RedRoundRobin

NORME DI PROGETTO

v0.0.1

www.redroundrobin.site — info@redroundrobin.site

INFORMAZIONI SUL DOCUMENTO

Versione	0.0.1
Uso	interno
Stato	in redazione
Destinatari	nome cognome
	nome cognome
Redattori	nome cognome
	nome cognome
Verificatori	nome cognome
	nome cognome
Approvazione	nome cognome

Descrizione

Il documento contiene tutta la normativa di progetto relativa al *way of working*, dalle convenzioni formali nei documenti allo stile di programmazione del software.

Registro delle modifiche

Versione	Descrizione	Data	Autore	Ruolo
0.0.1	Creazione iniziale del documento	02-12-2019	Mariano Sciacco	Redattore

Indice

1 Introduzione

1.1 Scopo del documento

Il documento ha lo scopo di definire quelle che sono le regole su cui si basa il way of working del gruppo Red Round Robin per lo svolgimento del progetto. Le attività che possono essere trovate all'interno di questo documento sono state prese da processi appartenenti allo standard ISO 12207. Tutti i membri del gruppo sono quindi tenuti a prendere visione di questo documento così da garantire uniformità e coesione all'interno del progetto.

1.2 Scopo del prodotto

Il capitolato C6 si pone come obiettivo quello di creare una web-application che permette di analizzare grosse moli di dati ricevuti da sensori eterogenei tra loro. Tale applicazione mette a disposizione un'interfaccia che permette di visualizzare alcuni dati di interesse od eventuali correlazioni tra i dati stessi. Infine, per ogni tipologia di dato è possibile assegnarne il monitoraggio ad un particolare ente, ruolo o gruppo.

1.3 Glossario e Documenti esterni

Per evitare possibili ambiguità relative alle terminologie (che andranno indicate in MAIUSCOLETTO)utilizzate nei vari documenti, verranno utilizzate due simboli:

- Una D al pedice per indicare il nome di un particolare documento.
- Una G al pedice per indicare un termine che sarà presente nel GLOSSARIO v0.0.1.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Standard ISO/IEC 12207:1995:** https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
- **Capitolato d'appalto C6 - ThiReMa:** https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C6.pdf

1.4.2 Riferimenti informativi

- Da aggiungere man mano che si fa riferimento alle slide del prof
- Guardare bene gli approfondimenti sul sito: https://www.math.unipd.it/~tullio/IS-1/2019/
- **Documentazione git:** https://git-scm.com/docs

- **Documentazione GitHub:** https://help.github.com/en/github
- **Documentazione LaTeX:** https://www.latex-project.org/help/documentation/

2 Processi Primari

2.1 Fornitura

2.1.1 Scopo

2.1.2 Aspettative

2.1.3 Descrizione

2.1.4 Attività

2.1.4.1 Studio di fattibillità

2.1.4.2 Piano di progetto

2.1.4.3 Piano di qualifica

2.1.5 Strumenti

2.2 Sviluppo

2.2.1 Scopo

Il processo di sviluppo definisce i compiti e le attività da intraprendere volte al raggiungimento del prodotto finale richiesto dal proponente.

2.2.2 Aspettative

Per una corretta implementazione di questo processo è necessario fissare:

- Obbiettivi di sviluppo;
- Vincoli tecnologici e di design.

Il prodotto finale deve rispettare i requisiti e le aspettative del proponente, superando i test definiti dalle norme di qualità.

2.2.3 Descrizione

Il processo di sviluppo, secondo lo standard ISO/IEC 12207:1995, si articola nelle seguenti attività:

- Analisi dei Requisiti;
- Progettazione;
- Codifica.

2.2.4 Attività

Di seguito verranno analizzate dettagliatamente le attività menzionate nella sezione precedente.

2.2.4.1 Analisi dei requisiti

Scopo

Gli Analisti si occupano di stilare il documento di Analisi dei Requisiti, il cui scopo è appunto quello di definire ed elencare tutti i requisiti del capitolato. Il documento finale conterrà:

- Descrizione generale del prodotto;
- Argomentazioni precise ed affidabili per i Progettisti;

- Casi d'uso rappresentati tramite diagrammi UML;
- Fissare funzionalità e requisiti concordi con le richieste del cliente;
- Stima dei costi.

Classificazione dei Requisiti

I requisiti verranno classificati per facilitarne la comprensione e vengono identificati, in maniera univoca, secondo il seguente schema identificativo:

R[Priorità]-[Tipologia]-[Identificativo]

Dove:

- **R:** Requisito
- **Priorità:** ogni requisito assumerà uno dei seguenti valori:
  - **A:** obbligatorio, strettamente necessario;
  - **B:** desiderabile, non strettamente necessario;
  - **C:** opzionale, relativamente utile o contrattabile in corso d'opera.
- **Tipologia:** ogni requisito assumerà uno dei seguenti valori:
  - **F:** funzionale;
  - **P:** prestazionale;
  - **Q:** qualitativo;
  - **V:** vincolo.
- **Identificativo:** numero progressivo per contraddistinguere il requisito, in forma gerarchica padre/figlio strutturato come segue:

[codicePadre].[codiceFiglio]

Classificazione dei Casi d'Uso

Gli Analisti, dopo la stesura dei requisiti, hanno anche il compito di identificare ed elencare i casi d'uso. Ognuno di essi è identificato, in maniera univoca, secondo il seguente schema identificativo:

UC[codicePadre].[codiceFiglio]

Ogni caso d'uso oltre al codice di identificazione deve contenere i seguenti campi:

- **Diagrammi UML:** diagrammi realizzati usando la versione 2.0 del linguaggio;
- **Attori primari:** attori principali del caso d'uso;
- **Attori secondari:** attori secondari del caso d'uso;
- **Descrizione:** breve descrizione del caso d'uso;
- **Attori secondari:** attori secondari del caso d'uso;
- **Estensioni:** eventuali estensioni coinvolte;
- **Inclusioni:** eventuali inclusioni coinvolte;
- **Precondizione:** condizioni identificate come vere prima del verificarsi degli eventi del caso d'uso;
- **Postcondizione:** condizioni identificate come vere dopo il verificarsi degli eventi del caso d'uso;
- **Scenario principale:** flusso degli eventi come elenco numerato con eventuale riferimento ad ulteriori casi d'uso.

2.2.4.2 Progettazione

L'attività di progettazione avviene una volta concluso il documento di Analisi dei Requisiti, dove i Progettisti hanno il compito di definire una soluzione soddisfacente del problema e di realizzare l'architettura del sistema. Questa fase si divide nelle seguenti fasi:

- **Tecnology baseline:** specifiche della progettazione del prodotto e delle sue componenti, insieme dei diagrammi UML dell'architettura ed i test di verifica;
- **Product baseline:** specifica maggiormente l'attività di progettazione e definisce i test necessari per la verifica.
- **Diagrammi UML:** Diagrammi necessari per rendere più chiare le soluzioni progettuali utilizzate e si suddividono in:
  - **Diagrammi delle attività:** descrivono un processo o un algoritmo;
  - **Diagrammi delle classi:** rappresentano gli oggetti del sistema e loro relazioni;
  - **Diagrammi dei casi d'uso:** descrivono le funzioni offerte dal sistema;
  - **Diagrammi dei package:** descrivono le dipendenze tra classi raggruppate in package;
  - **Diagrammi di sequenza:** descrivono la una sequenza di processi o funzioni.
- **Tecnologie utilizzate:** elenco dettagliato delle tecnologie impiegate;

2.2.4.3 Codifica

Scopo

In questa attività vengono stese le norme alle quali i Programmatori devono affidarsi durante l'attività di programmazione ed implementazione.

Aspettative

L'obiettivo è quello di sviluppare il software richiesto dal proponente utilizzando le norme di programmazione stabilite in modo da ottenere:

- codice leggibile ed uniforme per i Programmatori;
- agevolare le fasi di manutenzione, verifica e validazione.

Stile di codifica

Per garantire l'uniformità del codice, ciascun Programmatore dovrà attenersi alle seguenti regole norme di programmazione:

- **Indentazione:** i blocchi del codice devono essere indentati, per ciascun livello, con tabulazione la cui larghezza sia impostata a quattro (4) spazi. Ogni programmatore dovrà configurare il proprio editor di testo secondo questa regola;
- **Univocità dei nomi:** classi, metodi e variabili dovono avere nomi univoci e che ne descrivano il più possibile la funzione dove la prima lettera deve essere sempre minuscola e, nel caso in cui la classe/metodo/variabile sia una concatenazione di più parole, i programmatori devo attenersi al CamelCase$_G$.
- **Spazi:** prima di ogni apertura di parentesi graffa, tonda e quadra ci deve essere uno (1) spazio. Ogni chiusura di parentesi graffa per metodi, classi e condizioni va fatta andando a capo;

ALTRE?

2.2.5 Strumenti

Di seguito verranno elencati gli strumenti che verranno utilizzati nella fase di sviluppo.

Chrome

Browser web sviluppato da Google, basato sul motore di rendering Blink.

Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Include il supporto per debugging.

Draw.io

Strumento open source semplice ed intuitivo per la creazione dei diagrammi UML.

Bootstrap

Raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS per le varie componenti dell'interfaccia.

DA AGGIUNGERE?

3 Processi di supporto

3.1 Documentazione

3.1.1 Scopo

3.1.2 Aspettative

3.1.3 Descrizione

3.1.4 Ciclo di vita

3.1.5 Template LaTeX

3.1.6 Struttura dei documenti

3.1.6.1 Frontespizio

3.1.6.2 Registro modifiche

3.1.6.3 Indice

3.1.6.4 Contenuto principale

3.1.6.5 note a piè di pagina

3.1.7 Classificazione dei documenti

3.1.7.1 Documenti ufficiosi

3.1.7.2 Documenti ufficiali

3.1.7.3 Verbali

3.1.7.4 Glossario

3.1.7.5 Lettere

3.1.8 Norme tipografiche

3.1.8.1 Convenzioni sui nomi dei file

3.1.8.2 Glossario

3.1.8.3 Stile del testo

3.1.8.4 Elenchi puntati

3.2 Gestione della Configurazione

3.2.1 Scopo

La gestione della configurazione definisce i principi normativi utili a predisporre il WORKSPACE$_G$ per tutto il gruppo, semplificando e automatizzando la conservazione dei documenti e del software.

3.2.2 Repository

La repository è un luogo remoto in cui vengono mantenuti, salvati e versionati tutti i file che riguardano il progetto per tutto il ciclo di vita del prodotto. Questa risiede in modo condiviso all'interno di GITHUB$_G$ ed è accessibile solamente ai membri del team.

3.2.2.1 Integrazione di VCS e ITS con Github

La respository fa uso di un *Version Control System* (VCS) di tipo distribuito sotto il motore *Git*, che permette la condivisione dal locale al remoto del proprio spazio di lavoro su un luogo comune. Attraverso l'utilizzo di un web browser è possibile collegarsi a *Github* e controllare i file contenuti nella repository, usando come indirizzo web:

http://project.redroundrobin.site

Inoltre, *Github* integra un *Issue Tracking System* (ITS) che permette di tracciare lo sviluppo tramite ticketing, assegnando a ciascun membro del team una o più ISSUE$_G$ in base alle necessità.

3.2.2.2 Configurazione del Workflow

Usando *Git*, è possibile clonare e scaricare da remoto tutto il contenuto della repository per averne una copia in locale su cui poterci lavorare, visionando la cronologia dei file modificati ad ogni COMMIT$_G$ da parte di un membro del gruppo. *Git*, inoltre, include la possibilità di creare dei BRANCH$_G$ (locali e remoti) in cui poter sviluppare in maniera indipendente una funzionalità, che potrà essere integrata successivamente, senza bisogno di stare al passo con gli aggiornamenti della repository.

Pertanto, si è deciso di stabilire il seguente canone di WORKFLOW$_G$ per quanto concerne la documentazione:

- `master` branch: branch principale su cui vengono fatti i rilasci ad ogni MILESTONE$_G$;
- `develop` branch: branch di sviluppo su cui viene fatta integrazione di nuove funzionalità concluse;
- `feature/x` branch: branch indipendente usato da uno o più membri del gruppo per sviluppare una singola funzionalità, sezione o revisione di un documento;
  - **X:** si riferisce al numero della ISSUE$_G$ o al titolo della funzionalità da integrare.
  - il branch riferisce generalmente una singola ISSUE$_G$ ed è rimosso alla sua chiusura.

Ad ogni milestone viene associata una release interna alla repository, così da tenere traccia di una BASELINE$_G$.

3.2.2.3 Formati dei File

Configurazione

In generale, vengono utilizzati alcuni file con formati speciali o mancanti per la configurazione della repository. Questi file vanno modificati solamente dall'AMMINISTRATORE$_G$, o su richiesta, in base alle necessità.

- **.gitignore** contiene tutte le regole per evitare di caricare nella repository dei formati non autorizzati (es: file eseguibili);
- **.yml** contiene la configurazione di una GITHUB ACTIONS$_G$ per dirigere il WORKFLOW$_G$;
- **File senza formati** contengono configurazioni aggiuntive per GITHUB$_G$ (es: template delle ISSUE$_G$);

Documentazione

Per la documentazione si usano principalmente i seguenti tipi di file:

- **file .tex:** che contiene il codice sorgente LaTeX del documento;
- **file .pdf:** che è il documento compilato;
- **file .png:** che identifica una immagine;
- **file .md:** che identifica un file scritto in MARKDOWN$_G$, generalmente usato per gli appunti senza bisogno di compilazione.

3.2.2.4 Struttura della repository

Configurazione

La repository si compone di diverse tipologie di cartelle. Oltre alle cartelle che riguardano la documentazione e il software, è presente una cartella che viene usata ai fini della configurazione di GITHUB$_G$.

- `.github/`: cartella per la repository di GITHUB$_G$ con i file di configurazione per le GITHUB ACTIONS$_G$ e l'ISSUE TRACKING SYSTEM$_G$.

Documentazione

A livello di struttura, la documentazione nella repository si compone di 4 cartelle principali:

- `interni/`: contiene tutta la documentazione interna;
- `esterni/`: contiene tutta la documentazione esterna;
- `template/`: contiene tutti i template delle varie tipologie di documento;
- `notes/`: contiene tutte le note aggiuntive (*non*-LaTeX) sui seminari, le guide e l'organizzazione.

3.2.2.5 Countinous Delivery dei documenti

La repository è stata configurata per garantire una buona accessibilità e correttezza nello sviluppo dei documenti di progetto da parte dei membri del team. In particolare è stato messo in atto un processo di CONTINOUS DELIVERY$_G$ per avere sempre disponibili gli ultimi documenti modificati.

1. Ad ogni COMMIT$_G$ del branch `develop` che riguarda la modifica di un documento, viene fatta una BUILD$_G$ di tutti i documenti.
2. La BUILD$_G$ crea un ARTEFATTO$_G$ con tutti i documenti PDF che vengono salvati online, oltre che nella repository, e resi disponibili per essere visionati da remoto.
3. Tramite gli appositi canali di comunicazione, vengono notificati tutti i membri del team delle nuove modifiche.

Nel caso di errori nella compilazione dei file, viene inviato un avviso all'ultima persona che ha eseguito il COMMIT$_G$ e vengono mantenuti gli ultimi file correttamente compilati.

3.2.3 Versionamento e Rilascio

3.2.3.1 Documentazione

Tutti i file che riguardano la documentazione vengono conservati in una repository e vengono versionati per mezzo di un identificativo, in base alla loro fase di avanzamento. Questo permette di poter fare riferimento alle nuove versioni del documento durante tutto il ciclo di vita del software.

Codice di Versionamento

Ciascun documento possiede un identificativo di versionamento su ogni pagina che ha il seguente formalismo:

$$\mathrm{v}[\alpha].[\beta].[\gamma]$$

- (α): numero identificativo del rilascio del documento;
  - parte da 0 e non si resetta mai;

  - viene incrementato solo dal responsabile a seguito di una approvazione.

- (β): numero identificativo che rappresenta un avanzamento sostanziale dopo le revisioni del documento;
  - parte da 0 e si resetta solo a incrementi di α;
  - viene incrementato solo dai revisori.

- (γ): numero rappresentativo di una aggiunta, modifica o eliminazione fatta al documento
  - parte da 0 e si resetta solo a incrementi di β o di α;
  - viene incrementato da un redattore o da un revisore in base alle necessità.

Esempi Codice di Versionamento

- `v0.2.1`
- `v1.12.1`
- `v3.0.2`

Rilascio

Un documento viene rilasciato alle parti proponenti solamente quando vi è un incremento del primo numero (α), che ne deduce una approvazione da parte del responsabile.
Per quanto concerne la distribuzione interna, tutti gli ARTEFATTI$_G$ dei documenti realizzati durante la fase di sviluppo sono resi disponibili in modo rapido e automatizzato a tutti i membri del gruppo, a cui vengono notificate tutte le modifiche tramite il WORKSPACE$_G$ di *Slack*.

3.2.3.2 Software

I sorgenti del software che riguardano la codifica e la configurazione del prodotto da realizzare sono mantenuti nella repository insieme alla documentazione. Ogni file viene versionato con un apposito storico delle modifiche, mentre l'intero software viene versionato in quanto BASELINE$_G$ sulla base delle funzionalità presenti e dei requisiti obbligatori implementati.

Codice di Versionamento

Il versionamento del software viene eseguito sulla base delle implementazioni effettuate a livello di codifica. In aggiunta, si definisce una sintassi di stato nella versione per definirne l'usabilità.

$$\mathrm{v}[\delta].[\epsilon].[\mu]\text{-}[\lambda]$$

- (δ): numero identificativo del rilascio software per una MAJOR RELEASE$_G$;
  - parte da 0 e non si resetta mai;
  - viene incrementato solo dopo aver implementato tutti i requisiti obbligatori.

- (ϵ): numero identificativo per una MINOR RELEASE$_G$;
  - parte da 0 e si resetta solo a incrementi di δ;
  - viene incrementato solo dopo l'implementazione di uno o più requisiti.
- (μ): numero rappresentativo per una PATCH$_G$;
  - parte da 0 e si resetta solo a incrementi di ϵ o di δ;
  - viene incrementato a ogni modifica di uno o più requisiti e ad ogni cambio di configurazione del software.
- (λ): sigla che identifica uno stato del software; può avere i seguenti significati in ordine crescente di stato:
  - `dev`: derivante da **dev**elopment, versione ancora in sviluppo.
    - software non completo, che ha ricevuto aggiunte, modifiche o eliminazioni recenti;
    - può contenere errori che fanno fallire i test;
    - **non** si presta all'uso di un utente finale.
  - `rc`: **r**elease **c**andidate, versione candidata al rilascio;
    - software che contiene tutti o una parte dei requisiti obbligatori richiesti;
    - passa tutte le tipologie di test;
    - si presta all'uso di un utente finale.
  - `stable`: versione **stabile**, pronta al rilascio pubblico;
    - software completo che implenta tutti i requisiti obbligatori;
    - passa tutte le tipologie di test ed è validato;
    - può essere collaudato con il proponente e/o pubblicato.

Una versione del software subisce un incremento di stato in base alla verifica da parte del meccanismo automatico di TEST$_G$. In particolare:

- una versione può ricevere lo stato di `rc` solo se il primo numero (δ) o il secondo numero (ϵ) è diverso da 0;
- una versione può ricevere lo stato di `stable` solo se il primo numero (δ) è diverso da 0;

Esempi Codice di Versionamento

- `v0.3.5-dev` : versione non completa, non usabile e forse funzionante.
- `v0.7.0-rc` : versione non completa, usabile, funzionante e con alcuni requisiti implementati.
- `v1.0.0-rc` : versione forse completa, usabile, funzionante e in attesa di validazione.
- `v1.0.0-stable` : versione completa, usabile, funzionante, verificata e validata.

Rilascio

Le release del software vengono eseguite internamente nella REPOSITORY$_G$ in base alle funzionalità sviluppate. Inoltre, i rilasci interni saranno normati come segue:

- le versioni `stable` e/o MAJOR RELEASE$_G$ devono essere approvate dall'AMMINISTRATORE$_G$ e dal RESPONSABILE$_G$.
- le versioni `rc` e/o MINOR RELEASE$_G$ devono essere approvate, previa notifica di conferma di tutti i TEST$_G$, dall'AMMINISTRATORE$_G$ e dai VERIFICATORI$_G$.
- le versioni `dev` e/o PATCH$_G$ non richiedono approvazione e possono essere rilasciate autonomamente dal PROGRAMMATORE$_G$.

3.2.3.3 Tecnologie

Le tecnologie coinvolte riguardano principalmente GITHUB$_G$

3.2.3.4 Repository

3.2.3.5 Gestione delle modifiche

3.3 Garanzia della qualità

3.3.1 Scopo

3.3.2 Aspettative

3.3.3 Descrizione

3.3.4 Controllo qualità prodotto

3.3.5 Controllo qualità di processo

3.3.6 Classificazioni metriche

3.3.7 Strumenti

3.4 Verifica

3.4.1 Scopo

Il processo di verifica ha lo scopo di capire se il prodotto è realizzato nel modo corretto secondo delle regole stabilite.

3.4.2 Aspettative

Lo svolgimento del processo di verifica sarà garantito seguendo determinati punti:

- Definizione di criteri di accettazione;
- Prescrizione delle attività di verifica con relativa documentazione;
- Test di verifica;
- Correzione di eventuali difetti.

3.4.3 Descrizione

La verifica consiste nel cercare e risolvere possibili difetti all'interno della documentazione e del codice prodotto. Il completamento del processo di verifica rende possibile l'esecuzione del processo di validazione.

3.4.4 Attività

3.4.4.1 Analisi statica e dinamica

Analisi statica

L'analisi statica viene effettuata sulla documentazione e sul codice senza necessità di eseguire il prodotto e serve per verificare che non ci siano errori o difetti. I due tipi di analisi statica sono:

- Walkthrough: consiste nell'analizzare i vari documenti e file in tutto il loro contenuto per trovare eventuali difetti. Il verificatore controlla se sono presenti difetti e, in caso ne trovi, la correzione verrà effettuata dagli sviluppatori;
- Inspection: in questa tecnica si conosce dove possono trovarsi i possibili difetti, quindi non si analizzano i documenti e file per intero, ma solo le parti in cui di solito sono presenti. Il verificatore compone una lista di controllo (checklist) inserendo i punti in cui si possono rilevare possibili difetti, controlla in quei punti della lista e, se trova delle incorrettezze, la correzione viene poi effettuata dagli sviluppatori.
  Di seguito alcuni possibili punti in cui trovare difetti all'interno della documentazione:

- Elenchi puntati;
- Formato Date;
- Parole/frasi in grassetto/corsivo;
- Uso di riferimenti appropriati al Glossario/Documento.

Analisi dinamica

L'analisi dinamica è una tecnica per cui è necessaria l'esecuzione dell'oggetto di verifica, e consiste nell'attività di test.

3.4.4.2 Test

I test fanno parte dell'attività di analisi dinamica e servono per individuare possibili errori di funzionamento del codice. Per effettuare i test, essi devono essere automatizzati, tramite strumenti appositi, e ripetibili, ovvero devono essere definiti:

- l'ambiente di sviluppo e lo stato iniziale;
- le istruzioni eseguite;
- i dati di input e i dati di output attesi.

Test di unità

Test eseguiti sul funzionamento di unità di software in modo automatico: viene definito l'input e l'output atteso per verificare il corretto funzionamento dell'unità.

Test di integrazione

Test eseguiti su componenti del software per verificare se l'insieme di unità si interfaccia come dovrebbe. Questo test è eseguito in modo ricorrente, ogni volta che un insieme di unità esegue correttamente, esso viene integrato con altri insiemi di unità, fino al test completo sul sistema.

Test di regressione

Test eseguito ogni volta che un'unità viene modificata allo scopo di trovare difetti nelle funzionalità già testate, e per questo non dover retrocedere. Si rieseguono tutti i test necessari affinchè si possa essere certi che la modifica non causa il funzionamento scorretto di altre unità collegate all'unità modificata.

3.4.5 Strumenti

3.4.5.1 Verifica ortografica

3.5 Validazione

3.5.1 Scopo

Lo scopo del processo di validazione consiste nel garantire che il prodotto rispetta le richieste del committente, e quindi che esegue correttamente.

3.5.2 Aspettative

Per garantire che venga raggiunto lo scopo del processo di validazione, si eseguono le attività con dei test documentati e l'output generato corrisponde a quello aspettato.

3.5.3 Descrizione

Il processo di validazione esegue il test completo sul sistema affinchè sia chiaro se sono state rispettate le necessità del committente, il che porta a definire se il prodotto esegue in modo corretto. Per poter effettuare questo processo è necessario che venga eseguito dopo il processo di verifica, in modo che tutte le unità del sistema permettano il test completo su di esso.

3.5.4 Attività

3.5.4.1 Test

I test eseguiti in questo processo riguardano il test sul sistema, eseguito internamente, e il test di accettazione, eseguito insieme al committente.

Test di sistema

Dopo aver eseguito i test su tutte le unità e sulla loro integrazione, si testa il sistema nella sua interezza. Viene testato se le interazioni tra le varie componenti del sistema ritornano il risultato atteso o meno in concordanza con ciò che è stato definito nel processo di analisi dei requisiti.

Test di accettazione

Anche detto "test di collaudo" è il test eseguito su input definiti dal committente in modo da verificare se l'output atteso da esso è corretto o meno.

4 Processi organizzativi

4.1 Gestione dei processi

4.1.1 Scopo

4.1.2 Descrizione

4.1.3 Ruoli di progetto

4.1.3.1 Responsabile di progetto

4.1.3.2 Amministratore di progetto

4.1.3.3 Analista

4.1.3.4 Progettista

4.1.3.5 Programmatore

4.1.3.6 Verificatore

4.1.4 Procedure

4.1.4.1 Gestione delle comunicazioni

4.1.4.2 Gestione degli incontri

4.1.4.3 Gestione degli strumenti di coordinamento

4.1.4.4 Gestione dei rischi

4.1.5 Strumenti

4.2 Formazione del personale

4.2.1 Scopo

I membri del gruppo Red Round Robin sono tenuti a formarsi individualmente sulle tecnologie richieste per il completamento del progetto e, nel caso di incomprensioni o problemi nell'utilizzo di esse, il proponente si dichiara disponibile a corsi di formazione e chiarimenti.

4.2.2 Descrizione

Ogni persona dovrà quindi auto documentarsi sui linguaggi e gli strumenti di programmazione che verranno utilizzati per tutto il periodo di sviluppo del progetto, se sconosciuti.

4.2.3 Piano di formazione

Di seguito verranno elencate le tecnologie ed i linguaggi di programmazione necessari per lo svolgimento del capitolato con i relativi link al sito ufficiale:

- **Java**: www.java.com
- **LaTeX:** www.latex-project.org;
- **GitHub**: www.github.com.
- **Bootstrap**: www.getbootstrap.com
- **Apache Kafka**: www.kafka.apache.org