

2 marzo 2015



Piano di Qualifica

Informazioni sul documento

Nome Documento	Piano di Qualifica
Versione	1.0
Stato	<i>Formale</i>
Uso	<i>Interno</i>
Data Creazione	2 marzo 2015
Data Ultima Modifica	2 marzo 2015
Redazione	
Approvazione	
Verifica	
Lista distribuzione	
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Proponente Zucchetti S.p.a.

Registro delle modifiche

Versione	Autore	Data	Descrizione
1.0			Prima stesura del documento.

Tabella 1: Versionamento del documento

Storico

pre-RR

Versione 1.0	Nominativo
Redazione	
Verifica	
Approvazione	

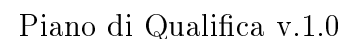
Tabella 2: Storico ruoli pre-RR

Indice

1	Introduzione	5
1.1	Scopo del documento	5
1.2	Scopo del Prodotto	5
1.3	Glossario	5
1.4	Riferimenti	5
1.4.1	Normativi	5
1.4.2	Informativi	5
2	Obiettivi di qualità	7
2.1	Qualità di processo	7
2.2	Qualità di prodotto	7
2.2.1	Funzionalità	7
2.2.2	Affidabilità	7
2.2.3	Efficienza	8
2.2.4	Usabilità	8
2.2.5	Manutenibilità	8
2.2.6	Portabilità	8
2.3	Procedure di controllo di qualità di processo	8
3	Visione generale delle strategie di verifica	10
3.1	Organizzazione	10
3.2	Pianificazione strategica e temporale	10
3.3	Responsabilità	10
4	Risorse	11
4.1	Risorse Necessarie:	11
4.1.1	Risorse umane	11
4.1.2	Risorse Hardware	11
4.1.3	Risorse software	11
4.2	Risorse disponibili	12
4.2.1	Risorse software	12
4.3	Strumenti,tecniche e metodi	13
4.3.1	Strumenti	13
4.3.2	Tecniche	13
4.3.3	Metodi	14
4.3.4	Metriche	15
5	Gestione amministrativa della revisione	16
5.1	Comunicazione e risoluzione di anomalie	16

Sommario

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



1.1 Scopo del documento

1.2 Scopo del Prodotto

1.3 Glossario

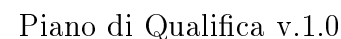
1.4 Riferimenti

1.4.1 Normativi

- ### 1.4.2 Informativi

- Università degli studi di Padova - 2014/2015

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



2.1 Qualità di processo

2.2 Qualità di prodotto

```

graph TD
    Root[Modello ISO/IEC 9126] --> QEsterna[Qualità esterna]
    Root --> QInterna[Qualità interna]
    QInterna --> QInUse[Qualità in uso]
    QInterna --> Funz[Funzionalità]
    QInterna --> Affid[Affidabilità]
    QInterna --> Effici[Efficienza]
    QInterna --> Usab[Usabilità]
    QInterna --> Manut[Manutenibilità]
    QInterna --> Portab[Portabilità]
    QInUse --- QInUseList["efficacia  
produttività  
soddisfazione  
sicurezza"]
    Funz --- FunzList["appropriatozza  
accuratezza  
interoperabilità  
conformità  
diversità"]
    Affid --- AffidList["mutualità  
tolleranza agli errori  
rispondibilità  
coerenza"]
    Effici --- EfficiList["comportamento  
rispetto al tempo  
utilizzo di risorse  
conformità"]
    Usab --- UsabList["comprensibilità  
apprendibilità  
operabilità  
attrattivo  
consuetudine"]
    Manut --- ManutList["analizzabilità  
modificabilità  
stabilità  
testabilità"]
    Portab --- PortabList["adattabilità  
installabilità  
conformità  
sostenibilità"]
  
```

2.2.1 Funzionalità

2.2.2 Affidabilità

Università degli studi di Padova - 2014/2015



2.2.3 Efficienza

E' un requisito non funzionale che indica il rapporto tra le prestazioni e le risorse disponibili, si valuta cioè 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 di elaborazione in determinate condizioni, e rispetto all'uso delle risorse, ossia la sua capacità di utilizzare la quantità adeguata di risorse per 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 impegnerà a non appesantire ulteriormente tali componenti.

2.2.4 Usabilità

E' 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 dalla Google dato l'alto tasso di penetrazione che hanno avuto nel mercato rispetto ad altre linee guida.

2.2.5 Manutenibilità

E' 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

E' 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.

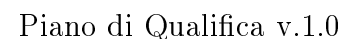
2.3 Procedure di controllo di qualità di processo

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.

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



- **Plan:** si stabiliscono gli obiettivi e i processi necessari a 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 divergenze di implementazione rispetto al piano;
- **Act:** se la fase di Check evidenzia che gli obiettivi fissati nel Plan e implementati sul Do rappresentano un miglioramento rispetto alla baseline precedente, si stabilisce una nuova baseline. In caso contrario la baseline resta la precedente. In entrambi i casi se la fase di Check ha evidenziato differenze rispetto alle aspettative, sarà necessario svolgere nuovamente il ciclo di PDCA.



3.1 Organizzazione

Questo documento è distribuito sotto licenza [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

- 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

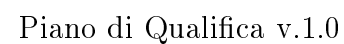
Saranno necessari:

- 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.





4.3 Strumenti, tecniche e metodi

4.3.1 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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$;
- **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** : estensione di Firefox che permette di registrare test tramite browser;
- **ApacheBench** : 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.

4.3.2 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. 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 ossia devono venir eseguiti nello stesso ambiente e con gli stessi ingressi in modo da potersi aspettare 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. E' 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:** Valuta 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 epr motivi SEO , in quanto un sito lento verrà analizzato molto meno frequentemente dai search engine crawlers dei motori di ricerca ;
 - **Test di collaudo:** attività formale supervisionata dal committente il cui buon esito comporta la possibilità di rilasciare il prodotto.

4.3.3 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:**



- #### 4.3.4 Metriche

- **Complessità ciclomatica di McCabe:** è un'indicazione del numero di segmenti lineari in un metodo (ad esempio sezioni di codice senza ramificazioni) e quindi può essere usato per determinare il numero di test necessari per ottenere una copertura completa dei possibili cammini. Un metodo senza ramificazioni ha Complessità Ciclomantica pari a 1. Tale valore è incrementato ogni qualvolta si incontra una ramificazione. Con “ramificazione” si intendono i cicli e i costrutti “if” e simili.
- **Numero di istruzioni:** numero di istruzioni all'interno di un metodo. Un indice elevato non rappresenta necessariamente un cattivo codice ma suggerisce la possibilità di estrarre metodi contenenti gruppi di istruzioni correlate, aumentando il livello di astrazione



5 Gestione amministrativa della revisione

Un'*anomalia* corrisponde a:

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 adesso 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.