NaturalApi

Norme di progetto

Versione	1.0.0
Approvazione	Andrea Polo
Redazione	Edoardo Tinto
Verifica	Matteo Infantino
Stato	Approvato
Uso	Esterno
Destinato a	FourCats
	TealBlue
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
Email	fourcats.unipd@gmail.com

Descrizione

Descrizione delle regole, degli strumenti e delle convenzioni adottate dal gruppo FourCats nella realizzazione del capitolato NaturalApi

Versione	Data	Nominativo	Ruolo	Descrizione	Verificatore	Esito
1.0.0	03-05-2020	Matteo Munari	Amministratore	Aggiornata §2.2.7 - Coding e Testing	Edoardo Tinto	Positivo
0.9.0	01-04-2020	Edoardo Tinto	Amministratore	Aggiornata §4.1.6 - Versionamento	Matteo Infantino	Positivo
	27-03-2020	Edoardo Tinto	Amministratore	Correzioni segnalate in esito RP	Matteo Infantino	Positivo
0.0.2	13-02-2020	Matteo Munari	Amministratore	Aggiornata §4.2 - Training	Edoardo Tinto	Positivo
	11-02-2020	Francesco Battistella	Amministratore	Aggiornata §3.2 - Gestione della configurazione	Edoardo Tinto	Positivo
	10-02-2020	Andrea Polo	Amministratore	Aggiornata §2.2 - Sviluppo	Edoardo Tinto	Positivo
	08-02-2020	Francesco Battistella	Amministratore	Aggiornata §3.3 - Gestione della qualità	Matteo Infantino	Positivo
	07-02-2020	Andrea Polo	Amministratore	Aggiornata §3.7 - Risoluzione dei problemi	Matteo Infantino	Positivo
	06-02-2020	Francesco Battistella	Amministratore	Aggiornata §4.1 - Gestione	Matteo Infantino	Positivo
	04-02-2020	Matteo Munari	Amministratore	Aggiornata §2.1 - Fornitura	Matteo Infantino	Positivo
	31-01-2020	Andrea Polo	Amministratore	Aggiornata §3.1 - Documentazione	Matteo Infantino	Positivo
	27-01-2020	Matteo Munari	Amministratore	Aggiornamenti riferimenti, titoli e link secondo suggerimenti RR	Matteo Infantino	Positivo
0.0.1	27-11-2019	Matteo Infantino	Amministratore	Modifiche §3.3 - Gestione della qualità	Francesco Battistella	Positivo
	26-11-2019	Edoardo Tinto	Amministratore	Completata stesura §3.1 - Documentazione	Andrea Polo	Positivo

26-11-2019	Matteo Infantino	Amministratore	Completata stesura §4 - Processi organizzativi	Andrea Polo	Positivo
24-11-2019	Edoardo Tinto	Amministratore	Modifiche a §3 - Processi di Supporto	Andrea Polo	Positivo
24-11-2019	Simone Innocente	Amministratore	Stesura §3.4 - Processo di Verifica	Andrea Polo	Positivo
24-11-2019	Edoardo Tinto	Amministratore	Modifiche §3.1 - Documentazione	Andrea Polo	Positivo
23-11-2019	Matteo Infantino	Amministratore	Modifiche §4.1 - Processo di Miglioramento	Andrea Polo	Positivo
23-11-2019	Giulio Umbrella	Amministratore	Modifiche §2 - Processi Primari	Andrea Polo	Positivo
22-11-2019	Edoardo Tinto	Amministratore	Modifiche a §3 - Processi di Supporto	Andrea Polo	Positivo
21-11-2019	Matteo Infantino	Amministratore	Stesura §3.3 - Gestione della Qualità	Andrea Polo	Positivo
21-11-2019	Simone Innocente	Amministratore	Stesura §4.3 - Processo di Gestione	Andrea Polo	Positivo
20-11-2019	Edoardo Tinto	Amministratore	Iniziata stesura §3 - Processi di Supporto	Francesco Battistella	Positivo
18-11-2019	Matteo Infantino	Amministratore	Iniziata stesura §4 - Processi Organizzativi	Francesco Battistella	Positivo
18-11-2019	Simone Innocente	Amministratore	Iniziata stesura §2 - Processi Primari	Francesco Battistella	Positivo
18-11-2019	Giulio Umbrella	Amministratore	Stesura §1 - introduzione	Francesco Battistella	Positivo
18-11-2019	Giulio Umbrella	Amministratore	Creata struttura documento	Francesco Battistella	Positivo

Indice

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di definire le regole, gli strumenti, i dettagli organizzativi e le convenzioni adottate dal gruppo FourCats durante tutti i processi del ciclo di vita del software.

1.2 Scopo del prodotto

Lo scopo del prodotto è la creazione di un *toolkit*, NaturalAPI, in grado di generare automaticamente application programming interfaces (*APIs*) e i relativi unit test per un dato linguaggio di programmazione, partendo da feature e scenari in formato *Gherkin* e da documenti di testo inerenti al dominio di interesse.

1.3 Convenzioni Testuali

Al fine di eliminare ogni ambiguità di linguaggio, si consiglia di vedere la sezione 3.1 del presente documento, relativa alla documentazione, per comprendere le convenzioni testuali adottate durante la stesura dello stesso.

1.4 Riferimenti

1.4.1 Normativi

- Capitolato: https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C3.pdf

1.4.2 Informativi

- ISO/IEC 12207-1995: https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
- Guida per l'installazione e l'utilizzo di LaTeX, "L'arte di scrivere con Latex" - nuova edizione 2011: http://www.dei.unipd.it/~addetto/manuali_online/ArteLaTeX.pdf
- Guida al setup e all'utilizzo di GitFlow: https://danielkummer.github.io/git-flow-cheatsheet/index.it_IT.html
- Calcolo indice di Gulpease: https://farfalla-project.org/readability_static/
- Repository *GitHub* di "comics.works Studio", progetto a supporto per lo sviluppo: https://github.com/tealblue-team/comics-works-suite/tree/develop/comics/works/solutions/studio
- RolloutBlog, Best Practices When Versioning a Release: rollout.io/blog
- GitFlow: https://devdev.it/guida-gitflow/come-funziona-gitflow-branch-develop-e-master/
- Git: https://it.wikipedia.org/wiki/Git_(software)

2 Processi primari

2.1 Fornitura

2.1.1 Scopo

Questo processo ha lo scopo di gestire il rapporto con il proponente teal.blue e di stabilire le condizioni necessarie al corretto completamento della fornitura.

2.1.2 Descrizione

Il processo comprende tutte le attività che bisogna eseguire per aggiudicarsi l'appalto e, in seguito, pianificare, realizzare e consegnare il prodotto software.

2.1.3 Iniziazione

Si prende visione del capitolato fornito dal proponente e in seguito si decide se accettare o no l'incarico. La decisione viene presa analizzando il problema in dettaglio, valutando la realizzabilità del prodotto e individuando i costi e i benefici. L'analisi e la scelta finale vengono documentate attraverso la redazione dello studio di fattibilità in accordo con il processo di documentazione (vedi 3.1).
Il documento dovrà contenere:

- Descrizione delle tecnologie coinvolte;
- Aspetti positivi e criticità emerse dallo studio del capitolato;
- Decisione finale e relative motivazioni.

Lo studio di fattibilità è un documento interno, cioè consultabile solo dai membri del gruppo.

2.1.4 Interazione con il proponente

Al fine di comprendere al meglio quello che il proponente desidera, è necessario effettuare delle interazioni delucidanti con il proponente. Queste sono utili per tutte le attività presenti all'interno del processo di fornitura. Le domande da porre dovono essere precedentemente discusse all'interno del gruppo, attraverso l'utilizzo dei canali Slack inerenti all'argomento in questione. I quesiti devono rispettare le seguenti regole:

- la formulazione dev'essere chiara;
- eventuali riferimenti al capitolato devono essere presenti.

In seguito, le domande vengono poste nel canale Slack riservato con il proponente seguendo le seguenti regole:

- un'unica persona di riferimento pone i quesiti: il responsabile o un suo delegato;
- devono essere sfruttate a pieno le possibilità che lo strumento mette a disposizione per ordinare il contenuto (es. elenchi puntati o utilizzo di sezioni di codice);
- deve essere considerato l'invio di un documento contenente le domande o materiale ad esse inerenti se questo può migliorare la chiarezza espositiva.

Se la situazione lo richiede, si può domandare una video-chiamata. In questo caso le domande devono comunque essere poste anticipatamente, per permettere al proponente di formulare al meglio le risposte.

2.1.5 Pianificazione

Si pianifica in dettaglio la realizzazione di NaturalAPI, rispettando i requisiti documentati nell'Analisi dei requisiti con il processo di Sviluppo e garantendo la qualità richiesta. Per prima cosa è necessario stabilire un life cycle model. Il modello di riferimento è quello *incrementale*, orientato a una continuo miglioramento del software tramite incrementi verificabili. I feedback ottenuti serviranno sia per riorganizzare il lavoro all'interno del gruppo, sia per confrontarsi con il proponente, per dimostrare la correttezza di quanto fatto o per identificare un problema e reagire tempestivamente. Si devono poi scegliere le opzioni di sviluppo, che nel nostro caso verrà fatto con l'utilizzo di risorse interne ed esterne, come il *parser* utile per *NaturalAPI discover*. Si deve inoltre pianificare il coinvolgimento con il proponente, che sarà anche l'utente di NaturalAPI, per effetturare joint review e dimostrazioni del prodotto. In accordo con il modello *agile* il coinvolgimento del proponente è essenziale, perciò le revisioni dovranno essere frequenti. Come ultima cosa è necessario stabilire gli strumenti da utilizzare per organizzare e tracciare il lavoro e pianificare il training del personale. A questo proposito si useranno *Telegram* e *Github*, che permette di fissare delle *milestone* e verificare il punto di avanzamento. La pianificazione ha come esito la produzione di due documenti:

- **Piano di progetto**, che dovrà comprendere:
  - identificazione delle risorse disponibili;
  - organizzazione cronologica del lavoro in modo da massimizzare l'efficienza;
  - suddivisione dei compiti e dei ruoli tra i componenti del gruppo;
  - definizione del ciclo di vita del software;
  - analisi economica con relativo preventivo dei costi.
- **Piano di qualifica**, che dovrà comprendere:
  - principi da seguire per eseguire la verifica in modo efficiente e quantificabile;
  - principi da seguire per eseguire la validazione in modo efficiente e quantificabile.

2.1.6 Esecuzione e controllo

Si sviluppa il software secondo quanto stabilito nel Piano di progetto e nel Piano di qualifica. Durante lo sviluppo si deve monitorare in modo iterativo lo stato di avanzamento del progetto, tramite *Github*, e controllare che il software sviluppato rispetti la qualità richiesta. In caso contrario si attiva il processo di risoluzione dei problemi.

2.1.7 Revisione e valutazione

Si devono coordinare le comunicazioni con l'acquirente, nel nostro caso il proponente. I meeting informali e le richieste di informazioni si effettuano tramite *Slack* o email, mentre le revisioni congiunte sul progetto vengono effettuate in accordo con il processo di Joint Review (vedi 3.6). Le revisioni vengono effettuate settimanalmente, con possibili variazioni da concordare con il proponente. Durante le revisioni inoltre, bisogna dimostrare che il software sviluppato soddisfi i requisiti richiesti fornendo al proponente i report di valutazione e test ottenuti dai processi di verifica e validazione.

2.1.8 Consegna e completamento

Il prodotto finito, candidato all'accettazione, dovrà essere consegnato presso la sede del committente su supporto CD-ROM / DVD, comprensivo di:

- utilità di installazione e istruzioni per l'uso;
- sorgenti completi e utilità di compilazione;
- documentazione completa e finale;
- eventuali utilità di collaudo.

La consegna al committente deve avvenire previo appuntamento, entro la data ultimativa delle ore 17:00 di venerdì 11 Settembre 2020. La consegna con il proponente deve essere ancora normata, si presuppone essa possa avvenire tramite repository *GitHub*.

2.2 Sviluppo

2.2.1 Scopo

Lo scopo del processo è la realizzazione del prodotto software richiesto dal proponente.

2.2.2 Descrizione

Lo Sviluppo raggruppa tutte le attività che si svolgono per la produzione di NaturalAPI a partire dall'analisi e dal design, fino ad arrivare alla consegna finale.

2.2.3 Analisi dei requisiti di sistema

Si devono identificare e analizzare i requisiti di sistema e software richiesti dal capitolato. L'attività ha come esito la produzione del documento di Analisi dei requisiti (AR.1.0.0). Il documento viene redatto in accordo con lo standard IEEE 830-1998 e deve contenere:

- Descrizione generale di NaturalAPI, individuando funzionalità e capacità del sistema;
- Descrizione dettagliata delle componenti;
- I casi d'uso di NaturalAPI e i relativi diagrammi UML;
- Requisiti e vincoli, che comprendono:
  - Requisiti richiesti dal proponente;
  - Requisiti richiesti dall'utente (nel nostro caso, la figura coincide con quella del proponente);
  - Requisiti di interfaccia;
  - Requisiti di sicurezza (valutati ma non presenti al momento);
  - Vincoli di design;
  - Requisiti di qualità;
  - Requisiti funzionali e capacità del software;
  - Requisiti fisici, ambientali e di performance;
  - Requisiti di consegna e installazione del software;
  - Presenza o meno di documentazione e manuali per l'utente.

I requisiti identificati devono essere:

- **Tracciabili**, deve essere possibile risalire al documento da cui è stato ricavato;
- **Consistenti**, due o più requisiti non possono contraddirsi;
- **Verificabili**, deve essere possibile stabilire in modo certo se il prodotto rispetta ogni requisito. Ne consegue che i requisiti devono essere espressi quantitativamente;
- **Realizzabili**, ogni requisito deve essere soddisfacibile, altrimenti il software non può essere sviluppato.

Una volta stabiliti i requisiti, è necessario confrontarsi con il proponente, nelle modalità previste dal processo di joint review, per verificare che tutti i requisiti minimi siano stati identificati. Durante l'attività di analisi si devono stabilire le specifiche dei test di sistema, in accordo con il "Modello a V". Sono obbligatorie due caratteristiche:

- Ogni test deve testare il soddisfacimento di un requisito o di parte di esso;
- Per ogni requisito dev'essere specificato almeno un test di sistema.

Le specifiche dei test di sistema vanno inserite nel Piano di qualifica (PQ.1.0.0).

2.2.4 Modello a V

Per assicurare la qualità del software prodotto, viene adottato il "Modello a V", il quale mette in risalto le relazioni tra ogni fase del ciclo di vita di sviluppo del software e la fase di testing ad esso associata. Prevede che lo sviluppo dei test cominci già all'inizio del progetto, in parallelo alle attività di analisi e progettazione, consentendo così di risparmiare un ampio tempo di progetto.

2.2.5 Design architetturale del software

Il modello che si intende seguire è la Clean Architecture, che prevede una suddivisione del software in vari livelli indipendenti con diversi gradi di astrazione che comunicano tra di loro tramite interfacce. In questo modo si separa la logica che sta alla base del software dall'interazione con l'utente o con dispositivi esterni.

2.2.5.1 Strati della Clean Architecture

L'architettura scelta impone l'individuazione di quattro strati distinti, ossia:

- Entità: fanno riferimento ad elementi (oggetti, strutture dati o funzioni) critici del dominio coinvolti in diverse applicazioni e poco soggette a cambiamenti;
- Casi d'uso: questo strato implementa i casi d'uso utilizzando, senza modificare, le entità. In questo strato UI e database non sono coinvolti;
- *Interfaces adapters*: questo strato fa da tramite con l'esterno, ovvero funge da interfaccia tra i dati prodotti dallo strato dei casi d'uso e servizi esterni (per esempio UI o database);
- Framework: coinvolge gli strumenti esterni che interagiscono con il software come framework e librerie o servizi web.

Nel rispetto della Clean Architecture quindi si deve strutturare il software in quattro strati seguendo questo procedimento:

1. Identificazione delle entità, che si riferiscono a oggetti che idealmente possono esistere anche al di fuori del software;
2. Identificazione dei casi d'uso che vanno ad accedere e a manipolare le entità;
3. Identificazione dei framework esterni che interagiscono con il software.

Per realizzare l'interfaccia grafica si deve seguire il modello Model-View-Controller (MVC), dove il Model è identificabile nei casi d'uso. Il design architetturale non richiede di documentare formalmente le decisioni prese. Le scelte fatte dovranno essere comunque riportate in un documento informale insieme a diagrammi che mettono in relazione le varie componenti al fine di semplificare la progettazione. Durante i design architetturale devono essere specificati i test di integrazione, in accordo con il "Modello a V". I test devono essere prodotti nel seguente modo:

1. Si identificano le componenti indipendenti sviluppabili in parallelo;
2. Si definisce il comportamento atteso dell'integrazione di due componenti indipendenti.

Ogni test di integrazione riguarda al più due componenti. Le specifiche dei test di integrazione vanno inserite nel Piano di qualifica (PQ.1.0.0).

2.2.6 Progettazione di dettaglio

Il software deve essere progettato in dettaglio e, attraverso un approccio top-down, le varie unità minime che lo compongono vengono identificate. Dal capitolato è già emersa la suddivisione in moduli indipendenti, perciò per ogni modulo sarà necessario:

1. Identificare gli oggetti che compongono il software, le relative proprietà e operazioni che vengono effettuate su di essi (diagrammi delle classi);

2. Stabilire le relazioni che legano gli oggetti (diagrammi delle classi, diagrammi dei package);
3. Stabilire il comportamento e l'interazione tra i vari oggetti (diagrammi di sequenza).

La progettazione richiede la produzione di diagrammi UML per rendere facilmente consultabili e comprensibili le scelte fatte. Le scelte fatte durante la progettazione devono basarsi sul design architetturale. In particolare l'interazione tra le classi che comunicano con strati logici diversi deve avvenire tramite interfacce, come stabilito dalla Clean Architecture. Per progettare i test di unità si deve:

1. Identificare le unità da testare;
2. Stabilire il risultato corretto da ottenere con l'esecuzione dell'unità.

Tutti i test definiti durante la progettazione di dettaglio sono funzionali (Black-Box Test). La progettazione di ogni componente dev'essere documentata in un documento formale in cui verranno inseriti i diagrammi UML, mentre i test andranno inseriti nel Piano di qualifica (PQ.1.0.0).

2.2.7 Coding e testing

Le unità identificate durante la progettazione di dettaglio devono essere implementate utilizzando il linguaggio Java. Per rendere il codice comprensibile a tutti i membri del gruppo è necessario rispettare le seguenti norme:

- **Parentesizzazione**: le parentesi di delimitazione iniziale dei costrutti vanno inserite in linea e non al di sotto di essi. Le parentesi di chiusura vanno invece messe in una nuova linea, come si può vedere nel seguente esempio:

```
public List<WordCounter> getNouns() {
    return nouns;
}

public List<WordCounter> getVerbs() {
    return verbs;
}

public List<WordCounter> getPredicates() {
    return predicates;
}
```

Figura 1: Esempio di parentesizzazione

- **Indentazione**: i blocchi innestati devono essere correttamente indentati usando per ciascun livello di indentazione quattro spazi, facendo eccezione per i commenti, come si può vedere nell'esempio sotto. Si consiglia la configurazione automatica dell'indentazione del proprio IDE per rispettare questa regola;

```java
public class SuggestionFrequency implements SuggestionFeedback {

    private Bdl bdl;

    public SuggestionFrequency(Bdl bdl) {
        this.bdl = bdl;
    }

    @Override
    public int findActionInBdl(String nameAction) {
        //predicates research
        for (WordCounter wc : bdl.getPredicates()) {
            if (wc.getWord().equalsIgnoreCase(nameAction.replace( target: "_", replacement: " "))) {
                return wc.getCount();
            }
        }
    return 0;
    }
```

Figura 2: Esempio di indentazione, notazione sui nomi, commento, istruzione condizionale e lingua

- **Univocità dei nomi**: classi, metodi e variabili devono avere un nome univoco ed esplicativo al fine di evitare ambiguità e incomprensione, come si può vedere nell'esempio sopra;
- **Classi e interfacce**: i loro nomi devono presentare la notazione "CamelCase", con la prima lettera maiuscola, e devono essere il più possibile "parlanti", cioè deve essere subito chiara la sua funzione. Si può vedere un esempio nella figura sopra;
- **Metodi e variabili**: i loro nomi devono presentare la notazione "camelCase", con la prima lettera minuscola. Si può vedere un esempio nella figura sopra;
- **Commenti**: ogni metodo difficilmente comprensibile deve presentare un commento che ne spiega la funzione, come nell'esempio della figura sopra;
- **Istruzioni condizionali**: evitare quanto più possibile, se non indispensabile, l'uso di if annidati e la presenza di più di una clausola da testare;
- **Lingua**: il codice, come anche i commenti, deve essere scritto in lingua inglese;
- **Ricorsione**: l'utilizzo della ricorsione va evitato quanto più possibile per evitare rallentamenti o una maggiore occupazione di memoria rispetto a soluzione iterative.

E' consigliabile rispettare le convenzioni di stile per il linguaggio Java esposte nella "Google Java Style Guide". A supporto del coding, viene utilizzato il progetto consigliato dal proponente, "comics.works Studio", la cui repository viene indicata nella sezione riferimenti relativa all'introduzione di questo file. Terminata la codifica di un unità si devono implementare ed eseguire i test funzionali definiti durante la progettazione. Devono essere inoltre definiti ed eseguiti i test di unità strutturali (White-Box Test) per verificare la correttezza della logica interna dell'unità. Per ogni unità dev'essere implementata una batteria di test che copre ogni possibile cammino di esecuzione del codice. I test sulle singole unità vanno sviluppati con *JUnit*. *Maven* deve essere utilizzato per garantire che il codice rispetti i seguenti punti:

- Compili;
- Esegua i test;

- Rispetti le metriche di qualità.

Le metriche da seguire sono specificate nel Piano di qualifica (PQ.1.0.0). Queste devono essere trasformate in regole e definite in un file "checkstyle.xml". Nel file di configurazione di *Maven*, "pom.xml", si deve includere:

- Il file "checkstyle.xml";
- Eventuali librerie di terze parti.

L'inclusione di queste ultime è particolarmente utile alla loro automatizzazione.

2.2.8 Integrazione del software

Lo sviluppo del codice relativo alle unità deve essere svolta secondo la pratica dell'integrazione continua. L'integrazione del software con i cambiamenti effettuati dev'essere svolta spesso, almeno una volta al giorno, assicurando che il codice compili e limitando le possibilità di creare conflitti con le modifiche degli altri. A questo proposito si deve utilizzare *Travis CI*. Per il suo impiego sono necessari i seguenti passi:

1. Creazione di un account;
2. Collegamento con la repository di *GitHub*;
3. Inserimento del file di configurazione ".travis.yml" nella root del progetto.

In seguito ad una corretta configurazione del file, *Travis*, ad ogni nuova push sul repository di *GitHub*, si occuperà autonomamente di eseguire i test di unità e di stile, oltre a creare una build, riportando l'esito dell'esecuzione via e-mail. L'integrazione delle unità sviluppate in parallelo deve essere immediatamente seguita dall'implementazione ed esecuzione dei test di integrazione definiti durante l'attività di design architetturale.

2.2.9 Test di qualifica del software

Conclusa la fase di codifica del software devono essere implementati ed eseguiti i test di sistema definiti durante l'analisi dei requisiti, per verificare che tutti i requisiti siano stati soddisfatti. Nel caso dovessero insorgere delle problematiche, esse vanno gestite tempestivamente secondo quanto previsto dal processo di risoluzione dei problemi. Superati i test di sistema con esito positivo si deve sottoporre il software al collaudo e ai test di accettazione concordati con il proponente.

3 Processi di supporto

3.1 Documentazione

Il processo di documentazione gestisce la produzione e la manutenzione dei documenti durante lo sviluppo del progetto.

3.1.1 Scopo

Gli obiettivi del processo di Documentazione sono i seguenti:

- Definire i documenti in termini di struttura e contenuto;
- Definire tutte le attività relative ai documenti;
- Definire un piano di documentazione per illustrare tutti i documenti prodotti;
- Fornire una guida per l'accesso alle informazioni riportate nei documenti.

3.1.2 Descrizione

Tale processo si compone di quattro attività e ogni attività si compone a sua volta di diversi compiti. Ogni compito ha per responsabile un membro del team. Di seguito sono elencate le attività che compongono il processo di documentazione:

- Design e sviluppo;
- Produzione;
- Manutenzione.

Questa sezione è divisa per attività. Inoltre, sono state dedicate due ulteriori sezioni: una per le convenzioni interne e una per gli strumenti di sviluppo coinvolti nel processo di documentazione.

3.1.3 Tipologia documenti

3.1.3.1 Documenti formali I documenti formali contengono la way of working stabilita dal gruppo. Per questo motivo si è deciso di darli una struttura precisa. Questo tipo di documenti richiede verifica e validazione.

3.1.3.2 Documenti informali I documenti informali servono per condividere idee all'interno del gruppo o come preparazione per la stesura di altri documenti. Questo tipo di documenti sono pensati come usa e getta e hanno forma e struttura che più si adattano alla scopo. Questo tipo di documenti è ad uso interno e non richiede verifica.

3.1.3.3 Verbali Lo scopo dei verbali è quello di tenere traccia delle decisione prese dal gruppo. Per questo tipo di documenti si è deciso di adottare una struttura ad hoc, che metta in risalto il contenuto in modo sintetico.

3.1.4 Struttura

In questa sottosezione è descritto in che modo sono strutturati i documenti.

3.1.4.1 Struttura documenti formali

La struttura del documento completo sarà la seguente

1. Intestazione, all'interno dell'intestazione andranno riportate le seguenti informazioni
   - Titolo del documento;
   - Codice identificativo del documento;
   - Scopo del documento;
   - Riferimenti ad altri documenti o fonti.
2. Registro di versionamento, all'interno del registro di versionamento è contenuto, sotto forma di tabella, lo storico delle modifiche del documento successive alla prima approvazione;
3. Contenuto, diviso appunto in sezioni.

3.1.4.2 Struttura verbali

Tutti i verbali hanno tre paragrafi obbligatori:

- Convocazione, dove vengono riportati data, luogo d'incontro e partecipanti;
- Riassunto, che può essere diviso in sottoparagrafi;
- Decisioni, dove vengono riportate in una tabella tutte le decisioni prese con relativo ID.

Il codice identificativo scelto per questi documenti che evidenzia la data di creazione è il seguente:

- Verbali interni: VI_AAAA-MM-GG;
- Verbali esterni: VE_AAAA-MM-GG;

dove AAAA-MM-GG indica la data in cui si è tenuto l'incontro.

3.1.4.3 Nome dei documenti

Ogni documento viene identificato da un nome univoco, tale nome fa riferimento alla traduzione in italiano dei documenti individuati dagli standard ISO/IEC. Qualora un termine tradotto possa risultare ambiguo si autorizza l'utilizzo del temine in lingua originale, in questo caso il termine andrà inserito nel glossario (es. design). Qualora il documento non sia identificato in modo preciso nello standard si autorizza la creazione di un nome identificativo, anche in questo caso il nome del documento andrà riportato nel glossario (es. piano di documentazione).

3.1.4.4 Codice dei documenti

Ogni documento avrà un codice identificativo, volto a semplificare i riferimenti al documento stesso. Il codice del documento sarà nel seguente formato: **XY.[ID di versionamento]** dove X e Y sono le iniziali del nome del documento. Qualora le iniziali non fossero sufficienti a identificare il documento si autorizza ad aggiungere un'altra iniziale o ad aggiungere una lettera oltre all'iniziale per distinguere i due codici. Esempio: Analisi dei requisiti 0.1.1, codice del documento AR.0.1.1.

3.1.5 Procedure

3.1.5.1 Approvazione

Svolta da un membro del team, in genere il responsabile ma questo compito può essere delegato ad un verificatore. Il responsabile della revisione del documento dovrà verificare che il documento rispetti il formato standard, che faccia uso del glossario e, in caso fossero presenti parole chiave non presenti nel glossario, aggiornare il glossario stesso. L'approvazione finale del documento è svolta sempre dal responsabile e non può essere delegata.

3.1.5.2 Aggiornamento L'aggiornamento si suddivide in tre categorie:

- Strutturale, consiste in modifiche alla struttura del documento. Il compito è svolto dal responsabile o un suo delegato, richiede l'apertura di una *issue* nell'ITS. Il compito consiste nel definire la divisione in sezioni del documento ed eventualmente aggiungere sezioni;
- Contenutistico, comporta cambiamenti al contenuto di una sezione. L'attività è avviata dal responsabile o da un verificatore, i quali individuano le criticità nel documento e invitano la persona incaricata alla stesura del documento a rivederne le criticità o a integrare le carenze della sezione. L'attività si conclude con l'approvazione del documento;
- Formale, comporta la revisione del registro e dei termini utilizzati all'interno di un documento o di una sezione. L'attività è avviata dal responsabile o da un verificatore e svolta dal responsabile dello sviluppo del documento.

3.1.6 Norme tipografiche

3.1.6.1 Elementi di stile

- **Corsivo** utilizzato per evidenziare parole che appartengono al glossario ma sono usate poco frequentemente o possono avere significati differenti. Qualora una parola compaia in corsivo, il suo significato è proprio quello indicato nel glossario. Un'eccezione viene fatta per i termini delle formule matematiche;
- **Grassetto** in grassetto andranno evidenziate le parole chiave all'interno del testo o di un elenco puntato;
- **Elenchi puntati** lo stile degli elenchi è già definito all'interno del template per la stesura dei documenti. Gli elenchi si utilizzano per descrivere sinteticamente le caratteristiche degli elementi di una lista.

3.1.6.2 Sigle ricorrenti Le sigle vanno riportate all'interno del glossario. Per quanto riguarda eventuali riferimenti a documenti vanno utilizzati i codici identificativi.

3.1.6.3 Formati Di seguito una lista di formati che andranno adottati nel processo di documentazione

- Formato data, il formato è il seguente: dd-mm-yyyy;
- Formato ora, il formato è il seguente: hh:mm.

3.1.7 Elementi grafici

In questa sezione sono elencati gli strumenti per gestire elementi grafici all'interno dei documenti e le convenzioni che riguardano l'uso di grafici, diagrammi e immagini. I requisiti che un elemento grafico deve soddisfare prima di essere accettato sono i seguenti

- Il formato deve essere uno dei seguenti: pdf, gan, jpg o png;
- Una legenda deve essere fornita qualora non sia immediata l'identificazione dei dati;
- Una didascalia deve sempre essere presente;
- Ogni elemento grafico deve sempre essere numerato, la numerazione è progressiva; Per la numerazione è sufficiente utilizzare il comando LaTeX "caption";
- Qualora un immagine contenga al suo interno più di un elemento grafico, è richiesto che ad ogni elemento sia associata una lettera, così da identificare univocamente ciascun elemento.

3.1.7.1 Diagrammi UML Per la produzione di diagrammi UML abbiamo scelto di utilizzare draw.io. Il diagramma prodotto andrà esportato in un formato integrabile con LaTeX Shop, per esempio jpg e png.

3.1.7.2 Diagrammi di Gantt Per la produzione dei diagrammi di Gantt utilizziamo il software "GanttProject", con il quale è possibile rappresentare i diagrammi in questione. In seguito alla creazione del diagramma bisogna conservare il file .gan, il quale sarà soggetto a versionamento come parte del documento a cui appartiene. In caso si riscontrino problemi nell'esecuzione del programma, rivolgersi al responsabile.

3.1.7.3 Grafici Per quanto riguarda la creazione di grafici non sono stati definiti strumenti particolari per la loro creazione. Restano da soddisfare i requisiti elencati all'inizio di questo paragrafo.

3.1.8 Piano di documentazione

Il piano di documentazione contiene le specifiche di tutti i documenti prodotti e rappresenta uno strumento per definire la funzione, i destinatari e i responsabili di tutte le attività coinvolte nella produzione di un documento.

3.1.8.1 Analisi dei requisiti - AR

- Lo scopo del documento è identificare le funzionalità che il software dovrà offrire e i requisiti per il soddisfacimento degli stakeholder;
- Documento ad uso esterno;
- Il responsabile dell'Analisi dei requisiti è l'analista.

3.1.8.2 Piano di progetto - PP

- Lo scopo del documento è istanziare i processi del progetto, stimare costi e risorse, pianificare le attività e i compiti coinvolti in un processo e definire ruoli e compiti all'interno del team;
- Documento ad uso esterno;
- Il responsabile del Piano di progetto è l'amministratore.

3.1.8.3 Norme di progetto - NP

- Lo scopo del documento è di descrivere i processi appartenenti al ciclo di vita del prodotto;
- Documento ad uso interno;
- Il referente delle Norme di progetto è l'amministratore con l'approvazione del responsabile.

3.1.8.4 Piano di qualifica - PQ

- Lo scopo del documento è di stabilire regole per garantire qualità nel corso dello sviluppo del progetto;
- Documento ad uso esterno;
- I responsabili del Piano di qualifica sono i verificatori.

3.1.8.5 Glossario

- Lo scopo del documento è spiegare in maggiore dettaglio i termini utilizzati nella documentazione
- Documento ad uso esterno.

Il Glossario è un documento condiviso di supporto al resto della documentazione, perciò non c'è un responsabile unico.

3.1.8.6 Manuale del manutentore - MM

- Lo scopo del documento è di descrivere le scelte progettuali del team e di illustrare come configurare l'ambiente di sviluppo;
- Documento ad uso esterno;
- I responsabili del Manuale sono i programmatori.

3.1.8.7 Manuale utente - MU

- Lo scopo del documento è di guidare l'utente all'utilizzo del prodotto;
- Documento ad uso esterno;
- I responsabili del manuale sono i programmatori.

3.1.9 Strumenti

LaTeX: per documenti formali.
Google docs: per documenti informali e altro materiale.
Draw.io: per la produzione di diagrammi UML.

3.2 Configuration Management

Il processo di configuration management si occupa di gestire il codice prodotto nel ciclo di vita del software per garantire il raggiungimento degli standard di qualità del prodotto software. Questo processo coinvolge attività di gestione, storage e consegna del prodotto software. Il processo di configuration management verrà integrato con l'aggiunta di strumenti nel corso del ciclo di vita del progetto. La presente sezione è divisa in sottosezioni: una dedicata all'obiettivo del processo, una alla definizione di *configuration item* e le restanti alle attività di cui il processo si compone.

3.2.1 Scopo

L'obiettivo di questa sezione è identificare le attività e i compiti coinvolti nel processo di configuration management.

3.2.2 Configuration Items

Con il termine *configuration item* facciamo riferimento ad un elemento coinvolto nel processo di configuration management. Un *configuration item* può essere un componente hardware, software o un documento.

CI	Descrizione	Numero di versione	Dipendenze
Git	VCS, sistema di versionamento	2.20.1 o superiori	Nessuna
Maven	strumento di build automation	3.3.9 o successive (es. 3.5.0)	Nessuna
Java	linguaggio di sviluppo	Java SE 8	Nessuna
JUnit	framework di testing in Java	3.8.1 o successive (in particolare 5.5.2)	JUnit 5 lavora con Java SE 8 o superiori

Tabella 2: Tabella dei *configuration items*

3.2.3 Implementazione del processo

Questa attività ha il compito di stendere un piano per l'attuazione del processo di configuration management. Tale piano dovrà descrivere gli strumenti e le procedure coinvolti in questo processo. Questo compito viene svolto in seguito alle attività di identificazione di configurazione, quindi è svolto ogni qual volta venga integrato un *configuration item*.

3.2.4 Identificazione dei *configuration items*

In questa attività vengono elencati e sottoposti a versionamento i *configuration items*. Questa mansione può svolgersi in più momenti qualora si decida di integrare nuovi *configuration items*. L'attività ha il compito di definire per ogni *configuration item* le relative *baseline* e versioni di riferimento. Questo lavoro è svolto dal responsabile. Per la selezione di un *configuration item* si valutino i seguenti aspetti:

- Dipendenze da altri *configuration items*;
- Suggerimenti del proponente e del committente;
- Costo temporale di configurazione;
- In quale misura può tornare utile.

3.2.5 Registrazione dei cambiamenti

Questa attività ha il compito di registrare e gestire cambiamenti di configurazione nel prodotto software. La modifica di un CI deve passare attraverso una procedura di change request la quale può essere avviata da un membro del team, dal proponente o dal committente.

3.2.5.1 Change request

Richiesta di modifica ad un CI, da presentare al responsabile a cui spetta l'approvazione. La struttura di una change request è la seguente:

- Autore della richiesta;
- Motivo della richiesta;
- Urgenza (bassa, media o alta);
- Stima di fattibilità, contiene informazioni sul costo della modifica, in particolare:
  - CI coinvolti nella modifica;
  - Valutazione dell'impatto della modifica;
  - Stima del costo della modifica (temporale).

3.2.5.2 Approvazione della richiesta

È compito del responsabile accettare o meno le richieste a lui sottoposte in base alle stime. In particolare il responsabile dovrà tener conto dei seguenti aspetti:

- Costo di adozione della nuova configurazione;
- Impatto sui processi del team;
- Vantaggio apportato dalla nuova configurazione.

3.2.5.3 Tracking della modifica
Le modifiche ai CI sono sottoposte a monitoraggio e tracciate tramite l'ITS di GitHub (vedasi processo di Gestione nel presente documento). Lo stato di avanzamento della change request è quindi registrato all'interno di una issue aperta dal responsabile.

3.2.6 Determinare il soddisfacimento dei requisiti

Questa attività è svolta parallelamente alle attività di verifica e validazione. Il suo compito è stabilire se la completezza del prodotto software è tale da potersi candidare a *baseline*. La verifica dei requisiti dovrà determinare:

- Il soddisfacimento dei requisiti funzionali;
- L'associazione di una baseline ad un insieme di requisiti soddisfatti.

3.2.6.1 Candidatura per una baseline
La candidatura a baseline avviene in seguito al soddisfacimento di un requisito funzionale. La pianificazione delle baseline segue il calendario del Piano di progetto, dove ad ogni incremento è associato lo sviluppo di una funzionalità. La verifica per il raggiungimento di una baseline è l'esito di un processo di build.

3.2.7 Rilascio del prodotto

Questa attività ha il compito di rilasciare e consegnare il prodotto software rispettando i criteri formali e i vincoli (contrattuali e progettuali) sottoscritti dal team.

3.2.7.1 Rilascio del prodotto al committente
Il prodotto andrà consegnato secondo le modalità stipulate con il committente:

- Il prodotto finito andrà consegnato su un supporto fisico;
- Il prodotto andrà consegnato in due copie;
- Il prodotto comprende anche la documentazione scritta dai membri del team.

3.2.7.2 Rilascio del prodotto al proponente
Come concordato con il proponente, il codice sorgente sviluppato è open source. In particolare:

- Il prodotto software è disponibile su una repository pubblica su GitHub (github.com/fourcatsteam/NaturalAPI);
- Il link alla repository è condiviso con il proponente e con il committente.

3.3 Gestione della qualità

3.3.1 Scopo

Lo scopo del processo di gestione della qualità è di garantire che il prodotto software e i processi attuati siano conformi ai bisogni del proponente e che rispettino gli obiettivi di qualità.
Inoltre ci si assicura la riproducibilità dei prodotti ottenuti e che questi siano stati sottoposti a cicli di verifica.
Attraverso la gestione della qualità si intende ottenere:

- Qualità nel prodotto tramite standard di qualità, metodologie, procedure e tools;
- Qualità nell'organizzazione e nei suoi processi tramite procedure di revisione e coordinamento;
- Qualità nei documenti tramite procedure per l'identificazione, la raccolta, l'archiviazione e la manutenzione;
- Soddisfazione finale di cliente e proponente.

3.3.2 Descrizione

Per la trattazione approfondita della gestione della qualità si fa riferimento al PQ.1.0.0, dove sono descritte le modalità utilizzate per garantire la qualità nello sviluppo del progetto. In particolare, in tale documento:

- Sono individuati gli attributi del software più significativi per il progetto;
- Sono individuati i valori di misurazione che il team vuole soddisfare.

In questo caso si mira ad ottenere software e documentazione di qualità soddisfacente.
In particolare, il processo assicura l'esecuzione di attività e compiti di controllo della qualità, sia programmati che continui, per l'intero ciclo di vita del software. Durante queste attività si svolgeranno delle procedure interne atte al calcolo e alla verifica delle metriche prese in esame.

3.3.3 Qualità di prodotto

3.3.3.1 Documenti

Al fine di mantenere una elevata qualità e leggibilità nei documenti prodotti, il team prevede di applicare le seguenti metriche.

Metriche

Nome	Descrizione
Indice di Gulpease	È un indice di leggibilità del testo regolato sulla lingua italiana. Considera la lunghezza delle parole, il numero delle frasi ed il numero delle parole totali. Il valore risultante è un numero intero compreso tra 0 e 100, dove un indice più alto corrisponde ad un indice di leggibilità migliore. $IG = 89 + \frac{300 \cdot (Numero\ Di\ Frasi) - 10 \cdot (Numero\ Di\ Lettere)}{Numero\ Di\ Parole}$ Calcolato con *script python.*

3.3.3.2 Software

Per la qualità di prodotto si prende come riferimento lo standard ISO/IEC 9126 che fornisce un modello per valutare tutte le caratteristiche da tenere in considerazione per produrre un prodotto di qualità.
Per ogni caratteristica vengono descritti:

- Gli obiettivi da perseguire;
- Le metriche utilizzate.

Le caratteristiche sono:

3.3.3.3 Funzionalità

È la capacità del prodotto di fornire le funzioni, espresse ed implicite, necessarie per operare in determinate condizioni, cioè in un determinato contesto, rilevate durante l'analisi dei requisiti.
Il gruppo Fourcats pone come obiettivi:

1. Adeguatezza: capacità di fornire un appropriato insieme di funzioni che permettano agli utenti di svolgere determinati task e di raggiungere gli obiettivi prefissati;
2. Accuratezza: capacità di fornire i risultati o gli effetti attesi con il livello di precisioni richiesta;
3. Aderenza alla funzionalità: il prodotto deve aderire a determinati standard, convenzioni e regolamenti.

Metriche

Nome	Descrizione
Completezza dell'implementazione	Indica, in percentuale, la completezza del prodotto e il rispetto dei requisiti. $$C = (1 - \frac{N_{FNI}}{N_{FI}}) \cdot 100$$ dove N_{FNI} indica il numero di funzionalità non implementate e N_{FI} indica il numero di funzionalità individuate dall'analisi.
PROS: Percentuale Requisiti Obbligatori Soddisfatti	Indica la percentuale di requisiti obbligatori soddisfatti. $$PROS = \frac{Requisiti\ Obbligatori\ Soddisfatti}{Requisiti\ Obbligatori\ Totali} \cdot 100.$$
PRD: Percentuale Requisiti Desiderabili	Indica la percentuale dei requisiti desiderabili soddisfatti. $$PRD = \frac{Requisiti\ Desiderabili\ Soddisfatti}{Requisiti\ Desiderabili\ Totali} \cdot 100.$$
PROD: Percentuale Requisiti Opzionali e Desiderabili	Indica la percentuale dei requisiti opzionali e desiderabili soddisfatti. $$PROD = \frac{Requisiti\ Opz\ Soddisfatti + Requisiti\ Des\ Soddisfatti}{Requisiti\ Opzionali\ e\ Desiderabili\ Totali} \cdot 100.$$

3.3.3.4 Affidabilità Capacità del prodotto di mantenere il livello di prestazione anche quando viene utilizzato in condizioni anomale o critiche. Il gruppo Fourcats pone come obiettivi:

1. Maturità: capacità di evitare che si verifichino errori o siano prodotti risultati non corretti in fase di esecuzione;
2. Tolleranza ai guasti: capacità di mantenere il livello di prestazioni in caso di errori nel software o di un uso scorretto del prodotto.

Metriche

Nome	Descrizione

Condition coverage	Su ogni riga di codice contenente alcune espressioni booleane, la copertura della condizione risponde semplicemente alla seguente domanda: "Ogni espressione booleana è stata valutata sia vera che falsa?". Questa è la densità delle possibili condizioni nelle strutture di controllo del flusso che sono state seguite durante l'esecuzione dei test unitari. $$Condition\ coverage = \frac{CT + CF}{2 * B}$$ dove: • CT = condizioni valutate 'vero' almeno una volta; • CF = condizioni valutate 'false' almeno una volta; • B = numero totale di condizioni. Calcolato con *sonarqube*.
Line coverage	Su una determinata riga di codice, la copertura delle linee risponde semplicemente alla seguente domanda: questa riga di codice è stata eseguita durante l'esecuzione dei test unitari? È la densità delle linee coperte dai test unitari: $$Line\ coverage = \frac{LC}{EL}.$$ dove: • LC = linee coperte (linee da coprire - linee non coperte); • EL = numero totale di linee eseguibili. Calcolato con *sonarqube*.
Lines to cover	Numero di righe di codice che potrebbero essere coperte da unit test (ad esempio, le righe vuote o le righe di commento complete non sono considerate righe da coprire). Calcolato con *sonarqube*.
Uncovered lines	Numero di righe di codice che non sono coperte dai test unitari. Calcolato con *sonarqube*.

Coverage	È un mix di copertura della linea e copertura delle condizioni. Il suo obiettivo è fornire una risposta ancora più accurata alla seguente domanda: quanto del codice sorgente è stato coperto dai test unitari? $$Coverage = \frac{CT + CF + LC}{2 * B + EL}$$ dove: • CT = condizioni valutate 'vero' almeno una volta; • CF = condizioni valutate 'false' almeno una volta.; • LC = linee coperte (lines to cover - uncovered lines); • B = numero totale di condizioni; • EL = numero totale di linee eseguibili. Calcolato con *sonarqube*.
Unit test errors	Numero di test che hanno fallito. Calcolato con *sonarqube*.
Unit test failures	Numero di test che hanno fallito per motivi non attesi. Calcolato con *sonarqube*.
Unit test success density	$$\frac{Unit\ tests\ - (Unit\ test\ errors\ + Unit\ test\ failuers\ )}{Unit\ tests} \cdot 100$$ Calcolato con *sonarqube*.

3.3.3.5 Manutenibilità Capacità del prodotto di essere modificato in modo da essere migliorato, corretto o adattato negli ambienti, nei requisiti e nelle specifiche funzionali. Il gruppo Fourcats pone come obiettivi:

1. Analizzabilità: capacità di analizzare il codice ed individuare le cause di errori o malfunzionamenti;
2. Modificabilità: capacità di consentire lo sviluppo di modifiche al software originale.

Metriche

Nome	Descrizione
Facilità di comprensione	La facilità con cui è possibile comprendere cosa fa il codice può essere rappresentata dalla percentuale del numero di linee di commento nel codice. $$FC = 1 - \frac{Comment\ lines}{Lines\ of\ code\ + Comment\ lines}$$ Calcolato con *sonarqube*.

Complessità ciclomatica	È una metrica software che indica la complessità di un programma misurando il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. In tale grafo, i nodi rappresentano unità atomiche di istruzioni, mentre gli archi connettono due nodi se le istruzioni ad esse associate possono essere eseguite in modo consecutivo. Calcolato con *sonarqube*.
CBO: Accoppiamento tra classi	Indica l'accoppiamento di una classe ad una seconda classe. Una classe è accoppiata ad una seconda se usa metodi o variabili definiti nella seconda. Un eccessivo accoppiamento è negativo per la modularità, per il riuso e per la manutenibilità del codice. Le classi A e B sono accoppiate se: • A ha un attributo che si riferisce o è di tipo B; • A chiama un servizio di un oggetto B; • A ha un metodo che referenzia B (come tipo restituito o parametro); • A ha una variabile locale di tipo B; • A è una sotto-classe di (o implementa) la classe B. Calcolato con *codemr*.
Code Smell	Il Code smell viene usato per indicare una serie di caratteristiche che il codice sorgente può avere e che sono generalmente riconosciute come probabili indicazioni di un difetto di programmazione. Vengono individuati da *sonarqube*.
Technical Debt Ratio	Rapporto tra il costo di sviluppo e il costo di correzione del software. $$TDR = \frac{Remediation\ cost}{Development\ cost}$$ dove: • Remediation cost =tempo per correggere il codice; • Development cost = costo per una linea di codice * numero di linee di codice. Dove il costo di una linea di codice è 0.06 giorni. Calcolato con *sonarqube*.

Maintainability Rating	Giudizio data al progetto in base al valore del technical debt ratio. La scala di giudizio è: • A = $\leq$ 5% del tempo speso per la correzione; • B = tra 6-10% del tempo speso per la correzione; • C = tra 11-20% del tempo speso per la correzione; • D = tra 21-50% del tempo speso per la correzione; • E = $>$ 50% del tempo speso per la correzione. Calcolato con *sonarqube*.
WMC: Metodi pesati per classe	Numero di metodi di una classe, pesati dalla complessità ciclomatica di ciascun metodo. Quanto più alto è il valore di questa metrica, tanto più complessa è la classe. Valori alti indicano una dimensione eccessiva della classe e scarsa riusabilità dei metodi. Calcolato con *codemr*.
DIT: Profondità dell'albero di ereditarietà	La posizione della classe nell'albero di ereditarietà. Ha valore 0 (zero) per le classi root e non ereditate. Per l'ereditarietà multipla, la metrica mostra la lunghezza massima. Le classe più profonda nella struttura ereditaria, probabilmente ereditano. Pertanto, è più difficile prevederne il comportamento. Anche questa classe relativamente complessa da sviluppare, testare e mantenere. Calcolato con *codemr*.
NOC: Numero di figli	Il numero di sottoclassi dirette di una classe. La dimensione del NOC indica approssimativamente come un'applicazione si riutilizza. Si presume che maggiore è il numero di figli di una classe, maggiore è la responsabilità del manutentore della classe di non interrompere il comportamento dei figli. Di conseguenza, è più difficile modificare la classe e richiede ulteriori test. Calcolato con *codemr*.
RFC: Risposta per classe	Il numero dei metodi che possono essere potenzialmente invocati in risposta a un messaggio pubblico ricevuto da un oggetto di una particolare classe. Include il grafico di chiamata completo di qualsiasi metodo chiamato dal metodo di origine. Se il numero di metodi che possono essere invocati in una classe è elevato, la classe è considerata più complessa e può essere altamente accoppiata ad altre classi. Pertanto sono necessari ulteriori test e manutenzione. Calcolato con*codemr*.

LCOM: Mancanza di coesione nei metodi	Misura in che modo i metodi di una classe sono correlati tra loro. La bassa coesione significa che la classe attua più di una responsabilità. Una richiesta di modifica da parte di un bug o di una nuova funzionalità, su una di queste responsabilità comporterà il cambio di quella classe. La mancanza di coesione influenza anche la comprensibilità e implica che le classi dovrebbero probabilmente essere suddivise in due o più sottoclassi. $$LCOM3 = \frac{m - \frac{somma(mA)}{a}}{m - 1}$$ dove: • m numero di procedure (metodi) in classe; • un numero di variabili (attributi) in classe. a contiene tutte le variabili condivise (statiche) o meno; • numero mA di metodi che accedono a una variabile (attributo); • somma (mA) somma di mA rispetto agli attributi di una classe. Calcolato con *codemr*.

3.3.4 Qualità dei processi

3.3.4.1 Processo di gestione

Nome	Descrizione
BAC: Budget At Completion	Indica la somma di tutti i valori del budget che sono stati precedentemente stabiliti per il lavoro da eseguire sul progetto. Il BAC può essere individuato come il valore totale pianificato del progetto. La sua misurazione è un numero intero.
SV: Schedule Variance	Indica l'andamento del progetto. In particolare: denota lo stato di anticipo o di ritardo sullo svolgimento del progetto in base alla pianificazione. Si calcola mediante l'uso della formula: $SV = EV - PV$ dove EV sta per "Earned Value" e PV per "Planned Value".
EV: Earned Value	È il valore delle attività e del lavoro realizzato alla data corrente. Corrisponde al denaro guadagnato fino a quel momento. **PV: Planned Value** Si tratta del costo pianificato, in euro, per realizzare le attività di progetto alla data corrente. Corrisponde al preventivo di periodo.
AC: Actual Cost	Indica il denaro speso fino al momento del calcolo. La sua misurazione è un numero intero.
CV: Cost Variance	Indica la linea di base dei costi del progetto. Individua se in quel momento si sta sforando il budget o meno: se il suo valore è positivo si è sotto budget, se negativo sopra budget. Si misura così: $CV = EV - AC$.
BV: Budget Variance	Indica se si è speso di più o di meno rispetto al budget previsto alla data corrente. Per calcolare questa metrica si utilizza la formula: $BV = PV - AC$ Se il suo valore è maggiore di zero, significa che il progetto spende il budget meno velocemente di quanto pianificato, il contrario se negativo.
EAC: Estimated at Completion	Indica la revisione del valore stimato per la realizzazione del progetto ovvero il costo previsto del progetto, man mano che il progetto procede. Un valore dell'EAC negativo indica che il progetto sembra rientrare nella cifra di base del budget originale. Un valore positivo indica invece che sia in eccesso di spesa rispetto alla cifra di base. Si calcola: $EAC = AC + ETC$ dove ETC = *Valore stimato per la realizzazione delle rimanenti attività necessarie al completamento del progetto.*
VAC: Variance at Completion	Indica la proiezione dell'eccedenza o del deficit di un bilancio. È indicata come differenza tra BAC e EAC. Si calcola con: $VAC = BAC - EAC$. Se il valore del VAC è positivo, indica che il progetto è sotto budget. Se il risultato è negativo, allora significa che il progetto è fuori budget.

3.4 Verifica

3.4.1 Scopo

Il processo di verifica accerta che l'esecuzione delle attività attuate nel periodo in esame non abbia introdotto errori. È costituito da attività e compiti atti a determinare se i requisiti del prodotto sono completi e corretti e se essi sono soddisfatti.

Il Verificatore e tutto il team hanno il compito di rispettare ogni norma definita in questo processo al fine di ottenere prodotti corretti, coesi, completi e in linea con le modalità di lavoro stabilite.

3.4.2 Descrizione

Il processo, dopo aver verificato il prodotto, lo restituisce in output conforme alle aspettative. Per ottenere tale risultato ci si affida ad attività di analisi e test.

3.4.3 Verifica processo

I processi vengono verificati tenendo conto dei criteri elencati di seguito:

- I requisiti di pianificazione del progetto devono essere adeguati e tempestivi;
- I processi selezionati per il progetto sono adeguati, attuati ed eseguiti come previsto;
- Gli standard, le procedure e gli ambienti per i processi del progetto sono adeguati.

Periodicamente verranno eseguiti dei controlli su ciascun processo, per poter verificare input e output. In particolare si analizzeranno:

- Gli output dei processi: che siano adeguati;
- Le risorse utilizzate: che non ci siano sprechi.

Per determinare l'andamento corretto del processo, si controllerà inoltre:

- Che il numero di bug identificati con *SonarQube* non aumenti;
- Il tempo dedicato per un processo o un'attività tramite l'uso di *Clockify*.

Sarà inoltre compito del Responsabile di progetto controllare i processi in modo da poter identificare i problemi e poterli adeguatamente sistemare.

3.4.4 Verifica requisiti

Per l'attività di verifica dei requisiti vengono utilizzate le seguenti tecniche:

- **Walkthrough**: strategia per una lettura preliminare dei prodotti non efficiente. Consiste in una lettura a largo spettro senza presupposti e solitamente effettuata da tutti i componenti del team per cercare anomalie, senza sapere inizialmente se vi siano difetti. La lettura è accompagnata da una discussione collettiva a cui seguono opportune correzioni;
- **Checklist**: si verificano le caratteristiche dei requisiti mediante queste domande:
  1. Identificabili: i requisiti hanno un ID?
  2. Tracciabili: ciascun requisito è mappato all'origine? Si può capire come è stato ricavato?
  3. Verificabili: è possibile ricavare un test per controllarlo?
  4. Classificabili: i requisiti sono stati classificati? In che modo?
  5. Completi: i requisiti includono tutte le funzioni e i vincoli richiesti dal proponente?
  6. Realizzabili: è possibile implementare qualcosa per soddisfare i requisiti?
  7. Consistenti: i requisiti non dovrebbero avere conflitti.
  8. Validi: i requisiti soddisfano i bisogni del proponente.
- **Dialogo con il proponente**: si apre una discussione su *Slack* con il proponente per avere conferma sui requisiti ricavati.
- **Stesura test**: Per valutare se un requisito è utile e fattibile, è stata stilata una lista di test.

3.4.5 Verifica codice

3.4.5.1 Analisi statica

L'analisi statica è una tecnica che non richiede l'esecuzione del software. Effettua controlli attraverso uno studio del codice e della documentazione per valutare la correttezza (intesa come assenza di errori e difetti), la conformità alle regole e la coesione dei componenti. Viene utilizzata durante tutto lo svolgimento del progetto tramite una delle seguenti strategie:

- **Inspection**: strategia efficiente per una lettura mirata dei prodotti. Consiste in una lettura dettagliata basata su errori presupposti appoggiandosi a risultati di test automatici;
- **Analisi di flusso di controllo**: strategia per localizzare codice non raggiungibile e cicli non terminanti. Inoltre si accerta la logica, la visibilità e la propagazione;
- **Analisi di flusso di dati**: strategia per accertare che nessun cammino d'esecuzione del programma acceda a variabili prive di valore. Permette di rilevare possibili anomalie e la violazione dell'incapsulazione;
- **Verifica formale del codice**: strategia per provare la correttezza del codice sorgente rispetto alla specifica algebrica dei requisiti e la correttezza parziale del programma;
- **Analisi di flusso d'informazione**: strategia per determinare quali dipendenze tra ingressi e uscite risultino dall'esecuzione di una unità di codice;
- **Analisi di limite**: strategia per verificare che i dati del programma restino entro i limiti del loro tipo e della precisione desiderata;
- **Analisi d'uso di stack**: strategia per determinare la massima domanda di stack richiesta a tempo d'esecuzione in relazione con la dimensione della memoria assegnata al processo. Verifica che non vi sia rischio di collisione tra stack e heap per qualche esecuzione;
- **Analisi temporale**: strategia per studiare le dipendenze temporali tra le uscite del programma e i suoi ingressi;
- **Analisi d'interferenza**: strategia per mostrare l'assenza di effetti di interferenza tra parti isolate del sistema.

3.4.5.2 Analisi dinamica

L'analisi dinamica è una tecnica di analisi del prodotto software che richiede la sua esecuzione.
Viene svolta mediante dei test che verificano il funzionamento del prodotto e cercano possibili difetti d'implementazione. I test sono una parte essenziale nell'analisi dinamica perché controllano che il codice scritto funzioni correttamente. Per questo motivo devono essere definiti, per ciascuno di essi, i seguenti parametri:

- **Ambiente**: sistema hardware e software sul quale verrà eseguito il test;
- **Stato iniziale**: lo stato di partenza del prodotto prima dell'esecuzione del test;
- **Input**: l'input inserito;
- **Output**: l'output atteso;
- **Istruzioni aggiuntive**.

Esistono diverse tipologie di test, ognuno dei quali ha un diverso oggetto di verifica e scopo.

- **Test di unità**
  I test di unità si eseguono su singole unità di software, dove per unità si intende il minimo componente di un programma dotato di funzionamento autonomo.
  Questi test si concentrano sul funzionamento di queste unità individuali.
  I test devono seguire le proprietà **A TRIP**, ossia devono essere:

- **Automatic**: automatici, deve essere presente un'automazione che permette di eseguire il test in modo completamente automatico. Inoltre devono durare pochi secondi e non richiedere attività umana;
- **Thorough**: esaustivi e accurati, devono verificare il comportamento di qualsiasi parte del progetto che potrebbe creare degli errori;
- **Repeatable**: ripetibili, devono produrre sempre lo stesso risultato. Per considerarsi ripetibili, i test devono essere:
  - Indipendenti dall'ordine di esecuzione: l'ordine dei test non deve influire sul risultato;
  - Indipendenti dall'ambiente di esecuzione: non deve dipendere da risorse esterne.
- **Indipendent**: indipendenti dall'ambiente di esecuzione, da elementi esterni e dall'ordine di esecuzione;
- **Professional**: devono essere scritti e mantenuti con la stessa professionalità del codice di produzione.

Oltre alle proprietà A TRIP, i test di unità seguiranno i principi del **RIGHT BICEP**:

- **Are the results *RIGHT*?**: verificare se i risultati che vengono prodotti siano corretti. Per corretto si intende che il risultato atteso sia uguale al risultato prodotto dall'unità;
- ***B*oundary conditions**: vengono identificate e verificate le condizioni limite (es. valori conformi al formato atteso, valori all'interno di un range);
- **check *I*nverse relationship**: alcune unità possono essere verificate tramite l'applicazione della loro funzionalità inversa;
- ***C*ross-check using other means**: utilizzo di uno strumento esistente per verificare se la nuova unità ha lo stesso comportamento;
- **force *E*rror condition**: ricreare condizioni di errore e verificare che il prodotto funzioni come ci si aspetta;
- ***P*erformance characteristic**: i test devono essere veloci per poter essere eseguiti molto spesso.

- **Test di integrazione**
  Le unità, dopo aver superato i test, vengono assemblate formando una porzione più grande del software.
  I test di integrazione si concentrano sulle interfacce tra le componenti: prevedono l'analisi di questa porzione, in modo da verificare che le unità messe assieme funzionino correttamente.
  Ogni volta che una porzione supera il test di integrazione costituisce una nuova unità per una porzione di grandezza maggiore.
  Questa procedura viene iterata fino al raggiungimento della dimensione totale del sistema. I test di integrazione si dividono in due categorie:
  - Test di integrazione a scope ridotto, ovvero test che coinvolgono un particolare modulo dell'architettura.
  - Test di integrazione ad ampio scope, ovvero test che coinvolgono diverse classi appartenenti a package o moduli diversi. Con questi test si vuole testare le funzionalità più che i moduli del prodotto.

- **Test di sistema**
  Il sistema viene testato dopo aver integrato tutte le sue componenti: si controlla l'interazione tra le parti e tutte le funzionalità del sistema.
  Viene verificato che soddisfi a pieno i requisiti e rispetti le specifiche definite nell'Analisi dei requisiti.

- **Test di accettazione** Questo tipo di test viene eseguito in presenza del committente. Viene svolto un collaudo verificando il gradimento del cliente.
  Se il test viene superato, il prodotto è pronto per il rilascio.

3.4.5.3 Specifica dei test Per assicurare la qualità del software prodotto, viene adottato il "Modello a V", il quale mette in risalto le relazioni tra ogni fase del ciclo di vita di sviluppo del software e la fase di testing ad esso associata. Prevede che lo sviluppo dei test cominci già all'inizio del progetto, in parallelo alle attività di analisi e progettazione, consentendo così di risparmiare un ampio tempo di progetto.

Ogni test avrà un nome univoco identificativo, volto a semplificarne la comprensione. Di seguito la specifica nomenclatura dei test per ogni categoria.

Test di unità e test di integrazione a scope ridotto
Questi test verranno indicati nel seguente modo:

T[Tipo][ID della classe][Codice]

dove:

- **Tipo** può assumere i seguenti valori:
  - U per i test di unità;
  - I per i test di integrazione a scope ridotto.
- **ID della classe** indica la classe alla quale è associato il test; Per ogni modulo andrà specificato un codice associato ad ogni classe, da riportare sotto forma di tabella.
- **Codice** è un numero intero incrementale.

Test di integrazione ad ampio scope, test di sistema e test d'accettazione
Questi test verranno indicati nel seguente modo:

T[Tipo][Use Case di riferimento][Codice]

dove:

- **Tipo** può assumere i seguenti valori:
  - I per i test di integrazione ad ampio scope;
  - S per i test di sistema;
  - A per i test di accettazione;
- **Use Case di riferimento** indica il caso d'uso al quale è associato il test;
- **Codice** è un numero intero incrementale.

Stato del test
I test inoltre hanno uno **stato**, che può assumere i seguenti valori:

- **I**: Indica che il test è stato implementato;
- **NI**: Indica che il test non è ancora stato implementato;
- **S**: Indica che il test è stato superato;
- **NS**: Indica che il test non è stato superato.

3.4.5.4 Strumenti per la verifica del codice

Per l'analisi statica del codice si utilizzerà lo strumento *SonarQube*, una piattaforma software per la revisione automatica del codice sorgente e controllo continuo della sua qualità, utile per trovare "codesmells", ossia debolezze di progettazione che riducono la qualità del software.
SonarQube permette, inoltre, di definire dei "Quality Gates", ovvero, una serie di misure di soglia sul progetto. Queste possono essere:

- La copertura del codice;
- La misura del debito tecnico;
- Il numero di problemi critici/di blocco;
- La valutazione della sicurezza;
- Il tasso di superamento dei test unitari.

Per l'analisi dinamica del codice si utilizzerà *Maven*, e *Travis CI*.
Maven è uno strumento di gestione di progetti software. Tramite questo tools si avrà la possibilità di eseguire automaticamente i test di unità.
Travis CI, invece, permette di implementare la pratica dell'integrazione continua. Questa consiste nell'allineamento frequente degli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso.

3.4.6 Verifica documentazione

Al fine di verificare la documentazione si effettueranno dei controlli mediante le tecniche di analisi statica walkthrough e inspection:

- **Walkthrough**: strategia per una lettura preliminare dei prodotti non efficiente. Consiste in una lettura a largo spettro senza presupposti e solitamente effettuata da tutti i componenti del team per cercare anomalie, senza sapere inizialmente se vi siano difetti. La lettura è accompagnata da una discussione collettiva a cui seguono opportune correzioni;
- **Inspection**: strategia efficiente per una lettura mirata dei prodotti. Consiste in una lettura dettagliata basata su errori presupposti appoggiandosi ad una lista stilata precedentemente. Normalmente viene effettuata da una sola persona.

A seguire è descritta una prima lista di controllo utilizzabile per le ispezioni. Con la progressione di ispezioni ed errori riscontrati, è previsto un ampliamento della stessa.

Controlli	Obiettivo
Sintassi	Le frasi devono essere semplici e chiare
Parte Mancante	Il documento deve essere uniforme, senza spazi vuoti o sezioni mancanti
Elenchi	Ogni voce deve iniziare con la lettera maiuscola
Punteggiatura elenchi	Ogni voce deve terminare con ";" eccetto l'ultima che termina in "."
Termini glossario	Il riferimento a termini di glossario devono essere espressi in corsivo
Nome gruppo	Il nome del gruppo deve essere scritto così: FourCats
Ruoli	Il nome dei ruoli deve essere scritto in minuscolo
ID documenti	I documenti si identificano con [SiglaIniziali].[Versione]

Tabella 6: Ispezioni

3.4.6.1 Strumenti per la verifica della documentazione

La documentazione viene stilata utilizzando il linguaggio LaTeX. Per la verifica ortografica vengono utilizzati gli strumenti integrati all'interno degli IDE Texmaker, TEXstudio e TEXshop. Per il calcolo dell'indice di Gulpease il team utilizza uno script Python che analizza documenti pdf.

3.5 Validazione

3.5.1 Scopo

Questo processo ha lo scopo di confermare che i requisiti vengano rispettati quando uno specifico prodotto è utilizzato nell'ambiente destinatario: soddisfa l'utilizzo per cui è stato creato.
Con la validazione il team garantisce che i prodotti verificati siano effettivamente conformi e che soddisfino i bisogni del committente.

3.5.2 Procedure

Per effettuare la validazione è necessario che i test di unità e di integrazione abbiano esito positivo. La validazione dev'essere svolta secondo il seguente procedimento:

1. I verificatori effettuano i test di sistema;
2. L'esito dei test viene registrato in un documento;
3. Il responsabile valuta l'esito dei test.

Se l'esito dei test è soddisfacente il responsabile sottopone al proponente i risultati. Altrimenti il responsabile fornisce al gruppo indicazioni sulle modifiche da effettuare per le criticità emerse dai test.

3.6 Joint Review

3.6.1 Scopo

Lo scopo del processo è la valutazione dello stato e dei prodotti di un'attività di progetto.

3.6.2 Descrizione

Una Joint Review consiste in una revisione, o meglio, in un confronto con il proponente per verificare ciò che è stato realizzato, al fine di restituire un feedback, una proposta, una soluzione per supportare l'avanzamento di un'attività di progetto.

3.6.3 Attività

3.6.3.1 Implementazione

In una Joint Review sono previsti i seguenti punti:

- **Convocazione**: La revisione viene effettuata sia in base a delle milestone prestabilite, come specificato nel Piano di progetto, sia quando viene ritenuto necessario. Per assicurare un corretto avanzamento del progetto, infatti, è indispensabile eseguire delle revisioni durante l'intera durata del contratto. Il compito di convocarle è delegato al responsabile, in seguito ad aver discusso con il gruppo di eventuali domande e/o problematiche da porre al proponente e averle riportate in un documento condiviso su *Drive* intitolato "GG_MM_AA_Domande". La convocazione deve avvenire attraverso canali di comunicazione dedicati con il proponente (es. *Slack* o e-mail). In questo task le due parti devono concordare le seguenti voci:
  - **Data**: decisa in base alle disponibilità dei membri del gruppo e del proponente;
  - **Motivo**: ragione principale della richiesta.
- **Risorse**: In seguito alla convocazione, le risorse necessarie per condurre la revisione devono essere concordate tra le parti. Il responsabile, dopo aver discusso con gli altri componenti, si deve accordare con il proponente per decidere quali risorse siano necessarie per lo svolgimento della revisione. Questo include quali membri del gruppo ne prendono parte, in quale struttura o in quale piattaforma di comunicazione svolgerla, o, ancora, quali software e/o strumenti possano essere necessari per il corretto svolgimento del processo. Si devono quindi decidere i seguenti punti:
  - **Partecipanti**: membri interessati nella revisione;
  - **Ora di inizio**: decisa in base alle disponibilità dei partecipanti;
  - **Luogo e/o piattaforma di incontro**: decisione che deve soddisfare le comodità dei partecipanti (es. Aula didattica TA 1BC50 in collegamento Skype con il proponente);
  - **Strumenti e/o software necessari**: eventuali strumenti o software richiesti per le finalità dell'incontro.
- **Ordine del giorno**: All'inizio di ogni revisione vanno concordati con il proponente i seguenti punti: ordine del giorno della riunione, prodotti software (risultati di un'attività) e problemi da esaminare; scopi e procedure.
  - **Prodotti di attività e problemi da esaminare**: i membri del gruppo devono esporre le loro perplessità e mostrare ciò che hanno realizzato con spirito critico, al fine di aiutare il proponente ad individuare eventuali problematiche.
- **Problematiche**: I problemi rilevati durante la revisione devono essere registrati e gestiti come specificato nel processo di risoluzione dei problemi (vedi 3.7);
- **Risultati**: I risultati della revisione devono essere documentati e distribuiti su *Drive*, sotto forma di verbale. Per la struttura da seguire, riferirsi al processo di documentazione.

3.6.3.2 Revisione della gestione del progetto

Lo stato di avanzamento del progetto deve essere valutato in relazione al Piano di progetto, alle scadenze, agli standard e alle linee guida applicabili. L'esito della revisione dovrebbe essere discusso tra le due parti e prevedere quanto segue:

- **Gestione delle attività**: Fare progredire le attività secondo i piani, sulla base di una valutazione dello stato dell'attività o del prodotto software;
- **Allocazione delle risorse**: Mantenere il controllo globale del progetto attraverso un'adeguata allocazione delle risorse;
- **Pianificazione**: Cambiare la direzione del progetto o determinare la necessità di una pianificazione alternativa;
- **Rischi**: Valutazione e gestione dei rischi che possono compromettere il successo del progetto.

3.6.3.3 Revisioni tecniche

Più nello specifico, in una Joint Review, vengono eseguite delle revisioni tecniche per valutare i prodotti software o i servizi in questione al fine di fornire la prova che:

- Sono completi;
- Sono conformi alle loro norme e specifiche;
- Sono candidabili a baseline;
- Le modifiche ad essi sono attuate correttamente e riguardano solo le aree identificate dall'autorità competente (vedi 3.2, processo di Configuration Management);
- Lo sviluppo è condotto secondo i piani, i tempi, le norme e le linee guida del progetto.

3.7 Risoluzione dei problemi

3.7.1 Scopo

Lo scopo di questo processo è quello di fornire un mezzo tempestivo, responsabile e documentato per assicurare che tutti i problemi scoperti siano analizzati e risolti.

3.7.2 Descrizione

Il processo favorisce una gestione più efficiente delle problematiche (comprese le non conformità), indipendentemente dalla loro natura o fonte, per le quali non vi è una soluzione immediatamente applicabile. Queste problematiche possono essere individuate durante l'esecuzione dello sviluppo, della verifica o di altri processi. L'idea è quella di fornire una base di partenza comune per la definizione dei problemi e favorire la risoluzione degli stessi.

3.7.3 Definizione del problema

Il problema individuato deve essere prontamente segnalato su Slack e definito strutturando una *issue*. Questa deve seguire le specifiche e il *workflow* imposto nel processo di gestione e deve contenere una definizione chiara della problematica e dell'obiettivo da raggiungere. A tale scopo, devono essere presenti i seguenti punti:

- Titolo: significativo, deve illustrare in breve di cosa la problematica tratta;
- Descrizione: dettagliata, deve contenere più informazioni possibili al fine di rendere il problema facilmente e rapidamente individuabile;
- Etichetta: Ogni problematica deve essere classificata per categoria attraverso l'assegnazione di etichette specifiche. Ad esempio, un problema nel modulo di NaturalAPI Design deve essere indicato utilizzando i seguenti tag:
  - Bug: qualcosa nel software non funziona come dovrebbe;
  - Design: il problema riguarda il modulo NaturalAPI Design.

Altre etichette possono essere assegnate alla stessa *issue* se rende più chiara l'identificazione del problema;

- Assegnato a: la persona incaricata a risolvere il problema. Se sconosciuta al momento della definizione, sarà compito del responsabile assegnare la/le risorse adeguate alla risoluzione dello stesso. Un membro del team può anche decidere di auto-assegnarsi una *issue*.

Questa attività viene tipicamente implementata dai verificatori ma chiunque può identificare un problema di qualsiasi tipo e riportarlo con la modalità sopra descritta.

3.7.4 Selezione e generazione delle soluzioni

L'incaricato deve spostare la *issue* nello stato "in progress" e trovare delle soluzioni al problema in seguito ad averne individuato le cause. Se si incontrano difficoltà nella generazione di una soluzione o se più soluzioni possono essere adottate, è bene che queste vengano discusse con gli altri membri del gruppo, sfruttando i canali *Slack* inerenti all'argomento trattato. La scelta ideale deve ricadere su una soluzione il più possibile efficiente, efficace e che abbia il minor numero di effetti collaterali.

3.7.5 Implementazione della soluzione

La soluzione individuata deve essere implementata con un approccio a cicli di implementazione brevi, supportati da verifiche e feedback. L'incaricato, attraverso la sezione "commenti" della *issue*, deve descrivere cosa è stato fatto per arrivare alla soluzione. In particolare, devono essere presenti le seguenti informazioni:

- Causa: il motivo del problema;
- Modifiche effettuate: le principali azioni intraprese per risolverlo.

In seguito, deve spostare la *issue* nello stato "verification".

3.7.6 Revisione del risultato

La risoluzione finale del problema deve essere controllata da un verificatore, che si deve occupare di spostare la *issue* nello stato "done" in seguito ad essersi assicurato che la problematica sia stato effettivamente risolta. In questo caso la *issue* deve essere definitivamente chiusa.

4 Processi organizzativi

4.1 Gestione

4.1.1 Scopo

Gli obiettivi della gestione di processo sono stabilire tutti gli elementi necessarie per avviare i processi, definirne gli obbiettivi e monitorarne lo svolgimento. Lo svolgimento complessivo delle attività del progetto viene illustrato nel Piano di progetto, utilizzato per pianificare le attività da svolgere e organizzare i ruoli svolti da ogni componente.

4.1.2 Descrizione

In questo capitolo vengono illustrati i ruoli. Vengono inoltre illustrate tutte le procedure e gli strumenti necessari per mettere in atto i processi. Vengono quindi trattati i seguenti argomenti:Questo processo definisce il controllo dell'organizzazione e conduzione dei processi del progetto. Viene trattata la gestione dei seguenti argomenti:

- Ruoli di progetto;
- Comunicazione;
- Coordinamento;
- Versionamento;
- Strumenti di analisi;
- Rischi.

Ciascun membro del gruppo, a rotazione, e per un periodo di tempo, ricoprirà il ruolo corrispondente alla figura aziendale. Le attività svolte da ciascun ruolo vengono organizzate e pianificate nel Piano di progetto.

4.1.3 Ruoli di progetto

4.1.3.1 Responsabile di Progetto Il Responsabile di progetto è il rappresentante, per conto del gruppo, dei risultati del progetto. Partecipa per tutta la sua durata con responsabilità di scelta e approvazione. I suoi compiti sono:

- Organizzazione e elaborazione di piani e scadenze;
- Approvazione dei documenti;
- Gestire le attività del gruppo;
- Si relaziona con il controllo di qualità interno al progetto;
- Approva l'offerta e i relativi allegati.

4.1.3.2 Amministratore Controlla e gestisce tutto l'ambiente di sviluppo, è il responsabile della sua efficienza e capacità operativa. Si occupa della redazione e attuazione di piani e procedure di gestione per la qualità, è inoltre il controllore delle versioni e delle configurazioni del prodotto. I suoi compiti sono:

- Amministra le infrastrutture;
- Gestisce i problemi legati alla gestione dei processi;
- Controlla versioni e configurazioni di progetto;
- Gestisce la documentazione di progetto.

4.1.3.3 Analista Sono i responsabili delle attività di analisi, si occupano di capire a fondo il problema definendone i suoi requisiti I suoi compiti sono:

- Analizzare e interpretare la complessità del problema;
- Studio del dominio del problema;
- Definizione dei requisiti del problema.

4.1.3.4 Progettista Sono i responsabili delle attività di progettazione, elaborano una soluzione al problema in base ai vincoli di risorse e di tempo. Gestiscono gli aspetti tecnici del progetto. I suoi compiti sono:

- Descrive il funzionamento del sistema;
- Definisce l'architettura del prodotto.

4.1.3.5 Programmatore Il programmatore si occupa delle attività di codifica atte alla realizzazione del prodotto e alla sua verifica e validazione. I suoi compiti sono:

- Implementare le scelte fatte dal progettista scrivendo codice documentato e mantenibile;
- Implementare i test per la parte di verifica.

4.1.3.6 Verificatore Il verificatore controlla il prodotto di lavoro svolto dagli altri membri del team, sia codice che documentazione. I suoi compiti sono:

- Trovare i difetti del prodotto preso in verifica;
- Segnalare i difetti e gli errori all'autore del prodotto preso in considerazione.

4.1.3.7 Esecuzione e controllo

Il responsabile di progetto avvia l'attuazione del piano per il soddisfacimento degli obiettivi e dei criteri stabiliti. Questa attività si suddivide nei seguenti compiti:

Monitoraggio del piano

Ogni processo viene monitorato dal responsabile in modo che aderisca al way of working. Vengono monitorati l'aderenza al piano stabilito, lo stato di avanzamento del processo e l'identificazione dei problemi.
Per la gestione a livello di ore lavorative, verrà utilizzato *Clockify* per poter rendere misurabile il progetto e rendicontare il lavoro di ogni ruolo.
Inoltre, per una migliore visualizzazione delle varie attività nel tempo, verrà utilizzato "GanttProject" in modo da poter rappresentare graficamente durata, sequenzialità e parallelismo.

Gestione dei problemi

Il responsabile indaga e analizza i problemi scoperti durante l'esecuzione del processo. Verrano poi aggiunti come issue nella repository di segnalazione ed assegnati ai componenti del gruppo.
La risoluzione dei problemi può comportare modifiche ai piani, sarà responsabilità del manager assicurare che l'impatto di qualsiasi cambiamento sia determinato,controllato e monitorato. I problemi e la loro risoluzione devono essere documentati. È il processo di Risoluzione dei problemi (vedi 3.7) che definisce il procedimento per la loro gestione.

4.1.3.8 Revisione e valutazione

Il responsabile deve garantire che i prodotti software e i piani soddisfino i requisiti iniziali. Inoltre, dovrà controllare che i risultati della valutazione dei prodotti software siano corretti. Vengono quindi svolte revisioni e valutazioni secondo le modalità descritte nel processo di Fornitura.

4.1.4 Gestione delle comunicazioni

4.1.4.1 Riunioni

Questa sezione ha lo scopo di definire le riunioni e le modalità di svolgimento delle stesse.
Le riunioni si dividono in

- **Incontri** : Sono incontri di persona, trovandosi fisicamente nello stesso luogo;
- **Hangouts**: Video chiamate di gruppo utilizzando Google Hangouts.

L'organizzazione delle riunioni spetta al responsabile che consulta la disponibilità dei singoli componenti del team e stabilisce l'ordine del giorno. Ogni membro è tenuto a presentarsi puntualmente alle riunioni oppure comunicare e ritardi al responsabile di progetto. Ogni componente inizialmente spiega su cosa sta lavorando in quel periodo e successivamente si parte con l'ordine del giorno.

4.1.4.2 Comunicazioni interne

Messaggistica
Le comunicazioni interne sono gestite utilizzando principalmente *Slack*, un software dedicato alla collaborazione aziendale che permette di inviare messaggi organizzati in canali tematici accessibili a tutto il team o solo ad alcuni membri. Ogni canale è specifico per una determinata attività o argomento, in questo modo le conversazioni sono più focalizzate. Inoltre è possibile comunicare anche attraverso chat individuali private con due o più membri del team.
Le funzionalità più usate di slack sono le seguenti:

- Apertura e gestione di un canale;
- Apertura e gestione di un thread, ovvero un flusso di messaggi interno ad un canale;
- Notifica ad un membro del team, tramite una quote, riguardo a issue a lui assegnate.

Un'altra piattaforma utilizzata dal team per le comunicazioni informali è Telegram. Tramite Telegram si scambiano messaggi non direttamente inerenti al progetto.

Chiamate Per le videochiamate si è deciso di utilizzare *Google Hangouts* un software di messaggistica istantanea che dà la possibilità di effettuare videochiamate in gruppo. Hangouts è gratuito e consente, oltre al normale servizio di videochiamate, di condividere lo schermo.
Le funzionalità più utilizzate di Hangouts sono le seguenti:

- Videochiamate, tra due o più membri;
- Condivisione schermo.

4.1.4.3 Comunicazioni esterne

Questa sezione riguarda le norme per comunicare con soggetti esterni al team FourCats, in particolare:

- Il proponente **TealBlue** rappresentato da Marco Piccolino, col quale si intende stabilire un rapporto di collaborazione in merito alla realizzazione del prodotto;
- I Committenti **Prof. Tullio Vardanega** e **Prof. Riccardo Cardin**, ai quali verrà fornita tutta la documentazione richiesta in ciascuna revisione di progetto, con i quali si intende stabilire un rapporto utile volto al miglioramento continuo dei processi e delle strategie.

Per le comunicazioni con il proponente si è predisposto un canale slack in cui sono presenti tutti i membri del gruppo e il proponente stesso. Le comunicazioni con il committente avvengono tramite dell'indirizzo email del gruppo:

fourcats.unipd@gmail.com

Le comunicazioni esterne sono affidate al Responsabile che ha il compito di preparare i materiali e le domande da inviare attraverso il processo specifico.

4.1.5 Gestione dei rischi

È compito del responsabile di progetto quello di individuare i rischi indicati nel Piano di progetto (PP.1.0.0)
Se dovesse trovarne di nuovi, dovrà aggiungerli nell'analisi dei rischi.
La procedura da seguire per gestire i rischi è la seguente:

- Individuazione dei problemi non calcolati e monitoraggio dei rischi già previsti;
- Registrazione di ogni riscontro previsto dei rischi nel Piano di progetto (PP.1.0.0);
- Aggiunta dei nuovi rischi individuati nel Piano di progetto (PP.1.0.0);
- Se necessario, ridefinizione delle strategie.

4.1.5.1 Codice identificativo

Per ogni rischio viene assegnato un codice identificativo

R{Classificazione}.{Indice}

Dove:

- **Classificazione**: identifica una delle seguenti categorie
  - **V**: Valuazione;
  - **O**: Organizzazione;
  - **P**: Personali;
  - **R**: Requisiti;
  - **T**: Tecnologici;
  - **S**: Strumenti.
- **Indice**: Indica il numero del rischio all'interno di una data classe, è espresso come un intero tra 0 e 99.

4.1.6 Versionamento

Per il versionamento dei file viene utilizzato un Version Control System distribuito. In questo modo è possibile tracciare e conservare tutte le modifiche effettuate a file, condividere i file fra i membri del gruppo e in caso di errori ripristinare i file a versioni precedenti.

4.1.6.1 ID di versionamento

L'ID di versionamento serve a distinguere versioni distinte di un medesimo documento. L'ID è scritto nel seguente formato **vx.y.z** dove x, y e z rappresentano rispettivamente

- x il numero dell'ultima *major release* da cui il documento proviene;
- y il numero di *minor release* dall'ultima *major release*;
- z il numero di *patch* dall'ultima *minor release*.

Esempio: un documento precedente al primo rilascio al committente, sottoposto a due *minor release*, di cui è stata rilasciata una sola *patch* avrà il seguente ID di versionamento: v0.2.1

Major release
Con *major release* si fa riferimento ad un rilascio non retrocompatibile. In particolare la prima *major release* effettuata dal team è prevista per la consegna della RA.

Minor release
Con *minor release* si intende un rilascio retrocompatibile del prodotto. Con le *minor release* si rilasciano funzionalità aggiuntive del prodotto che si integrano alla *major release* di riferimento. In particolare i rilasci di funzionalità del prodotto sviluppate in seguito al periodo di progettazione saranno considerati *minor release*.

Patch
Con *patch* si intendono delle modifiche al prodotto che non comportano l'aggiunta di funzionalità. Alcuni esempi di *patch* sono i rilasci per risolvere bug e migliorare servizi. Essendo la documentazione parte integrante del prodotto, le modifiche alla documentazione precedenti alla prima *major release* saranno registrate come *patch*.

Prodotti non versionati
Di seguito sono elencati i prodotti sviluppati dal team che non sono soggetti a versionamento.

- Proof-of-Concept, tale prodotto non sarà soggetto a sviluppi incrementali. L'obiettivo di questo prodotto è quello di fornire una dimostrazione di come le tecnologie coinvolte nel progetto possano essere integrate e di come i tre moduli possano integrarsi l'un l'altro.
- Verbali, questi documenti non sono versionati in quanto non soggetti a modifiche contenutistiche nel corso del ciclo di vita del progetto.

4.1.6.2 Repository
Per gestire i file del progetto sono presenti due repository, una per la documentazione e l'altra per il software. I motivi delle scelta sono legati all'elevato numero di file presenti e alla necessità di tenere separati i flussi di lavoro di documenti e software. Inoltre una singola repository sarebbe troppo complessa da gestire. Le repository utilizzate sono:

- **NaturalAPI_Documentazione**: repository privata che contiene tutta il documenti formali del progetto soggetti a versionamento;
- **NaturalAPI**: repository pubblica che contiene il codice sorgente del software. In questo modo committente e proponente possono ispezionare il codice prodotto.

Struttura del repository
Il repository **NaturalAPI_Documentazione** contiene una cartella per ogni documento da realizzare. Ogni cartella contiene un file chiamato *0_NomeDocumento.tex* che contiene tutte le sezioni del documento e per ognuna di esse viene creato un file .tex nuovo. All'interno di ogni cartella, se necessario, è stata creata una sotto-cartella *images* la quale contiene tutte le immagini utilizzate per quel documento.

Il repository **NaturalAPI** è stato strutturato in tre cartelle, una per ciascun modulo previsto dal capitolato. Le directory con i PoC sono indicate dal prefisso "poc" seguito dal nome del modulo. Le successive versioni conterrano regole più precise per il codice sorgente del software da consegnare.

4.1.6.3 Commit
Le modifiche ai file sono chiamate commit che rappresentano tutte le righe dei file che sono state modificate o aggiunte. I commit devono essere accompagnate da messaggi per spiegare le modifiche fatte. I messaggi di commit devono avere la seguente struttura:

Soggetto
E' la parte più importante del messaggio e riassume le modifiche. Le convenzioni per il soggetto sono:

- Scrivere il primo carattere in maiuscolo;
- Limitarsi a 50 caratteri;
- Non usare il punto finale;
- Utilizzare l'imperativo per i messaggi.

Corpo
Se necessario aggiungere un corpo al messaggio per motivare le modifiche all'incirca di 72 caratteri. I paragrafi vanno separati da una nuova linea e possono essere aggiunti elenchi.

Issue
Se presenti una o più issue relative alla modifica, vanno aggiunte dopo il corpo del messaggio.

Template
Questo è un possibile template per un messaggio completo.

```
Riassunto corto (50 caratteri o meno)

Maggiori dettagli, se necessario, la singola linea dovrebbe avere all'incirca 72 caratteri.
Il soggetto e il corpo vanno separati da una linea vuota. In alcuni contesti
la prima linea è il soggetto del commit mentre il resto è il corpo.
La linea vuota che li separa è fondamentale (a meno che tu non includa il corpo);
diversi strumenti fanno confusione se soggetto e corpo non sono separati.

Illustra il problema che questo commit sta risolvendo. Concentrati sul
perchè delle modifiche anziche sul come (questo è spiegato dal codice).
Di sono effetti secondari o altre consequenze controintuitive?
Questo è il luogo dove illustrarle.

Ulteriori paragrafi sono separati da linea vuota.

- Elenchi puntati vanno bene.

- Tipicamente viene usato il segno meno o l'asterisco per i punti che
sono separati da linee vuote.

I riferimenti a issue (eg #42) vanno aggiunti alla fine.
```

4.1.7 Gestione dei conflict

E' possibile che due membri dei team lavorino sugli stessi file causando un conflict, ossia uno stato non consistente dei file. In caso di conflict il membro del team deve segnalarlo al resto del gruppo comunicando il nome del file e l'origine del conflict. La segnalazione avviene attraverso il canale di comunicazione slack del team. In caso di problemi gravi è il Responsabile a decidere come risolvere il conflict. Una volta deciso come agire, il membro del team che ha causato il conflict lo risolve il locale e carica la versione corretta nel repository remoto.

4.1.8 Workflow Patterns

4.1.8.1 Branch

Per l'aggiunta o la modifica di funzionalità vengono utilizzati i branch. Nel VCS il branch è un puntatore ad un singolo commit. Quando un membro del team deve modificare dei file già presenti o aggiungerne di nuovi non va modificare i file già presenti ma crea un nuovo branch separato dove eseguire il lavoro. Un branch può subire modifiche e contenere più di un commit che ne rappresenta la storia. A lavoro finito, le modifiche prodotte vengono analizzate e il branch può essere chiuso eseguendo un'operazione di merge oppure possono essere fatte ulteriore modifiche.

4.1.8.2 Git Flow Per la gestione dei branch viene utilizzato il modello gitflow in cui viene creato un ramo feature a partire dal ramo develop. A lavoro compiuto, viene eseguito il merge del ramo feature nel ramo develop.

4.1.9 Coordinamento

4.1.10 Issue Tracking System

Per organizzare il lavoro viene utilizzato un issue tracking system. Gli obbiettivi da raggiungere vengono espressi in milestone. Le milestone a loro volta sono formate da issue.

4.1.10.1 Issue

Le issue rappresentano l'attività lavorativa fondamentale. E' compito del Responsabile identificare le issue e assegnarle ai membri del team. Ciascuna issue ha la seguente struttura:

- Titolo del issue
- Descrizione delle issue
- Data di completamento
- Tag
- Membro del team a cui è assegnato il compito
- Milestone di appartenenza, ma non per tutte le issue
- Descrizione del compito assegnato

Le issue possono avere dei tag per indicarne la funzione, ad esempio bug può essere utilizzato per segnalere un bug del codice. I membri del team possono creare delle issue per segnalare problemi o eventuali nuove issue da portare a termine. Per la documentazione è presente un tag per ciascuno dei documenti prodotti.

4.1.10.2 Issue work flow

Le issue hanno diversi stati che attraversano nel loro ciclo di vita. Le issue e i work flow individuati vengono quindi utilizzati per rappresentare i processi e i loro avanzamento. E' compito del Responsabile strutturare le issue e i work flow per attuare i processi. I membri del team devono portare avanti le issue e aggiornare il loro stato.

Per la documentazione, le issue hanno i seguenti **stati**:

- **To Do**: elenco di issue da completare.
- **In Progress**: issue in lavorazione.
- **Verification**: issue da verificare.
- **Done**: issue completate.

Il responsabile identifica le issue da completare e le istanzia nello stato To Do, assegnandole ad analisti e programmatori. I membri del team ricevono notifica attraverso mail dell'assegnazione o controllando la board del progetto. Quando le attività vengono iniziate, passo allo stato In progress per segnalare che sono in lavorazione. Una volta completate passano allo stato Verification per le attività di verifica. Infine vengono messe nello stato Done.

4.1.10.3 Milestone Le milestone sono gruppi di issue che corrispondono al raggiungimento di determinati obbiettivi. Le milestone vengono identificate sulla base degli obbiettivi riportati nel Piano di progetto. E' compito del responsabile preparare le milestone negli strumenti di coordinamento e identificare tutte le attività necessarie per portarle a termine. Una milestone ha i seguenti campi:

- Titolo del issue
- Descrizione delle issue
- Data di completamento
- Un elenco di issue associate

Il Responsabile nel corso del progetto deve verificare il completamente della milestone entro i tempi previsti. E' sua responsabilità esercitare una sorveglianza attiva sull'andamento dei lavori per verificare che gli obbiettivi siano realizzabili. In caso contrario, deve riorganizzare il lavoro e aggiornare il Piano di progetto.

La milestone può essere chiusa quando tutte le issue associate sono state chiuse, questo compito spetta al Responsabile. Se alla data di completamento le issue non sono ancora chiuse, il Responsabile deve ispezionare le issue non ancora chiuse e contattare i membri del team a cui sono state assegnate per chiedere chiarimenti. Una volta individuati i problemi emersi e identificata una soluzione, deve aggiornare il Piano di progetto e le milestone.

4.1.11 File condivisi

Per i file non versionati viene utilizzato un servizio di file sharing. Le cartelle più importanti sono:

- **Verbali**: Varbali delle riunioni;
- **Presentazioni**: Ci sono le presentazioni delle varie fasi;
- **Materiali di studio**: Questa cartella contiene i diversi materiali di studio e le guide per i tool che utilizziamo;
- **Documenti**: Ogni documento ha una sua cartella, al suo interno ci sono dei file di utilità per quel determinato documento;
- **Standard**: Contiene gli standard;
- **Varie**: I file meno importanti vengono messi in questa cartella, come ad esempio il logo del gruppo .

Tutti i documenti informali vengono custoditi qui.

4.1.12 Diagrammi di Gantt

4.1.13 Strumenti

4.1.13.1 Git Git è un software di controllo di versione distribuito open source.

4.1.13.2 GitHub Servizion hosting per progetti software che usa il sistema di controllo di versione Git. Viene utilizzato come repository remoto utilizzato da tutti i membri del gruppo per la gestione dei file. Oltre ai servizi di versionamento, vengono utilizzati i servizi di issue tracking system.

4.1.13.3 GitKraken Gitkraken è un GUI per il software Git che semplifica l'utilizzo di Git. Permette di dare una rappresentazione grafica dei commit e dei branch molto più intuitiva rispetto al terminale. Un ulteriore vantaggio è che offre nella stessa applicazione l'interazione con l'ITS di Github e una rappresentazione temporale dei commit.

GitKraken Glo Per la gestione dell'ITS viene utilizzato Gitkraken Glo, un GUI per interagire con le issue di Github. Il vantaggio principale di Glo è riunisce in un singola applicazione tutte le operazioni per il coordinamento e il versionamento, riducendo i tempi morti tra le operazioni.

GitKraken Glo Le issue vengono raggrupate in board a loro volta organizzate per colonne che rappresentano il work flow issue. Le board sono flessibili e permettono di creare colonne personalizzate per meglio rappresentare i processi. Sono inoltre disponibili funzioni di ricerca e filtro delle operazioni per nome o per tag. Attraverso la board il Responsabile può avere un'istantanea dell'andamento del progetto e controllare il carico di lavoro assegnato ai membri del team. Oltre alla board è presente una funzione calendario che permette di mappare le issue sui giorni del mese.

4.1.13.4 Google Drive Google Drive è un servizio per la memorizzazione e sincronizzazione online, utilizzato per condividere file per l'organizzazione del gruppo. È stato scelto per la condivisione veloce di file come guide, documentazioni, riferimenti e documenti informali tramite Google Docs.

4.1.13.5 GanttProject GanttProject è un software per la gestione del progetto che facilita la programmazione di compiti e la gestione delle risorse. Deve essere utilizzato per redigere i diagrammi di Gantt. Questi illustrano in modo grafico la pianificazione di una determinata fase del progetto. All'interno dei diagrammi devono essere presenti processi ed attività che devono seguire le seguenti regole base:

- Le attività di verifica devono essere indicate con il colore verde chiaro;
- Tutte le altre attività devono essere indicate con il colore predefinito di GanttProject: azzurro;
- Eventuali dipendenze tra attività vanno indicate attraverso l'opzione "predecessori" presente all'interno delle proprietà di un'attività.

GanttProject crea dei file in formato .gan che devono essere conservati all'interno del repository per permettere, se necessario, eventuali modifiche ai diagrammi già creati.

4.1.13.6 Clockify Come strumento di monitoraggio delle ore di orologio, usiamo clockify, un tool online che offre un sistema comodo per monitorare e tenere traccia del tempo speso per ogni attività. Questo strumento viene utilizzato per rendicontare le ore di ogni ruolo del team.

4.2 Training

4.2.1 Scopo

Lo scopo del processo di training è mantenere i membri del gruppo il più possibile preparati e fare in modo che tutti abbiano le stesse competenze al termine del progetto.

4.2.2 Descrizione

L'acquisizione, la fornitura, lo sviluppo e il mantenimento del prodotto software dipendono fortemente dalla preparazione del personale. È necessario possedere competenze essenziali di gestione del software e di ingegneria del software, ma è opportuno che ci sia un piano di istruzione pianificato in anticipo affinché la preparazione di tutti i membri sia disponibile fin dal principio del ciclo di vita del software.

4.2.3 Training iniziale

Ogni persona deve rendere note le sue conoscenze di partenza in relazione ad ogni aspetto affrontato nell'analisi dei requisiti, in particolare riguardo linguaggi di programmazione e tools. Tutti i membri, a prescindere dal livello di partenza, devono impegnarsi in un periodo iniziale di apprendimento personale, per uniformare il più possibile il livello di conoscenza del gruppo. Ogni documento o link a materiale utile dev'essere caricato sulla cartella Drive del gruppo.

4.2.4 Piano di training

Ogni volta che emerge il bisogno di utilizzare una tecnologia o un tool sconosciuti a parte del gruppo si dovrà:

1. Verificare chi all'interno del gruppo ha familiarità con la nuova tecnologia o tool;
2. Assegnare a uno dei membri identificati il compito di redigere un documento informale con le indicazioni principali necessarie alla comprensione della tecnologia o all'utilizzo del tool.

Se la nuova tecnologia o tool è completamente sconosciuta al gruppo allora:

1. Viene assegnato a un membro qualsiasi il compito di effettuare ricerche;
2. Il membro del gruppo dovrà redigere un documento informale contenete l'esito delle ricerche.

La formazione dei restanti membri del gruppo avviene perlopiù individualmente. Ogni membro del gruppo dovrà studiare in autonomia i documenti prodotti. Per confronti e chiarimenti sulle nuove tecnologie tra più membri del gruppo si deve creare un apposito canale Slack. L'utilizzo di un nuovo tool richiede invece pratica individuale per esplorarne tutte le funzionalità richieste. In caso un membro del gruppo abbia dubbi sull'utilizzo di un tool deve chiedere chiarimenti sul canale Slack dedicato (o crearne uno se non presente). Gli altri membri del gruppo dovranno rispondere appena possibile per risolvere la perplessità. Se il confronto collettivo con il gruppo non porta a una comprensione accettata e condivisa da tutti si può optare per richiedere chiarimenti al proponente o ai committenti.

4.2.5 Documentazione

Lo studio di una nuova tecnologia deve essere accompagnato dalla produzione di un documento informale. Il documento dovrà contenere:

- Nome della tecnologia studiata;
- Breve descrizione della tecnologia;
- Sintetica spiegazione sul suo utilizzo;
- Link alle fonti;
- Link al download (in caso di software);
- Eventuali link a tutorial.

Il documento dovrà poi essere caricato nel Drive del gruppo.