MaaP: MongoDB as an admin Platform

Piano di Qualifica

Versione	1.2.0
Data creazione	2013-11-28
Data ultima modifica	2013-12-16
Stato del Documento	Formale
Uso del Documento	Esterno
Redazione	Mattia Sorgato, Andrea Perin, Alberto Garbui, Giacomo Pinato
Verifica	Fabio Miotto, Alessandro Benetti
Approvazione	Giacomo Pinato
Distribuzione	Aperture Software Prof. Tullio Vardanega Prof. Riccardo Cardin CoffeeStrap

Sommario

Questo documento ha lo scopo di presentare le strategie adottate dal gruppo Aperture Software nell'ottica del miglioramento continuo e assicurazione della qualità.

Diario delle modifiche

Versione	Data	Autore	Modifiche effettuate
1.2.0	2013-12-16	Giacomo Pinato (RE)	Approvazione documento
1.1.1	2013-12-16	Alessandro Benetti (VE)	Verifica documento
1.1.0	2013-12-15	Fabio Miotto (VE)	Verifica documento
1.0.6	2013-12-13	Giacomo Pinato (RE)	Aggiunto resoconto attività di verifica e strandard di qualità
1.0.4	2013-12-04	Mattia Sorgato (AM)	Aggiunta metriche
1.0.3	2013-12-03	Alberto Garbui (AN)	Aggiunta analisi
1.0.2	2013-12-01	Andrea Perin (RE)	Aggiunta strategie di verifica
1.0.1	2013-11-28	Andrea Perin (RE)	Creazione documento

Tabella 1: Registro delle modifiche

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il piano di qualifica ha l'obiettivo di definire le strategie adottate dal gruppo Aperture Software per garantire la qualità del prodotto che verrà sviluppato. Il presente documento descriverà la qualità desiderate che il software dovrà avere, le metriche utilizzate per rendere il prodotto e i processi quantificabili. Verrà inoltre definito cosa significa software di alta qualità, in riferimento a questo specifico dominio, così da avere un ideale di riferimento per gli obiettivi del team.

1.2 Scopo del prodotto

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

1.3 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_v1.2.0.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.

1.4 Riferimenti

1.4.1 Normativi

- **Norme di Progetto**: *Norme_di_progetto_v1.2.0* ;
- **Capitolato d'appalto C1**: MaaP as an admin Platform
  http://www.math.unipd.it/~tullio/IS-1/2013/Progetto/C1.pdf.

1.4.2 Informativi

- **Piano di Progetto**: *Piano_di_progetto_v1.2.0*;
- **Glossario**: *Glossario_v1.2.0*;
- **Slides del corso di Ingegneria del software Modulo A, AA 2013/2014 del prof. Tullio Vardanega**:
  http://www.math.unipd.it/~tullio/IS-1/2013/;
- **Wikipedia**: http://it.wikipedia.org;
- Ian Sommerville, Software Engineering, 9 edizione (2011);
- **Standard ISO/IEC TR 15504 Software process assessment**:
  http://en.wikipedia.org/wiki/ISO/IEC_15504;

- **Standard ISO/IEC 9126:2001 Software engineering-product quality**:
  http://en.wikipedia.org/wiki/ISO_9126;
- **Budget Variance e Schedule Variance - Dati empirici**:
  http://office.microsoft.com/en-us/project-help/determine-the-right-threshold-for-project-cost
  aspx.

2 Strategia di verifica

2.1 Qualità di processo

Per garantire la qualità del prodotto finale è necessario migliorare la metodologia che porta alla qualità dei processi che compongono il prodotto. Per fare questo si è deciso di utilizzare lo standard ISO/IEC 15504[1] denominato SPICE[2]. Per applicare il modello appena citato si deve utilizzare il ciclo di Deming[3] che ha come obiettivo il miglioramento continuo dei processi nel loro ciclo di vita.

2.2 Qualità di prodotto

Per cercare di realizzare e progettare un prodotto software, in accordo con specifiche e standard definiti, ed essere privo di non conformità o difetti, è necessario usare lo standard ISO/IEC 9126[4], il quale redige e descrive obiettivi qualitativi e fornisce delle linee guida l'utilizzo di metriche al fine di tracciare il progresso nel miglioramento continuo di processo e prodotto.

2.3 Procedure di controllo di qualità di processo

Per garantire la qualità dei processi si utilizza il ciclo PDCA[5]. Questo principio permette un continuo miglioramento della qualità di tutti i processi coinvolti nella realizzazione del prodotto finale. Per controllare la qualità bisogna che i processi siano pianificati dettagliatamente, che le risorse siano individuate e ripartite in maniera quantificabile e ci deve essere un controllo sui processi. Per rendere quantificabile la qualità dei processi si utilizzano le metriche descritte nella sezione 4 di questo documento.

2.4 Procedure di controllo di qualità di prodotto

Per garantire il controllo di qualità si utilizza:

- **Quality Assurance**: tradotta in assicurazione di qualità, è l'insieme di processi che hanno come fine il miglioramento e il perseguimento della qualità. L'intenzione di un team di lavoro consiste nell'ottenere quella che si dice correction by construction, ovvero correttezza per costruzione;
- **Verifica**: è la valutazione che un prodotto, servizio o sistema è conforme a regole, requisiti, specifiche o condizioni imposte. E' spesso un processo interno e differisce dalla validazione. In analogia, l'attività di Verifica deve rispondere alla domanda: did i built the system right?, ovvero ho costruito il sistema in modo corretto?;
- **Validazione**: è l'assicurazione che un prodotto, servizio o sistema incontra nelle necessità che i clienti o gli stakeholders identificano. Spesso comporta l'accettazione e l'idoneità con clienti esterni. In questo caso la domanda è: did i built the right system?, tradotto in ho costruito il sistema giusto?.

[1] vedi Appendice, sezione A.1
[2] Software Process Improvement and Capability Determination
[3] Vedi appendice, sezione A.3
[4] vedi appendice, sezione A.2
[5] Alias, Ciclo di Deming, vedi appendice, sezione A.3

2.5 Organizzazione

Per ogni processo attuato si ci sono delle attività di verifica e per ogni processo realizzato viene verificata la qualità del processo stesso e la qualità dell'eventuale prodotto ottenuto da esso. Ogni fase descritta nel Piano di Progetto necessita di attività di verifica:

- **Analisi**: si seguono i metodi di verifica descritti nelle Norme di Progetto sui documenti prodotti e i processi attuati. La messa in opera di tali tecniche è descritta in (SEZIONE DI TEST SUI DOCUMENTI).

Questa sezione conterrà i riferimenti alle successive attività di verifica, aggiornati dopo ogni fase di produzione e di test del prodotto. La stesura dei documenti è l'attività principale e costante nello svolgimento del progetto, il processo di Verifica viene diviso in due attività. In ogni documento è presente un diario delle modifiche per mantenere una cronologia delle attività svolte e di chi le ha svolte.

2.6 Pianificazione strategica e temporale

Avendo scadenze prefissate nel Piano di Progetto, dobbiamo garantire che le attività di verifica di tutti i documenti e prodotti deve essere sistematica, disciplinata e quantificabile. Procedendo in questa maniera si correggono gli errori il prima possibile. La metodologia da seguire è descritta nelle Norme di Progetto.

2.7 Responsabilità

Per garantire che il processo di verifica sia disciplinato, sistematico e quantificabile, bisogna attribuire responsabilità a specifici ruoli di progetto. I ruoli sono Responsabile di Progetto e Verificatore. I compiti di ciascun ruolo sono descritti nelle Norme di Progetto.

2.8 Risorse

Per raggiungere gli obiettivi di qualità prefissati sono necessarie risorse umane e tecnologiche, suddivise rispettivamente in strumenti software e hardware utilizzati dai componenti del gruppo per effettuare verifica su processi e prodotti. I ruoli maggiormente coinvolti nella responsabilità delle attività di Verifica e validazione sono il Responsabile di Progetto e il Verificatore, e i rispettivi compiti sono descritti dettagliatamente nelle Norme di Progetto.

3 Analisi

3.1 Analisi statica

Questa tipologia di analisi può essere applicata sia al codice che alla documentazione, dato che prevede l'utilizzo di tecniche generali per ogni tipo di prodotto del team.

3.1.1 Walkthrough

Questa tecnica utilizza una scansione ampia e non mirata dell'oggetto in verifica, data la mancanza di esperienza e best practice del Verificatore. L'attuazione di questa tecnica di Analisi è quindi molto onerosa, per questo sarà nostro obiettivo renderla più parallelizzabile possibile, così da ridurre i costi di Verifica. Dopo ogni attività di verifica tramite Walkthrough, sperabilmente avremo trovato la maggior parte degli errori, fornendoci una visione delle erroneità commesse, di conseguenza potremo raffinare l'Analisi e avvicinarci alla metodologia di Inspection.

3.1.2 Inspection

Questa tecnica è un'evoluzione del Walkthrough e applica una ricerca più mirata e specifica. E' possibile utilizzare questa tecnica dopo aver acquisito dimestichezza con l'attività di Verifica, quindi non sarà possibile utilizzarla fin da subito. E' obiettivo di una fase di Inspection la ricerca mirata di errori, aumentando l'efficienza della Verifica e riducendo i costi in termini di tempo e risorse.

3.2 Analisi dinamica

Questa particolare tipologia di Analisi si applica solamente ai prodotti software sviluppati dal team, mediante l'utilizzo di test progettati e scritti appositamente per verificare la correttezza dei prodotti e la loro effettiva Validazione. L'importanza di un test si attua nella sua automazione, in quanto riduce il tempo dedicato alla Verifica manuale del codice, certamente più onerosa. Per questo motivo la proprietà più importante di un test è la sua ripetibilità. Per fare in modo che un test abbia questa qualità, è fondamentale definire a priori certe caratteristiche, ovvero:

- **Ambiente**: deve essere specificato l'insieme di componenti hardware e software su cui verrà eseguito il prodotto software, al fine di evitare problemi di incompatibilità e malfunzionamento;
- **Variabili**: si deve conoscere e garantire la corretta struttura delle variabili in ingresso ai test, in modo da prevedere gli output attesi e verificare la loro correttezza;
- **Procedure**: deve essere chiara la sequenza delle operazioni e la metodologia di applicazione dei test.

Di seguito analizzeremo i diversi tipi di test attuati nelle varie parti del progetto:

- **Test di unità**: attività di Verifica svolta su ogni singola unità software del sistema, mediante l'utilizzo di stub, driver e logger.;
- **Test di integrazione**: attività di Verifica che controlla la corretta integrazione di più unità software. Grazie a questo test si possono rilevare errori non riscontrabili nei test di unità. Anche in questa fase, per poter simulare le unità nell'integrazione, vengono create unità fittizie ad hoc;
- **Test di sistema**: questo test si propone di validare il prodotto, una volta che si è stabilita la sua versione definitiva;

- **Test di regressione**: attività di Verifica che consiste nel ripetere i test già effettuati su una componente, ogni qualvolta quella componente venga modificata o aggiornata;
- **Test di accettazione**: test finale effettuato dal proponente del software, al cui superamento segue il rilascio del prodotto ultimato.

4 Metrica

La definizione di metriche di prodotto e di processo è imprescindibile per garantire che il nostro prodotto sia quantificabile. Grazie l'utilizzo di metriche si stabilisce un riferimento per quantificare la maturità, la consistenza e la conformità agli standard. Data la variabilità delle metriche, si suddivideranno gli obiettivi per ciascuna metrica in due range possibili:

- **Sufficiente**: range stabilito come minimo accettabile, sotto il quale ogni unità o documento non verrà accettato come completo;
- **Ottimale**: range consigliato e da usare come riferimento, dal quale è necessario scostarsi il meno possibile.

4.1 Metrica documenti

Dopo aver vagliato diverse metriche di misurazione nell'ambito dei documenti, abbiamo scelto di utilizzare una metrica di complessità di leggibilità di un testo, tarata sulla lingua italiana. L'eccessiva variabilità nei metodi di Analisi della sillabazione dei termini e la conseguente non predicibilità dei test basati sulla sillabazione, hanno portato a scartare diverse metriche internazionali. Inoltre non sono state trovate metriche (oltre all'indice Gulpease) specifiche della lingua italiana e con esito certo.

4.1.1 Indice Gulpease

L'indice Gulpease[6] è un indice di leggibilità del testo che basa il suo calcolo su componenti del testo enumerabili meccanicamente, così da rendere automatico il processo di Verifica. L'indice prevede un intervallo di valori tra 0 e 100, dove 100 esprime la leggibilità massima. I nostri obiettivi per l'indice Gulpease sono i seguenti:

- **Obiettivo ottimale**: [50–100];
- **Obiettivo sufficiente**: [40–100].

4.2 Metrica software

Sebbene nella fase di Analisi dei Requisiti non sia prevista la stesura di codice, è comunque utile riportare metriche per l'Analisi della complessità del codice, da utilizzare nei processi di Verifica del software. Di seguito verranno riportate le metriche che in questa fase sono ritenute le più efficaci, che potranno tuttavia subire variazioni in corso d'opera, soprattutto nella fase di Codifica software.

4.2.1 Complessità ciclomatica

La complessità ciclomatica è una metrica software applicabile singolarmente a funzioni, moduli, metodi e classi di un programma. Questa metrica è calcolata utilizzando il grafo di controllo di flusso del programma. Alti valori di questa metrica implicano una scarsa manutenibilità del software, mentre valori troppo bassi possono indicare un'altrettanta bassa efficienza del software. Degli obiettivi ragionevoli per questa metrica sono i seguenti:

- **Obiettivo Sufficiente**: [1–15];
- **Obiettivo Ottimale**: [1–10][7].

[6] vedi appendice, sezione B.1
[7] Il valore 10 come massimo è stato calcolato da T.J.McCabe, inventore della metrica.

4.2.2 Livelli di annidamento

Il numero di livelli di annidamento dei metodi rappresenta la quantità di richiami di altri metodi all'interno di uno stesso metodo. Un elevato livello di annidamento definisce un'elevata complessità del codice e di conseguenza comprensione dello stesso. I nostri obiettivi stimati per questa metrica sono:

- **Obiettivo Sufficiente**: [1-6];
- **Obiettivo Ottimale**: [1-3].

4.2.3 Attributi per classe

Il numero di attributi per classe esprime, appunto, la quantità di proprietà di una classe. Un elevato numero di attributi può denotare un'eccessiva dimensione della classe stessa, che potrebbe piuttosto essere suddivisa in più classi più piccole. Gli obiettivi per questa metrica sono:

- **Obiettivo Sufficiente**: [0-16];
- **Obiettivo Ottimale**: [3-8].

4.2.4 Parametri per metodo

Un alto numero di parametri per metodo denota un'eccessiva complessità del metodo stesso, comportandone probabilmente un'ulteriore lunghezza non accettabile. E' buona norma scrivere dei metodi con pochi parametri, al fine di ottenere procedure specifiche e atomiche, di conseguenza facilmente assegnabili e verificabili. Degli obiettivi validi per questa metrica sono:

- **Obiettivo Sufficiente**: [0-8];
- **Obiettivo Ottimale**: [0-4].

4.2.5 Linee di codice per linee di commento

Questo numero indica il rapporto tra linee di codice e linee di commento, per avere un fattore di commenti all'interno di un'unità software. In generale, un alto grado di commento del codice porta ad una maggiore manutenibilità ed informazione per uno sviluppatore. Gli obiettivi stimati sono:

- **Obiettivo Sufficiente**: [> 0.25];
- **Obiettivo Ottimale**: [0.30][8].

4.2.6 Flusso di informazioni

Valore che calcola il flusso di informazioni passanti per un modulo. Definiti:

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

il valore calcolato è: $(lunghezza\ funzione)2 * Fan - in * Fan - out$

[8] Il valore di 0.30 è stato calcolato dal rapporto 22/78, derivato dalle medie di Ohloh https://www.ohloh.net/p/firefox/factoids#FactoidCommentsLow

4.2.7 Accoppiamento

- **Accoppiamento afferente**: questo fattore indica la quantità di di classi esterne ad un package che dipendono da classi interne allo stesso. Un alto valore di accoppiamento in una singola classe del package influisce sull'accoppiamento dell'intero package. Questo fatto non è necessariamente un errore di progettazione, ma il package in esame può rappresentare un punto critico del software. Per contro, un package con basso fattore di accoppiamento può delineare una scarsa utilità del package stesso, che probabilmente andrebbe inglobato con altri package;
- **Accoppiamento efferente**: questo fattore indica l'accoppiamento contrario, ovvero il numero di classi interne al package che dipendono da classi esterne. Più questo indice è basso, più indipendente è il package stesso.

4.2.8 Instabilità

Il fattore di instabilità di un package indica la possibilità di modifica del package senza influire sulla stabilità del software ad esso dipendente. Questo indice e' calcolato con la formula seguente:

$$I = Ce/(Ca + Ce)$$

dove Ca è l'accoppiamento afferente e Ce l'accoppiamento efferente. Gli obiettivi forniti da *best practice*[9] sono i seguenti:

- **Obiettivo Sufficiente**: [0.0 - 0.8];
- **Obiettivo Ottimale**: [0.0 - 0.3].

4.2.9 Copertura del codice

Questo fattore indica la percentuale di codice coperto durante l'esecuzione dei test. Più alta sarà la percentuale, minore sarà la possibilità di errori riscontrabili nell'esecuzione del software. Il valore ideale di 100% indica che tutte le porzioni di codice sono testate da uno o più test. Gli obiettivi stimati per questa metrica sono:

- **Obiettivo Sufficiente**: [42% - 100%];
- **Obiettivo Ottimale**: [65% - 100%].

4.3 Metrica processo

Delle metriche ragionevoli per rendere i processi quantificabili si sono scelti due indicatori che si basano sui costi e i tempi spesi per un processo. Data la scarsa esperienza di cooperazione e di pianificazione delle attività del gruppo, gli obiettivi di questi indici fanno riferimento a convenzioni comuni.

4.3.1 Schedule Variance

Indica se si è in linea, in anticipo o in ritardo rispetto alla schedulazione delle attività di progetto pianificate nella baseline. E' un indicatore di efficacia e se il suo valore è > 0 allora il progetto sta avanzando con maggiore velocità rispetto a quanto pianificato. Viceversa se negativo. Gli obiettivi fissati sono:

- **Obiettivo Sufficiente**: $[\geq -(costo\,preventivo\,per\,fase * 5\%)]$;
- **Obiettivo Ottimale**: $[\geq 0]$.

[9] Range ricavati da http://staff.unak.is/andy/StaticAnalysis0809/metrics/i.html

4.3.2 Budget Variance

Indica se alla data corrente si è speso di più o di meno rispetto a quanto si era pianificato alla data corrente. Se tale valore è > 0 allora il progetto sta consumando il proprio budget con minor velocità rispetto a quanto pianificato. Viceversa se negativo.

- **Obiettivo Sufficiente**: $[\geq -(costo\,preventivo\,per\,fase * 10\%)]$;
- **Obiettivo Ottimale**: $[\geq 0]$.

5 Resoconto attività di verifica

5.1 Processi

Di seguito vengono riportati i valori degli indici di SV e BV calcolati per i documenti durante la fase di Analisi.

Attività	SV	BV
Studio Fattibilità		
Analisi dei Requisiti		
Glossario		
Norme di Progetto		
Piano di Progetto		
Piano di Qualifica		

Tabella 2: SV e BV dei processi durante l'Analisi

5.2 Documenti

Di seguito vengono riportati gli indici di Gulpease calcolati per i vari documenti durante la fase di Analisi. L'esito di ciascun documento sarà superato o bocciato in base all'indice di Gulpease stabilito nel paragrafo(???).

Documento	Valore indice	Esito
Studio Fattibilità v1.2.0		
Analisi dei Requisiti v1.2.0		
Glossario v1.2.0		
Norme di Progetto v1.2.0		
Piano di Progetto v1.2.0		
Piano di Qualifica v1.2.0		

Tabella 3: Esiti dell'indice di Gulpease calcolato sui documenti durante l'Analisi

A Standard di qualità

A.1 ISO/IEC 15504

ISO/IEC 15504, anche conosciuto come SPICE(Software Process Improvement and Capability Determination, ovvero miglioramento di processi software e determinazione di capacità) è un insieme di documenti di standard tecnici per lo sviluppo software. Questo documento viene utilizzato nel perseguimento della qualità di processo in quanto stabilisce una struttura per la definizione degli obiettivi per il miglioramento dei processi stessi. Lo standard dichiara che ogni processo deve essere sottoposto ad un controllo continuo, ripetibile e quantificabile al fine di individuare i punti critici e misurare i miglioramenti.

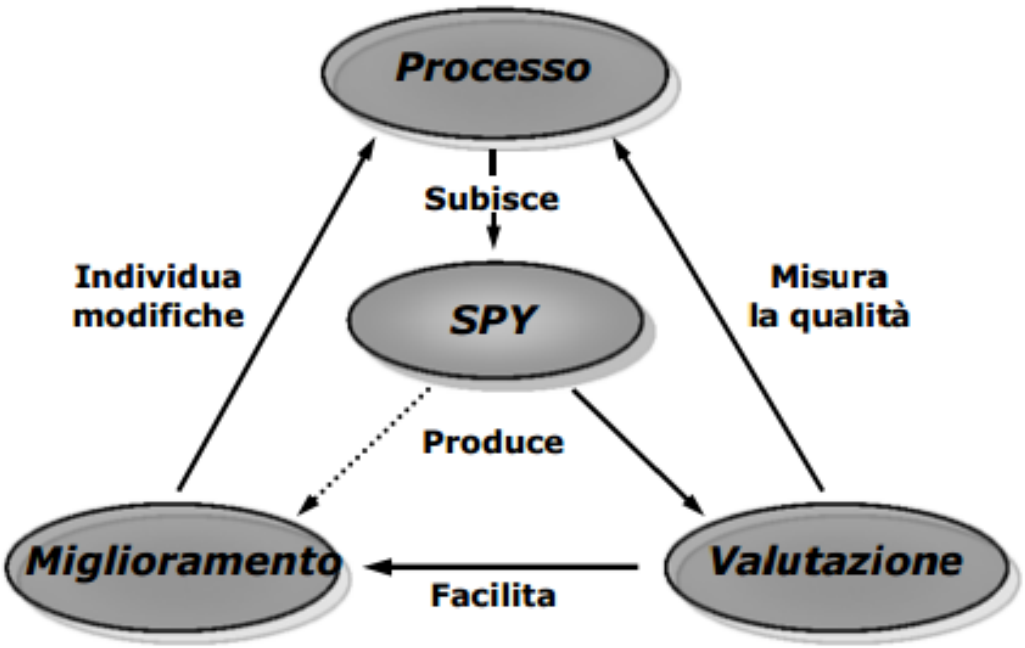

Figura 1: Software Process Assessment and Improvement

Secondo SPICE, un processo può essere classificato in base al suo livello di maturità, in una scala da 1 a 6, con annessi i livelli di capacità ad ogni livello:

- **Incomplete:** I risultati del processo non esistono o non sono appropriati;
- **Performed:** Si ottengono dei risultati, ma in un modo non specificato o non prevedibile;

  - *Process Performance*: capacità del processo di produrre degli output da dagli input.

- **Managed:** l'esecuzione è pianificata e tracciata, il prodotto è conforme a standard e requisiti specifici;
  - *Performance Management:* capacità del processo di produrre un output coerente con gli obiettivi del processo;
  - *Work Product Management:* capacità del processo di creare un risultato documentato, controllato e verificato.
- **Established:** il processo è eseguito e controllato riferendosi a dei buoni principi di ingegneria del software;
  - *Process Definition*: il processo fa riferimento a degli standard di processo per definire i risultati attesi;
  - *Process Deployment*: capacità del processo di utilizzare risorse appropriate per il raggiungimento degli obiettivi.
- **Predictable:** il processo è eseguito consistentemente con dei limiti di controllo definiti, per raggiungere altrettanto definiti obiettivi di processo;
  - *Process Measurement*: capacità di definizione di obiettivi e metriche di prodotto e di processo, con cui garantire il raggiungimento di obiettivi aziendali;
  - *Process Control*: capacità di controllo tramite metriche di progetto e prodotto definite, per puntare al miglioramento.
- **Optimizing:** l'esecuzione del processo è ottimizzata per soddisfare bisogni correnti e futuri, e il processo soddisfa ripetibilmente i suoi obiettivi prefissati;
  - *Process Innovation*: capacità di gestione di eventuali cambiamenti nel prodotto in modo controllato ed efficace;
  - *Continuous Optimization*: capacità di identificare e applicare modifiche atte al miglioramento dei processi aziendali.

Per finire, lo standard definisce 4 stadi di misurazione degli attributi di un processo, suddivisi in:

- **N**, non adeguato o non posseduto;
- **P**, parzialmente posseduto;
- **L**, largamente posseduto;
- **F**, completamente posseduto.

A.2 ISO/IEC 9126

Con la sigla ISO/IEC 9126 si individua una serie di normative e linee guida, sviluppate dall'ISO (Organizzazione internazionale per la normazione) in collaborazione con l'IEC (Commissione Elettrotecnica Internazionale), preposte a descrivere un modello di qualità del software. Il modello propone un approccio alla qualità in modo tale che le società di software possano migliorare l'organizzazione e i processi e, quindi come conseguenza concreta, la qualità del prodotto sviluppato.

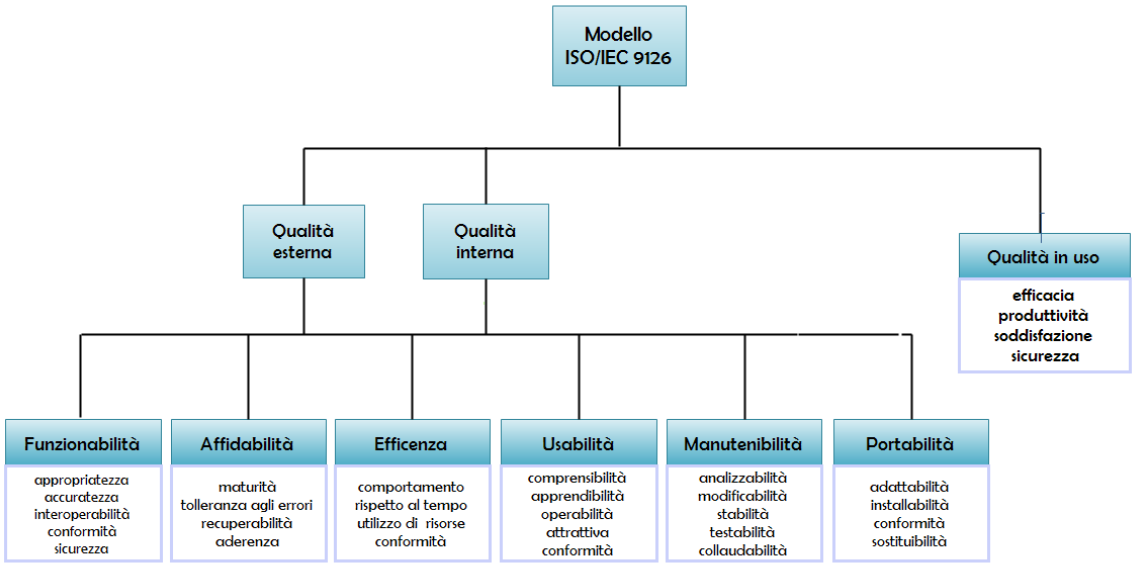

Figura 2: Modello di qualità ISO/IEC 9126

Il presente standard definisce 6 caratteristiche di qualità che ogni prodotto software deve perseguire, al fine di garantire la conformità agli standard con efficienza ed efficacia. Le caratteristiche delle qualità in uso esulano dal presente progetto didattico, in quanto non è prevista l'attività di manutenzione conseguente al rilascio del prodotto. Quindi ci soffermiamo all'analisi della qualità interna ed esterna dello standard ISO/IEC 9126. Le caratteristiche che un prodotto deve avere sono le seguenti:

- **Funzionalità**: capacità di un prodotto software di fornire funzioni che soddisfano esigenze stabilite, necessarie per operare sotto condizioni specifiche;
  - *Appropriatezza*: rappresenta la capacità del prodotto software di fornire un appropriato insieme di funzioni per gli specificati compiti ed obiettivi prefissati all'utente;
  - *Accuratezza*: la capacità del prodotto software di fornire i risultati concordati o precisi effetti richiesti;
  - *Interoperabilità*: la capacità del prodotto software di interagire ed operare con uno o più sistemi specificati;
  - *Conformità*: la capacità del prodotto software di aderire agli standard, convenzioni e regolamentazioni rilevanti al settore operativo a cui vengono applicati;
  - *Sicurezza*: la capacità del prodotto software di proteggere informazioni e i dati negando in ogni modo che persone o sistemi non autorizzati possano accedervi o modificarli, e che a persone o sistemi effettivamente autorizzati non sia negato l'accesso ad essi.
- **Affidabilità**: capacità del prodotto software di mantenere uno specificato livello di prestazioni quando usato in date condizioni per un dato periodo;

  - *Maturità*: capacità di un prodotto software di evitare che si verificano errori, malfunzionamenti o siano prodotti risultati non corretti;
  - *Tolleranza degli errori*: capacità di mantenere livelli predeterminati di prestazioni anche in presenza di malfunzionamenti o usi scorretti del prodotto;
  - *Recuperabilità*: capacità di un prodotto di ripristinare il livello appropriato di prestazioni e di recupero delle informazioni rilevanti, in seguito a un malfunzionamento. A seguito di un errore, il software può risultare non accessibile per un determinato periodo di tempo, questo arco di tempo è valutato proprio dalla caratteristica di recuperabilità;
  - *Aderenza*: capacità di aderire a standard, regole e convenzioni inerenti all'affidabilità.

- **Efficienza**: capacità di fornire appropriate prestazioni relativamente alla quantità di risorse usate;
  - *Comportamento rispetto al tempo*: capacità di fornire adeguati tempi di risposta, elaborazione e velocità di attraversamento, sotto condizioni determinate;
  - *Utilizzo delle risorse*: capacità di utilizzo di quantità e tipo di risorse in maniera adeguata;
  - *Conformità*: capacità di aderire a standard e specifiche sull'efficienza.

- **Usabilità**: capacità del prodotto software di essere capito, appreso, usato e benaccetto dall'utente, quando usato sotto condizioni specificate.
  - *Comprensibilità*: esprime la facilità di comprensione dei concetti del prodotto, mettendo in grado l'utente di comprendere se il software è appropriato;
  - *Apprendibilità*: capacità di ridurre l'impegno richiesto agli utenti per imparare ad usare la sua applicazione;
  - *Operabilità*: capacità di mettere in condizione gli utenti di farne uso per i propri scopi e controllarne l'uso;
  - *Attrattiva*: capacità del software di essere piacevole per l'utente che ne fa uso;
  - *Conformità*: capacità del software di aderire a standard o convenzioni relativi all'usabilità.

- **Manutenibilità**: capacità del software di essere modificato, includendo correzioni, miglioramenti o adattamenti;
  - *Analizzabilità*: rappresenta la facilità con la quale è possibile analizzare il codice per localizzare un errore nello stesso;
  - *Modificabilità*: capacità del prodotto software di permettere l'implementazione di una specificata modifica (sostituzioni componenti);
  - *Stabilità*: capacità del software di evitare effetti inaspettati derivanti da modifiche errate;
  - *Testabilità*: capacità di essere facilmente testato per validare le modifiche apportate al software.

- **Portabilità**: capacità del software di essere trasportato da un ambiente di lavoro ad un altro. (Ambiente che può variare dall'hardware al sistema operativo);
  - *Adattabilità*: capacità del software di essere adattato per differenti ambienti operativi senza dover applicare modifiche diverse da quelle fornite per il software considerato;
  - *Installabilità*: capacità del software di essere installato in uno specificato ambiente;
  - *Conformità*: capacità del prodotto software di aderire a standard e convenzioni relative alla portabilità;
  - *Sostituibilità*: capacità di essere utilizzato al posto di un altro software per svolgere gli stessi compiti nello stesso ambiente.

A.2.1 Qualità esterne

Le metriche esterne, specificate nella norma ISO/IEC 9126-2, misurano i comportamenti del software sulla base dei test, dall'operatività e dall'osservazione durante la sua esecuzione, in funzione degli obiettivi stabiliti in un contesto tecnico rilevante o di business.

A.2.2 Qualità interne

La qualità interna, più precisamente le metriche interne, è specificata nella norma ISO/IEC 9126-3 e si applica al software non eseguibile (ad esempio il codice sorgente) durante le fasi di progettazione e codifica. Le misure effettuate permettono di prevedere il livello di qualità esterna ed in uso del prodotto finale, poiché gli attributi interni influiscono su quelli esterni e quelli in uso. Le metriche interne permettono di individuare eventuali problemi che potrebbero influire sulla qualità finale del prodotto prima che sia realizzato il software eseguibile. Esistono metriche che possono simulare il comportamento del prodotto finale tramite simulazioni.

A.3 Ciclo di Deming (ciclo PDCA)

Il ciclo di Deming o Deming Cycle (ciclo di PDCA - plan–do–check–act) è un modello studiato per il miglioramento continuo della qualità in un'ottica a lungo raggio. Serve per promuovere una cultura della qualità che è tesa al miglioramento continuo dei processi e all'utilizzo ottimale delle risorse. Questo strumento parte dall'assunto che per il raggiungimento del massimo della qualità sia necessaria la costante interazione tra ricerca, progettazione, test, produzione e vendita. Per migliorare la qualità e soddisfare il cliente, le quattro fasi devono ruotare costantemente, tenendo come criterio principale la qualità. La sequenza logica dei quattro punti ripetuti per un miglioramento continuo è la seguente:

- **P** - Plan. Pianificazione.
- **D** - Do. Esecuzione del programma, dapprima in contesti circoscritti.
- **C** - Check. Test e controllo, studio e raccolta dei risultati e dei riscontri.
- **A** - Act. Azione per rendere definitivo e/o migliorare il processo.

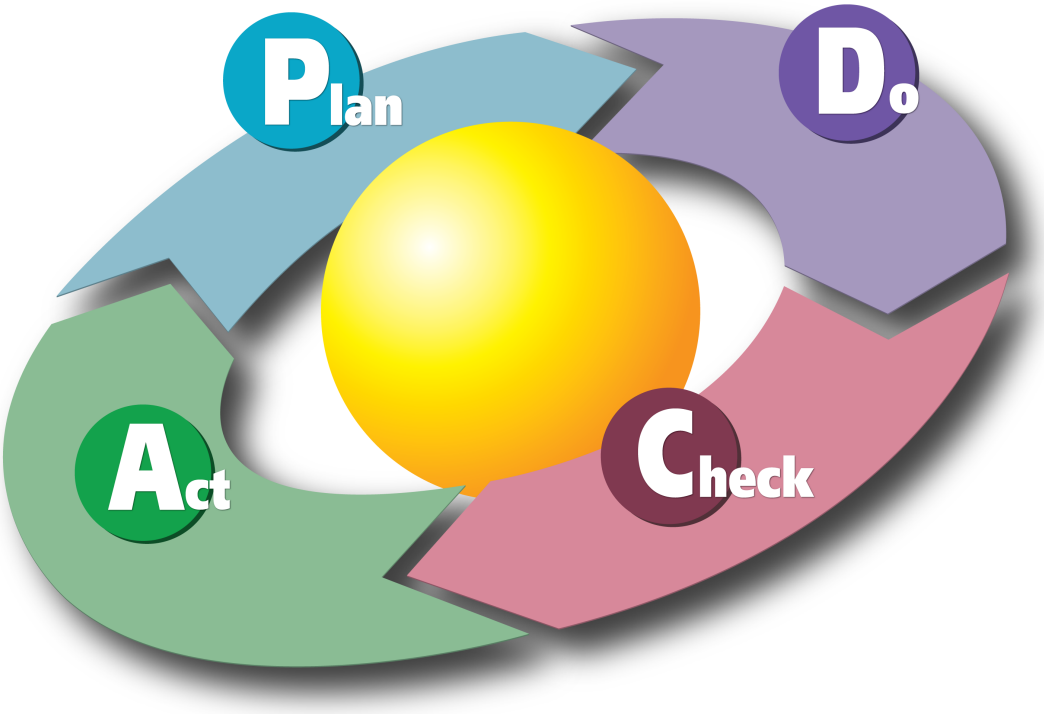

Figura 3: Ciclo di PDCA

B Descrizione

B.1 Indice Gulpease

L'**Indice Gulpease** è un 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.
Definito nel 1988 nell'ambito delle ricerche del GULP (Gruppo Universitario Linguistico Pedagogico) presso il Seminario di Scienze dell'Educazione dell'Università degli studi di Roma La Sapienza, si basa su rilevazioni raccolte tra il 1986 e il 1987 dalle cattedre di Filosofia del linguaggio e di Pedagogia dell'Istituto di Filosofia.
L'indice di Gulpease considera due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere.
La formula per il suo calcolo è la seguente:

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

.
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.

B.2 Metriche software

B.2.1 Complessità ciclomatica

La complessità ciclomatica (o complessità condizionale) è una metrica software. Sviluppata da Thomas J. McCabe nel 1976, è utilizzata per misurare la complessità di un programma. Misura direttamente il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. La complessità ciclomatica è calcolata utilizzando il grafo di controllo di flusso del programma: i nodi del grafo corrispondono a gruppi indivisibili di istruzioni, mentre gli archi connettono due nodi se il secondo gruppo di istruzioni può essere eseguito immediatamente dopo il primo gruppo. La complessità ciclomatica può, inoltre, essere applicata a singole funzioni, moduli, metodi o classi di un programma. La complessità ciclomatica di una sezione di codice è il numero di cammini linearmente indipendenti attraverso il codice sorgente. Per esempio, se il codice sorgente non contiene punti decisionali come IF o cicli FOR, allora la complessità sarà 1, poiché esiste un solo cammino nel sorgente (e quindi nel grafo). Se il codice ha un singolo IF contenente una singola condizione, allora ci saranno due cammini possibili: il primo se l'IF viene valutato a TRUE e un secondo se l'IF viene valutato a FALSE.
Matematicamente, la complessità ciclomatica di un programma è definita in riferimento ad un grafo diretto contenente i blocchi base di un programma con un arco tra due blocchi se il controllo può passare dal primo al secondo (il grafo di controllo di flusso). La complessità è quindi definita come:

$$v(G) = e - n + 2p$$

dove:

- **v(G)** = complessità ciclomatica del grafo G;

- **e** = numero di archi del grafo;
- **n** = numero di nodi del grafo;
- **p** = numero di componenti connesse.