuscola. In caso di elenchi di definizioni, al termine da definire seguiranno i 'due punti' (:) successivamente vi sarà la sua descrizione. Al termine della descrizione dell'elemento si inserirà il carattere 'punto e virgola' (;) nel caso di definizione semplice, mentre il carattere 'punto' (.) al termine di una definizione complessa (più di un periodo). Per l'ultimo elemento della lista si userà sempre il carattere 'punto' (.);
- **Formati ricorrenti:**
 - Date: Scritte con lo standard YYYY-MM-DD dove YYYY indica l'anno, MM il mese e DD il giorno;
 - Orari: Scritti nel formato 24h.
- **Componenti grafiche:**
 - Immagini: I formati ammessi sono PNG o JPG;
 - Tabelle: Devono rispettare lo stile del *template_{|g|} Latex_{|g|}* realizzato.

3.1.8 Strumenti utilizzati

La stesura dei documenti deve essere effettuata utilizzando il *linguaggio di markup_{|g|} LaTeX_{|g|}* e l'ambiente *Overleaf_{|g|}* con dizionario italiano ed inglese installati.

3.2 Processo di garanzia della qualità

3.2.1 Scopo

Ivi sono descritte le norme alle quali i componenti del gruppo devono sottostare in sede di redazione del *Piano di Qualifica v3.0.0*.

3.2.2 Aspettative

Il gruppo desidera redarre il documento *Piano di Qualifica v3.0.0* in modo coerente con quanto riportato nelle norme ivi indicate.

3.2.3 Classificazione dei processi

La qualità del lavoro è garantita dalla suddivisione di quest'ultimo in vari processi, ciascuno dei quali vede associate una o più metriche i cui valori di accettazione vengono definiti nel Piano di Qualifica.

Ad occuparsi di tale suddivisione sono gli Amministratori, che dovranno rispettare la seguente notazione:

PROC[num]

Dove num è un numero positivo, intero di quattro cifre che identifica univocamente il determinato processo. La numerazione parte da "0001".

3.2.4 Classificazione delle metriche

La rilevazione della qualità del lavoro è effettuata dagli amministratori attraverso le metriche. Onde evitare ambiguità le metriche dovranno seguire la seguente notazione:

M[categ][sottocateg][num]

All'interno della quale:

- **categ**: Indica la categoria della metrica riferendosi a prodotti, processi o test. Può assumere i seguenti valori:
 - PROC: Per indicare i processi;
 - PROD: Per indicare i prodotti;
- **sottocateg**: Indica la sotto-categoria della metrica, se esiste, in caso contrario non è presente.
 - Per quanto riguarda la categoria PROC, solo nel caso in cui si tratti di processi di test, possono assumere i seguenti valori:
 - * TA: per indicare tutti i tipi di test
 - * TM: per indicare i test di modulo
 - * TH: per indicare i test ad alto livello
 - Per quanto riguarda la categoria PROD, possono assumere i seguenti valori:
 - * D: per indicare i documenti;
 - * S: per indicare il software.
- **num**: Numero positivo, intero di quattro cifre che identifica univocamente la determinata metrica. La numerazione parte da "0001".

3.2.5 Procedure

L'implementazione di metriche tramite procedure verrà studiato con l'avanzare del progetto.

3.2.6 Metriche per la qualità di processo

Il gruppo ha individuato nello standard ISO/IEC 12207 alcuni processi il cui scopo è quello di garantire la qualità del prodotto finale. Ogni metrica di qualità di processo è stata inserita nella sezione relativa al processo di cui tratta.

3.2.6.0.1 M[PROC][0010] - Halstead Difficulty per-function

Misura il livello di complessità di una funzione.

$$DIF = \left(\frac{UOP}{2}\right) * \frac{OD}{UOD}$$

dove UOP è il numero di operatori distinti, OD è il numero totale di operandi e UOD è il numero di operandi distinti.

3.2.6.0.2 M[PROC][0011] - Halstead Volume per-function

Indica la dimensione dell'implementazione di un algoritmo basandosi sul numero di operazioni eseguite e sugli operandi di una funzione.

$$VOL = (OP+OD) * \log_2(UOP + UOD)$$

dove OP è il numero totale di operatori, OD è il numero totale di operandi, UOP è il numero di operatori distinti e UOD è il numero di operandi distinti.

3.2.6.0.3 M[PROC][0012] - Halstead Effort per-function

Rappresenta il costo necessario per scrivere il codice di una funzione.

$$E = DIF * VOL$$

dove DIF indica l'Halstead Difficulty e VOL è l'Halstead Volume.

3.2.7 Metriche per la qualità di prodotto

Di seguito sono spiegate nel dettaglio le metriche utilizzate per valutare la qualità di prodotto.

3.2.7.1 Metriche per i documenti

3.2.7.1.1 M[PROD][D][0001]: Indice di *Gulpease*_{|g|}

È un indice di leggibilità per la lingua italiana, con il vantaggio di contare le singole lettere e non le sillabe per definire la lunghezza delle parole. L'indice prende in considerazione la lunghezza delle parole e la lunghezza delle frasi rispetto al numero totale delle lettere. Come descritto nella seguente formula, restituisce poi un valore

che indica l'indice di difficoltà della leggibilità del testo.

$$89 + \frac{300 * (\text{numero delle frasi}) - 10 * (\text{numero delle lettere})}{\text{numero delle parole}}$$

Risultati:

- Minore di 80: difficile da leggere per chi ha licenza elementare;
- Minore di 60: difficile da leggere per chi ha licenza media;
- Minore di 40: difficile da leggere per chi ha un diploma superiore.

Ai fini di rendere il testo il più comprensibile possibile, si è deciso di porre importanza anche ad un'analisi diretta da parte di una persona, in quanto l'indice di leggibilità fornito dalla formula sopra riportata non garantisce un buon testo sotto tutti gli aspetti ritenuti importanti. Uno dei motivi che ha spinto il gruppo a questa scelta è dato dal fatto che un risultato considerato non buono secondo l'indice di Gulpease, potrebbe essere conseguenza di un testo con frasi complesse utilizzate per non ricadere in contenuti poco formali; inoltre il risultato non dà informazioni sulla logica del contenuto.

Per calcolare l'indice di Gulpease il gruppo ha tenuto conto solo dei contenuti testuali informativi, tralasciando contenuti come indice, intestazione e tabelle.

3.2.7.1.2 M[PROD][D][0002]: Errori ortografici

*Overleaf*_[g], l'editor utilizzato per redigere la documentazione, fa uso di un correttore ortografico per mettere in evidenza gli errori di ortografia. Sarà compito del Verificatore correggerli attraverso l'uso di tale strumento.

3.2.7.2 Metriche per il Software

3.2.7.2.1 M[PROD][S][0001]: Copertura requisiti obbligatori

Una volta eseguita l'analisi dei requisiti, essi vengono organizzati in unità di dimensioni gestibili e quelli obbligatori messi in evidenza. Tale metrica indica la percentuale di requisiti obbligatori implementati fino a un dato momento. La formula che la misura è dunque:

$$CRo = \frac{NRoS}{NRoT} * 100$$

dove NRoS è il numero dei requisiti obbligatori soddisfatti e NRoT è il numero dei requisiti obbligatori totali

3.2.7.2.2 M[PROD][S][0002]: Copertura requisiti accettati

Una volta eseguita l'analisi dei requisiti, essi vengono organizzati in unità di dimensioni gestibili e una parte di essi viene identificata come accettati ma non obbligatori. Tale metrica indica la percentuale di implementazione di questi requisiti fino a un

dato momento. La formula che la misura è dunque:

$$CRa = \frac{NRaS}{NRaT} * 100$$

dove NRaS è il numero dei requisiti accettati che sono stati soddisfatti e NRaT è il numero totale dei requisiti accettati.

3.2.7.2.3 M[PROD][S][0003]: Accuratezza rispetto alle attese

In fase di testing tale metrica misura la percentuale dei test che hanno restituito i risultati attesi. La formula è la seguente:

$$AC = (1 - \frac{NTf}{NTt}) * 100$$

dove NTf è il numero di test case falliti e NTt è il numero di test case totali.

3.2.7.2.4 M[PROD][S][0004]: Densità di *failure*_{|g|}

Rappresenta un dato opposto a quello dell'Accuratezza rispetto alle attese. Indica infatti la percentuale di test conclusi con un esito di failure, quindi negativo, rispetto al totale di test eseguiti.

$$DF = (\frac{Nfr}{Nte}) * 100$$

dove Nfr indica il numero di failure rilevati e Nte il numero di test case eseguiti.

3.2.7.2.5 M[PROD][S][0005]: Blocco di operazioni non corrette

È un valore espresso in percentuale che indica il livello di funzionalità con cui si gestiscono correttamente i fault.

$$BNC = (\frac{Nfe}{Non}) * 100$$

Nfe: numero di failure evitati durante i test; Non: numero di test case eseguiti che prevedono l'esecuzione di operazioni non corrette.

3.2.7.2.6 M[PROD][S][0006]: Tempo di risposta

Consiste nel tempo che intercorre tra la domanda al software per una precisa funzionalità e il risultato restituito all'utente.

$$TR = \frac{\sum_{k=1}^N Tk}{n}$$

dove a partire dalla richiesta di n funzionalità, Tk è il tempo trascorso tra una richiesta k per una determinata funzionalità e il termine delle operazioni necessarie a dare seguito a tale richiesta.

3.2.7.2.7 M[PROD][S][0007]: Comprensibilità delle funzioni offerte

Consiste nella percentuale di operazioni comprese nell'immediato senza il bisogno di consultare il manuale.

$$CFC = \left(\frac{Nfc}{Nfo} \right) * 100$$

dove Nfc è il numero di funzionalità comprese dall'utente e Nfo è il numero di funzionalità offerte ad esso offerte.

3.2.7.2.8 M[PROD][S][0008]: Facilità di apprendimento delle funzionalità

Indica il tempo, misurato in minuti, utilizzato dall'utente per apprendere nello specifico una determinata funzionalità e il modo di utilizzarla correttamente.

3.2.7.2.9 M[PROD][S][0009]: Utilizzo effettivo delle funzionalità

Percentuale che indica il livello di apprezzamento dell'utente rispetto alle funzionalità offerte dall'applicazione.

$$PA = \left(\frac{Nfu}{Nfo} \right) * 100$$

dove Nfu è il numero funzionalità utilizzate dall'utente e Nfo è il numero funzionalità offerte dall'app.

3.2.7.2.10 M[PROD][S][0010]: Capacità di analisi di failure

Rapporto, espresso in percentuale, delle failure di cui si conosce la causa rispetto al numero totale di failure attualmente rilevate.

$$CF = \left(\frac{Nfi}{Nft} \right) * 100$$

dove Nfi è il numero di failure di cui si è identificata la causa e Nft è il numero di failure totali trovate.

3.2.7.2.11 M[PROD][S][0011]: Impatto delle modifiche

Dato il numero di failure risolte, questo indice calcola, in percentuale, il rapporto tra quante di esse hanno generato nuove failure in seguito alla loro risoluzione e il loro numero totale.

$$IM = \left(\frac{Nfg}{Nfr} \right) * 100$$

dove Nfg è il numero di failure generanti (che generano altre failure) e Nfr è il totale di failure risolte.

3.2.7.2.12 M[PROD][S][0012]: Rapporto linee di commento su linee di codice

Si considera di grande importanza la presenza di commenti che descrivono il codice, in quanto aumentano la leggibilità e l'intelligibilità dello stesso e ne favoriscono la manutenzione, soprattutto in previsione di interventi di più persone.

$$RLC = \left(\frac{Nlcom}{Nlcod} \right) * 100$$

dove Nlcom è il numero di linee di commenti e Nlcod è il numero di linee di codice

3.2.7.2.13 M[PROD][S][0013]: Versioni di Android supportate

È considerato prioritario il supporto delle versioni più recenti del sistema operativo Android, tuttavia è considerato migliorativo un numero maggiore di versioni supportate dall'applicazione.

3.2.7.2.14 M[PROD][S][0014]: Copertura del framework *Octalysis*_{|g|}

Il prodotto include funzionalità che seguono i principi del framework di *gamification*_{|g|} Octalysis.

Per misurare la distribuzione di tali funzionalità vengono contate per ogni *Core Drive*_{|g|} quante appartengono ad ognuno di essi, e rapportati i valori su una scala da 0 a 10. Per la visualizzazione grafica della distribuzione delle funzionalità viene utilizzato il software online Octalysis Tool.

3.3 Processo di configurazione**3.3.1 Scopo**

Nella presente sezione sono descritte le direttive riguardanti la configurazione degli strumenti di condivisione attraverso i quali sviluppare il materiale da consegnare.

3.3.2 Aspettative

Il gruppo si attende alle seguenti norme al fine di condividere tra i vari membri del team il materiale da consegnare in modo ordinato e tracciabile.

3.3.3 Attività**3.3.3.1 Versionamento****3.3.3.1.1 Descrizione**

Si è scelto l'utilizzo della tecnologia *Git*_{|g|}, usando il servizio di GitHub per i seguenti contenuti:

- Parti versionabili;
- Documenti formali;
- Documenti informali.

Onde evitare confusione, per ogni tipo di contenuto è stato creato un *repository*_{|g|} dedicato, come meglio indicato nella seguente sezione §3.3.3.2.

3.3.3.1.2 Struttura delle repository

Il repository principale, denominato con il nome del progetto "P2PCS", è stato ampliato.

La cartella è stata suddivisa in due sottocartelle:

- Documentazione;
- Codice.

All'interno della cartella "Documentazione" sono state create quattro sottocartelle:

- RR;
- RP;
- RQ;
- RA.

All'interno di ognuna di queste cartelle sono presenti due sottocartelle: formali e non formali, la cui definizione è espressa in §3.1.3. All'interno la cartella "formali" vi sono tutti i documenti definitivi in formato .pdf, e presenta la seguente struttura:

- **Documenti interni**

- Studio Di Fattibilità;
- Norme Di Progetto;
- Verbali interni.

- **Documenti esterni**

- Piano di Progetto;
- Piano di Qualifica;
- Analisi dei Requisiti
- Glossario;
- User Manual;
- Developer Manual;
- Verbali esterni.

Nella cartella non formali, invece, sono presenti i file in formato .pdf non ancora conclusi. I sorgenti di tali file (formato .tex) sono salvati su un account condiviso da tutti i membri del gruppo sulla piattaforma *Overleaf*_[g], che ne permette anche la loro redazione.

3.3.3.1.3 Branch

In fase di RR si è deciso di non fare uso dei *branch*_[g] in quanto per lo sviluppo della documentazione da presentare e di contenere tutto nel master branch.

In fase di RP si è deciso di creare un branch per ogni membro del gruppo, al fine di gestire del codice in parallelo in uno stesso documento o sorgente per il codice.

Inoltre il singolo componente, se ne ha necessità, potrà creare ulteriori branch notificando la decisione al resto dei componenti del gruppo.

In fase di RQ si è deciso di utilizzare la stessa struttura dei branch creati in fase di RP, aggiungendo un branch particolare per il versionamento dei documenti.

3.3.3.1.4 Aggiornamento della repository

Ivi sono descritti i comandi base per il corretto utilizzo dello strumento Git, al fine di aggiornare correttamente i contenuti della relativa repository:

- git status: Mostra lo stato attuale del repository locale con i file modificati, tracciati e non tracciati da Git;
- git pull: aggiorna sovrascrivendo il repository locale con quello remoto;
- Nel caso in cui si verifichino dei conflitti:
 - git stash: per accantonare momentaneamente le modifiche apportate.
 - git pull: per ottenere ed applicare i *commit*_[g] mancanti.
 - git stash apply per ripristinare le modifiche.
- git add [files]: aggiungerà i file modificati e quelli nuovi specificati;
- git rm [files]: rimuove un file in modo che eseguendo un commit delle modifiche venga poi rimosso anche dal repository;
- git commit: successivamente richiede di riassumere le modifiche effettuate, in caso sia utile si può aggiungere un messaggio esteso di descrizione (aggiungendo al comando la dicitura -m "messaggio").
- git push: per completare l'operazione e fornire le modifiche agli altri membri del gruppo;
- git branch: permette di visualizzare dei branch presenti nel repository locale;
- git checkout: permette di cambiare il branch attivo nel repository locale, tutti i comandi eseguiti hanno effetto sul branch attivo;
- git fetch: scarica la versione aggiornata dal repository ma non esegue automaticamente il merge;

- **git merge:** effettua l'unione del repository remoto scaricato con il comando `fetch` con il repository locale;
- **git push:** aggiorna sovrascrivendo il repository remoto con quello locale.

3.3.3.2 Strumenti

- **Client git:** Secondo le preferenze dei membri del gruppo vengono usati principalmente due client *GitKraken*_[g] e *Git Bash*_[g].
- **Server git:** Vista la conoscenza preliminare e l'affidabilità di *Github*_[g] si è scelto questo server.

3.4 Processo di verifica

3.4.1 Scopo

Attraverso il processo di verifica si vuole evidenziare ed eliminare la presenza di errori nell'esecuzione degli altri processi per l'intero sviluppo del prodotto, in particolare alla conclusione di ogni incremento. Questa sezione norma gli strumenti e i metodi che verranno usati per la verifica del codice e dei documenti.

Il processo di verifica riguarda esclusivamente la produzione di documenti per quanto riguarda la fase antecedente la RR, mentre per quanto riguarda le fasi successive verterà anche sulla correzione del prodotto software consegnato.

3.4.2 Aspettative

Il processo ha lo scopo di ottenere i seguenti effetti:

- Ottenere un sistema di controllo degli errori non invasivo, efficiente ed efficace;
- Stabilire i criteri necessari per la verifica del prodotto;
- Sottolineare le problematiche trovate col fine di risolverle.

3.4.3 Descrizione

Il processo si divide nei seguenti momenti:

- **Controllo:** Attraverso le tecniche di analisi statica e analisi dinamica (descritte nelle sezioni successive) si riesce ad effettuare un controllo approfondito del codice sorgente e della sua corretta esecuzione;
- **Test:** Esecuzione test sul software.

3.4.4 Attività

3.4.4.1 Analisi statica

L'analisi statica è una tecnica di analisi applicabile sia alla documentazione che al codice e permette di effettuare la verifica di quanto prodotto individuando errori ed anomalie.

Vi sono due metodologie per effettuarla:

- **Walkthrough:** tecnica applicata quando non si sanno le tipologie di errori o problemi che si stanno cercando e quindi prevede una lettura da cima a fondo del codice o documento per trovare anomalie di qualsiasi tipo. Questo metodo risulta essere il più dispendioso in termini di efficienza ma si tratta anche del più semplice da imparare, diventando inevitabilmente il più utilizzato nelle prime fasi del progetto.
- **Inspection:** tecnica da applicare quando si ha idea delle possibili problematiche che si stanno cercando e si attua leggendo in modo mirato il documento o il codice. Questo procedimento risulta molto più veloce del precedente in quanto, utilizzando la lista di controllo degli errori e dalle misurazioni effettuate, permette un'analisi più efficace dei punti critici, tralasciando invece le parti senza problematiche.

3.4.4.2 Analisi dinamica

L'analisi dinamica consiste nella realizzazione ed esecuzione di una serie di test di vario tipo sul codice del software. Questa tecnica non è applicabile per trovare errori nella documentazione; pertanto tale tecnica è stata utilizzata dai membri del team solo nelle fasi successive alla RR.

Ivi sono presenti le norme per quanto riguarda i test che verranno utilizzati dai membri del team in fasi successive alla RR fino alla consegna del prodotto

3.4.4.2.1 Test

Questa sezione riguarda i test che verranno eseguiti durante le fasi di RP, RQ, RA. I test devono essere ripetibili, per questo per ognuno di essi andrà specificato, attraverso dei commenti in lingua inglese sul codice, l'ambiente di esecuzione, le attese e l'analisi dei risultati. Durante queste le fasi di RP, RQ e RA i membri del team, al fine di rilevare eventuali errori e risolverli, sono tenuti a eseguire i seguenti test (in ordine cronologico) sul prodotto consegnato:

- Test di unità: attività di testing sulla singola *unità_{|g|}* software, permettendo l'individuazione di eventuali bug e semplificare la futura integrazione con le altre parti del prodotto. Dal momento che questo tipo di test si concentra sulla singola componente software, non è in grado di individuare eventuali errori di integrazione tra le componenti. Il testing di questo tipologia sarà automatizzato, dal momento che questo approccio consente di isolare efficacemente una componente al fine di valutarne la correttezza ;

- **Test di integrazione:** test che verificano l'integrazione tra le varie componenti. Questo tipo di testing è necessario dal momento che sopperisce alle mancanze prima elencate dei test di unità. Dal momento che verifica l'integrazione tra le parti del sistema, il testing ha senso solo se preceduto dal testing sulle singole componenti (quindi test di unità).

3.4.5 Metriche

3.4.5.1 M[PROC][0008] - Percentuale di codice coperto da test

Indica il rapporto tra linee di codice per le quali è previsto un test di verifica su linee di codice totale in percentuale.

$$PL_{cc} = \left(\frac{L_{cc}}{L_{ct}}\right) * 100$$

dove L_{cc} indica linee codice coperte e L_{ct} le linee totali di codice.

3.4.5.2 M[PROC][0009] - Percentuale di test passati

Indica il rapporto tra i test passati e i test totali nella fase di sviluppo.

$$PT_p = \left(\frac{T_p}{T_t}\right) * 100$$

dove T_p indica i test passati e T_t i test totali.

3.4.5.3 M[PROC][0010] - Percentuale di test non passati

Indica il rapporto tra i test non passati e i test totali nella fase di sviluppo

$$PT_{np} = \left(\frac{T_{np}}{T_t}\right) * 100$$

dove T_{np} indica i test non passati mentre T_t sono i test totali.

3.4.6 Strumenti

- **Documenti:** Per quanto riguarda la verifica ortografica ci si affida allo strumento integrato di Overleaf, attraverso il quale le parole sbagliate vengono segnate in rosso, permettendo un controllo rapido ed efficace.
- **Indice Gulpease:** Per il calcolo dell'Indice di Gulpease il team ha utilizzato lo strumento disponibile online chiamato *Farfalla Project*_[g];
- **Octalysis Tool:** Per una valutazione della qualità degli aspetti relativi alla *Gamification*_[g] si è scelto di utilizzare lo strumento online Octalysis Tool disponibile sul sito del creatore del framework *Octalysis*_[g] Yu-kai Chou.
- **Gestione processi e feedback:** Anche per la verifica si è stato scelto di utilizzare il sistema di *issues*_[g] integrato in GitHub.

3.5 Processo di validazione

3.5.1 Scopo

Il processo ha l'obiettivo di verificare che il prodotto sia conforme a quanto pianificato e sia abile nel gestire e minimizzare gli effetti degli errori.

3.5.2 Aspettative

In questa sezione il gruppo intende sviluppare direttive utili per eseguire correttamente il processo di validazione.

3.5.3 Attività

I passi per compiere l'attività di validazione sono i seguenti:

- **Esecuzione di test di sistema:** tipo di testing che verrà eseguito in fase di RQ, segue il testing di unità e di integrazione. Questo tipo di testing ha lo scopo di accertare la copertura dei requisiti individuati;
- **Tracciamento:** Tracciamento dei test con rendiconto eseguito dai verificatori;
- **Analisi risultati:** In seguito a seguire il Responsabile di progetto controlla e analizza i report ricevuti sui test decidendo se:
 - Accettare la validazione e consegnare i risultati al proponente, informandolo sulle modalità di esecuzione della validazione;
 - Chiedere la ripetizione di alcuni o tutti i test con ulteriori nuove indicazioni.

4 Processi organizzativi

4.1 Processo di coordinamento

4.1.1 Scopo

In questa sezione vengono elencate direttive, linee guida e strumenti che il gruppo dovrà utilizzare a fini organizzativi all'interno dello svolgimento del lavoro.

4.1.2 Aspettative

Il gruppo si attende alle seguenti direttive al fine di organizzare, evitando incomprensioni di varia natura, gli incontri tra i componenti.

4.1.3 Comunicazione

In questa sezione vengono illustrati i metodi di comunicazione sia tra i membri all'interno del gruppo Cyber13 sia tra il gruppo e le entità esterne, come Committenti e Proponenti.

4.1.3.1 Comunicazioni interne

Le comunicazioni interne al gruppo avvengono tramite *Slack_{|g|}*, strumento di messaggistica nel quale sono stati predisposti dei canali tematici, suddivisi per argomento:

- **general:** Canale di carattere generale, per coordinare luoghi e orari di ritrovo, scambiare opinioni sul progetto e in generale inviare comunicati che non trovano spazio in alcuno degli altri canali specifici.
- **documentirr:** Canale predisposto per la coordinazione e l'aggiornamento dei lavori del team su tutti i documenti da approntare in vista della RR di Aprile 2019 (si procederà allo stesso modo anche per le fasi successive);
- **backend:** Canale predisposto per la coordinazione e l'aggiornamento dei lavori del team su l'avanzamento del lavoro sul backend;
- **frontend:** Canale predisposto per la coordinazione e l'aggiornamento dei lavori del team su l'avanzamento del lavoro sul frontend.

4.1.3.2 Comunicazioni esterne

Alcune norme regolano anche le comunicazioni con le parti esterne al gruppo Cyber13, nello specifico:

- La proponente GaiaGo rappresentata da Filippo De Pretto, con la collaborazione del quale si definiranno i *requisiti_{|g|}* che porteranno alla realizzazione del prodotto;

- I committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin, ai quali verrà fornita la documentazione richiesta in ciascuna revisione di progetto.

Comunicazioni esterne scritte

Le comunicazioni esterne scritte vengono effettuate utilizzando l'indirizzo e-mail del gruppo swe.cyber13@gmail.com a cui tutti i membri hanno accesso. Per comunicare con il Proponente viene utilizzato l'indirizzo e-mail f.depreto@gaiago.com. Anche le comunicazioni con i committenti avvengono esclusivamente tramite l'indirizzo e-mail del gruppo: ogni messaggio deve essere definito da un oggetto chiaro e conciso, così come deve essere il suo contenuto, e rivolgersi ai committenti in modo formale e cortese.

Videochiamate remote

Le videocchiamate remote si svolgeranno in sede di Technology Baseline e Product Baseline con il committente Riccardo Cardin, e con la proponente (rappresentata da Filippo De Pretto) ogni qualvolta che il team lo ritenga necessario. Durante le videocchiamate con il committente Riccardo Cardin, tutto il gruppo deve essere tassativamente presente (anche se non necessariamente tutti i componenti prendono parte attivamente alla discussione), mentre per quanto riguarda le videocchiamate con la proponente il numero minimo di componenti del gruppo è cinque (come meglio descritto nella sezione 4.1.4.2). Inoltre ogni membro del gruppo, qualora lo ritenga necessario, può richiedere un colloquio personale con la proponente al fine di chiarire dubbi in merito a uno specifico ambito. A termine della chiamata il membro interessato dovrà notificare l'esito della chiamata ai restanti membri del team.

4.1.4 Riunioni

Le riunioni interne ed esterne al team Cyber13 sono un momento fondamentale per il corretto avanzamento e conclusione del progetto. Alcune regole ne normano quindi l'organizzazione e lo svolgimento.

Per ogni riunione viene nominato un Segretario tra i membri del gruppo, il quale garantirà il rispetto dell'ordine del giorno, si occuperà di annotare i punti più importanti discussi e da tali appunti redigerà infine un Verbale di Riunione. Quest'ultimo non è un ruolo fisso, quindi il membro del team che ne ricoprirà la carica potrà variare da riunione a riunione se necessario.

Le riunioni che non potranno avere un luogo fisico di ritrovo avverranno in remoto attraverso il programma *Skype*_[g] o lo strumento *Google Meet*_[g], secondo quanto concordato preventivamente prima dell'appuntamento.

4.1.4.1 Riunioni interne

La partecipazione è ammessa ai soli membri del team Cyber13.

Ordine del giorno, data, ora e luogo della riunione vengono decisi dal Responsabile di progetto, che si occupa anche di approvare il Verbale di Riunione stilato dal Segretario.

I membri del team hanno il dovere di presentarsi in orario all'appuntamento, comunicando eventuali ritardi e partecipando attivamente e costruttivamente alla discus-

sione.

La riunione è considerata valida se almeno 5 dei 6 membri del team sono presenti. In circostanze eccezionali qualora un membro fosse impossibilitato ad essere presente di persona ma la sua presenza dovesse essere fondamentale per la discussione dell'ordine del giorno, esso potrà prendere parte alla riunione da remoto attraverso uno strumento di videochiamata come Skype o Google Meet.

4.1.4.2 Riunioni esterne

Prendono parte a queste riunioni i membri del team Cyber13 e la Proponente.

Essendo la sede di GaiaGo a Milano, le riunioni esterne avranno il più delle volte luogo da remoto, attraverso uno strumento quale Skype o Google Meet, deciso di volta in volta prima dell'appuntamento.

Le riunioni esterne cui la Proponente prenderà parte di persona saranno invece effettuate presso la Torre Tullio Levi Civita (ex Torre Archimede), previa disponibilità dei locali.

Come per le riunioni interne, il Verbale di Riunione sarà redatto da un Segretario eletto di volta in volta e sarà approvato in seguito dal Responsabile di Progetto.

Data l'elevata importanza delle riunioni esterne, esse vengono considerate valide anche alla presenza di un singolo membro del team Cyber13 e della Proponente.

4.1.4.3 Verbale di riunione

Il Segretario eletto di volta in volta si occupa di stilarlo secondo il seguente schema:

- **Verbale TIPO del DATA:** Il frontespizio del Verbale di Riunione permette di identificarne il TIPO (Interno/Esterno) e la DATA in cui la riunione ha avuto luogo.
- **Diario delle modifiche:** Come definito nella sezione 3.1.6.1.
- **Indice:** Semplice indice dei contenuti del Verbale.
- **Informazioni sulla riunione:** Deve contenere obbligatoriamente le seguenti informazioni:
 - **Luogo e data della riunione:** ad esempio Padova, 15 Marzo 2019;
 - **Partecipanti alla riunione:** a partire dai Proponenti/Committenti in caso di Riunione Esterna, seguiti dai nomi dei membri del team Cyber13 presenti elencati in ordine alfabetico.

Possono invece considerarsi informazioni facoltative ma utili:

- **Ora di inizio e di fine:** scritte nel formato 24h;
- **Motivo della riunione:** Illustra in forma riassuntiva i motivi generali della riunione;
- **Ordine del giorno:** Indica attraverso un elenco numerato gli argomenti da discutere così come definiti dal Responsabile di Progetto;
- **Resoconto:** riassunto di quanto discusso in riunione, redatto dal Segretario. Indicherà chiaramente quali punti dell'Ordine del Giorno sono stati effettiva-

mente affrontati, eventuali argomenti discussi che non fossero presenti nell'Ordine del Giorno e le relative decisioni prese.

Il tracciamento delle decisioni prese avverrà attraverso un sistema a codici del tipo VER-DATA.X dove VER significa Verbale, DATA è sostituito con la data della riunione e X è un numero sequenziale che identificherà univocamente la decisione cui si sta facendo riferimento.

Nomenclatura e archiviazione dei verbali: I Verbali di Riunione saranno archiviati con la nomenclatura VER-DATA dove DATA è la data della riunione nel formato YYYY-MM-DD per permetterne un posizionamento semplice e cronologico all'interno della cartella assegnata.

Trattandosi di documenti esterni e ufficiali, verranno archiviati nella cartella di Repository dedicata e saranno redatti in LaTeX.

4.2 Processo di pianificazione

4.2.1 Scopo

In questa sezione vengono elencate le norme relative al processo di pianificazione interno al gruppo.

4.2.2 Aspettative

Il gruppo si aspetta di organizzare in maniera ottimale ed evitando incomprensioni di varia natura la divisione del lavoro da svolgere. Per fare vengono stabiliti ruoli precisi all'interno del progetto.

4.2.3 Ruoli di progetto

Nel corso del progetto ogni componente del team avrà modo di ricoprire a rotazione il ruolo di ognuna delle figure aziendali tipicamente coinvolte nella realizzazione di un software.

Tale assegnazione cercherà di distribuire nel modo più omogeneo possibile il tempo che ciascun membro investirà in ogni ruolo, dando in ogni caso priorità alla continuità delle attività già in corso.

Il Responsabile di Progetto di turno dovrà inoltre garantire che non si creino situazioni di conflitto di interesse che potrebbero compromettere la qualità del prodotto finale.

4.2.3.1 Responsabile di progetto

Il "Responsabile di progetto", o "Project Manager", accentra le responsabilità di scelta e approvazione del gruppo, del quale è anche il rappresentante presso proponenti e committenti.

Si occupa quindi di gestire le risorse umane e di coordinare i membri del team, dell'approvazione dei documenti, della pianificazione e delle relazioni esterne.

4.2.3.2 Amministratore

L'amministratore è una figura di supporto che si occupa di fornire al team gli strumenti per lavorare in maniera rigorosa e regolamentata.

Tali strumenti consistono nella redazione e attuazione di piani e procedure di Gestione della Qualità, la redazione delle Norme di Progetto, il versionamento e la configurazione dei prodotti.

Pur non essendo sotto sua diretta responsabilità, collabora alla redazione del Piano di Progetto e si assicura che la documentazione sia corretta e verificata.

4.2.3.3 Analista

L'analista si occupa di analizzare e capire appieno il dominio del problema per identificarne i requisiti espliciti ed impliciti, e per riconoscere ed evitare gravi problemi di progettazione successivi.

Essendo colui che all'interno del team meglio conosce il problema e i requisiti che la sua realizzazione richiede, esso è coinvolto anche nella definizione degli accordi contrattuali, nella verifica delle implicazioni di costo e qualità, nella realizzazione dello Studio di Fattibilità e dell'Analisi dei Requisiti.

4.2.3.4 Progettista

Il progettista si occupa della definizione di una soluzione soddisfacente per tutti gli *stakeholder*_[g], perseguendo efficienza ed efficacia nella soddisfazione dei requisiti a partire dal lavoro dell'Analista.

In tal senso il progettista conosce e applica soluzioni di lavoro note e ottimizzate (*Best practice*_[g]), e si occupa di organizzare e suddividere il sistema in parti di complessità trattabile, per rendere il lavoro di codifica realizzabile e verificabile.

4.2.3.5 Programmatore

Il programmatore si occupa della codifica del progetto, realizzando quindi il prodotto finale attraverso l'implementazione dell'architettura definita dal Progettista.

Il codice prodotto deve essere documentato, versionato e mantenibile secondo le norme fissate.

Si occupa anche della scrittura del Manuale Utente e di realizzare le componenti necessarie alla verifica e alla validazione del codice.

4.2.3.6 Verificatore

Il verificatore è coinvolto in ogni fase del progetto. Il suo compito è appunto quello di verificare la conformità dei prodotti ai requisiti fissati di funzionalità e qualità.

Il suo coinvolgimento è richiesto in ogni fase poiché deve accertarsi che ad ogni attività di processo eseguita non siano stati introdotti errori.

Attraverso la redazione di un Piano di Qualifica si occupa di illustrare le verifiche effettuate sul progetto e i relativi esiti, secondo quanto previsto dal Piano di Progetto.

4.2.3.7 Rotazione dei ruoli

Per assicurare un buon esito del progetto e allo stesso tempo assicurare una rotazione dei ruoli assegnati a ciascun membro del gruppo durante la sua realizzazione, sono necessarie alcune semplici regole:

- Considerare impegni e interessi di ogni singolo membro del gruppo;
- Evitare conflitti di interesse, in particolare evitare che un ruolo assegnato ad un membro del gruppo comporti la verifica di lavoro svolto in precedenza dallo stesso mentre ricopriva un altro ruolo all'interno della stessa fase dello sviluppo.
- Garantire che al cambio di ruolo il nuovo assegnatario venga posto nelle migliori condizioni di lavoro possibile. In tal senso il precedente assegnatario deve assicurarsi di non lasciare al nuovo criticità da lui non gestibili. Inoltre si occuperà di lasciargli al nuovo una lista di consigli e procedure utili in un documento interno informale.
- Garantire che ciascun membro ricoprirà di volta in volta il ruolo a lui assegnato in modo esclusivo: ciascuno dovrà occuparsi di quanto è di sua competenza, senza interferire o incrociare incarichi e responsabilità.

4.2.4 Ticketing

4.2.4.1 Task list

Lo svolgimento del progetto prevede la suddivisione del modello di sviluppo in varie fasi. Le attività da svolgere in una fase sono contenute in una *task board*_[g]. Il responsabile ha il compito di creare le task board per ogni fase del modello su *GitHub*_[g], utilizzando il nome "X", dove X identifica in modo significativo la sezione di progetto cui si riferisce. Per esempio, per quanto riguarda i documenti, X segue la nomenclatura indicata nella sezione 3.1.4.1.

4.2.4.2 Task

Ogni singola *task*_[g] viene creata o approvata dal Responsabile di progetto ed è caratterizzata da un titolo significativo per la componente di progetto cui si riferisce.

4.2.4.3 Ticket

Il *ticket*_[g] sono lo strumento attraverso cui viene gestito il ciclo di vita del task, dalla sua definizione, alla sua assegnazione ad uno o più membri del gruppo, alla sua implementazione, verifica, approvazione e chiusura. L'assegnazione del *ticket*_[g] ad uno specifico membro del gruppo può avvenire secondo due modalità:

- **Direttamente:** L'assegnazione del task è già noto al momento della creazione dello stesso. In questo caso il $ticket_{|g|}$ viene assegnato direttamente allo specifico componente del gruppo da parte del Responsabile di progetto.
- **Indirettamente:** Nel caso in cui un membro non è presente agli incontri oppure il lavoro che richiede una $task_{|g|}$ è impegnativo, può non essere possibile assegnare direttamente un ticket ad un componente del gruppo. In questo caso il membro non presente è obbligato a controllare la sezione $issues_{|g|}$ presente su $GitHub_{|g|}$ dove troverà il lavoro da svolgere. Questi tasks inoltre possono essere assegnati qualora il componente finisse i $ticket_{|g|}$ a suo carico.

4.2.5 Metriche

4.2.5.1 M[PROC][0011] - Schedule Variance

Indica a che livello di avanzamento del progetto rispetto alla pianificazione delle attività.

$$SV = BCWP - BCWS$$

dove BCWP indica le attività svolte e BCWS le attività che dovrebbero essere state svolte finora.

4.2.5.2 M[PROC][0012] - Budget Variance

Indica se attualmente si è speso meno o più di quanto previsto.

$$BV = BCWS - ACWP$$

dove BCWS indica il costo pianificato delle attività svolte ad una certa data e ACWP indica il costo effettivo delle attività svolte a tale data.

4.2.5.3 M[PROC][0013] - Rischi non individuati

Indice del numero di rischi non individuati nella fase di analisi: è indicato con un numero intero incrementato partendo da zero per ogni rischio rilevato che non fosse stato individuato precedentemente in fase di analisi dei rischi. Viene resettato all'inizio di ogni fase di progetto.

4.3 Processo di formazione

La formazione del personale è da svolgersi autonomamente. I membri del team Cyber13 sono tenuti a studiare individualmente le tecnologie che verranno utilizzate durante lo svolgimento del progetto.

La documentazione di riferimento, oltre al materiale precedentemente citato nella sezione 1.4.2 Riferimenti informativi, comprende:

- Documentazione LaTeX:
<https://www.latex-project.org>
- Documentazione GitHub:
<https://github.com>
- Documentazione UML:
https://www.math.unipd.it/~tullio/IS-2/Guida_Notazioni_UML-14.pdf
- Documentazione sviluppo Android:
<https://developer.android.com/docs>
- Documentazione Kotlin:
<https://kotlinlang.org/docs/reference/>
- Documentazione PHP:
<https://www.php.net/docs.php>

Inoltre ogni membro del team, qualora lo ritenga necessario, potrà formarsi attraverso canali non ufficiali che siano in grado di risolvere eventuali dubbi in merito a uno specifico problema (per esempio canali sulla piattaforma YouTube, libri o articoli di varia natura presenti in Internet).

4.4 Procedure

4.4.1 Creazione e gestione dei task

Per la creazione di un task si procede nel seguente modo:

- Accesso alla task list relativa al progetto di interesse identificata con il nome dello stesso.
- Creazione di un nuovo task.
- Il Responsabile di Progetto può decidere di suddividerlo in sotto-task da assegnare a membri del gruppo differenti.

4.4.2 Gestione dei ticket

La vita di ogni ticket viene presentata nel seguente diagramma:

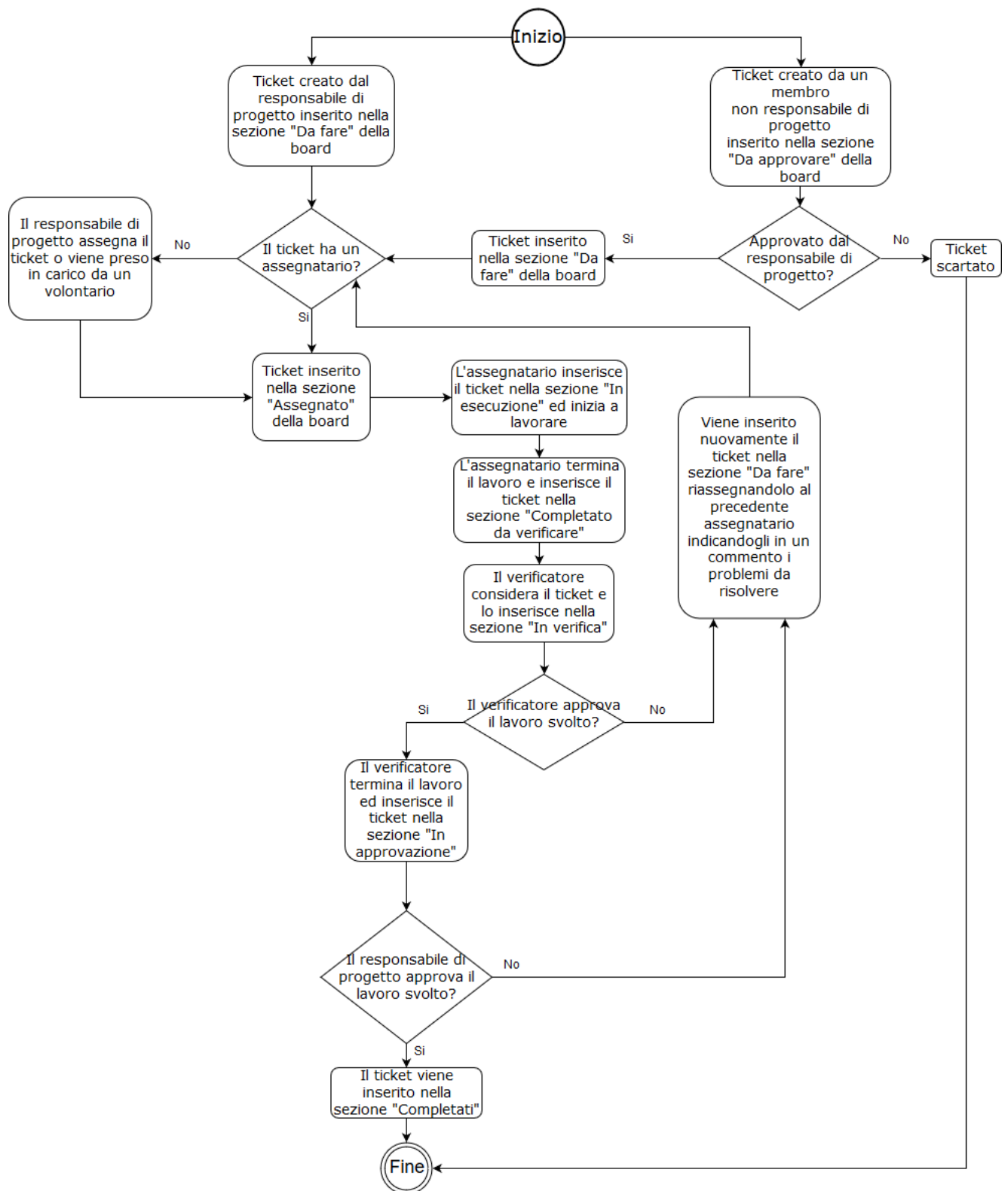


Figura 49: Diagramma del ciclo di vita di un ticket

4.4.3 Stesura del consuntivo

L'operazione di stesura del consuntivo viene eseguita dal Responsabile di Progetto nel seguente modo:

- Esporta da *Gantt project*_[g] un file in formato *Microsoft Excel*_[g] che indica le ore rendicontate nella fase corrente;
- Accede al *Foglio di Google*_[g] presente nel *Google Drive*_[g] associata alla mail del gruppo con relativo template e ne fa una duplicazione;
- Inserisce le ore rendicontate nelle apposite celle ottenendo la differenza con quelle preventivate;
- Riporta i valori ottenuti nel Piano di Progetto;
- Crea una tabella che, elaborando i dati derivanti dalla differenza tra le ore rendicontate e quelle preventivate, trova il budget effettivo rispetto a quello previsto;
- Infine con l'elaborazione di questi dati crea una valutazione complessiva del lavoro.

Per il periodo che precede la revisione RR ci si ferma al punto 3 senza ricavarne la differenza in quanto le ore preventivate non sono presenti. Successivamente ad ogni revisione sarà aggiornato il Piano di Progetto con i dati ottenuti.

4.5 Strumenti

4.5.1 Pianificazione

Per la pianificazione la scelta è ricaduta sulla piattaforma GitHub perchè offre la possibilità di organizzare le task di progetto in ticket, da assegnare a uno o più membri ed infine organizzarli in colonne. A tal proposito il gruppo ha deciso di creare varie colonne una per ogni fase del ciclo di vita del ticket in modo da avere un'idea chiara sul punto in cui sono i vari membri del gruppo con i relativi compiti.

4.5.2 Comunicazione

Per la comunicazione ci si è orientati per l'applicazione di messaggistica *Slack*_[g] appositamente per gruppi di lavoro, che offre la possibilità di realizzare canali telematici, in modo da creare comunicazioni specifiche per argomento. Qualora vi sia la necessità il team si affida inoltre a strumenti di videochiamata remota come *Skype*_[g] e *Google Meet*_[g] che permettono video chiamate di gruppo e la condivisione dello schermo (particolarmente utile per quel che riguarda discussioni in merito al codice prodotto).

4.5.3 Creazione diagrammi di Gantt

Per la realizzazione di diagrammi di *Gantt*_[g] si è scelto lo strumento *open source*_[g] e *multiplatforma*_[g] GanttProject.

4.5.4 Calcolo del consuntivo

Gli strumenti a disposizione dal Responsabile di Progetto sono i seguenti:

- Fogli Google
- GanttProject

A Ciclo di Deming

Il *ciclo di Deming*_[9] (o ciclo di PDCA, acronimo dall'inglese Plan–Do–Check–Act) è un metodo di gestione iterativo in quattro fasi utilizzato per il controllo e il miglioramento continuo dei processi e dei prodotti.

Serve per promuovere una cultura della qualità che è tesa al miglioramento continuo dei processi e all'utilizzo ottimale delle risorse. Questo strumento parte dall'assunto che per il raggiungimento del massimo della qualità sia necessaria la costante interazione tra ricerca, progettazione, test, produzione e vendita. Per migliorare la qualità e soddisfare il cliente, è necessario passare attraverso tutte e quattro le fasi costantemente, tenendo come criterio principale la qualità.

Le quattro fasi che compongono il ciclo possono essere riassunte nella seguente figura: Dove:

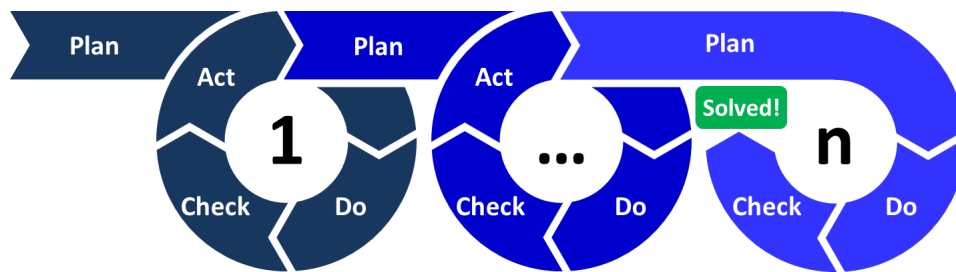


Figura 50: Ciclo PCDA

- **P:** Plan. In questa fase vengono stabiliti gli obiettivi e i processi necessari per fornire risultati in accordo con i risultati attesi, attraverso la creazione di attese di produzione, di completezza e accuratezza delle specifiche scelte. Quando possibile, avvio su piccola scala, per verificare i possibili effetti;
- **D:** Do. Esecuzione del programma, dapprima in contesti circoscritti. Attuare il piano, eseguire il processo, creare il prodotto. Raccogliere i dati per la creazione di grafici e analisi da destinare alla fase di "Check" e "Act";
- **C:** Check. Test e controllo, studio e raccolta dei risultati e dei riscontri. Studiare i risultati, misurati e raccolti nella fase del "Do" confrontandoli con i risultati attesi, obiettivi del "Plan", per verificarne le eventuali differenze. Cercare le deviazioni nell'attuazione del piano e focalizzarsi sulla sua adeguatezza e completezza per consentirne l'esecuzione. I grafici dei dati possono rendere questo molto più facile, in quanto è possibile vedere le tendenze di più cicli PDCA, convertendo i dati raccolti in informazioni. L'informazione è utile per realizzare il passo successivo : "Act";
- **A:** Act. Azione per rendere definitivo e/o migliorare il processo (estendere quanto testato dapprima in contesti circoscritti all'intera organizzazione). Richiede azioni correttive sulle differenze significative tra i risultati effettivi e previsti. Analizza le differenze per determinarne le cause e dove applicare le modifiche per ottenere il miglioramento del processo o del prodotto. Quando

un procedimento, attraverso questi quattro passaggi, non comporta la necessità di migliorare la portata a cui è applicato, il ciclo PDCA può essere raffinato per pianificare e migliorare con maggiore dettaglio la successiva iterazione, oppure l'attenzione deve essere posta in una diversa fase del processo.

Il gruppo Cyber13 ha ritenuto il ciclo di Deming come utile strumento per lo svolgimento delle seguenti attività:

- Procedure quotidiane di gestione per l'individuo e/o la squadra;
- Verifiche e revisioni.

B SOLID Principles

I principi ivi indicati sono norme di buona programmazione da utilizzare in fase di progettazione e sviluppo al fine di produrre *clean code*_[g].

I principi sono:

- **Single responsibility** (coesione): Ogni modulo, classe o funzione dovrebbe avere responsabilità su una singola parte della funzionalità fornita dal software e che tale responsabilità dovrebbe essere interamente incapsulata dalla classe;
- **Open-closed**: Le entità software (classi, moduli, funzioni, ecc.) dovrebbero essere aperte per estensione, ma chiuse per la modifica"; cioè, tale entità può consentire il suo comportamento da estendere senza modificare il suo codice sorgente;
- **Liskov substitution**: La sostituibilità è un principio nella programmazione orientata agli oggetti che afferma che, in un programma per computer, se S è un sottotipo di T, allora gli oggetti di tipo T possono essere sostituiti con oggetti di tipo S (cioè un oggetto di tipo T può essere sostituito con qualsiasi oggetto di un sottotipo S) senza alterare nessuna delle proprietà desiderabili del programma (correttezza, compito svolto, ecc.);
- **Interface segregation**: Il principio di separazione delle interfacce (ISP) afferma che nessun client dovrebbe essere costretto a dipendere da metodi che non usa. L'ISP divide le interfacce che sono molto grandi in quelle più piccole e specifiche in modo che i clienti debbano solo conoscere i metodi che sono di loro interesse. Tali interfacce ridotte sono anche chiamate interfacce di ruolo. L'ISP ha lo scopo di mantenere un sistema disaccoppiato e quindi più semplice da rifattorizzare, modificare e ridistribuire;
- **Dependency inversion**: Il principio di inversione delle dipendenze è una forma specifica di moduli software di disaccoppiamento. Seguendo questo principio, le relazioni di dipendenza convenzionali stabilite da moduli di alto livello, che regolano le politiche a moduli di dipendenza di basso livello, vengono invertite, rendendo i moduli di alto livello indipendenti dai dettagli di implementazione del modulo di basso livello. Il principio afferma:
 - I moduli di alto livello non dovrebbero dipendere da moduli di basso livello. Entrambi dovrebbero dipendere dalle astrazioni (ad esempio interfacce);
 - Le astrazioni non dovrebbero dipendere dai dettagli. I dettagli (implementazioni concrete) dovrebbero dipendere dalle astrazioni.

Stabilendo che sia gli oggetti di alto livello che quelli di basso livello devono dipendere dalla stessa astrazione, questo principio di progettazione inverte il modo in cui alcune persone possono pensare alla programmazione orientata agli oggetti. L'idea alla base dei punti precedenti di questo principio è che quando si progetta l'interazione tra un modulo di alto livello e uno di basso livello, l'interazione dovrebbe essere considerata come un'interazione astratta

tra di essi. Ciò non ha solo implicazioni sul design del modulo di alto livello, ma anche su quello di basso livello: quello di basso livello dovrebbe essere progettato tenendo presente l'interazione e potrebbe essere necessario cambiare la sua interfaccia di utilizzo. In molti casi, pensare all'interazione in sé come un concetto astratto consente di ridurre l'accoppiamento dei componenti senza introdurre schemi di codifica aggiuntivi, consentendo solo uno schema di interazione più leggero e meno dipendente dall'implementazione. Quando lo schema/i di interazione astratti scoperti tra due moduli è/sono generici e la generalizzazione ha senso, questo principio di progettazione porta anche al seguente modello di codifica di inversione delle dipendenze.

C Stati di progresso per SEMAT

L'istituzione $SEMAT_{|g|}$ ha individuato stati di progresso per quanto riguarda il sistema software:

- **Architecture Selected:** viene selezionata l'architettura e le diverse tecnologie da utilizzare per l'implementazione del software;
- **Demonstrable:** sono dimostrate le principali caratteristiche dell'architettura e le decisioni sulle principali interfacce e configurazioni del sistema;
- **Usable** Il sistema è utilizzabile, le funzionalità sono testate e accettate;
- **Ready:** anche la documentazione per l'utente è pronta, gli stakeholder hanno accettato il prodotto;
- **Operational:** Sistema è pienamente in uso su un sistema operativo con tutte le funzionalità disponibili;
- **Retired:** il sistema non è più supportato, è stato sostituito.