24-05-2015

Piano di Qualifica

Informazioni sul documento

Nome Documento	Piano di Qualifica
Versione	1.1.0
Stato	*Formale*
Uso	*Esterno*
Data Creazione	20-03-2015
Data Ultima Modifica	24-05-2015
Redazione	Fossa Manuel, Petrucci Mauro
Approvazione	Tollot Pietro
Verifica	Gabelli Pietro
Lista distribuzione	*LateButSafe*

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

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Il presente documento contiene le norme e le convenzioni che il gruppo LateButSafe intende 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$_g$ di verifica continua sulle attività svolte; questo consentirà di rilevare e correggere anomalie e incongruenze in modo tempestivo e senza spreco di Risorse$_g$.

1.2 Scopo del Prodotto

Lo scopo del Progetto$_g$ è la realizzazione un Software$_g$ 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 e di dominio, gli acronimi e le parole che necessitano di essere chiarite sono riportate nel documento Glossario_v.2.0.0.pdf. Ogni occorrenza di vocaboli presenti nel Glossario è marcata da una "g" minuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- Norme di Progetto$_g$: NormeDiProgetto_v.2.0.0.pdf;
- Capitolato d'appalto C4: Premi: Software$_g$ di presentazione "better than Prezi" http://www.math.unipd.it/~tullio/IS-1/2014/Progetto/C4.pdf.

1.4.2 Informativi

- Piano di Progetto$_g$: PianoDiProgetto_v.2.0.0.pdf;
- Slide dell'insegnamento Ingegneria del Software$_g$ modulo A: http://www.math.unipd.it/~tullio/IS-1/2014/ ;
- SWEBOK – Version 3 (2004): capitolo 11 – Software$_g$ Quality http://www.computer.org/portal/web/swebok/html/ch11;
- Ingegneria del Software$_g$ - Ian Sommerville - 9a Edizione (2010):
  - Capitolo 24 - Gestione della qualità;
  - Capitolo 26 - Miglioramento dei Processi$_g$.
- Standard ISO$_g$ /IEC TR 15504: Software$_g$ process assessment http://en.wikipedia.org/wiki/ISO/IEC_15504;

- Standard ISO_g /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$_g$ che lo definiscono; per questo motivo si è deciso di utilizzare lo standard ISO/IEC 15504 denominato SPICE$_g$, che rende disponibili strumenti adatti a valutarli.

Per applicare correttamente questo modello si deve utilizzare il ciclo di Deming (ciclo PDCA) il quale definisce una metodologia di controllo dei processi durante il loro ciclo di vita che consente di migliorarne in modo continuativo la qualità.

Tutti i Processi$_g$ dovranno quindi essere sottoposti a valutazione in modo da verificarne la qualità ed eventualmente facilitarne il miglioramento. A tale scopo lo SPICE$_g$ definisce nove attributi di Processo$_g$ per effettuare una migliore valutazione:

1. **Process performance**
   Gli indicatori della performance di Processo$_g$ sono:
   - I lavori identificati come input al Processo$_g$ (input work products);
   - I lavori identificati come output del Processo$_g$ (output work products);
   - Le azioni compiute per trasformare gli input work products in output work products.
2. **Performance Management**
   L'attuazione di un Processo$_g$ è pianificata e controllata al fine di generare risultati che rispondono agli obiettivi attesi;
3. **Work Product Management**
   L'attuazione di un Processo$_g$ è pianificata e controllata al fine di generare risultati che siano adeguatamente documentati, controllati e verificati;
4. **Process Definition**
   L'attuazione di un Processo$_g$ si basa su approcci standardizzati;
5. **Process Resource**
   Il Processo$_g$ può contare su adeguate risorse umane$_g$, di infrastrutture, ecc. per essere attuato;
6. **Process Measurement**
   I risultati conseguiti e le misure rilevate durante l'attuazione di un Processo$_g$ sono utilizzati per assicurarsi che l'attuazione di tale Processo$_g$ supporti efficacemente il raggiungimento di obiettivi specifici;
7. **Process Control**
   Un Processo$_g$ è controllato tramite la raccolta, analisi ed utilizzo delle misure di prodotto e di Processo$_g$ rilevate, con l'obbiettivo di correggere, se necessario, le sue modalità di attuazione;
8. **Process Change**
   Le modifiche alla definizione, gestione e attuazione di un Processo$_g$ sono controllate;

9. **Continuous Integration**
   Le modifiche ad un Processo$_g$ sono identificate ed implementate con lo scopo di assicurare il continuo miglioramento nel raggiungere gli obbiettivi definiti per l'organizzazione.

Sono inoltre stabiliti quattro differenti livelli di possesso di ciascuno degli attributi:

- **N - Non posseduto** (0 - 15% di possesso): non c'è evidenza oppure ce n'è poca del possesso di un attributo;
- **P - Parzialmente posseduto** (16 - 50% di possesso): c'è evidenza di approccio sistematico al raggiungimento del possesso di un attributo e del raggiungimento di tale possesso, ma alcuni aspetti del possesso possono essere non prevedibili;
- **L - Largamente posseduto** (51 - 85% di possesso): vi è evidenza di approccio sistematico al raggiungimento del possesso di un attributo e di un significativo livello di possesso di tale attributo, ma l'attuazione del Processo$_g$ può variare nelle diverse unità operative dell'organizzazione;
- **F - (Fully) Pienamente posseduto** (86 - 100% di possesso): vi è evidenza di un totale e sistematico approccio e di un completo raggiungimento del possesso dell'attributo; non esistono significative differenze nel modo di attuare il Processo$_g$ tra le diverse unità operative.

Vi sono poi vari livelli di maturità dei Processi$_g$ che dipendono dal diverso livello di possesso degli attributi:

- **Livello 0** - Processo$_g$ incompleto: il Processo$_g$ non è implementato o non raggiunge gli obiettivi. Non vi è evidenza di approcci sistematici agli attributi definiti;
- **Livello 1** - Processo$_g$ semplicemente attuato: il Processo$_g$ viene messo in atto e raggiunge i suoi obiettivi. Non vi è evidenza di approcci sistematici agli attributi definiti. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process performance";
- **Livello 2** - Processo$_g$ gestito: il Processo$_g$ è attuato, ma anche pianificato, tracciato, verificato ed aggiustato se necessario, sulla base di obiettivi ben definiti. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Performance management" e "Work product management";
- **Livello 3** - Processo$_g$ definito: il Processo$_g$ è attuato, pianificato e controllato sulla base di procedure ben definite, basate sui principi del Software$_g$ engineering. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process definition" e "Process resource" ;
- **Livello 4** - Processo$_g$ predicibile: il Processo$_g$ è stabilizzato ed è attuato all'interno di definiti limiti riguardo i risultati attesi, le performance, le Risorse$_g$ impiegate ecc. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process measurement" e "Process control";

- **Livello 5** - Processo$_g$ ottimizzante: il Processo$_g$ è predicibile ed in grado di adattarsi per raggiungere obiettivi specifici e rilevanti per l'organizzazione. Il raggiungimento di questo livello è dimostrato attraverso il possesso degli attributi di "Process change" e "Continuous integration".

L'applicazione dello standard ISO/IEC 15504 porta a benefici sia agli sviluppatori del Software$_g$ che ai suoi utilizzatori o acquirenti. Per gli sviluppatori porta vantaggi nell'ottimizzazione dell'uso delle Risorse$_g$, un contenimento dei costi, una maggiore tempestività di consegna del prodotto ultimato, migliore stima dei rischi e degli impegni e la possibilità di confrontarsi con delle best practice. Per gli utenti invece abbiamo una maggior facilità nella selezione dei fornitori, una migliore valutazione dei rischi di Progetto$_g$, controllo dello stato di avanzamento in corso d'opera, riduzione dei costi di correzione degli errori ed un controllo dei rischi e delle varianti in corso d'opera.

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$_g$, individuando sei caratteristiche principali atte a rendere il prodotto qualitativamente accettabile.

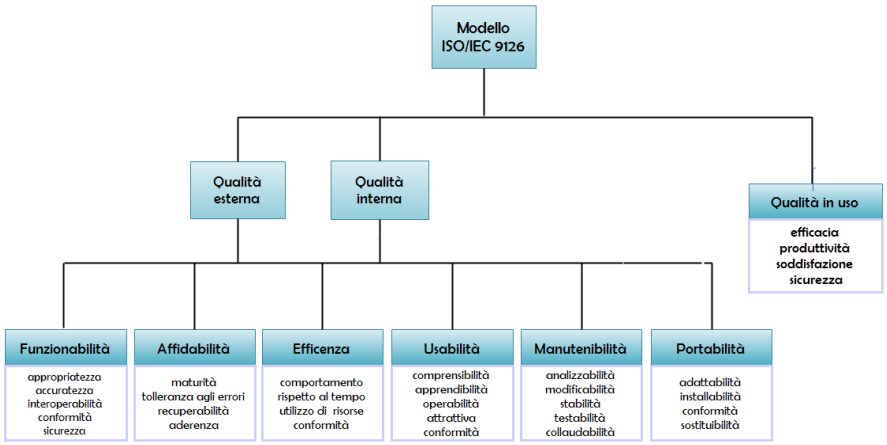

Fig 1: Rappresentazione del modello ISO/IEC 9126:2001

2.2.1 Funzionalità

È un Requisito$_g$ funzionale che indica la capacità del Software$_g$ di soddisfare le esigenze esposte dal capitolato ed individuate durante l'analisi dei Requisiti$_g$. Per valutare questa caratteristica si considerano l'appropriatezza e l'accuratezza delle funzioni$_g$ offerte, l'interoperabilità del prodotto rispetto ai diversi sistemi e la sicurezza offerta per la protezione dei dati.
Quantificazione del raggiungimento dell'obbiettivo di qualità: la misurazione del raggiungimento di questo obiettivo si calcolerà verificando la quantità di requisiti soddisfatti che

avranno un riscontro in elementi funzionanti nell'applicazione prodotta. La soglia di sufficienza sarà quindi data dal soddisfacimento di tutti i requisiti obbligatori previsti dal capitolato d'appalto.

2.2.2 Affidabilità

È un Requisito$_g$ non funzionale che indica la capacità del Software$_g$ 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.
Quantificazione del raggiungimento dell'obbiettivo di qualità: la misurazione del raggiungimento di questo obiettivo si calcolerà confrontando il numero di esecuzioni totale con quelle andate a buon fine e che hanno mantenuto un livello di prestazioni tali da poter permettere l'utilizzo previsto del prodotto

2.2.3 Efficienza

È un Requisito$_g$ non funzionale che indica il rapporto tra le prestazioni e le Risorse$_g$ disponibili. Si valuta se il Software$_g$ utilizza al meglio le Risorse$_g$ 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$_g$, data dalla capacità d'utilizzarne una quantità adeguata ad eseguire le funzioni$_g$ richieste.
Quantificazione del raggiungimento dell'obbiettivo di qualità: Un modo per valutare l'efficienza di un Software$_g$ 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$_g$ non funzionale che indica la capacità del Software$_g$ 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.
Quantificazione del raggiungimento dell'obbiettivo di qualità: Questa caratteristica non è facilmente misurabile in quanto non esistono metriche$_g$ per quantificarla, perciò si farà affidamento alle linee guida del material design$_g$ fornite da Google, dato l'alto tasso di adozione rispetto ad altre linee guida.

2.2.5 Manutenibilità

È un Requisito$_g$ non funzionale che indica la capacità del Software$_g$ di essere corretto, migliorato o adattato con impegno contenuto; a tale scopo esso deve essere facilmente analizzabile e modificabile, deve garantire stabilità a seguito di modifiche e la testabilità di tali modifiche.

Quantificazione del raggiungimento dell'obbiettivo di qualità: Per la misurazione di questo requisito si fa riferimento alle metriche$_g$ descritte nella sezione 3.6.

2.2.6 Portabilità

È un Requisito$_g$ non funzionale che indica la capacità del Software$_g$ di adattarsi al cambio di dispositivo e sistema operativo, limitando la necessità di apportare cambiamenti.
Quantificazione del raggiungimento dell'obbiettivo di qualità: Per soddisfare questa caratteristica, come espresso dal capitolato, è necessario che il Software$_g$ funzioni$_g$ sia su computer (indipendentemente dal loro sistema operativo) e su dispositivi mobile$_g$ Android$_g$, iOS e Windows$_g$ Phone.

2.3 Procedure di controllo di qualità di processo

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

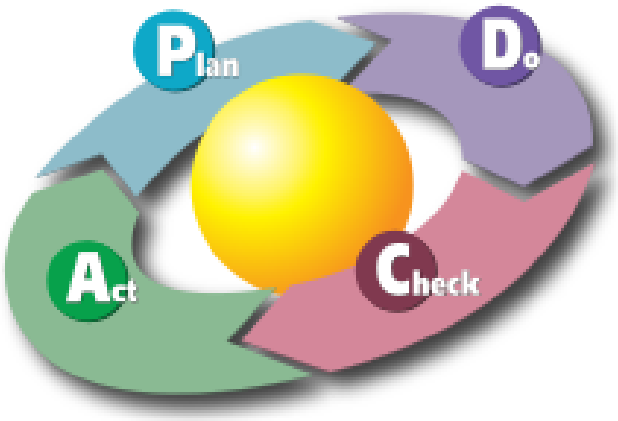

Fig 2: Schema PDCA

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

- **Plan**: si stabiliscono obiettivi e Processi$_g$ necessari ad ottenere risultati conformi agli obiettivi attesi;
- **Do**: si implementa il piano, si esegue il Processo$_g$ 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 entrambi i casi se la fase di Check ha evidenziato differenze rispetto alle aspettative, sarà necessario svolgere nuovamente il ciclo di PDCA.

Una descrizione di come il gruppo applicherà il PDCA è riportata nelle *Norme di Progetto v.2.0.0*(§5.1).

3 Visione generale delle strategie di verifica

3.1 Organizzazione

Ogniqualvolta avvenga un cambiamento sostanziale nello sviluppo del prodotto, si istanzierà il Processo$_g$ di verifica.
Nello specifico durante ogni fase (Analisi, Progettazione, Realizzazione e Validazione$_g$) 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$_g$ di Codice$_g$;
- 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 NormeDiProgetto_v.2.0.0.pdf;
- Conclusione della modifica sostanziale di un File$_g$ di Codice$_g$, 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 NormeDiProgetto_v.2.0.0.pdf.

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$_g$ 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$_g$.

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$_g$ il ramo di Codice$_g$ che le contiene.
È 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

Al fine di rendere sistematica l'attività di verifica, per poter rispettare le scadenze fissate nel Piano di Progetto$_g$ ed evitare la propagazione di errori all'interno dei documenti o di File$_g$ di Codice$_g$ prima della loro verifica, la loro redazione sarà anticipata da una fase di studio preliminare.
Questa fase permetterà di ridurre la necessità di grossi interventi nelle fasi successive, quando

la correzione di imprecisioni concettuali e tecniche potrebbe risultare particolarmente gravosa. Come da Piano di Progetto$_g$ di seguito si riportano le quattro Milestone$_g$ prima delle quali si effettuerà una verifica del prodotto:

- Revisioni formali:
  - Revisione dei Requisiti$_g$ (28/04/2015)
  - Revisione di Accettazione (06/07/2015)
- Revisioni di progresso:
  - Revisione di Progettazione (29/05/2015)
  - Revisione di Qualifica (18/06/2015)

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

3.3 Responsabilità

I principali ruoli di responsabilità individuati sono:

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

3.4 Risorse

Per assicurare che gli obiettivi qualitativi vengano raggiunti è necessario l'utilizzo di Risorse$_g$ sia umane che tecnologiche. Per una dettagliata descrizione dei ruoli e delle loro responsabilità fare riferimento alle Norme di Progetto$_g$. Per Risorse$_g$ tecniche e tecnologiche sono da intendersi tutti gli strumenti Software$_g$ e hardware che il gruppo intende utilizzare per attuare le attività di verifica su Processi$_g$ e prodotti. Affinché il lavoro dei Verificatori venga agevolato si sono predisposti numerosi strumenti automatici che eseguono controlli sistematici sui prodotti generati. Tali strumenti sono descritti in modo accurato nelle Norme di Progetto$_g$.

3.5 Tecniche di analisi

- **Analisi statica**: consiste nell'analizzare il Codice$_g$ 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:
  - **Walkthrough**: Si svolge effettuando una lettura critica a largo spettro. È una tecnica che viene utilizzata soprattutto nelle prime attività del Progetto$_g$, quando ancora non è presente una adeguata esperienza da parte dei membri del gruppo, che permetta di attuare una verifica più mirata e precisa. Con l'utilizzo di questa tecnica, il *Verificatore* sarà in grado di stilare una lista di controllo con gli errori più frequenti in modo da favorire il miglioramento di tale attività nelle fasi future. Questa è un'attività onerosa e collaborativa che richiede l'intervento di più persone per essere efficace ed efficiente. Dopo una prima fase di lettura e individuazione degli errori, segue una fase di discussione con la finalità di esaminare i difetti riscontrati e di proporre le dovute correzioni. L'ultima fase consiste nel correggere gli errori rilevati e nello scrivere un rapporto che elenchi le modifiche effettuate.
  - **Inspection**:Questa tecnica consiste nell'analisi mirata di alcune parti del documento o del Codice$_g$ che sono ritenute fonti maggiori di errore. La *lista di controllo*, che deve essere seguita per svolgere efficacemente questo Processo$_g$, deve essere redatta anticipatamente ed è frutto dell'esperienza maturata dai verificatori attraverso la tecnica di Walkthrough. L'Inspection è una strategia più rapida del Walkthrough in quanto consente l'analisi di alcune parti dei prodotti ritenute critiche dalla checklist e non necessità della lettura integrale dei documenti in oggetto. Diversamente dal Walkthrough, tale tecnica viene svolta esclusivamente dai verificatori che dopo aver individuato gli errori procedono alla loro correzione e alla redazione di un rapporto di verifica che tenga traccia del lavoro svolto. Durante l'applicazione del Walkthrough ai documenti, sono state riportate le tipologie di errori più frequenti.
    La *lista di controllo* risultante è in appendice delle *Norme di Progetto v.2.0.0.*
- **Analisi dinamica**: consiste nel verificare e validare il Software$_g$ 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$_g$, generalmente prodotte da un singolo programmatore, in modo da verificare che rispettino i Requisiti$_g$. Può essere svolto con un alto grado di parallelismo 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$_g$ Software$_g$ individuati nell'analisi dei Requisiti$_g$ permettendo la Validazione$_g$ del sistema prodotto;
  - **Test di regressione**: stabilisce se modifiche all'implementazione di un Programma$_g$ alterano elementi$_g$ 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$_g$ tra utente e sistema;
- **Test di collaudo**: attività formale supervisionata dal Committente$_g$ il cui buon esito comporta la possibilità di rilasciare il prodotto;

3.6 Metriche

Il Processo$_g$ di verifica, per essere informativo, deve essere quantificabile. Le misure rilevate dal Processo$_g$ di verifica devono quindi essere basate su metriche$_g$ stabilite a priori.
Una Metrica$_g$ è la misura di una proprietà relativa ad una porzione di un documento Software$_g$, allo scopo di fornire informazioni significative sulla qualità del Codice$_g$ prodotto.
Per valutare la bontà del lavoro svolto non è sufficiente basarsi solo sulle metriche$_g$, che sono solamente degli indicatori valutati a posteriori, perché un'importanza ancora maggiore la riveste il controllo sulla qualità del Processo$_g$.

3.6.1 metriche$_g$ per la progettazione

- **Numero di classi, coesione tra di esse e peso**: il peso di una classe è identificato dalla somma della complessità ciclomatica di tutti i metodi appartenenti alla classe;
- **Complessità di flusso**: misura la quantità di informazioni in entrata ed uscita da una Funzione$_g$ (fan-in e fan-out).

  - **Fan-in**: numero di moduli che passano informazioni dentro al modulo in esame;
  - **Fan-out**: numero di moduli a cui il modulo in esame passa informazioni.

  Il valore è calcolato come:
  $(lunghezzafunzione)^2 \times fan-in \times fan-out$.
  Un Fan-in elevato indica un buon design strutturale: significa che il modulo in esame viene usato frequentemente, con conseguente riuso del modulo e riduzione della ridondanza del Codice$_g$.
  Un Fan-out elevato indica un elevato accoppiamento tra moduli: significa che un modulo ha molte dipendenze da altri moduli e ciò mostra un povero design strutturale; implica anche un aumento dei costi di manutenibilità: ogni cambiamento di un modulo scatena la manutenzione dei moduli dipendenti.

3.6.2 metriche$_g$ per il codice

- **Complessità Ciclomatica di McCabe**: è indicazione del numero di segmenti lineari in un metodo (ad esempio sezioni di Codice$_g$ 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 ogniqualvolta 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$_g$ semplice con pochi rischi, superato questo limite il Codice$_g$ diventa più complesso, instabile e difficilmente manutenibile;

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

- **Halstead**: la Metrica$_g$ di Halstead$_g$ non è solamente un indice di complessità, ma identifica le proprietà misurabili del Software$_g$ e le relative relazioni. Si basa sull'osservazione che una Metrica$_g$ dovrebbe valutare l'implementazione di un algoritmo in linguaggi differenti ed essere indipendente dal esecuzione su una specifica piattaforma.

  Calcolo:
  Prima di tutto bisogna ricavare, dal Codice$_g$ sorgente, i seguenti valori:

  - n1 = numero distinti operatori;
  - n2 = numero distinti operandi;
  - N1 = numero totale operatori;
  - N2 = numero totale operandi.

  Successivamente possono essere calcolati i seguenti valori:

  - **Program length:**
    $$N = N1 + N2$$
  - **Program vocabulary:**
    $$n = n1 + n2$$
  - **Volume:** il volume descrive la dimensione dell'implementazione di un algoritmo e si basa sul numero di operazioni eseguite e sugli operandi di una Funzione$_g$. Il volume di una function senza parametri composta da una sola linea è 20, mentre un indice superiore a 1000 indica che probabilmente la Funzione$_g$ esegue troppe operazioni.
    $$V = N \times \log_2(n)$$

    Parametri utilizzati

    * Range-accettazione: [20-1500];
    * Range-ottimale: [20-1000];

- **Indice di manutenibilità**: Questa Metrica$_g$ è una scala logaritmica con valore massimo 171. Rappresenta quanto manutenibile è il Codice$_g$, ossia quanto facile è da supportare e migliorare.
  L'indice di manutenibilità è calcolato tramite una fattorizzazione di altre metriche$_g$ come Linee di Codice(LOC), Complessità Ciclomatica(CC), volume di Halstead(VH) e percentuale di commenti(COM).

Un elevato valore indica un'ottima manutenibilità, bassi valori al contrario indicheranno una difficoltà nella fasi di manutenzione e incremento del Codice$_g$:

$$M = 171 - 5.2\ln(HV) - 0.23(CC) - 16.2\ln(LOC) + 50.0\sin(\sqrt{2.46 * COM})$$

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

  Parametri utilizzati:

  - Range-sufficiente: [60%-80%];
  - Range-ottimale: [80%-100%].

3.6.3 metriche$_g$ per i documenti

- **Indice Gulpease**: misura l'indice di leggibilità di un testo; è tarato sulla lingua italiana. Rispetto ad altri indici 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].

A Riassunto delle attività di verifica

A.1 Revisione dei Requisiti

Durante questa fase sono stati prodotti solamente documenti di testo quindi sono state applicate le tecniche di analisi statica$_g$ descritte nella sezione §3.5.
Nella verifica dei documenti sono stati riscontrati soprattutto errori grammaticali e di battitura dovuti a disattenzioni durante la stesura.
È stato trovato anche qualche errore più grave, come il mancato rispetto delle regole di formattazione riportate nelle *Norme di Progetto v.2.0.0*e alcune mancanze all'interno del documento di *Analisi dei Requisiti v.2.0.0*.

A.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$_g$ descritte su §3.6.3.

Documento	Valore indice	Esito
Piano di Progetto v.2.0.0	89	Superato
Analisi dei Requisiti v.2.0.0	91	Superato
Norme di Progetto v.2.0.0	75	Superato
Piano di Qualifica v.2.0.0	82	Superato
Studio di Fattibilità v.2.0.0	82	Superato
Specifica Tecnica v.1.0.0	69	Superato
Glossario v.2.0.0	97	Superato

Tab 1: 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.

A.3 Progettazione

Viene qui riportata una tabella riassuntiva che riporta il calcolo dei parametri di accoppiamento afferente ed efferente per i componenti individuati nella progettazione.

Componente	Afferente	Efferente
Premi::Model	4	5
Premi::Model::SlideShow	1	3
Premi::Model::SlideShow::SlideShowActions::InsertEditRemove	21	6
Premi::Model::SlideShow::SlideShowElements	3	7
Premi::Model::SlideShow::SlideShowActions::Command	2	21
Premi::Model::ServerRelations	12	3
Premi::Model::ServerRelations::Loader	3	1
Premi::Model::ServerRelations::AccessControl	4	1
Premi::Model::ServerRelations::DBConsistency	0	2
Premi::Model::ServerRelations::Presenter	5	4

Tab 2: Tabella accoppiamento afferente ed efferente delle componenti

Come si può vedere dalla tabella, l'accoppiamento efferente è generalmente molto basso e quindi positivo, ad eccezione del package Command, per il quale però questo è accettato a causa della natura intrinseca di tale componente. L'accoppiamento afferente mostra invece la stabilità richiesta dalle classi del Model.