MaaP: MongoDB as an admin Platform

Norme di Progetto

Versione	5.2.0
Data creazione	2013-11-20
Data ultima modifica	2014-07-16
Stato del Documento	Formale
Uso del Documento	Interno
Redazione	Mattia Sorgato
Verifica	Andrea Perin
Approvazione	Giacomo Pinato
Distribuzione	Aperture Software

Sommario

Questo documento si propone di presentare le norme che il gruppo Aperture Software ha stabilito per la realizzazione del prodotto MaaP: MongoDB as an admin Platform.

Diario delle modifiche

Versione	Data	Autore	Modifiche effettuate
5.2.0	2014-07-16	Giacomo Pinato (RE)	Approvazione documento.
5.1.0	2014-07-15	Andrea Perin (VR)	Verifica documento.
5.0.1	2014-07-14	Mattia Sorgato (AM)	Effettuate correzioni segnalate dal Committente.
4.2.0	2014-04-17	Mattia Sorgato (RE)	Approvazione documento.
4.1.0	2014-04-17	Alberto Garbui (VR)	Verifica documento.
4.0.4	2014-04-14	Michele Maso (AM)	Stesura strumenti.
4.0.3	2014-04-10	Michele Maso (AM)	Stesura attività.
4.0.2	2014-04-09	Michele Maso (AM)	Stesura processi.
4.0.1	2014-04-07	Michele Maso (AM)	Riorganizzazione documento.
3.2.0	2014-01-24	Fabio Miotto (RE)	Approvazione documento.
3.1.1	2014-01-23	Alessandro Benetti (VE)	Verifica documento.
3.1.0	2014-01-21	Fabio Miotto (VR)	Verifica documento.
3.0.2	2014-01-15	Alberto Garbui (AM)	Incremento documento.
3.0.1	2014-01-13	Alberto Garbui (AM)	Effettuate correzioni segnalate dal Committente.
2.2.0	2014-01-04	Alberto Garbui (RE)	Approvazione documento.
2.1.0	2014-01-03	Giacomo Pinato (VR)	Verifica documento.
2.0.2	2013-12-27	Fabio Miotto (AM)	Sezione Progettazione, Riorganizzazione.
2.0.1	2013-12-23	Fabio Miotto (AM)	Modifica sezione Comunicazioni.
1.2.0	2013-11-27	Andrea Perin (RE)	Approvazione documento.
1.1.0	2013-11-26	Alessandro Benetti (VR)	Verifica documento.
1.0.2	2013-11-25	Giacomo Pinato (RE)	Ampliamento documento.
1.0.1	2013-11-20	Andrea Perin (RE)	Creazione documento.

Tabella 1: Registro delle modifiche

Indice

1 Introduzione

1.1 Scopo del documento

Questo documento è volto a definire le norme che dovranno essere osservate da tutti i componenti del team per l'intera durata del progetto. Tali norme sono volte a garantire la qualità finale del prodotto attraverso la rigida regolamentazione dei processi e delle strategie di produzione. In queste pagine vengono delineate le linee guida e le norme per tutte le *attività*$_G$ che concorreranno allo sviluppo del *software*$_G$, della documentazione e del progetto in generale.

1.2 Scopo del prodotto

Lo scopo del prodotto è produrre un *framework*$_G$ per generare interfacce web di amministrazione dei dati di business basato sullo stack *Node.js*$_G$ e *MongoDB*$_G$.
L'obiettivo è quello di semplificare il lavoro allo *sviluppatore*$_G$ che dovrà rispondere in modo rapido e standard alle richieste degli esperti di *business*$_G$.

1.3 Ruoli di Progetto

Per la piena riuscita del progetto è indispensabile distinguere i vari ruoli che concorrono alla creazione del prodotto finale e le loro diverse responsabilità e competenze. Ogni ruolo avrà una specifica area di competenza, degli specifici compiti, oneri e particolari autorizzazioni. Ogni componente dovrà limitarsi ai compiti ad esso assegnati e, nel caso qualcosa esuli dal suo campo di pertinenza, lo stesso dovrà rivolgersi al collega occupante il ruolo corrispondente.
Tutti i membri, a rotazione, dovranno occupare come minimo una volta ciascuno dei ruoli descritti sottostante. Per fare in modo che la rotazione dei ruoli non causi problematiche o conflitti, è necessario che ciascuna attività venga pianificata e le persone interessate devono attenersi agli incarichi a loro assegnati.
I ruoli vengono assegnati in base al carico di lavoro che le attività richiedono (per essere svolte entro le scadenze fissate) e anche in base al budget massimo prefissato. A causa di quanto descritto in precedenza il gruppo ha deciso di assegnare più ruoli per persone, garantendo sempre che non vadano in conflitto tra di loro e che nella stessa giornata lavorativa una persona non svolga più ruoli contemporaneamente. Solo dopo aver garantito quanto scritto sopra, il Verificatore dovrà visionare il diario delle modifiche di ciascun documento per scovare incongruenze.
Verrà di seguito fornita una descrizione dei ruoli di progetto, accompagnata dalle responsabilità e le modalità operative alle quali le persone incaricate dovranno attenersi.

1.3.1 Responsabile di Progetto

Il Responsabile di Progetto incentra su di sé le responsabilità di scelta ed approvazione dei lavori. Ricopre il ruolo di rappresentante del gruppo nei contatti con l'esterno e durante la presentazione dei lavori.
Le sue competenze principali comprendono:

- Pianificazione, coordinamento e controllo delle attività;
- Gestione e controllo delle risorse;
- Approvazione delle analisi di gestione e rischio;
- Approvazione dei documenti;

- Comunicazioni con i Committenti/Proponenti.

Il Responsabile ha il compito di assicurarsi che le attività di Verifica vengano svolte sistematicamente seguendo le *Norme_di_Progetto_v5.2.0*. Deve al contempo accertarsi che vengano rispettati i ruoli e le competenze assegnate nel *Piano_di_Progetto_v5.2.0* e che non vi siano conflitti di interesse tra redattori e Verificatori. A tale proposito, sarà sua responsabilità stilare un *Piano di Progetto* in cui ogni redattore di uno specifico documento o codice potrà solamente verificare prodotti altrui.
Ha inoltre l'onere di gestire la creazione e l'assegnazione dei ticket di pianificazione e di assegnare ad un membro del gruppo il ruolo di Responsabile di quest'ultimo, nel caso riguardi una sotto-attività.
Redige il Piano di Progetto e collabora alla stesura del Piano di Qualifica, per quanto riguarda gli ambiti di pianificazione delle attività.

1.3.2 Amministratore

L'*Amministratore*$_G$ è il responsabile del controllo, dell'efficienza e dell'operatività dell'ambiente di lavoro e degli strumenti per la condivisone e la sincronizzazione.
Egli deve garantire:

- L'individuazione e la gestione di strumenti per automatizzare quanto più possibile processi o attività;
- L'individuazione e la gestione di strumenti per il controllo dei processi e delle risorse;
- L'individuazione e la gestione di strumenti e strategie per il controllo della qualità;
- Gestione del versionamento.

Redige la sezione delle Norme di Progetto, nei paragrafi e sezioni relativi agli strumenti utilizzati a supporto dei processi. Inoltre redige la sezione del Piano di Qualifica inerente agli strumenti di Verifica.

1.3.3 Analista

L'Analista è il responsabile delle attività di Analisi all'interno del progetto. Il suoi compiti comprendono:

- La piena comprensione del problema e il suo dominio;
- La stesura di una specifica precisa, completa e comprensibile dal Proponente, dal Committente e dal Progettista.

L'Analista redige il documento di Studio di Fattibilità, successivamente l' Analisi dei Requisiti e in parte il Piano di Qualifica.

1.3.4 Progettista

Il progettista è colui il quale ricerca e delinea la giusta struttura del sistema per soddisfare il problema posto dal Committente e dal Proponente.
I suoi compiti principali sono:

- Disegnare una soluzione fattibile e adatta a soddisfare i requisiti delineati dagli Analisti;
- Implementare soluzioni di progettazione ottimali, ottimizzate e collaudate;

- Utilizzare tecnologie e standard che rendano facile la realizzazione e la manutenzione del prodotto.

Redige la Specifica Tecnica e la Definizione di Prodotto, oltre a partecipare alla stesura delle metriche di verifica della progettazione del Piano di Qualifica.

1.3.5 Verificatore

Il Verificatore è responsabile delle attività di Verifica. Ha il compito di assicurare che i documenti e il codice siano conformi alle norme stabilite nel documento *Norme_di_Progetto_v5.2.0*. Inoltre deve controllare che lo stato di ciclo di vita sia corrispondente a quello attuale. Redige la sezione del Piano di Qualifica che illustra gli esiti delle prove e delle verifiche effettuate.

1.3.6 Programmatore

Il *Programmatore*$_G$ è responsabile delle attività di Codifica e delle componenti di ausilio necessarie per l'esecuzione delle prove di Verifica e Validazione. Le responsabilità di tale ruolo sono:

- Implementare rigorosamente le soluzioni descritte dal Progettista per la Realizzazione del progetto;
- Scrivere codice e la sua relativa documentazione affinché entrambi rispettino gli standard stabiliti per la loro scrittura;
- Implementare i test sul codice scritto, necessari per prove di Verifica e Validazione.

Redige il Manuale Utente e tutta la documentazione relativa al codice.

1.4 Glossario

Al fine di evitare ogni ambiguità nella comprensione del linguaggio utilizzato nel presente documento e, in generale, nella documentazione fornita dal gruppo Aperture Software, ogni termine tecnico di difficile comprensione o di necessario approfondimento verrà inserito nel documento *Glossario_v5.2.0.pdf*.
Saranno in esso definiti e descritti tutti i termini in corsivo e allo stesso tempo marcati da una lettera "G" maiuscola in pedice nella documentazione fornita.

1.5 Riferimenti

1.5.1 Informativi

- **Slide dell'insegnamento Ingegneria del Software modulo A**:
  Ingegneria dei requisiti: http://www.math.unipd.it/~tullio/IS-1/2013/;
- **Software Engineering - Ian Sommerville - 9th Edition (2010)**;
- **ISO/IEC 12207 Systems and software engineering**;
- **Standard ISO 8601** - Data elements and interchange formats: http://it.wikipedia.org/wiki/ISO_8601;
- **Jenkins:** http://en.wikipedia.org/wiki/Jenkins_(software);

- **Piano di Qualifica**: *Piano_di_Qualifica_v5.2.0*;
- **Piano di Progetto**: *Piano_di_Progetto_v5.2.0*.

2 Processi

L'intero progetto sarà sviluppato mediante dei processi, ovvero mediante un insieme di attività coerenti e coese. Ogni progetto è suddiviso nelle attività che lo compongono.

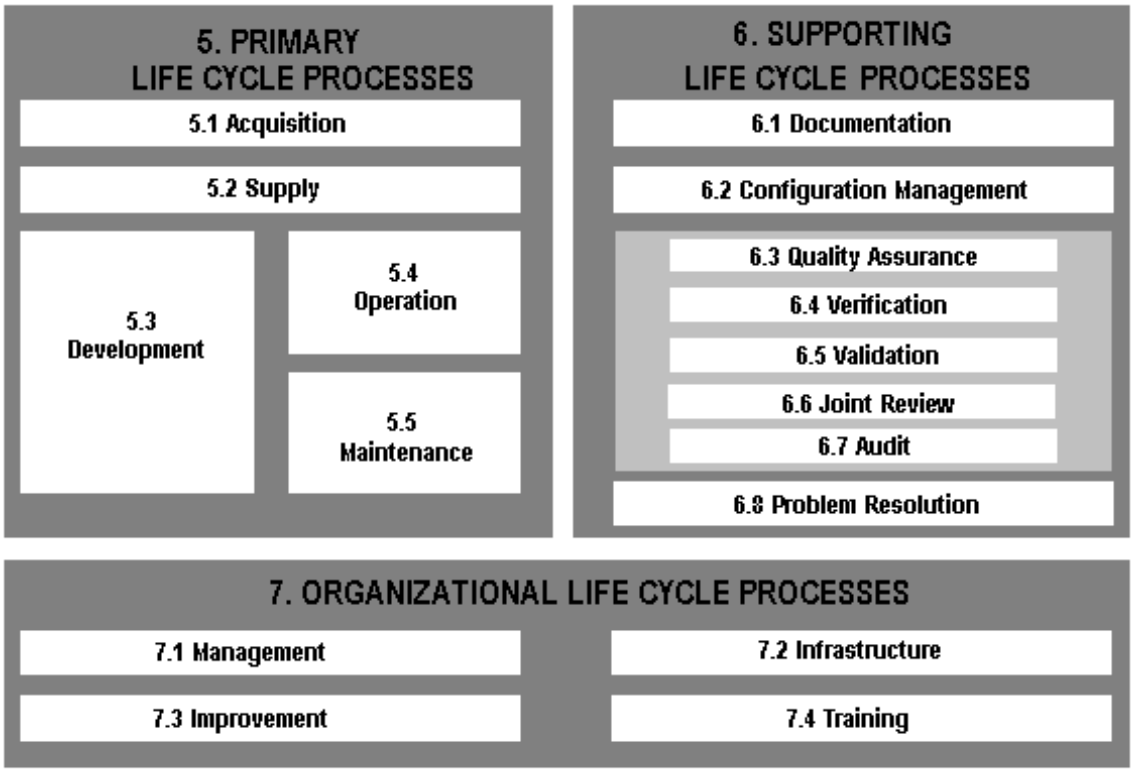

Figura 1: I principali processi descritti in ISO/IEC 12207

2.1 Acquisizione

Lo scopo del processo di acquisizione è quello di ottenere la definizione di un prodotto o servizio che soddisfi le richieste espresse dall'acquirente. Questo processo inizia con l'identificazione delle necessità del cliente e termina con l'accettazione da parte di quest'ultimo dell'analisi e delle specifiche definite.
Il processo di acquisizione comprende tutte le attività volte ad identificare, elaborare e catalogare i requisiti, impliciti e non, che il progetto possiede.

2.1.1 Analisi dei Requisiti

2.1.1.1 Identificazione dei Requisiti

Questo compito si prefigge di identificare tutti i requisti, espliciti e non, che il cliente desidera, che il sistema necessita per un buon funzionamento e che sono necessari alla riuscita finale del progetto.

2.1.1.1.1 Norme generali
Le principali fonti di requisiti saranno:

- Colloqui con il committente;
- Capitolato;
- Casi d'uso individuati;
- Verbali derivanti da discussioni di gruppo.

Dopo aver analizzato e compreso la natura e le funzionalità del prodotto atteso, gli analisti dovranno elaborare dei casi d'uso che dimostrino le funzionalità disponibili e le loro relazioni. A partire da questi ultimi e dalle fonti sopra citate, dovranno individuare tutti i requisiti necessari a definire non ambiguamente funzionalità e vincoli necessari al procedimento del progetto.

2.1.1.2 Stesura casi d'uso
La stesura dei casi d'uso è un compito utile per comprendere più a fondo le funzionalità del sistema e per identificare requisiti impliciti precedentemente mancati.

2.1.1.2.1 Norme generali

Ciascun caso d'uso descritto dagli analisti dovrà rispettare la seguente notazione:

UC[codice progressivo di livello]

Ogni caso d'uso ha un $codice_G$ univoco gerarchico.
Il codice progressivo può includere diversi livelli di gerarchia separati da un punto. A ogni descrizione di caso d'uso viene riportato il numero del diagramma associato di cui fa parte, ovvero la rappresentazione del caso d'uso generale a cui esso appartiene. Ogni diagramma associato è scritto nella seguente forma:

Figura[numero progressivo]:[codice univoco e nome del caso d'uso].

2.1.1.2.2 Procedure

- Aprire Astah;
- Cliccare su "Nuovo diagramma dei casi d'uso";
- Disegnare il caso d'uso;
- Nominare il file "*UC*" seguito dal codice del caso d'uso senza spazi. Ad esempio *UC1.2.1.3*;
- Salvare il caso d'uso nell'apposita cartella in formato png.

2.1.1.2.3 Strumenti
Astah.

2.1.1.3 Classificazione dei requisiti
Questo compito mira a classificare i requisiti in base alle loro caratteristiche.

2.1.1.3.1 Procedure

Ogni requisito andrà catalogato secondo il seguente schema:

R[importanza][tipo][codice]

dove:

- Importanza può assumere i seguenti valori:
  - **O:** *requisito*$_G$ obbligatorio;
  - **D:** requisito desiderabile;
  - **F:** requisito facoltativo.
- Tipo può assumere i seguenti valori:
  - **F:** funzionale;
  - **Q:** di qualità;
  - **P:** prestazionale;
  - **V:** vincolo.
- Codice è il codice che identifica univocamente ogni requisito.

A seguito della catalogazione, il requisito deve essere inserito nello strumento di tracciamento automatico.

2.1.1.3.2 Strumenti

ApertureTrace.

2.1.1.4 Tracciamento

Deve essere realizzata una tabella di tracciamento che collega ogni requisito alla sua fonte. La tabella deve avere la seguente struttura:

- **Fonte:** identifica la sorgente dei requisiti associati;
- **Requisiti:** Elenca tutti i requisiti derivati dalla fonte.

2.1.1.4.1 Procedure

2.1.1.4.2 Inserimento requisito

Ogni volta che un requisito diventa stabile, solido e atomico, esso deve venire inserito nel software di tracciamento automatico ApertureTrace. Per l'inserimento si deve:

- Selezionare inserimento requisito;
- Inserire il codice del requisito;
- Inserire il grado di importanza del requisito;
- Inserire il tipo del requisito;
- Inserire una descrizione del requisito;
- Inserire la fonte del requisito;
- Confermare l'inserimento.

2.1.1.4.3 Generazione tabelle
Per generare le tabelle di tracciamento si deve:

- Entrare su ApertureTrace;
- Selezionare la tabella Requisiti-Fonti;

2.1.1.4.4 Strumenti
ApertureTrace.

2.1.2 Studio di fattibilità

Questa attività comprende tutti i compiti volti ad analizzare e valutare la fattibilità del progetto proposto.

2.1.2.1 Valutazione fattibilità
Questo compito consiste nel valutare i vari aspetti che concorrono a quantificare la complessità del progetto e, in base alle risorse disponibili, determinare se sia fattibile intraprendere lo sviluppo del progetto.

2.2 Fornitura

Lo scopo di questo processo è fornire un piano di gestione del progetto contente informazioni riguardanti pianificazione delle attività, scadenze, milestone e tempi di consegna. Serve inoltre a stimare il costo totale del progetto ed il monte ore necessario per ciascun ruolo visto precedentemente. Questo piano sarà poi seguito durante tutta la durata del progetto.
Questo processo comprende attività volte a gestire la pianificazione, le risorse, i rischi e i controlli sul progetto. Le responsabilità di gestione dell'intero progetto, dalla nascita alla conclusione, sono da attribuire al Responsabile di Progetto.

2.2.1 Pianificazione

La pianificazione mira a fissare le scadenze e le milestone dei vari processi che si andranno ad attuare al fine di poter controllare il carico di lavoro, la sua durata e poter meglio gestire le risorse.

2.2.1.1 Norme generali
All'inizio di ciascuna delle fasi delineate nel *Piano di Progetto v5.2.0*, il Responsabile di Progetto deve realizzare un *diagramma di* Gantt_{GG} su $\textit{Redmine}_{G}$, come descritto in sezione 2.2.1.2.1 di questo documento, per tracciare e gestire le attività dei processi utilizzati dal progetto.
Dopo ogni sessione di lavoro di un componente o gruppo di componenti, gli stessi devono tracciare il loro lavoro modificando l'avanzamento dell'attività e registrando le ore/persona spese mediante il sistema offerto da Redmine. Dovranno inoltre modificare lo stato dell'attività in base al ruolo ricoperto e lo stato di avanzamento della stessa.
Durante l'avanzamento del progetto il Responsabile deve monitorare costantemente il verificarsi dei rischi descritti nel *Piano di Progetto v5.2.0* così come la possibilità che l'avanzamento comporti nuovi rischi non previsti precedentemente.

2.2.1.1.1 Creazione dei ticket

I ticket possono essere creati da:

- Responsabile di Progetto: Crea i ticket per le macro fasi evidenziate durante la pianificazione;
- Responsabile di Sotto-progetto: Crea i ticket per i processi minori, non pianificati o di secondaria importanza;
- Verificatore: Crea i ticket per la segnalazione di errori rilevati durante i processi di verifica.

2.2.1.1.2 Tipologie di ticket

Si possono aprire le seguenti tipologie di ticket:

- Ticket di pianificazione: Rappresentano attività di maggiore importanza. Sono organizzati gerarchicamente in base all'importanza dell'attività relativa;
- Ticket di realizzazione: Rappresentano ticket di realizzazione dei documenti;
- Ticket per la verifica: Rappresentano ticket di verifica assegnati ai verificatori;
- Ticket per la segnalazione degli errori: Rappresentano errori e problematiche riscontrate dai verificatori.

2.2.1.1.3 Stati d'avanzamento

I possibili stati di avanzamento di un ticket sono i seguenti:

- **Nuovo:** Il compito è stato creato ed è in attesa di essere assegnato;
- **In Elaborazione:** Il compito è stato assegnato ed è in lavorazione;
- **Chiuso:** la lavorazione è terminata;
- **Risolto:** Stato disponibile solo per i ticket di errore. Indica che le correzioni hanno effettivamente risolto il problema riscontrato.

2.2.1.2 Procedure

2.2.1.2.1 Realizzazione diagramma di Gantt

Redmine genera automaticamente il diagramma di Gantt in base alle attività ed ai compiti inseriti. Per inserire un nuovo compito si deve seguire la procedura illustrata nel sottoparagrafo successivo.

2.2.1.2.2 Ticket di pianificazione

- Entrare nel progetto;
- Cliccare, nel menù in alto, sulla voce corrispondente a Nuova segnalazione;
- Nel menù a tendina con la voce Tracker selezionare *compito*;
- Specificare un oggetto per il ticket, che corrisponde ad una breve sintesi del compito;

- Specificare una descrizione, che corrisponde ad una stesura dettagliata del compito;
- Specificare una priorità;
- Specificare a chi assegnare il ticket;
- Specificare la data di inizio;
- Specificare la data di scadenza;
- Specificare il tempo stimato.

2.2.1.2.3 Ticket di realizzazione

- Entrare nel progetto;
- Cliccare, nel menù in alto, sulla voce corrispondente a Nuova segnalazione;
- Nel menù a tendina con la voce Tracker selezionare *Realizzazione*;
- Specificare un oggetto per il ticket, che corrisponde ad una breve sintesi del compito;
- Specificare una descrizione, che corrisponde allo scopo specifico di quel ticket;
- Specificare una priorità;
- Specificare a chi assegnare il ticket;
- Specificare la data di inizio;
- Specificare la data di scadenza;
- Specificare il tempo stimato.

2.2.1.2.4 Ticket di verifica

- Entrare nel progetto;
- Cliccare, nel menù in alto, sulla voce corrispondente a Nuova segnalazione;
- Nel menù a tendina con la voce Tracker selezionare *Verifica*;
- Specificare un oggetto per il ticket;
- Specificare una priorità;
- Specificare la data di inizio;
- Specificare la data di scadenza;
- Specificare il tempo stimato.

2.2.1.2.5 Ticket di segnalazione errori

- Entrare nel progetto;
- Cliccare, nel menù in alto, sulla voce corrispondente a Nuova segnalazione;
- Nel menù a tendina con la voce Tracker selezionare *Bug*;
- Specificare un oggetto per il ticket, che corrisponde ad una breve sintesi dell'errore;
- Specificare una descrizione, che corrisponde ad una stesura dettagliata dell'errore, e specificare anche la posizione dell'errore;
- Specificare una priorità;
- Specificare la data di inizio;
- Specificare la data di scadenza;
- Specificare il tempo stimato.

2.2.1.2.6 Gestione dei cambiamenti

In caso di errori, in seguito alla notifica da parte del Verificatore tramite ticket, il responsabile di progetto dovrà assegnare la correzione mediante la procedura di ticketing descritta precedentemente. Al termine della correzione, sarà compito del responsabile di sotto-progetto accettare o respingere la modifica, e richiederne di conseguenza il rifacimento. Nel caso la correzione riguardi un'attività di codifica, sarà compito del responsabile di sotto-progetto programmare una nuova esecuzione dei test di unità e di integrazione correlati al modulo modificato.

2.2.1.3 Strumenti

Diagrammi di Gantt, Redmine, OpenProj.

2.2.2 Analisi e gestione dei rischi

L'analisi e la gestione dei rischi si prefiggono di identificare e monitorare costantemente i rischi in cui si potrebbe incorrere durante lo svolgimento del progetto, identificando possibili strategie di riduzione di probabilità e dei danni.

2.2.2.1 Procedure

Nel caso in cui si identifichi un nuovo rischio, quest'ultimo deve essere inserito nell'apposita sezione del piano di progetto. Deve inoltre essere elaborata una strategia per ridurre la probabilità che tale rischio si verifichi e una per il controllo dei potenziali danni. Nel caso in cui uno dei rischi già individuati si verifichi, il Responsabile deve immediatamente attuare la strategia per il controllo dei danni precedentemente disegnata.

2.2.2.2 Strumenti

Piano di Progetto, Redmine.

2.2.3 Gestione delle risorse

La gestione delle risorse si di prefigge di gestire al meglio le risorse assegnate a tutte le altre attività e compiti in attuazione.

2.2.3.1 Procedure
Per gestire le risorse umane assegnate ad ogni attività, il Responsabile deve sfruttare la struttura precedentemente creata su Redmine ed assegnare ogni attività ai membri del gruppo secondo il sistema di ticketing descritto nella sezione 2.2.1.2. Dovrà inoltre controllare che il numero di risorse sia sempre adeguato all'attività in questione al fine di garantire il suo completamento entro i termini pianificati.

2.2.3.2 Strumenti
Redmine.

2.3 Sviluppo

Lo scopo di questo processo è progettare, implementare e testare il prodotto al fine di renderlo adatto ad essere rilasciato ai clienti.
Comprende tutte le attività volte a definire l'architettura software del progetto, ovvero la divisione del sistema in sottocomponenti, le interfacce per la loro interazione e i loro modelli di composizione, e tutte le attività di stesura del codice.

2.3.1 Progettazione architetturale

Questa attività mira a fornire una definizione precisa e formale di come sarà l'architettura software di alto e medio livello del prodotto finale.

2.3.1.1 Realizzazione diagrammi UML
Questo compito consiste nel realizzare tutti i diagrammi UML necessari al documento di specifica tecnica. Tutti i diagrammi UML devono rispettare lo standard UML 2.0. Devono essere realizzati i seguenti diagrammi:

- Diagrammi delle attività;
- Diagrammi di sequenza;
- Diagrammi delle classi;
- Diagrammi dei *package*$_G$.

La quantità e la precisione dei diagrammi devono essere condizionate alla loro utilità effettiva.

2.3.1.2 Definizione delle componenti
Questo compito consiste nell'identificare tutte le varie componenti che andranno poi a costituire il prodotto finale. Ogni componente progettato deve essere riportato all'interno della Specifica Tecnica secondo lo schema definito nei formalismi di specifica, nel documento Specifica Tecnica.
Il componente verrà modellato secondo la seguente struttura:

- **Nome:** specifica il nome del componente; ogni nome di ciascun componente deve essere scritto secondo la seguente notazione: E1::E2::E3::E4;la struttura è la seguente:
  - **E1:** è il nome del componente generale, ovvero il *namespace*$_G$ globale;
  - **E2:** è il sotto-componente di E1;
  - **E3:** è il sotto-componente di E2;
  - **E4:** è una singola unità la cui responsabilità è affidata ad un singolo programmatore.

- **Informazioni sul package:** contiene un'immagine che mostra il componente corrente;
- **Descrizione:** descrive in maniera testuale la funzionalità del componente;
- **Sotto-componenti**: specifica se il componente ha altri sotto-componenti;
- **Interazioni con altri componenti:** specifica se il componente ha relazioni con altri componenti.
- **Classi:** specifica se il componente ha eventuali classi associate; se c'è una $classe_G$ deve essere scritta secondo la struttura definita nella successiva sezione.

2.3.1.3 Definizione delle classi

Questo compito consiste nell'identificare i moduli e le classi interne ad ogni componente e stabilire il loro ruolo e le loro interrelazioni. Ogni modulo o classe progettata deve essere riportata all'interno della Specifica Tecnica secondo lo schema definito nei formalismi di specifica, nel documento Specifica Tecnica. La classe verrà modellata secondo la seguente struttura:

- **Nome:** specifica il nome del modulo/classe, in cui tra parantesi si mette se è astratta oppure no; il nome deve essere scritto secondo la seguente notazione: E1::E2::E3::E4::E5;la struttura è la seguente:
  - **E1:** è il nome della componente generale, ovvero il namespace globale;
  - **E2:** è il sotto-componente di E1;
  - **E3:** è il sotto-componente di E2;
  - **E4:** è il sotto-componente di E3;
  - **E5:** è il nome della classe/modulo.
- **Descrizione:** descrive in maniera testuale la funzionalità della classe/modulo;
- **Utilizzo:** descrizione testuale di come viene utilizzata la classe/modulo;
- **Relazioni con altre classi:** viene specificato se la classe/modulo corrente ha relazioni con delle classi di altri componenti;
- **Classe da cui eredita (superclasse):** specifica se la classe eredita da altre classi; si usa la seguente notazione: E1::E2::E3::E4::E5; la struttura è così formata:
  - **E1:** è il nome del namespace globale;
  - **E2:** package utilizzato;
  - **E3 (eventuale):** componente generico del package E2;
  - **E4 (eventuale):** componente generico del sotto-package E3;
  - **E5:** nome della classe.

2.3.1.4 Procedure

Come paradigma di progettazione si userà il *meet in the middle*, ovvero si inizierà con il metodo top-down, scomponendo macro problemi in problemi più piccoli fino ad arrivare a funzionalità sufficientemente atomiche. Dopodiché si passerà al metodo bottom-up, iniziando a raggruppare soluzioni che concorrono allo stesso obiettivo ed assegnandole a moduli precedentemente individuati.

2.3.2 Progettazione di dettaglio

I Progettisti devono descrivere l'architettura di basso livello del sistema e dei singoli componenti nella Definizione di Prodotto in modo chiaro, formale e non ambiguo, integrando quanto scritto nella Specifica Tecnica.

2.3.2.1 Realizzazione diagrammi UML

Questo compito consiste nel realizzare tutti i diagrammi UML necessari al documento di definizione di prodotto. Tutti i diagrammi UML devono rispettare lo standard UML 2.0. Devono essere realizzati i seguenti diagrammi:

- Diagrammi delle attività;
- Diagrammi di sequenza;
- Diagrammi delle classi;
- Diagrammi dei *package*$_G$.

La quantità e la precisione dei diagrammi devono essere condizionate alla loro utilità effettiva.

2.3.2.2 Definizione dei moduli

Questo compito consiste nel definire con la massima precisione il ruolo di ogni modulo, le sue funzioni e le sue relazioni con le altre componenti. Ogni modulo deve essere riportato nella definizione di prodotto secondo il seguente schema:

- **Nome:** specifica il nome del modulo, deve essere preceduto da @ e deve essere il nome del paragrafo corrispondente al modulo;
- **Descrizione:** descrive in maniera testuale la funzionalità del modulo;
- **Metodi:** Devono essere descritti i metodi interni al modulo. Dopo il metodo devono essere elencati i parametri in ingresso tra parentesi tonde. Si dovrà poi procedere a descrivere ogni singolo parametro in ingresso, le dipendenze, se presenti, da altri moduli ed infine il comportamento del metodo nella sua interezza.
  All'interno della definizione di ogni modulo, il riferimento ad altri moduli dovrà essere identificato da una notazione in **grassetto**. Quando vengono menzionati i metodi, invece, viene riportato il loro nome in *corsivo*.

2.3.2.3 Procedure

Si dovrà per prima cosa identificare chiaramente tutte le responsabilità di ogni singola classe/modulo, definendo con un alto livello di astrazione tutti i suoi metodi. Quando si sarà sicuri che tutti i metodi sono correttamente integrati e assegnati alla classe/modulo corretta, si dovranno delineare i parametri di tali metodi e i loro tipi di ritorno.

2.3.3 Progettazione test

2.3.3.1 Progettazione test di sistema

La tabella dei test di sistema avrà la seguente forma:

- **Test:** codice univoco che identifica il test. Per i test di sistema inizierà con TS;
- **Descrizione:** descrizione testuale del test;

- **Requisito:** codice univoco del requisito a cui si fa riferimento;
- **Stato:** lo stato in questo caso sarà uguale per tutti, ovvero D.S. (da sviluppare), in quanto i test verranno eseguiti più avanti.

2.3.3.2 Progettazione test di integrazione

La tabella dei test d'integrazione avrà la seguente forma:

- **Test:** codice univoco che identifica il test. Per i test d'integrazione si inizierà con TI;
- **Descrizione:** descrizione testuale del test;
- **Componente:** associa il componente al corrispondente test;
- **Stato:** lo stato in questo caso sarà uguale per tutti, ovvero D.S. (da sviluppare), in quanto i test verranno eseguiti più avanti.

2.3.3.3 Progettazione test di unità

La tabella dei test d'unità avrà la seguente forma:

- **Test:** codice univoco che identifica il test. Per i test di unità si inizierà con TU;
- **Descrizione:** descrizione testuale del test;
- **Metodi testati:** elenca le funzionalità testate;
- **Stato:** lo stato dovrà essere D.S. nel caso in cui il test non sia ancora stato sviluppato.Nel caso in cui invece il test sia stato eseguito lo stato potrà essere superato oppure fallito.

2.3.3.4 Procedure

2.3.3.4.1 Test di unità

Per progettare un test d'unità si deve:

- Identificare un'unità su cui effettuare il test. Questa unità deve essere quanto più piccola possibile;
- Identificare delle classi di equivalenza che rappresentino tutti i possibili input;
- Identificare tutti gli output attesi in base agli input, dividendo quelli corretti da quelli non;
- Implementare un test che verifichi che il comportamento dell'unità sia uguale a quello atteso;
- Inserire il test in ApertureTrace.

2.3.3.4.2 Test di integrazione

Per progettare un test d'integrazione si deve:

- Identificare due o più componenti che interagiscono tra loro.
- Identificare come questi componenti interagiscono e per quale obiettivo;
- Disegnare un test che verifica il corretto funzionamento dell'integrazione;
- Inserire il test in ApertureTrace.

2.3.3.4.3 Test di sistema

Per progettare un test di sistema si deve:

- Identificare una serie di requisiti funzionali che concorrono alla realizzazione di una singola funzionalità;
- Disegnare un insieme di passi che vadano ad utilizzare tutte le sotto-funzionalità descritte dai requisiti.
- Definire il risultato atteso per ogni singolo passo;
- Inserire il test in ApertureTrace.

2.3.4 Tracciamento

Deve essere realizzata una tabella di tracciamento che collega ogni requisito al componente che lo soddisfa e viceversa. Lo stesso deve essere fatto con i requisiti e le classi. Deve essere inoltre realizzate due tabelle per ogni tipologia di test:

- Test - Unità / Unità - Test
- Test - Componenti / Componente - Test
- Test - Requisiti / Requisito - Test

Le tabelle devono avere la seguente struttura:

2.3.4.1 Tracciamento requisito-componenti

- **Requisito:** specifica il requisito;
- **Componenti:** specifica le componenti che soddisfano il requisito.

2.3.4.2 Tracciamento componente-requisiti

- **Componente:** specifica la componente;
- **Requisiti:** specifica i requisiti che vengono soddisfatti dalla componente.

2.3.4.3 Tracciamento classe-requisiti

- **Classe:** specifica il nome della classe o del modulo;
- **Requisiti:** specifica i requisiti.

2.3.4.4 Tracciamento requisito-classi

- **Requisito:** specifica il requisito;
- **Classi:** specifica i nomi delle classi o moduli.

2.3.4.5 Tracciamento test-unità

- **Test:** specifica il test;
- **Unità:** specifica i nomi delle unità testate.

2.3.4.6 Tracciamento unità-test

- **Unità:** specifica l'unità testata;
- **Classi:** specifica i test effettuati.

2.3.4.7 Tracciamento test-componenti

- **Test:** specifica il test;
- **Componenti:** specifica i nomi dei componenti coinvolti.

2.3.4.8 Tracciamento componente-test

- **Componente:** specifica il componente testato;
- **Test:** specifica i test effettuati.

2.3.4.9 Tracciamento test-requisiti

- **Test:** specifica il test;
- **Requisiti:** specifica i requisiti testati.

2.3.4.10 Tracciamento requisito-test

- **Requisito:** specifica il requisito;
- **Test:** specifica i test effettuati.

2.3.5 Codifica

Questa attività comprende tutte i compiti volti alla realizzazione pratica dell'architettura definita precedentemente e all'implementazione delle sue componenti.

2.3.5.1 Norme generali

Gli sviluppatori dovranno attenersi alle comuni Coding Conventions, in particolare a quelle suggerite dal Prof. Crockford nel suo sito web.[1]
Dovranno inoltre attenersi a tutte le strategie che assicurano la qualità di processo definite nel Piano di Qualifica v.4.2.0 sezione 3. I nomi di classi, metodi, variabili e funzioni dovranno essere in *camel case*$_G$, le parole componenti separate da underscore e scritte in inglese. La prima lettera dovrà essere minuscola, fatta eccezione per le classi che inizieranno sempre con lettera maiuscola. Nessun nome dovrà mai iniziare con un numero.
La ricorsione deve essere implementata solo in caso di vera necessità, altrimenti và evitata. Nel caso in cui venga implementata si dovrà fornire una prova di corretta terminazione e un calcolo approssimato delle risorse richieste per la corretta esecuzione.

2.3.5.1.1 Header dei file

Ogni file dovrà contenere un header strutturato come segue:

- **File:** nome del file;
- **Module:** modulo di appartenenza;
- **Author:** autore;
- **Created:** data di creazione;
- **Description:** descrizione del file;
- **Modification History:** tabella dei cambiamenti effettuati sul file.

2.3.5.1.2 Documentazione dei metodi

Ogni metodo/funzione di codice dovrà essere preceduto/a da un commento contenente le seguenti informazioni:

- **Descr:** breve descrizione del comportamento del metodo;
- **Return:** cosa ritorna la funzione.

2.3.5.1.3 Versionamento del software

Nel versionare il software, le tre cifre di versionamento verranno modificate in base ai seguenti parametri:

- La prima cifra decimale verrà aumentata nel caso in cui vengano introdotti cambiamenti non retro compatibili al *framework*$_G$. Possono essere inclusi cambiamenti minori. Nel caso in cui la prima cifra venga aumentata la seconda e la terza ripartono da 0;
- La seconda cifra decimale verrà aumentata nel caso in cui vengano rilasciate nuove funzionalità retro compatibili. Deve necessariamente essere aumentata se una qualsiasi funzionalità pubblica del framework viene marcata deprecata. È possibile aumentare la seconda cifra anche in caso di rilascio di nuove funzionalità o miglioramenti sostanziali. Al cambiamento della seconda cifra la terza deve ripartire da 0;

[1] http://javascript.crockford.com/code.html

- La terza cifra decimale verrà aumentata nel caso di correzione di errori o altri piccoli cambiamenti retro compatibili;
- La cifra X a 0 è riservata per lo sviluppo iniziale. Le versioni con X a 0 non sono da considerarsi stabili;
- La versione 1.0.0 definisce la prima versione stabile del framework che si andrà a sviluppare.

2.3.5.2 Scrittura del codice

Al fine di uniformare l'ambiente di sviluppo e di testing viene messo a disposizione sul repository un ambiente preconfigurato con tutto l'occorrente per lo sviluppo. Le procedure sottostanti illustrano come avvalersi di tale strumento.

2.3.5.2.1 Procedure

Per installare sul proprio computer l'ambiente di lavoro completo si deve:

- Assicurarsi di avere MongoDB installato localmente. Il file di installazione è disponibile sul sito ufficiale;
- Assicurarsi di avere node.js installato localmente. Il file di installazione è disponibile sul sito ufficiale;
- Aggiornare la propria copia del repository locale all'ultimo commit;
- Entrare nella cartella package_ npm;
- Digitare sul terminale npm install;
- Digitare su terminale npm link per creare un link fittizio.
- Entrare dentro la cartella maap_ project;
- Digitare sul terminale npm link maaperture;
- Digitare sul terminale npm install;

Una volta completato il processo il framework maaperture sarà disponibile e configurato correttamente.

2.3.5.2.2 Strumenti

Eclipse.

2.3.5.3 Scrittura dei test

2.3.5.3.1 Procedure

Per la scrittura dei testi di unità del client si deve:

- Creare un nuovo file nella cartella test.
- Chiamare il nuovo file con il nome della componente da testare
- Copiare il template per i test del client e adattarlo alla componente da testare.
- Scrivere il test.

2.3.5.3.2 Strumenti

Eclipse,Mocha.

3 Processi di supporto

3.1 Comunicazioni

3.1.1 Comunicazioni interne

Le comunicazioni interne sono divise in comunicazioni formali e informali, in base alla natura della comunicazione stessa. A questa categoria appartengono tutti gli avvisi di impegni collettivi del team, la segnalazione di imprevisti organizzativi o la convocazione di riunioni.

3.1.1.1 Comunicazioni interne formali Le comunicazioni interne formali appartengono all'area di organizzazione generale del gruppo.

3.1.1.1.1 Procedure Ogni *email*$_G$ inviata a tale indirizzo, per segnalazioni di imprevisti o convocazioni di riunioni, verrà inoltrata alla email personale di ogni componente nel gruppo.

3.1.1.1.2 Strumenti È stata creata una mailing list all'indirizzo aperture-team@googlegroups.com. La mailing list è collegata ad un gruppo su *Google*$_G$ Groups, che permette di velocizzare le comunicazioni implementando una *simil-chat*$_G$ tramite email distribuite; inoltre mantiene uno storico di qualsiasi comunicazione. Va usato quindi come strumento per discussioni e comunicazioni ufficiali. Una mail usata per la comunicazione interna deve avere la seguente struttura:

- **Destinatario:** l'unico indirizzo possibile è aperture-team@googlegroups.com;
- **Mittente:** è l'indirizzo email personale di chi scrive il messaggio;
- **Oggetto:** deve essere una sintesi del contenuto della email;
- **Corpo:** contiene una descrizione completa di tutte le informazioni che si vogliono comunicare; esse devono essere facilmente capibili e possibilmente strutturate in maniera che siano leggibili; inoltre le frasi devono essere corte e contenere termini di linguaggio comune;
- **Allegati:** si possono usare, qualora se ne ritenga necessario, degli allegati per completare l'informazione che si vuole spedire. Un esempio utile potrebbe essere quello di allegare un verbale o un resoconto di incontri con il Proponente o con il Committente.

3.1.1.2 Comunicazioni interne informali Le comunicazioni riguardanti dubbi o chiarimenti marginali o comunque non inerenti all'organizzazione del gruppo, vengono definiti informali. Per loro natura, queste comunicazioni devono essere veloci e non complesse, quindi abbiamo deciso di utilizzare strumenti immediati di comunicazione, quali le *chat*$_G$.

3.1.1.2.1 Procedure Un membro del gruppo esprime il dubbio come messaggio istantaneo, il quale sarà visibile da ogni altro componente.

3.1.1.2.2 Strumenti

3.1.1.2.3 *Skype*$_G$

Per facilitare le comunicazioni istantanee è stata creata su Skype una chat di gruppo dove sono presenti tutti i membri del team. Tale chat deve essere usata solo per conversazioni non ufficiali o per discussioni non importanti come lo scambio di articoli, consigli o comunicazioni non rilevanti.

3.1.1.2.4 Gruppo *Facebook*$_G$

Data la grande diffusione di questo *social network*$_G$ e l'aderenza di tutti i membri del gruppo ad esso, abbiamo deciso di creare un gruppo apposito su questa piattaforma. Facebook fornisce un accesso ancora più immediato di Skype, data la sua diffusione su dispositivi mobile.

3.1.2 Comunicazioni esterne

3.1.2.1 Procedure L' email può essere usata soltanto dal Responsabile di Progetto e verrà utilizzata per tutte le comunicazioni che il gruppo terrà con l'*ambiente*$_G$ esterno.

3.1.2.2 Strumenti L' email ufficiale del gruppo è ApertureSWE@gmail.com.
Una mail usata per la comunicazione interna deve avere la seguente struttura:

- **Destinatario:** l'indirizzo email può essere quello del Proponente oppure quello del Committente, che può essere o il Prof. Tullio Vardanega o il Prof. Riccardo Cardin;
- **Mittente:** l'unico indirizzo possibile è ApertureSWE@gmail.com e deve essere usato solo dal Responsabile di Progetto;
- **Oggetto:** deve essere una sintesi del contenuto della email;
- **Corpo:** contiene una descrizione completa di tutte le informazioni che si vogliono comunicare; esse devono essere facilmente capibili e possibilmente strutturate in maniera che siano leggibili; inoltre le frasi devono essere corte e contenere termini specifici dell'ambiente *business*$_G$ inerenti al progetto;
- **Allegati:** si possono, qualora se ne ritenga necessario, usare degli allegati, per completare l'informazione che si vuole spedire. Un esempio utile potrebbe essere quello di allegare un verbale o un resoconto di incontri con il Proponente o con il Committente.

3.1.3 Riunioni

Ogni membro del team può richiedere che venga organizzata una riunione al Responsabile che, una volta verificata la motivazione della richiesta, accoglie o meno la stessa. Il Responsabile di Progetto ha la facoltà di indire riunioni qualora sentisse la necessità di farlo. Alle riunioni è gradita, ma non richiesta, la partecipazione di tutti i membri del gruppo. È giustificata l'assenza nel caso in cui la riunione riguardi temi non inerenti al ruolo di progetto che si sta ricoprendo o nel caso di impegni validi, giustificabili e improrogabili. Ad ogni riunione verrà eletto un segretario che avrà il compito di annotare gli argomenti di discussione e le decisioni prese durante la stessa. Dovrà inoltre redigere un verbale che verrà ufficialmente raccolto e archiviato come documentazione interna entro tre giorni dalla data della riunione. La struttura del Verbale deve contenere i seguenti elementi:

- Data;
- Luogo;
- Ora;

- Durata;
- Partecipanti interni;
- Partecipanti esterni;
- Motivazione della riunione/incontro;
- Argomenti trattati.

3.1.3.1 Procedura La riunione deve essere segnalata sul calendario di gruppo con almeno due giorni di anticipo e non deve andare a sovrapporsi con impegni precedenti di altri componenti, a meno che la loro presenza non possa essere esclusa. Deve inoltre essere inviata una email di promemoria a tutti i membri del gruppo indicante giorno, ora e motivazione della riunione.

3.1.3.2 Strumenti Google Calendar.

3.1.4 Comunicazioni con i Committenti e Proponenti

Prima di richiedere un colloquio personale con il Proponente e/o Committente il team deve preparare un documento che riassume i punti che verranno discussi contenente argomenti, dubbi e domande da porre.
Alla fine del colloquio verrà redatto un verbale, vedi sezione 2.3.3 di questo documento, che riassume gli argomenti trattati, le conclusioni, nonché le strategie che si sono delineate in concordanza col Committente/Proponente.

3.1.4.1 Procedura I colloqui con i committenti di progetto possono essere richiesti dal Responsabile mediante la email ufficiale del team. Tutti i membri devono essere avvisati prima di richiedere un incontro con il Committente.

3.1.4.2 Strumenti Registratore audio.

3.2 Verifica

Questa attività comprende tutte i compiti volti a verificare che i processi precedentemente attuati non abbiano introdotto errori. Per i documenti il processo di verifica viene istanziato nel momento in cui gli stessi passano dalla versione X.0.Z alla versione X.1.Z. È sempre compito del responsabile assegnare ai Verificatori il ticket di verifica per il processo in questione.

3.2.1 Metriche per errori

Sono riportate delle metriche utili alla valutazione degli errori che si potranno riscontrare durante le attività di Verifica.

Errore	Gravità	Priorità
Indici Errati	Alta	Urgente
Tracciamento errato	Media	Breve
Errore di progettazione	Alta	Alta
Errore ortografico o di formattazione	Bassa	Entro Milestone

Valore Gulpease fuori norma	Bassa	Entro Milestone
Violazione norme di codifica	Media	Breve

Tabella 2: Tabella errori

La gravità di un errore può essere:

- **Bassa**: l'errore ha impatto minimo e su aspetti marginali;
- **Media**: l'errore ha un impatto significativo e causa un ritardo o un aumento di costo considerevole;
- **Alta**: l'errore rende il prodotto inutilizzabile e porta potenzialmente al fallimento del progetto.

I tempi di risoluzione possono essere:

- **Entro milestone**: l'errore va corretto prima della milestone;
- **Breve**: l'errore va corretto entro qualche giorno dalla sua scoperta;
- **Urgente**: l'errore va corretto il prima possibile.

A seguito del riscontro di un errore il verificatore apre un ticket per la correzione dell'errore, come descritto in sezione 2.2.1.2.5, e la persona responsabile dell'attività procederà nella correzione, chiudendo così il relativo ticket aperto.

3.2.2 Walkthrough

Questa tecnica prevede di scorrere l'intero documento alla ricerca di errori senza nessun criterio di ricerca specifico.

3.2.2.1 Procedure

Partendo dall'inizio del documento, lo si legge fino alla fine segnalando ogni errore trovato e si registrano i più frequenti affinchè siano poi aggiunti alla Lista di Controllo.

3.2.2.2 Strumenti

TeXstudio, GNU Aspell.

3.2.3 Inspection

Questa tecnica prevede di scorrere l'intero documento alla ricerca mirata di errori.

3.2.3.1 Procedure

Partendo dall'inizio del documento, si controllano tutte le parti che più di frequente contengono errori, segnalando eventuali riscontri.

3.2.3.2 Strumenti

Lista di Controllo, TeXstudio, GNU Aspell.

3.2.4 Analisi statica

L'analisi statica si basa sull'analisi del codice scritto senza che esso venga eseguito, e sul risultato di test eseguiti su di esso.

3.2.4.1 Procedure

L'analisi statica viene effettuata direttamente dall'IDE in tempo reale durante la stesura del codice.

3.2.4.2 Strumenti

Eclipse Metrics ,JSHint.

3.2.5 Test Unità

I testi di unità sono un metodo per testare singole unità di codice, come ad esempio un modulo od una classe. I test di unità dovranno essere fatti sul minimo quantitativo di codice che abbia senso testare.

3.2.5.1 Procedure

Per eseguire i test di unità sul client sarà sufficiente aprire un terminale nella cartella del progetto contenente il client e digitare karma.
I test verranno eseguiti e la copertura generata automaticamente.

3.2.5.2 Strumenti

Mocha, Istanbul, Karma

3.2.6 Statement e branch coverage

Statement e branch coverage sono delle metriche che consentono di sapere quanta percentuale di codice è stata testata. Lo statement coverage indica quante linee di codice sono state testate rispetto alla totalità, il branch coverage indica quanti rami logici sono stati testati rispetto a tutti quelli intraprendibili. Istanbul è stato integrato con Karma per far sì che la copertura venga calcolata e il resoconto generato automaticamente all'esecuzione dei test. Nel caso in cui si volesse effettuare l'operazione manualmente sono di seguito riportare le procedure da seguire.

3.2.6.1 Procedure

3.2.6.1.1 Verificare copertura di un test

Per verificare la copertura di un test bisogna usare il seguente comando:

istanbul cover mytest.js

3.2.6.1.2 Generare un report

Per generare un report della copertura del codice bisogna usare il seguente comando:

istanbul report

Sono disponibili i seguenti formati di output:

- html;
- lcovonly, produce un file lcov.info;
- lcov, produce html + lcov files. Questo è il formato di default;
- cobertura, produce un file xml contenente la copertura;
- text-summary, produce un resoconto testuale della copertura, tipicamente su console;
- text, produce una tabella di testo dettagliata con la copertura di ogni file.

3.2.6.1.3 Verificare la copertura minima

Istanbul permette di impostare una soglia minima di copertura e di verificare automaticamente se tale soglia è rispettata. Per verificare la copertura attuale contro quella minima bisogna usare il seguente comando:

istanbul check-coverage

Il comando ritorna 1 se la copertura minima è rispettata, altrimenti ritorna 0.

3.2.6.2 Strumenti

Istanbul.

3.2.7 Tracciamento

3.2.7.1 Norme generali

Devono essere generate le seguenti tabelle di tracciamento per i test effettuati:

3.2.7.2 Tracciamento componente-test d'unità

La tabella per il tracciamento componente-test d'unità avrà la seguente forma:

- **Unità:** il nome del componente associato al corrispondente test;
- **Test:** codice univoco che identifica il test.

3.2.7.3 Tracciamento test d'unità-componente

La tabella per il tracciamento test d'unità-componente avrà la seguente forma:

- **Test:** codice univoco che identifica il test;
- **Unità:** il componente associato al corrispondente test.

3.2.7.4 Tracciamento componente-test d'integrazione

La tabella per il tracciamento componente-test d'integrazione avrà la seguente forma:

- **Componente:** il nome del componente associato al corrispondente test;
- **Test:** codice univoco che identifica il test.

3.2.7.5 Tracciamento test d'integrazione-componente

La tabella per il tracciamento test d'integrazione-componente avrà la seguente forma:

- **Test:** codice univoco che identifica il test;
- **Componenti:** i componenti associati al corrispondente test.

3.2.7.6 Tracciamento requisito-test di sistema

La tabella per il tracciamento requisito-test di sistema avrà la seguente forma:

- **Requisito:** il codice del requisito soddisfatto dal test;
- **Descrizione:** descrizione testuale del requisito.
- **Test:** codice univoco che identifica il test.

3.2.7.7 Tracciamento test di sistema-requisiti

La tabella per il tracciamento test di sistema-requisiti avrà la seguente forma:

- **Test:** codice univoco che identifica il test;
- **Requisiti:** i requisiti associati al test.

3.3 Documentazione

Questo processo di supporto comprende tutte le attività volte alla realizzazione della documentazione di supporto necessaria al progetto.
La stesura della documentazione è un processo di supporto fondamentale per documentare progressi e risultati del lavoro del gruppo, fornire regole e procedure su come eseguire tale lavoro e fornire una documentazione precisa sul prodotto finale, sulle sue componenti e sulle sue modalità di utilizzo.

3.3.1 Norme generali

Viene fornito un template in LaTeX per la Realizzazione della documentazione, sia interna che esterna, che i membri del gruppo dovranno seguire nella stesura dei documenti.
Ogni documento ha un header che riporta logo e nome del gruppo sulla sinistra ed un footer che riporta il nome e la versione del documento a sinistra e il numero della pagina a destra. Entrambi sono presenti in ogni pagina, eccetto la prima.

3.3.1.1 Prima pagina

La prima pagina di ogni documento deve riportare nel seguente ordine:

- Il logo esteso del gruppo;
- Il nome del progetto;
- Il nome del documento;

- Informazioni sul documento (versione, data creazione, data ultima modifica, stato del documento, uso del documento, i redattori del documento, i Verificatori, chi ha approvato il documento e la *distribuzione*$_G$ del documento);
- Breve sommario del documento.

3.3.1.2 Seconda pagina

La seconda pagina deve riportare il diario delle modifiche apportate al documento dalla sua creazione fino alla versione corrente, strutturato nella seguente maniera:

Versione	Data	Autore	Modifiche effettuate

Le modifiche dovranno essere ordinate cronologicamente dalla più recente alla meno recente.
La terza pagina deve riportare l'indice del documento.
Ogni tabella ed ogni immagine inserita deve essere descritta da una didascalia, in cui compare un numero identificativo incrementale per la tracciabilità.

3.3.1.3 Norme tipografiche e stili di testo

Nella stesura della documentazione si dovranno seguire le seguenti indicazioni:

- I documenti dovranno essere grammaticalmente e sintatticamente corretti e scritti in modo fluido;
- L'elenco puntato termina con il punto e virgola oppure con il punto se è l'ultimo elemento;
- Un carattere di punteggiatura non deve mai seguire un carattere di spaziatura;
- Il testo racchiuso tra parentesi non deve aprirsi o chiudersi con un carattere di spaziatura e non deve terminare con un carattere di punteggiatura;
- Le lettere maiuscole vanno poste solo dopo il punto, il punto di domanda, il punto esclamativo e all'inizio di ogni elemento di un elenco puntato, oltre che dove previsto dalla lingua italiana. È inoltre utilizzata l'iniziale maiuscola nel nome del team, del progetto, dei documenti, dei ruoli di progetto, delle fasi di lavoro e nelle parole Proponente e Committente;
- Nel caso in cui ci si riferisca ad un documento, il titolo di quest'ultimo dovrà essere scritto in corsivo e si dovrà riportare la versione riferita;
- I ruoli di progetto (es. Analista) dovranno essere scritti con la lettera iniziale maiuscola;
- Gli acronimi dovranno essere scritti in lettere maiuscole;
- È preferibile usare la forma attiva a quella passiva;
- Quando possibile usare elenchi puntati invece che frasi;
- Usare termini specifici e segnare i termini del glossario in corsivo e con la G in pedice;
- Dividere i documenti in sezioni e sottosezioni titolate;
- Le date dovranno essere espresse nella forma AAAA-MM-GG secondo lo standard *ISO*$_G$ 8601:2004;

- Le attività (es. Verifica) vanno scritte con la lettera iniziale maiuscola;
- Gli elenchi puntati con un primo livello di profondità sono formati da dei pallini neri pieni, tranne quando si deve elencare una sequenza numerata di istruzioni da fare in un ordine stabilito, allora in quel caso si usa un elenco numerato;
- Gli elenchi puntati con un secondo livello di profondità hanno un trattino.

3.3.1.4 Sigle e abbreviazioni

Le sigle e le abbreviazioni dovranno essere utilizzate solo in contesti in cui lo spazio è limitato come tabelle e diagrammi. Sono previste le seguenti abbreviazioni:

- AdR = Analisi dei Requisiti;
- GL = Glossario;
- NdP = Norme di Progetto;
- PdP = Piano di Progetto;
- PdQ = Piano di Qualifica;
- SdF = Studio di Fattibilità;
- ST = Specifica Tecnica;
- DdP = Definizione di Prodotto;
- RA = Revisione di Accettazione;
- RP = Revisione di Progettazione;
- RQ = Revisione di Qualifica;
- RR = Revisione dei Requisiti;
- AS = Aperture Software;
- DP = Design Pattern.

Per i ruoli si useranno le seguenti abbreviazioni:

- AN = Analista;
- VR = Verificatore;
- RE = Responsabile;
- AM = Amministratore;
- PT = Programmatore;
- PR = Progettista.

3.3.1.5 Versionamento

Il versionamento verrà effettuato sia sul codice che sui documenti prodotti dal gruppo al fine di differenziare e rendere immediatamente riconoscibile la fase di sviluppo in cui ci si trova attualmente, mantenendo uno storico organizzato dei cambiamenti effettuati.
Il numero di versionamento deve essere nella forma X.Y.Z, con X,Y e Z numeri interi non negativi. Tutti gli elementi devono salire numericamente di una unità alla volta. Ogni qualvolta che una versione viene rilasciata non può più essere effettuato alcun cambiamento ad essa. Ogni modifica sarà inserita nella versione successiva.
Nel versionare i documenti, le tre cifre di versionamento verranno modificate in base alle seguenti regole:

- X: indica il numero di uscite formali del documento, diviso come segue:
  1. Fase di Analisi, si estende fino alla Revisione dei Requisiti;
  2. Fase di Analisi in Dettaglio, si estende fino all'ingresso nella fase di Progettazione;
  3. Fase di Progettazione Architetturale, si estende fino alla Revisione di Progettazione;
  4. Fase di Progettazione di Dettaglio e Codifica,si estende fino alla Revisione di Qualifica;
  5. Fase di Verifica e Validazione, si estende fino alla Revisione di Accettazione e alla fine del progetto.
- Y: indica la fase di sviluppo del documento e varia come segue:
  0. Fase di stesura del documento, dove il documento è ancora modificabile;
  1. Fase di Verifica del documento, dove il documento è formalmente corretto, è modificabile e sta venendo controllato dal Verificatore;
  2. Documento ultimato e approvato.

  Sarà compito del Responsabile impostare la versione a X.1.0 nel momento in cui assegnerà il *ticket*$_G$ di controllo ai Verificatori e impostare la versione a X.2.0 nel caso in cui il documento risulti approvato dagli stessi.
- Z: indica il numero di modifiche minori apportare al documento. Aumenta al termine di ogni sessione di lavoro sul documento. Non ha un limite massimo.

3.3.1.6 Ciclo di vita

Ogni documento prodotto attraversa uno specifico percorso di vita, riassunto dai seguenti 4 stati:

1. **Creazione:** viene creata la struttura base del documento senza alcun contenuto;
2. **Lavorazione in corso:** il contenuto del documento viene scritto, ed è soggetto a continue modifiche;
3. **Verifica:** il contenuto del documento viene completato e viene assegnato ai Verificatori che dovranno svolgere la verifica del documento;
4. **Approvazione:** per ogni documento che è stato verificato, il cui esito della Verifica è stato soddisfacente, viene inviato al Responsabile di Progetto, che lo approva. Passato questo ultimo stato, il documento si trova nello stato finale per quella specifica versione.

Ciascun documento può trovarsi in uno stesso stato più volte: quando il documento approvato necessita di una revisione con il Committente, il documento ricomincia il *ciclo di vita*$_G$ che al completamento porterà ad una nuova versione incrementata.

3.3.1.7 Norme stilistiche

Di seguito vengono elencate le norme che dovranno essere seguite durante la stesura di una definizione:

- Reperire la definizione da una fonte autorevole, oppure ottenerla da più fonti;
- Non superare le 400 lettere per definizione;
- Preferibilmente, iniziare la definizione con un sostantivo;
- Non iniziare mai una definizione con " É un ".

3.3.1.8 Marcatore termini

Il nostro gruppo si avvale di uno *script*$_G$ automatico in linguaggio *Python*$_G$ per la marcatura dei termini definiti nel Glossario. Questo strumento esegue una scansione di ogni documento basandosi sull'elenco dei termini nel Glossario e marca ogni termine (se presente) con una G in pedice.

3.3.1.9 Procedure

3.3.1.9.1 Modifica di un documento

- Modificare od ampliare il documento;
- Aggiungere una riga nella tabella delle modifiche con data, autore e modifiche effettuate;
- Eseguire lo script per il calcolo dell'indice di Gulpease e verificare che rientri nei valori accettati, altrimenti correggere il documento.
- Eseguire il commit nel repository.

3.3.1.9.2 Inserimento termine di glossario

Ogniqualvolta un membro del gruppo incontra un termine di questo tipo, durante la stesura di qualsiasi documento, deve seguire la seguente *procedura*$_G$:

1. Controllare che il termine non sia già presente all'interno del Glossario;
2. Nel caso non fosse presente, interrompere la stesura del documento corrente per inserire il termine nel Glossario;
3. Collocare il termine rispettando l'ordine alfabetico delle definizioni;
4. Dare una definizione chiara e concisa del termine;
5. Nel caso in cui la definizione del termine corrente causi la definizione di altri termini, bisogna tenerne traccia e definirli in ordine;
6. Aggiornare la tabella delle modifiche all'interno del documento Glossario, modificando conseguentemente il numero di versione del documento stesso.

3.3.1.9.3 Avanzamento di versione

Per procedere con l'avanzamento di versione di un documento, il Responsabile deve:

1. Assicurarsi che i lavori di stesura del documento siano stati ultimati;
2. Aggiornare la versione del documento a X.1.0;
3. Assegnare un ticket di verifica sul documento ad uno o più Verificatori secondo la procedura descritta in 2.2.1.2.4;
4. Nel caso in cui il verificatore approvi il lavoro, il responsabile aggiorna la versione a X.2.0 e il documento diventa definitivo;
5. Nel caso in cui il verificatore rilevi degli errori, quest'ultimo aprirà dei ticket di errore come descritto in 2.2.1.2.5 e attenderà la loro risoluzione prima di ricominciare il processo di verifica;

3.3.1.9.4 Finalizzazione dei lavori

Al termine della lavorazione di un documento, prima che esso venga passato ai verificatori, si deve eseguire la seguente procedura:

- Eseguire lo script per il marcamento dei termini da glossario;
- Eseguire lo script per il calcolo dell'indice di Gulpease e, se non dovesse risultare sufficiente, modificare il documento.

3.3.1.10 Strumenti

TeXstudio,Astah, script glossario, script Gulpease;

3.4 Gestione di configurazione

Questo processo di supporto comprende tutte le attività volte a gestire l'evoluzione del progetto ed a tenere traccia di tutte le sue versioni e configurazioni.

3.4.1 Gestione del repository

Viene messo a disposizione un repository Git_G su $GitHub_G$ per la gestione e il versionamento di codice e documenti tra i vari membri del gruppo.
Il repository pubblico è disponibile all'indirizzo:

https://github.com/ApertureSoftware/AperturePublic.git.

3.4.1.1 Norme generali

Nell' utlizzo del repository di gruppo, tutti i membri dovranno attenersi strettamente alle seguenti regole:

- È severamente vietato introdurre sul repository qualsiasi file che non rispetti in qualche sua parte le norme descritte in questo documento;
- È severamente vietato introdurre nel branch *master* del repository file contenenti errori, come ad esempio file non compilanti, incompleti o che hanno evidenziato errori durante i test;

- Nel caso ci fosse necessità di condividere file incompleti o contenenti errori, dovrà essere creato un branch apposito. Quest'ultimo, una volta terminati i lavori, verrà unito al branch principale mediante la procedura descritta in 3.4.1.3.4, previa approvazione del Responsabile;
- Ogni commit deve essere accompagnato da un messaggio che ne descrive il contenuto. Tale messaggio deve essere sufficientemente chiaro e significativo.

3.4.1.2 *Branch*$_G$

Nel repository saranno disponibili vari branch per favorire lo sviluppo del codice da parte degli sviluppatori.

3.4.1.2.1 Master

Il branch principale sarà chiamato *master* e conterrà l'ultima versione del software stabile rilasciata. Affinché una nuova versione possa essere caricata nel branch master, quest'ultima deve compilare senza errori o warning e deve aver superato tutti i test disegnati per verificarne la qualità.

3.4.1.2.2 Secondari

Gli sviluppatori possono richiedere la creazione di un branch secondario, provando nuove strategie di sviluppo senza alterare il branch master. Il Responsabile di Progetto è il responsabile dell'approvazione della richiesta.

3.4.1.3 Procedure

3.4.1.3.1 Sincronizzare il repository

Per copiare in locale il contenuto del repository bisogna eseguire il comando *git pull* nella cartella che contiene il repository locale.

3.4.1.3.2 Salvare una modifica in locale

Con il comando git add *nome_file* si aggiunge un file a quelli soggetti a controllo di versione. Usando il comando *git commit -m "Messaggio di commit "* si salvano le modifiche effettuate ai file.

3.4.1.3.3 Inviare le modifiche al repository remoto

Per inviare le modifiche al repository remoto si deve usare il comando *git push*.

3.4.1.3.4 Effettuare il merge tra due branch

La sequenza di operazioni corretta per effettuare il merge del branch test nel branch master è la sequente:

- git checkout master;
- git pull origin master;
- git merge test;
- git push origin master.

3.4.1.3.5 Gestione dei conflitti

Nel caso si verifichino conflitti sarà necessario usare il comando *git mergetool* e risolvere il conflitto tramite l'apposito strumento.

3.4.1.4 Strumenti

Git.

A Descrizione degli strumenti di lavoro

A.1 Coordinamento

A.1.1 Redmine

Come piattaforma per la gestione del progetto è stato scelto Redmine. Le principali funzioni che esso fornisce sono:

- Creazione di un nuovo progetto;
- Un sistema di gestione dei ticket;
- Il grafico Gantt delle attività;
- Un calendario per organizzare i compiti e le attività;
- La visualizzazione del *repository*$_G$ associato al progetto;
- Un sistema di gestione del tempo e delle milestone.

A.1.2 Google Drive

Drive è uno strumento per la condivisione dei file online, per velocizzare e semplificare l'organizzazione di alcune attività. I file caricati su Drive non sono soggetti a versionamento, pertanto si dovranno caricare soltanto file informativi come guide, tutorial o comunque file di cui non si deve tenere traccia.

A.1.3 Google Calendar

Viene messo a disposizione del team un calendario per coordinare temporalmente le attività e gli impegni di ciascuno dei componenti. Tutti i membri del team useranno il calendario messo a disposizione su Google Calendar per segnalare i giorni in cui per loro non sarà possibile lavorare al progetto o partecipare a riunioni e sessioni di lavoro di gruppo. I componenti si impegnano a segnare sul calendario quanto prima qualsiasi impegno o impedimento dovesse sorgere e a controllare periodicamente il suddetto per essere aggiornati su eventuali impegni altrui e/o di gruppo. Il calendario verrà inoltre usato dal Responsabile per la segnalazione di riunioni o sessioni di lavoro di gruppo.

A.1.4 Jenkins

Si è scelto di adottare Jenkins per applicare l'integrazione continua allo sviluppo del progetto. Tale software permette di pianificare ed impostare la compilazione del codice e dei documenti. Mette inoltre a disposizione un'interfaccia su cui è possibile visualizzare lo stato del codice prodotto. Esso è infatti in grado di interagire con il software di versionamento e, se disponibile, con software per l'esecuzione di test sul codice prodotto.

A.2 Strumenti per la pianificazione

A.2.1 OpenProj

Si è scelto di utilizzare il software OpenProj per gestire e pianificare le attività e le risorse inerenti al progetto. Questo software è gratuito e open source, sostituisce i più costosi software di project management, inoltre è multipiattaforma, ovvero è disponibile per più sistemi operativi. Questo software dispone delle seguenti caratteristiche:

- Open-source;
- Fornisce diagrammi di Gantt;
- Fornisce diagrammi di PERT;
- Generazione automatica della Work Breakdown Structure(WBS), partendo dal Gantt;
- Permette di calcolare le metriche Schedule Variance (SV) e Budget Variance (BV).

A.3 Strumenti per la documentazione

A.3.1 LaTeX

Per la stesura dei documenti verrà utilizzato il *linguaggio di markup*$_G$ LaTeX. Esso rende possibile definire template di documenti standardizzati da poter applicare a qualsiasi contenuto, separando così formattazione da contenuto e facilitando il lavoro di stesura. LaTeX inoltre dispone di qualsiasi strumento di formattazione di cui si possa avere bisogno, eliminando il bisogno di strumenti ausiliari alla stesura dei documenti.

A.3.2 Script Gulpease

Script realizzato dal team per il calcolo automatico dell'indice di Gulpease.

A.3.3 Astah

Come strumento per rappresentare i diagrammi delle classi, diagrammi di sequenza e diagrammi di attività, è stato utilizzato il software di modellazione Astah. Questo strumento è stato consigliato dai docenti del corso di Ingegneria del Software, in quanto è gratuito, semplice da usare e adeguato alle nostre esigenze. Tutti i diagrammi seguono gli standard UML 2.0.

A.3.4 TeXstudio

TeXstudio è l'editor di LaTeXconsigliato per la scrittura della documentazione.

A.4 Strumenti per la codifica

A.4.1 Eclipse

Come ambiente di sviluppo si consiglia Eclipse, in quanto dispone di numerosi plug in per gestire tutti i linguaggi che il team andrà ad utilizzare e anche molti per la verifica.

A.5 Strumenti per il tracciamento

A.5.1 ApertureTrace

Al fine di semplificare e automatizzare le attività di tracciamento è stato realizzato un software che genera automaticamente tutte le tabelle a partire da un database in cui sono stati precedentemente inseriti i dati. ApertureTrace genera automaticamente le tabelle in LaTeXper il tracciamento di requisiti, componenti e test.

A.6 Strumenti per la verifica

A.6.1 GNU Aspell

GNU Aspell è uno strumento Open Source per il controllo degli errori di scrittura. Sottolinea in rosso le parole non riconosciute e ne suggerisce una correzione.

A.6.2 Eclipse Metrics

Metrics è un plug in per Eclipse che consente di calcolare automaticamente una serie di metriche utili a sviluppare codice di qualità. Tra le principali metriche calcolabili sono presenti:

1. Linee di codice totali;
2. Accoppiamento afferente;
3. Accoppiamento efferente;
4. Complessità ciclomatica;
5. Numero di parametri per metodo;
6. Indice di instabilità.

A.6.3 JSHint

JSHint è uno strumento open source di analisi statica per il riconoscimento di errori e di potenziali problemi nel codice Javascript. È inoltre utile per imporre l'utilizzo delle norme di codifica agli sviluppatori.

A.6.4 Mocha

Mocha è un framework di test per JavaScript basato su node.js. È estremamente ricco di funzionalità. Tra le principali ci sono:

- Supporto ai testi asincroni e alle *promises*;
- Mappa le eccezioni non catturate;
- Timeout dei test asincroni;
- Calcolo della durata di ogni test;
- Controllo del *memory leak*;
- Supporto al debugger di node.js.

A.6.5 Karma

Karma è uno strumento per l'esecuzione dei test direttamente sul browser ed è perfettamente integrabile con AngularJS, il che lo rende ideale per i test sul client.

A.6.6 Istanbul

Istanbul è uno strumento per il calcolo della copertura di codice Javascript. Le sue funzionalità principali sono:

- Tracciamento di statement, branch e function coverage;
- Indipendenza dal gestore dei test;
- Generazione di report in HTML, testo semplice e LCOV;
- Compatibilità con Mocha;
- Consente di impostare un limite minimo di copertura e di testare se tale limite è rispettato.

B Lista di Controllo

Durante le attività di Walkthrough sono stati riscontrati principalmente i seguenti errori:

- **Stili di testo:**
  - Elenco puntato non termina con ; nel caso in cui la riga non sia l'ultima, o non termina con . nel caso lo sia;
  - Elenco puntato non inizia con lettera maiuscola;
  - Le definizioni del glossario iniziano con È un;
  - È viene scritta incorrettamente come E';
  - Alcuni riferimenti ai documenti non vengono scritti in corsivo;
  - I vari ruoli non cominciano con la lettera maiuscola;
  - Le parole Proponente e Committente non vengono scritte con la lettera iniziale maiuscola;
  - Alcune tabelle non rispettavano lo standard stabilito;
  - I nomi delle tecnologie utilizzate vengono scritti in maniera erronea, ad esempio è stato scritto Angularjs al posto di AngularJS oppure Nodejs al posto di Node.js;
  - Nella scrittura della Specifica Tecnica non è stato mandato a capo il testo che descrive una sezione.

- **Lingua italiana:**
  - Alcune parole accentate vengono scritte con l'apostrofo e non con il relativo accento;
  - In alcune frasi era omesso lo spazio tra la virgola e la parola successiva.
- **Sintassi UML:**
  - Il nome di alcuni casi d'uso non rispecchiavano la descrizione dello stesso.
- **LaTeX**
  - La parola LaTeX non viene scritta con il comando giusto;
  - Per le parole accentate non viene usato il comando apposito.