2 marzo 2015

Piano di Qualifica

Informazioni sul documento

Nome Documento	Piano di Qualifica
Versione	1.0
Stato	*Formale*
Uso	*Interno*
Data Creazione	2 marzo 2015
Data Ultima Modifica	2 marzo 2015
Redazione	
Approvazione	
Verifica	
Lista distribuzione	*LateButSafe*

Prof. Tullio Vardanega
Prof. Riccardo Cardin
Proponente Zucchetti S.p.a.

Registro delle modifiche

Tab 1: Versionamento del documento

Versione	Autore	Data	Descrizione
0.1		2 marzo 2015	Prima stesura del documento

latebutsafe@gmail.com

Storico

pre-RR

Versione 1.0	**Nominativo**
Redazione	
Verifica	
Approvazione	

Tab 2: Storico ruoli pre-RR

Indice

Sommario

Il presente documento contiene le norme e le convenzioni che il gruppo LateButSafeintende adottare durante l'intero ciclo di vita del prodotto software Premi.

1 Introduzione

1.1 Scopo del documento

Il Piano di Qualifica ha lo scopo di descrivere le strategie che il gruppo di lavoro ha deciso di adottare per perseguire obiettivi qualitativi da applicare al proprio prodotto. Per ottenere tali obiettivi è necessario un processo di verifica continua sulle attività svolte; questo consentirà di rilevare e correggere anomalie e incongruenze in modo tempestivo e senza spreco di risorse.

1.2 Scopo del Prodotto

Lo scopo del progetto è la realizzazione un software per la creazione ed esecuzione di presentazioni multimediali favorendo l'uso di tecniche di storytelling e visualizzazione non lineare dei contenuti.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento (((Glossario v?.?.?))). Ogni occorrenza di vocaboli presenti nel Glossario è marcata da una "G" maiuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- Norme di Progetto: (((Norme di Progetto v?.?.?)));
- Capitolato d'appalto C4: Premi: Software di presentazione "better than Prezi" http://www.math.unipd.it/~tullio/IS-1/2014/Progetto/C4.pdf.

1.4.2 Informativi

- Piano di progetto: (((Piano di progetto v?.?.?)));
- Slide dell'insegnamento Ingegneria del Software modulo A: http://www.math.unipd.it/~tullio/IS-1/2014/ ;
- SWEBOK – Version 3 (2004): capitolo 11 – Software Quality http://www.computer.org/portal/web/swebok/html/ch11;
- Ingegneria del software - Ian Sommerville - 9a Edizione (2010):
  - Capitolo 24 - Gestione della qualità;
  - Capitolo 26 - Miglioramento dei processi.
- Standard ISO /IEC TR 15504: Software process assessment http://en.wikipedia.org/wiki/ISO/IEC_15504;

- Standard ISO /IEC 9126: Product quality
  http://en.wikipedia.org/wiki/ISO/IEC_9126;

2 Obiettivi di qualità

2.1 Qualità di processo

Al fine di garantire la qualità del prodotto in ogni fase di realizzazione, si deve garantire la qualità dei processi che lo definiscono; per questo motivo si è deciso di utilizzare lo standard ISO/IEC 15504 denominato SPICE, che rende disponibili strumenti adatti a valutarli.

2.2 Qualità di prodotto

Per garantire la qualità del prodotto si è deciso di seguire le indicazioni fornite dallo standard ISO/IEC 9126:2001 sostituito dal successivo ISO/IEC 25010:2011. Questo documento fornisce un modello per valutare la qualità esterna (nell'ambiente di utilizzo) ed interna (indipendente dall'ambiente) di un software, individuando sei caratteristiche principali atte a rendere il prodotto qualitativamente accettabile.

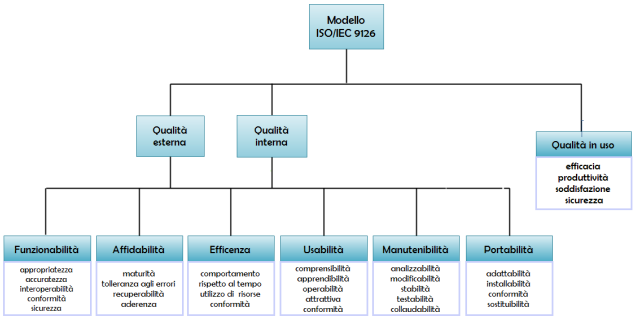

Fig 1: Rappresentazione del modello ISO/IEC 9126:2001

2.2.1 Funzionalità

È un requisito funzionale che indica la capacità del software di soddisfare le esigenze esposte dal capitolato ed individuate durante l'analisi dei requisiti. Per valutare questa caratteristica si considerano l'appropriatezza e l'accuratezza delle funzioni offerte, l'interoperabilità del prodotto rispetto ai diversi sistemi e la sicurezza offerta per la protezione dei dati.
Si sarà ottenuto un buon risultato in questo settore quando il software avrà superato in maniera positiva tutti i test e assicurerà copertura a tutti i requisiti obbligatori.

2.2.2 Affidabilità

È un requisito non funzionale che indica la capacità del software di svolgere correttamente il suo compito, mantenendo delle buone prestazioni anche al variare dell'ambiente nel tempo; vengono considerate la sua tolleranza agli errori, la capacità di evitare fallimenti nell'esecuzione a seguito di malfunzionamenti (detta maturità), e la recuperabilità dei dati e delle prestazioni nell'eventualità di un malfunzionamento inevitabile. Il prodotto può considerarsi affidabile se il numero di esecuzioni andate a buon fine è sufficientemente grande rispetto al numero di esecuzioni totali.

2.2.3 Efficienza

È un requisito non funzionale che indica il rapporto tra le prestazioni e le risorse disponibili. Si valuta se il software utilizza al meglio le risorse a sua disposizione per fornire le funzionalità richieste, considerando il suo comportamento rispetto al tempo, ossia la velocità di risposta e d'elaborazione in determinate condizioni, che rispetto all'uso delle risorse, data dalla capacità d'utilizzarne una quantità adeguata ad eseguire le funzioni richieste.
Un modo per valutare l'efficienza di un software è calcolarne i tempi di attesa in seguito all'esecuzione di un comando, tuttavia, nel caso del prodotto Premi l'efficienza è limitata anche dallo stato della rete e dall' utilizzo di componenti grafiche quali video o immagini; per questo motivo il gruppo non può garantire tempi di risposta brevi per ogni azione compiuta dall'utente, ma si impegna a non appesantire ulteriormente tali componenti.

2.2.4 Usabilità

È un requisito non funzionale che indica la capacità del software di essere compreso, appreso ed usato con soddisfazione dall'utente. Per far ciò il prodotto deve soddisfare condizioni di comprensibilità, apprendibilità ed operabilità; deve inoltre avere una certa attrattiva nei confronti dell 'utente allo scopo di rendergliene piacevole l'utilizzo. Questa caratteristica non è facilmente misurabile in quanto non esistono metriche per quantificarla, perciò si farà affidamento alle linee guida del material design fornite da Google, dato l'alto tasso di adozione rispetto ad altre linee guida.

2.2.5 Manutenibilità

È un requisito non funzionale che indica la capacità del software di essere corretto, migliorato o adattato con un impegno contenuto; a tale scopo esso deve essere facilmente analizzabile e modificabile, deve garantire stabilità a seguito di modifiche e testabilità di tali modifiche. Per misurare questa caratteristica esistono una serie di metriche descritte nella sezione —-> Metriche e quantificabili.

2.2.6 Portabilità

È un requisito non funzionale che indica la capacità del software di adattarsi al cambio di dispositivo e sistema operativo, limitando la necessità di apportare cambiamenti.
Per soddisfare questa caratteristica, come espresso dal capitolato, è necessario che il software funzioni sia su computer (indipendentemente dal loro sistema operativo) e su dispositivi mobile Android, iOS e Windows Phone.

2.3 Procedure di controllo di qualità di processo

Per applicare il modello SPICE si utilizzerà il ciclo di Deming. Il ciclo di Deming è un sistema iterativo per il miglioramento continuo della qualità dei processi e dei prodotti da essi risultanti che permette di riconoscere lo stato di avanzamento di un progetto, fornendo un metodo di lavoro logico e sistematico.

È chiamato anche ciclo PDCA, in quanto è definito dall'iterazione delle quattro fasi:

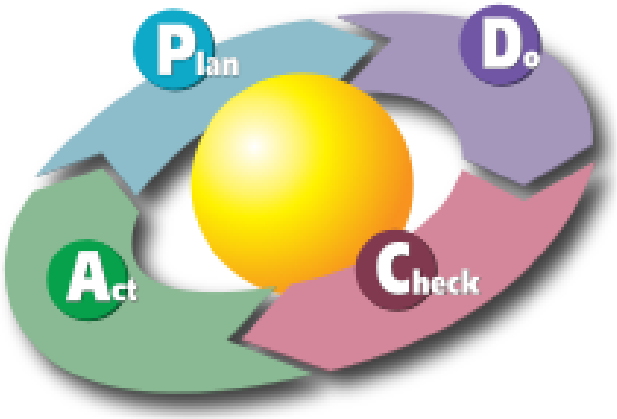

Fig 2: Schema PDCA

- **Plan**: si stabiliscono obiettivi e processi necessari ad ottenere risultati conformi agli obiettivi attesi.
- **Do**: si implementa il piano, si esegue il processo e si realizza il prodotto. Si raccolgono dati da analizzare nei passi successivi.
- **Check**: si studiano i risultati ottenuti tramite la raccolta dei dati nella fase Do e si paragonano con i risultati attesi (gli obiettivi stabiliti nella fase Plan), per verificare la presenza di incongruenze. Si evidenziano le differenze nell'implementazione rispetto al piano.
- **Act**: se la fase di Check evidenzia che gli obiettivi fissati nel Plan e implementati nel Do rappresentano un miglioramento rispetto alla baseline precedente, si stabilisce una nuova baseline; in caso contrario la baseline non cambia. In enrambi i casi se la fase di Check ha evidenziato differenze rispetto alle aspettative, sarà necessario svolgere nuovamente il ciclo di PDCA.

3 Visione generale delle strategie di verifica

3.1 Organizzazione

Ogni qualvolta avvenga un cambiamento sostanziale nello sviluppo del prodotto, si istanzierà il processo di verifica. Nello specifico durante ogni fase (Analisi, Progettazione, Realizzazione e Validazione) saranno applicate le tecniche di verifica qui descritte nei seguenti casi:

- Conclusione della prima redazione di un documento;
- Conclusione della prima redazione di un file di codice;
- Conclusione della modifica sostanziale di un documento: quando il versionamento passa da .x.y.z a .x.y+1.0 oppure a .x+1.0.0. Si veda per approfondimento il paragrafo relativo al versionamento nel documento xxxxxx;
- Conclusione della modifica sostanziale di un file di codice, quando cioè il versionamento passa da .x.y.z a .x.y +1.0 oppure a .x+1.0.0. Si veda per approfondimento il paragrafo relativo al versionamento nel documento xxxxxx.

L'obiettivo delle attività di verifica è quello di trovare e rimuovere i problemi presenti. Un problema può verificarsi a vari livelli, e per ogni livello assume un nome diverso:

- Fault (difetto): è l'origine del problema, ciò che fa scaturire il malfunzionamento;
- Error (errore): è lo stato per cui il software si trova in un punto sbagliato del flusso di esecuzione o con valori sbagliati rispetto a quanto previsto dalla specifica;
- Failure (fallimento, guasto): è un comportamento difforme dalla specifica, cioè la manifestazione dell'errore all'utente del software.

Esiste una relazione di causa-effetto fra questi tre termini:

$$DIFETTO \longrightarrow ERRORE \longrightarrow FALLIMENTO$$

Non sempre un errore dà origine ad un fallimento: ad esempio potrebbero esserci alcune variabili che si trovano in stato erroneo ma non vengono lette, o non viene percorso il ramo di codice che le contiene.
E' necessario prestare particolare attenzione a questo tipo di errori (detti anche quiescenti), avvalendosi anche di strumenti per il rilevamento dei bug.

3.2 Pianificazione strategica e temporale

Ai fini di rendere sistematica l'attività di verifica, per poter rispettare le scadenze fissate nel Piano di Progetto ed evitare la propagazione di errori all'interno dei documenti o di file di codice prima della loro verifica, la loro redazione sarà anticipata da una fase di studio preliminare. Questa fase permetterà di ridurre la necessità di intervenire con grossi interventi a posteriori, quando la correzione di imprecisioni concettuali e tecniche potrebbe risultare particolarmente gravosa. Come da Piano di Progetto di seguito si riportano le quattro milestone prefissate prima delle quali si effettuerà una verifica del prodotto:

- Revisioni formali:
  - Revisione dei Requisiti (27/04/2015)
  - Revisione di Accettazione (06/07/2015)
- Revisioni di progresso:
  - Revisione di Progettazione (29/05/2015)
  - Revisione di Qualifica (18/07/2015)

  Sarà necessario, infine, assicurarsi che ogni requisito sia tracciato consistentemente nel documento di Analisi dei Requisiti.

3.3 Responsabilità

I principali ruoli di responsabilità individuati sono:

- Amministratore di Progetto:
  - Assicura la funzionalità dell'ambiente di lavoro;
  - Redige i piani di gestione della qualità e ne verifica l'applicazione.
- Responsabile del progetto:
  - Assicura lo svolgimento delle attività di verifica;
  - Assicura il rispetto dei ruoli e delle competenze come descritti nel Piano di Progetto;
  - Approva e sancisce la distribuzione di un documento o di un file di codice;
  - Assicura il rispetto delle scadenze.

4 Risorse

4.1 Risorse Necessarie:

4.1.1 Risorse umane

I ruoli necessari a garantire la qualità del prodotto sono:

- Responsabile di Progetto;
- Amministratore;
- Verificatore;
- Programmatore.

4.1.2 Risorse Hardware

Saranno necessari:

- computer con installato software necessario allo sviluppo del progetto in tutte le sue fasi;
- luoghi in cui svolgere riunioni, preferibilmente dotato di connessione ad Internet.

4.1.3 Risorse software

Saranno necessari:

- Strumenti per automatizzare i test;
- Framework per eseguire test di unità;
- Piattaforma di versionamento per la creazione e gestione di ticket;
- Debugger per i linguaggi di programmazione scelti;
- Browser come piattaforma di testing dell'applicazione da sviluppare;
- Strumenti per effettuare l'analisi statica del codice per misurare le metriche.

4.2 Risorse disponibili

Sono disponibili:

- Computer personali dei membri del gruppo;
- Computer presenti nelle aule infomatiche del Dipartimento di Matematica;
- Aule disponibili per incontri nel Dipartimento di Matematica;
- Un dispositivo Raspberry Pi 2 Model B, utilizzato come server per programmi organizzativi e di testing.

4.2.1 Risorse software

Si rimanda alla sezione Strumenti 5.1

5 Strumenti,tecniche e metodi

5.1 Strumenti

Di seguito verranno elencati gli strumenti software che sono o saranno utilizzati dal gruppo per effettuare le operazioni di verifica e validazione:

- **Aspell**: correttore ortografico per documenti redatti in LaTeX;
- **Aptana**: scelto dal gruppo per la stesura del codice ha integrato al suo interno varie funzioni di debugging ed esecuzione del codice;
- **W3C validator**: sito che controlla la validità dei markup nei documenti scritti in HTML;
- **Jenkins**: sistema di per l'integrazione continua del codice e dei file latex;
- **jSHint**: tool di supporto per la ricerca di eventuali errori nel codice javascript;
- **Selenium IDE**: esentione di Firefox che permette di registrare test tramite browser;
- **ApacheBrench**: strumento a linea di comando utilizzato per misurare l'efficienza di un server web ed in grado di simulare situazioni di sovraccarico della rete;
- **SpeedTracer**: plugin per Google Chrome che permette di verificare l'efficienza di un'applicazione web durante la sua esecuzione.

  Verranno utilizzati anche tutti gli strumenti già integrati all'interno del browser per lo sviluppo delle pagine web.

5.2 Tecniche

- **Analisi statica**: consiste nell'analizzare il codice tramite tools e letture senza tuttavia eseguirlo. Data la natura di questo tipo di analisi, è possibile applicarla anche per il controllo di tutti i documenti testuali prodotti. Si esegue applicando i due seguenti metodi:
  - **Inspection**: l'obiettivo di questa tecnica di analisi è l'individuazione di difetti attraverso la lettura mirata del codice. Un prerequisito per questa metodologia di verifica è la definizione di una lista di controllo che elenca le possibili sezioni o passaggi maggiormente soggetti ad errori. La verifica deve essere condotta da soggetti nettamente distinti dai programmatori. La correzione degli errori individuati va eseguita in ogni fase e documentata tramite un rapporto delle attività svolte;
  - **Walkthrough**: l'obiettivo di questa tecnica di analisi è l'individuazione di difetti eseguendo una lettura integrale di tutto il codice senza l'assunzione di presupposti. Viene eseguita da gruppi misti di ispettori e sviluppatori. Per evitare incomprensioni è importante che al termine della lettura gli elementi coinvolti discutano i difetti trovati e che nessuna persona possa coprire entrambi i ruoli allo stesso tempo. Al termine della fase di discussione si applicherà la fase di correzione dei difetti che apporterà le modifiche concordate. Anche in questo caso è importante tenere un rapporto delle attività svolte.

- **Analisi dinamica**: consiste nel verificare e validare il software o un suo componente osservandone il comportamento in esecuzione durante lo svolgimento di test. Tali test devono essere svolti in maniera ripetibile: significa che se eseguiti nello stesso ambiente e con gli stessi ingressi, devono produrre i medesimi risultati.
  - **Test di unità**: esamina la correttezza di piccole unità di codice, generalmente prodotte da un singolo programmatore, in modo da verificare che esse rispettino i loro requisiti. È possibile svolgerlo con un alto grado di parallelismo possibilmente servendosi di un automa.
  - **Test di integrazione**: verifica che l'integrazione delle unità che hanno superato il test precedente non produca problemi. Tali problemi, non potendo essere relativi alle singole unità, saranno da ricercare nell'interfaccia che le aggrega.
  - **Test di sistema**: accerta la copertura dei requisiti software individuati nell'analisi dei requisiti permettendo la validazione del sistema prodotto.
  - **Test di regressione**: stabilisce se modifiche all'implementazione di un programma alterano elementi precedentemente funzionanti. Per far ciò si eseguono nuovamente i test di unità e integrazione sulle parti modificate.
  - **Test funzionali**: mettono alla prova le funzionalità del sistema, simulando l'iterazione tra Utente e sistema.
  - **Test Prestazionali**: valutano le prestazioni dell'applicazione in molti modi e da molti punti di vista. Questo tipo di test mostra ciò che proverà l'utente in termini di caricamento e velocità del sito. Le prestazioni sono importanti anche per motivi SEO, in quanto un sito lento verrà analizzato molto meno frequentemente dai web crawler dei motori di ricerca.
  - **Test di collaudo**: attività formale supervisionata dal committente il cui buon esito comporta la possibilità di rilasciare il prodotto.

5.3 Metodi

Il gruppo ha deciso di utilizzare i seguenti metodi per applicare le tecniche sopra descritte, aiutandosi con gli strumenti elencati:

- **Documenti LaTeX**:
  1. Rilettura approfondita;
  2. Controllo ortografico tramite lo strumento Aspell;
  3. Controllo dell'applicazione delle regole tipografiche esposte nel documento Norme di progetto;
  4. Verifica della corretta formattazione del file pdf prodotto.
- **Codice**: il codice verrà analizzato dagli strumenti integrati all'interno dell'IDE Aptana e dagli strumenti di sviluppo forniti dai singoli browser.
- **Schemi UML**:

1. data l'impossibilità di controllare la correttezza ortografica degli schemi, con Aspell è necessario esaminare attentamente e più volte i nomi, gli identificativi e i testi nei diagrammi;
2. controllo della correttezza degli identificativi dei casi d'uso rispetto alla nomenclatura stabilita nel documento Norme di Progetto e rispetto alle sezioni dell'Analisi dei Requisiti in cui sono inseriti;
3. controllo della numerazione dei casi d'uso rispetto la loro gerarchia;
4. controllo che i casi d'uso soddisfino tutte le esigenze espresse nel capitolato.

5.4 Metriche

Il processo di verifica, per essere informativo, deve esse quantificabile. Le misure rilevate dal processo di verifica devono quindi essere basate su metriche stabilite a priori. Una metrica è la misura di una qualche proprietà relativa ad una porzione di un documentosoftware allo scopo di fornire informazioni significative sulla qualità del codice prodotto. Non bisogna tuttavia basarsi solamente sulle metriche, che sono solo indicatori a posteriori della bontà del lavoro svolto: un'importanza ancora maggiore la riveste il controllo sulla qualità del processo.

5.4.1 Metriche per il codice

- **Complessità Ciclomatica di McCabe**: è indicazione del numero di segmenti lineari in un metodo (ad esempio sezioni di codice senza ramificazioni), può quindi essere usato per determinare il numero di test necessari per ottenere una copertura completa dei possibili cammini. Un metodo senza ramificazioni ha Complessità Ciclomatica di McCabe pari a 1; tale valore è incrementato ogni qualvolta si incontra una ramificazione. Con "ramificazione" si intendono cicli, costrutti "if" e simili;
  Secondo McCabe una complessità ciclomatica nel range 1-10 individua un codice semplice con pochi rischi , superato questo limite il codice diventa più complesso , instabile e difficilmente manutenibile.

- **Numero di istruzioni**: rappresenta il numero di istruzioni all'interno di un metodo. Un indice elevato non rappresenta necessariamento un cattivo codice ma suggerisce la possibilità di estrarre metodi contenenti gruppi di istruzioni correlate, aumentando il livello di astrazione;

- **Indice di mantenibilità**: calcola un indice dal valore compreso tra 0 e 100 che rappresenta la facilità di mantenibilità del codice. Un elevato valore indica un'ottima mantenibilità: un punteggio tra 20 e 100 indica che il codice ha mantenibilità buona; tra 10 e 19, una manteniblità moderata; un punteggio tra 0 e 9 indica bassa mantenibilità.

- **Copertura del codice**: è indicazione di quanto codice sorgente sia stato testato. Un elevato indice di copertura indica che il codice sorgente è stato testato in profondità e che difficilmente può contenere dei bug.

  Parametri utilizzati:

  - Range-accettazione: [42%-100%];
  - Range-ottimale: [65%-100%].

5.4.2 Metriche per i documenti

- **Indice Gulpease**: Misura l'indice di leggibilità di un testo tarato sulla lingua italiana. Rispetto ad altri ha il vantaggio di utilizzare la lunghezza delle parole in lettere anziché in sillabe, semplificandone il calcolo automatico. Permette di misurare la complessità dello stile di un documento.
  L'indice Gulpease considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere.
  L'indice è calcolato secondo la seguente formula:

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

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

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

  Parametri utilizzati:

  - Range-accettazione: [40-100];
  - Range-ottimale: [50-100].

6 Gestione amministrativa della revisione

6.1 Comunicazione e risoluzione di anomalie

Un'*anomalia* corrisponde a:

- Violazione delle norme tipografiche in un documento;
- Uscita dal range d'accettazione degli indici di misurazione;
- Incongruenza del prodotto con funzionalità presenti nell'analisi dei requisiti;
- Incongruenza del codice con il design del prodotto.

In caso un verificatore riscontri un'anomalia, aprirà un ticket nel sistema di ticketing con le modalità specificate nelle Norme di Progetto. Le modalità di risoluzione di quest'ultimo e la sua struttura vengono descritte in modo dettagliato all'interno del documento NormeDiprogetto-v1.0.pdf. Quando viene rilasciata una nuova versione di un documento od un modulo, il Verificatore controlla il registro delle modifiche ed in base ad esso effettua una verifica alla ricerca di anomalie da correggere. Se ne trova, apre un ticket e lo comunica all'Amministratore; s'occuperà della correzione la persona che ha apportato la modifica al documento o modulo. Le nuove modifiche dovranno essere approvate dall'Amministratore.

7 Resoconto delle attività di verifica

7.1 Riassunto delle attività di verifica

7.1.1 Revisione dei Requisiti

Durante questa fase sono stati prodotti solamente documenti di testo quindi sono state applicate le tecniche di analisi statica effettuando *walkthrough* e rispettando i metodi definiti nella sezione 5.3.
Nella la verifica dei documenti sono stati riscontrati soprattutto errori grammaticali e di battitura dovuti a disattenzioni durante la stesura.
E stato trovato anche qualche errore più grave come il non rispetto delle regole di formattazione riportate nelle *Norme di progetto* e alcune mancanze all'interno del documento di *Analisi dei requisiti*.

7.1.2 Documenti

Vengono qui riportati i valori dell'indice Gulpease per ogni documento durante la fase di **Analisi**. Un documento è considerato valido soltanto se rispetta le metriche descritte su 5.4.2.

Documento	Valore indice	Esito
Piano di Progetto v.1.0.0	?	?
Analisi dei Requisiti v.1.0.0	?	?
Norme di Progetto v.1.0.0	?	?
Piano di Qualifica v.1.0.0	?	?
Studio di Fattibilità v.1.0.0	?	?
Glossario v.1.0.0	?	?

Tab 3: Esiti verifica documenti, Analisi

Come si può notare dalla tabella, tutti gli indici Gulpease dei documenti rientrano nel range ottimale precedentemente definito e quindi i documenti redatti hanno raggiunto la leggibilità desiderata.