Piano di Qualifica

Gruppo SteakHolders — Progetto MaaP

Informazioni sul documento	
Versione	1.3.1
Redazione	Enrico Rotundo
Verifica	Giacomo Fornari
Approvazione	Nicolò Tresoldi
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo SteakHolders
	CoffeeStrap

Descrizione
Questo documento descrive le operazioni di verifica e validazione seguite dal gruppo SteakHolders relativi al progetto MaaP.

Registro delle modifiche

Versione	Data	Persone coinvolte	Descrizione
1.3.1	2013-12-20	Nicolò Tresoldi (Responsabile in deroga)	Approvazione.
1.2.3	2013-12-19	Giacomo Fornari (Verificatore)	Verifica.
1.2.2	2013-12-19	Gianluca Donato (Verificatore)	Stesura resoconto verifiche.
1.2.1	2013-12-17	Giacomo Fornari (Verificatore)	Verifica.
1.1.7	2013-12-15	Enrico Rotundo (Progettista)	Stesura Pianificazione strategica.
1.1.6	2013-12-14	Enrico Rotundo (Progettista)	Stesura Gestione amministrativa.
1.1.5	2013-12-13	Enrico Rotundo (Progettista)	Stesura Appendice Qualità.
1.1.4	2013-12-12	Enrico Rotundo (Progettista)	Stesura Misure e Metriche.
1.1.3	2013-12-11	Enrico Rotundo (Progettista)	Stesura Tecniche.
1.1.2	2013-12-10	Enrico Rotundo (Progettista)	Stesura Introduzione.
1.1.1	2013-12-10	Enrico Rotundo (Progettista)	Creazione documento.

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

L'obiettivo primario è la *qualità*$_G$ del prodotto e dei suoi processi, ottenibile tramite una serie di controlli proattivi stabiliti al tempo zero. L'assenza di tali verifiche abbinata ad un team di più soggetti senza particolari accortezze e competenze, portano al progressivo deterioramento del materiale prodotto, sia esso codice sorgente o documentazione. Questo fenomeno è noto sotto il nome di *Broken windows theory*$_G$ ed è intrinseco alla componente sociale dell'uomo. Il concetto chiave è prevenire l'inserimento di materiale non aderente alle *Norme di Progetto v1.3.1* poiché, secondo la teoria succitata, innescherebbe un meccanismo che deteriora la qualità all'interno del *repository*$_G$.

Si vogliono gestire le componenti *accidentali*$_G$ dei processi, ossia tutte quelle problematiche non intrinseche alla produzione, ma che ne sono direttamente collegate; si desidera scongiurare il pericolo di operare *by correction*$_G$ per evitare modifiche in corso d'opera che posso bloccare la maturazione del prodotto e richiedere dispendiose correzioni.

1.1 Scopo del documento

Il Piano di Qualifica illustra la strategia di verifica e validazione che il gruppo SteakHolders ha deciso di adottare. È necessario dare una dimensione alla qualità dei prodotti e dei processi, operazioni che non rientrano nei normali ruoli di progetto, bensì rappresentano una *funzione aziendale*. Vi sono molteplici punti di vista della qualità, il *committente*$_G$ sarà in grado di valutare su basi oggettive quanto prodotto. Inoltre, la direzione del progetto potrà fare affidamento sulla consistenza dello stato dello stesso e il proponente avrà una solida base di verifica ideata e funzionante.

Questo documento si colloca nella parte relativa al *Project* delle *quattro P del Software Engineering* (People, Product, Project, Process) perché tratta una parte fondamentale sulle attività a supporto della produzione di MaaP.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un *framework*$_G$ per generare interfacce web di amministrazione dei dati di *business*$_G$ basato su *stack*$_G$ *Node.js*$_G$ e *MongoDB*$_G$. L'obbiettivo è quello di semplificare il processo di implementazione di tali interfacce che lo sviluppatore, appoggiandosi alla produttività del framework MaaP, potrà generare in maniera semplice e veloce ottenendo quindi un considerevole risparmio di tempo e di sforzo. Il fruitore finale delle pagine generate sarà infine l'esperto di business che potrà visualizzare, gestire e modificare le varie entità e dati residenti in *MongoDB*$_G$. Il prodotto atteso si chiama *MaaP*$_G$ ossia *MongoDB as an admin Platform*.

Il prodotto atteso si chiama *MaaP*$_G$ ossia *MongoDB as an admin Platform*.

1.3 Glossario

Ogni occorrenza di termini tecnici, di dominio e gli acronimi sono marcati con una G in pedice e riportati nel documento *Glossario v1.3.1*.

1.4 Riferimenti

Vengono elencanti i riferimenti su cui si è basata l'organizzazione dell'attività di qualifica.

1.4.1 Normativi

- **Norme di Progetto:** *Norme di Progetto*;
- **Capitolato d'appalto C1:** http://www.math.unipd.it/~tullio/IS-1/2013/Progetto/C1.pdf.

1.4.2 Informativi

- **Piano di Progetto:** *Piano di Progetto v1.3.1*;
- **Slide di Ingegneria del Software mod. A:** http://www.math.unipd.it/~tullio/IS-1/2013/;
- **SWEBOK 2004 Version - capitolo 11:** http://www.computer.org/portal/web/swebok/htmlformat;
- **Software Engineering 9th - I. Sommerville (Pearson, 2011) - capitoli: 8, 24, 26:** http://www.pearsoned.co.uk/bookshop/detail.asp?item=100000000377819;
- **Software Engineering Method and Theory (SEMAT)** http://www.ivarjacobson.com/semat/;
- **ISO/IEC 15504** http://en.wikipedia.org/wiki/ISO/IEC_15504;
- **ISO/IEC 9126** http://en.wikipedia.org/wiki/ISO/IEC_9126.

2 Visione generale della strategia di verifica

La strategia generale adottata è quella di automatizzare il più possibile il lavoro di verifica; questo richiede scelta e uso di *tools*$_G$ adeguatamente configurati. L'obiettivo è avere un riscontro affidabile e numericamente trattabile che permetta di assicurare il grado di qualità predeterminato. Il lavoro manuale verrà così ridotto al minimo e confinato all'opera di validazione. La speranza è che dei buoni processi portino ad un buon software.

2.1 Organizzazione

Viene verificata la qualità di ogni processo e di ogni output da esso prodotto. Ogni fase descritta nel *Piano di Progetto v1.3.1* produce output di diverso tipo, per questo è necessario programmare l'attività di verifica in modo mirato:

- **Analisi**: in questa fase si controllano che i processi e la documentazione prodotta rispettino le *Norme di Progetto*, verrà verificato che ogni requisito abbia corrispondenza in un caso d'uso utilizzando l'applicativo web *Requisteak*$_G$;
- **Progettazione Architetturale**: in questa fase vanno verificati i processi incrementali relativi all'analisi e ai nuovi documenti di progettazione;
- **Progettazione di Dettaglio e Codifica**: in questa fase vanno verificati i processi incrementali relativi alla progettazione.

Il *Diario delle modifiche* viene incluso in ogni documento al fine di tracciarne uno storico dell'evoluzione.

2.2 Pianificazione strategica e temporale

2.2.1 Strategia

Il *Piano di Progetto* fissa una serie di scadenze improrogabili, pertanto è necessario definire con chiarezza una strategia di qualifica efficace. Gli incrementi sulla documentazione o sul codice possono essere di natura programmata, quindi prefissati nel calendario, oppure posso insorgere come inaspettati, in questo caso sarà necessario programmare le dovute modifiche; è questo il caso di *bug*$_G$ o *errori* (vedi paragrafo 3.1). La qualità di ogni incremento si basa sul fatto che la struttura di qualifica garantisce il rispetto delle *Norme di Progetto*. Questo lavoro verrà svolto con l'aiuto di automatismi che segnaleranno le problematiche rilevate in modo da permettere una rapida correzione, in particolare:

- Documentazione: tramite uno script di *pre-commit*$_G$ viene bloccato il commit nel *repository*$_G$ se le modifiche eseguite impediscono la compilazione dei documenti o se gli strumenti di verifica automatica rilevano delle anomalie. In questo modo viene sempre garantita una qualità minima per il codice caricato sul *repository*$_G$.

Per migliorare l'efficienza dei processi si utilizzano quanto più possibile automatismi, in modo da poter destinare le risorse umane ad altri lavori. L'utilizzo di software apposito permette di eseguire controlli mirati senza consumare risorse umane. L'implementazione di tali controlli viene descritta nelle *Norme di Progetto v1.3.1*.

2.2.2 Tempistiche

La pianificazione strategica si basa sul modello $SEMAT_G$. Le seguenti specifiche non coprono tutti gli stati indicati da questo schema considerando solo l'aspetto relativo alla **qualità**.

Le sezioni coinvolte sono:

- *Solution* → *Software System*: qualità del software;
- *Endeavor* → *Way of Working*: qualità del ambiente di lavoro garantita dalle norme;
- *Endeavor* → *Work*: qualità dei processi.

In questo modello viene fatta una distinzione tra lavoro e modo di lavorare, la qualità però è trasversale a queste due entità, per cui andrebbero considerati anche alcuni aspetti della sezione *work*. Per chiarezza è stato scelto di considerare il *Way of Working* perché, in accordo con lo schema fornito da $SEMAT_G$, guida il lavoro stesso determinando la qualità prodotta.

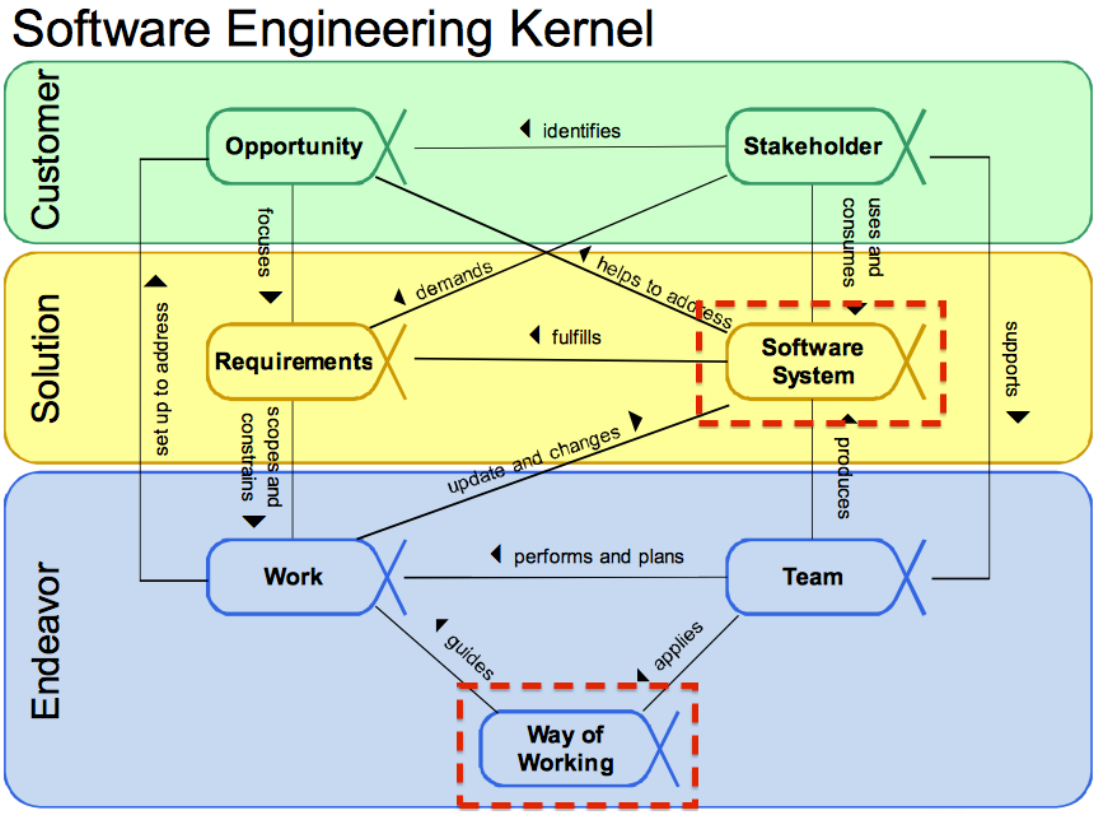

Figura 1: Software Engineering Kernel

Di seguito viene descritto lo stato che ogni fase descritta dal *Piano di Progetto* deve soddisfare, in relazione alle sezioni del $SEMAT_G$ sopracitate.

2.2.2.1 Analisi

- Software System:

1. *Architecture Selected*: le piattaforme, le tecnologie e i linguaggi da usare sono stati selezionati. Vengono individuate le metriche per il software, i documenti ed i processi.

- Way of Working:
  1. *Principles Established*: i vincoli sono stati definiti nelle *Norme di Progetto v1.3.1*, i $tools_G$ sono stati individuati ed il team comincia ad operare secondo quanto stabilito;
  2. *Foundation Established*: i $tools_G$ per la qualità sono configurati e le buone pratiche da seguire sono definite;
  3. *In Use*: alcuni membri del team utilizzano le pratiche indicate e i tools vengono usati e migliorati. Vengono raccolti feedback sull'utilizzo degli strumenti per la qualità.

2.2.2.2 Progettazione Architetturale

- Software System:
  2. *Demonstrable*: le caratteristiche chiave dell'architettura software sono state individuate e gli $stakeholders_G$ sono in accordo. Vengono individuate le unità da testare per soddisfare la richiesta del $proponente_G$ di verificare il funzionamento di almeno il 70% del software prodotto, come stabilito nel *Verbale del 2013-12-05*.

- Way of Working:
  4. *In Place*: tutti i membri utilizzano e comprendono l'intero ambiente di lavoro che è configurato secondo le *Norme di Progetto v1.3.1*. Il $repository_G$ è configurato per accettare solo materiale in linea con le norme stabilite;
  5. *Working Well*: tutto il team lavora in modo coordinato secondo le norme applicando le pratiche individuate nelle norme e i $tools_G$ supportano a pieno regime l'attività di lavoro.

2.2.2.3 Progettazione di Dettaglio e Codifica

- Software System:
  3. *Usable*: il sistema è parzialmente usabile. Vengono praticati gli opportuni test che saranno convalidati ed il livello di anomalie sarà accettabile;
  4. *Ready*: la documentazione utente è disponibile ed è conforme alle norme. Vengono forniti i risultati dell'analisi statica e dei test eseguiti, si correggono le anomalie e gli $stakeholders_G$ accettano il sistema.

2.2.2.4 Validazione

- Software System:
  5. *Operational*: il sistema è in uso in un ambiente pienamente funzionante. Vengono ultimati gli ultimi test;
  6. *Retired*: il ritiro del software non viene considerato all'interno di questo progetto didattico.

- Way of Working:
  6. *Retired*: il ritiro del software non viene considerato all'interno di questo progetto didattico.

2.3 Responsabilità

La responsabilità della verifica viene attribuita al *Responsabile* di progetto e ai *Verificatori*. I compiti e le modalità di attuazione sono definiti nel *Piano di Progetto*.

2.4 Risorse

La qualifica dei processi essendo un processo, consuma delle risorse che si dividono in due categorie:

- **Umane**: le figure coinvolte sono il *Responsabile* di progetto e il *Verificatore*. I processi da loro effettuati consumano ore di produttività contabilizzate e schedulate secondo il *Piano di Progetto*. Le ore di produttività sono fissate dalle regole di progetto (http://www.math.unipd.it/~tullio/IS-1/2013/Progetto/PD01b.html) in un minimo di 85 e un massimo di 105 ore individuali. Il *Piano di Progetto* determina la distribuzione di tali quote orarie con la relativa retribuzione. Ai fini della qualifica si potrà parlare di ore di produttività tralasciandone l'aspetto economico, in quanto non rientra nel dominio del documento succitato;
- **Tecnologiche**: riguardano i *mezzi* utilizzati per gli automatismi per la qualità e la loro gestione. Trattandosi esclusivamente di mezzi informatici, vengono consumate unità di calcolo considerate a costo nullo. Tale considerazione si basa sul fatto che tutti i tipi di elaborazioni informatiche sono svolte su mezzi per i quali non è richiesto né un contributo economico, né un quantitativo temporale abbastanza consistente da poter essere considerato degno di nota.

Le modalità del loro impiego sono descritte dettagliatamente nelle *Norme di Progetto*.

2.5 Tecniche

Le *Software Quality Managment Techinques*$_G$ adottate dal gruppo sono suddivise in quattro categorie. Di seguito ne vengono date le definizioni, tuttavia non sempre si tratta di classi nettamente distinte e la loro applicazione molto spesso prevede sovrapposizioni.

2.5.1 Statiche

Consiste nello studiare la documentazione ed il software senza istanze di esecuzione del sorgente. Queste tecniche possono includere attività *people-intensive* o di *analisi* condotte da singoli individui con o senza l'assistenza di *tools*$_G$ automatizzati.

2.5.1.1 Walkthrough

Si svolge effettuando una lettura a largo spettro. È un'attività onerosa e collaborativa che richiede la cooperazione di più persone, si tratta di una tecnica non efficiente pertanto se ne sconsiglia l'attuazione. Ne è previsto l'utilizzo principalmente durante la prima parte del progetto quando non tutti i membri del gruppo hanno piena padronanza e conoscenza delle *Norme di Progetto* e del *Piano di Qualità*.

2.5.1.2 Inspection

Si tratta di una lettura mirata e strutturata, volta a localizzare l'errore con il minor costo possibile. Richiede una buona padronanza di tutti gli strumenti utilizzati e conoscenza di tutti i documenti di progetto. Solitamente viene effettuata da un singolo individuo. La ricerca mirata si basa sugli errori ricorrenti, uno script dovrà ricercare tali anomalie e segnalarle al *Verificatore* in modo efficiente. Le anomalie più frequenti sono:

- utilizzo di comandi locali LaTeX deprecati, ossia quei comandi che erano stati definiti alla prima creazione del *repository*$_G$ ma che poi sono stati ridefiniti;

- utilizzo di caratteri di underscore senza prima anteporre una backslash;
- errori ortografici come rappresentare la *È* con una *e* maiuscola apostrofata;
- inserimento di parole comuni come placeholder, e.g. ciao, blabla, asd ...;
- formato delle date errato o non conforme alle *Norme di Progetto v1.3.1*;
- formato dei nomi errato, le *Norme di Progetto v1.3.1* indicano prima il nome e poi il cognome;
- elenchi puntati non conformi alle norme, in particolare i punti spesso non terminano con il punto e virgola;

2.5.2 People-intensive

Sono attività che coinvolgono almeno due persone impegnate a svolgere compiti anche parzialmente non automatizzabili. Solitamente l'efficienza non è massima in quanto si paga il coordinamento tra le persone. Le risorse necessarie sono checklists e i risultati dei tests e delle altre tecniche di analisi.

2.5.3 Analitiche

Le tecniche analitiche comprendono tutte quelle azioni volte a valutare criticamente un componente del prodotto. Tali tecniche si servono di metodi come l'analisi della complessità, degli algoritmi e del controllo di flusso. L'analisi di complessità è utile per valutare quanto un componente è articolato da progettare, implementare, testare o manutenere. Il controllo di flusso è volto al riconoscimento delle anomalie e può essere usato a supporto di altre attività. Infine, l'analisi degli algoritmi è fondamentale in quei software in cui vi è una componente algoritmica importante e vi è la necessità di assicurare output consistenti e corretti.

2.5.4 Dinamiche

Le tecniche dinamiche vengono generalmente utilizzate durante lo sviluppo e la manutenzione. Si eseguono dei test ripetibili basati sull'effettiva esecuzione del codice. È quindi opportuno costruire un set di strumenti per riprodurre insiemi di *input* e portare il software in uno stato iniziale, al fine di verificare velocemente che gli output siano quelli attesi. Un file di log_G permetterà, in base al contenuto, di ricostruire le sequenze di operazioni effettuate. Di seguito vengono elencati i test che si andranno ad effettuare in ordine di granularità.
Il $proponente_G$, nel corso del *Verbale* del 2013-12-05, ha indicato che dovrà essere testato almeno il 70% del codice prodotto.

2.5.4.1 Test di unità

Si tratta di testare il funzionamento delle $unità_G$ tramite l'utilizzo di $stub_G$, $test\ driver_G$, e $logger_G$. Un'$unità_G$ consiste nella più piccola quantità di software che è utile verificare singolarmente. Questi test dovrebbero procedere quanto più in parallelo, assegnando priorità alle $unità_G$ che mostrano dei risultati utilizzabili per definire dei $prototipi_G$. Il corretto funzionamento di tutte le unità riduce al minimo la presenza di errori di programmazione e permette di integrare queste componenti secondo le specifiche di progetto con la garanzia che esse funzionino.

2.5.4.2 Test di integrazione

Consiste nel test di una parte di due o più *unità*$_G$ e ne valuta globalmente i risultati. Le componenti non ancora sviluppate vanno simulate con dei sostituti fittizi.

2.5.4.3 Test di sistema

Consiste nella validazione del sistema per accertare la copertura dei requisiti software e il suo collaudo viene supervisionato dal committente per mostrare la conformità del prodotto.

2.5.4.4 Test di regressione

Consiste nell'eseguire nuovamente i test riguardanti le componenti software che hanno subito modifiche. Tale operazione è aiutata dal tracciamento, il quale permette di individuare e ripetere facilmente i test di unità, di integrazione e di sistema che sono stati potenzialmente influenzati dalla modifica.

2.5.4.5 Test di accettazione

Si tratta del collaudo del prodotto in presenza del proponente. Al superamento di tale collaudo segue il rilascio ufficiale del prodotto sviluppato.

2.6 Misure e Metriche

Vengono adottate delle metriche per rendere misurabili e valutabili i processi, i documenti ed il software prodotto. La visione non vuole essere comparativa, ma serve al gruppo per monitorare l'andamento dei processi e la qualità del prodotto.

2.6.1 Metriche per i processi

Le seguenti metriche rappresentano un indicatore volto a monitorare e prevedere l'andamento delle principali variabili critiche del progetto (i tempi e i costi). Sono state scelte metriche di tipo *consuntivo*$_G$ perché danno un riscontro immediato sullo stato attuale del progetto; ad ogni incremento verranno valutati tali indici e, se necessario, verranno stabiliti opportuni provvedimenti da parte del *Responsabile* di progetto.

2.6.1.1 Schedule Variance

Indica se si è in linea, in anticipo o in ritardo rispetto alla schedulazione delle attività di progetto pianificate.

$$SV = BCWP - BCWS$$

Dove $BCWP$ indica il valore delle attività realizzate alla data corrente e $BCWS$ rappresenta il costo pianificato per realizzare le attività di progetto alla data corrente. È un indicatore di efficacia soprattutto nei confronti del Cliente. Se $SV > 0$ significa che il progetto sta producendo con maggior velocità a quanto pianificato, viceversa se negativo.

2.6.1.2 Budget Variance

Indica se alla data corrente si è speso di più o di meno rispetto a quanto previsto.

$$BV = BCWS - ACWP$$

Dove $BCWS$ indica il costo pianificato per realizzare le attività di progetto alla data corrente e $ACWP$ rappresenta il costo effettivamente sostenuto alla data corrente. È un indicatore che ha un valore unicamente contabile e finanziario. Se $BV > 0$ significa che il progetto sta spendendo il proprio budget con minor velocità di quanto pianificato, viceversa se negativo.

2.6.2 Metriche per i documenti

La *leggibilità* dei documenti è indispensabile per garantirne la qualità. Si è scelto di adottare un indice per misurare la leggibilità dei testi in lingua italiana:

2.6.2.1 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. Permette di misurare la complessità dello stile di un documento. 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:

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

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].

2.6.3 Metriche per il software

La prima release di *NodeJS*$_G$ risale a Maggio 2009, è stata riscontrata una forte differenza tra le metriche disponibili per l'analisi statica rispetto a quelle per i linguaggi meno recenti. Inoltre, nessun membro del gruppo ha conoscenza di tale linguaggio e delle sue particolarità come l'aspetto *funzionale*$_G$. Tali differenze con i linguaggi studiati nel percorso universitario si sono tradotte nella difficoltà di individuare metriche non incentrate sulla visione ad oggetti del codice. Infine, si è osservata l'assenza di strumenti per la misurazione di metriche tradizionali come la coesione e l'instabilità dei package. Di seguito vengono elencate le metriche per il software prodotto.

2.6.3.1 Complessità ciclomatica

La complessità ciclomatica è una metrica software che indica la complessità di un programma misurando il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. Nel grafo sopracitato i *nodi* 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. Questo indice può essere applicato indistintamente a singole funzioni, moduli, metodi o package di un programma. Si vuole utilizzare tale metrica per limitare la complessità durante la fase di sviluppo. Durante il testing è utile per determinare il numero di casi di test necessari, infatti l'indice di complessità è un limite superiore al numero di test necessari per raggiungere il coverage completo del modulo testato. Inoltre, uno studio ha mostrato forti corrispondenze tra le metriche di complessità e il livello di coesione nei package presi in esame[1].
Parametri utilizzati:

- Range-accettazione: [0 - 15];
- Range-ottimale: [0 - 10][2] .

2.6.3.2 Numero di metodi - NOM

Il *Number of methods* è una metrica usata per calcolare la media delle occorrenze dei metodi per package. Un package non dovrebbe contenere un numero eccessivo di metodi. Valori superiori al range ottimale massimo potrebbero indicare una necessità di maggiore decomposizione del package.
Parametri utilizzati:

- Range-accettazione: [3 - 10];
- Range-ottimale: [3 - 7].

2.6.3.3 Bugs per lines of code

Questa metrica misura il numero di bugs trovati su un certo quantitativo di linee di codice. L'aumentare del sorgente implica un incremento delle probabilità di nascondere errori, per questo è bene mantenere il codice più chiaro e semplice possibile. Con la crescita del prodotto è utile monitorare il rapporto tra i difetti trovati e il codice incrementale, tale indice dovrebbe restare costante o diminuire nel tempo. Il gruppo fissa questa metrica ad un massimo di 60, considerando il fatto che nessun membro ha conoscenze dello *stack*$_G$ tecnologico utilizzato, l'obiettivo è di giungere alla *Revisione di Accettazione* con valori compresi tra 0 e 20. Lo sforamento di tali valori determina l'intervento del *Responsabile* di progetto che dovrà individuare tempestivamente la causa del problema.

2.6.3.4 Variabili non utilizzate e non definite

La presenza di variabili non utilizzate viene considerata *pollution*$_G$ pertanto non viene tollerata. Tali occorrenze vengono rilevate analizzando l'*Abstract syntax tree*$_G$ (AST) eseguendo una cross-reference tra le variabili dichiarate e quelle inizializzate. Per sua natura, *Javascript*$_G$ non blocca l'insorgenza di tali occorrenze, pertanto si rischia di dichiarare una variabile e poi utilizzarne una con nome leggermente diverso, oppure semplicemente dichiarare una variabile che in seguito non verrà mai utilizzata.
Parametri utilizzati:

[1] Stein, C., G. Cox and L. Etzkorn, 2005. Exploring the Relationship between Cohesion and Complexity. J. Comput. Sci., 1: 137-144.
[2] McCabe (dicembre 1976). A Complexity Measure. IEEE Transactions on Software Engineering: 308–320.

- Range-accettazione: [0 - 0];
- Range-ottimale: [0 - 0].

2.6.3.5 Numero parametri per metodo

Un numero elevato di parametri per un metodo potrebbe evidenziare un metodo troppo complesso.
Parametri utilizzati:

- Range-accettazione: [0 - 8];
- Range-ottimale: [0 - 4].

2.6.3.6 Numero funzioni d'interfaccia per package

Un numero elevato di funzioni d'interfaccia in un package evidenzia un possibile errore di progettazione.
Parametri utilizzati:

- Range-accettazione: [0 - 20];
- Range-ottimale: [1 - 10].

2.6.3.7 Halstead

La metrica di Halstead non è solamente un indice di complessità, ma identifica le proprietà misurabili del software e le relative relazioni. Si basa sull'osservazione che una metrica dovrebbe valutare l'implementazione di un algoritmo in linguaggi differenti ed essere indipendente dal esecuzione su una specifica piattaforma. In un problema vengono identificati:

- n_1 = il numero di operatori distinti
- n_2 = il numero di operandi distinti
- N_1 = il numero totale di operatori
- N_2 = il numero totale di operandi

Da cui vengono calcolati:

- $n = n_1 + n_2$: vocabolario della funzione
- $N = N_1 + N_2$: lunghezza della funzione

Data la bassa disponibilità nella rete di valori di riferimento, i range specificati sono frutto di un confronto tra il $report_G$ sulla complessità di una libreria *open source*$_G$ presa come esempio (https://github.com/philbooth/complexity-report/blob/master/EXAMPLE.md) e i valori dichiarati in http://www.mccabe.com/pdf/McCabeIQMetrics.pdf. Tali valori vengono dichiarati momentanei (RR) e saranno da rivalutare sia considerando altre fonti, sia considerando i valori rilevati in parti del codice che il gruppo considera come riferimento.

Halstead difficulty per-function Il livello di difficoltà di una funzione misura la propensione all'errore ed è proporzionale al numero di operatori presenti.

$$D = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right)$$

Parametri utilizzati:

- Range-accettazione: [0 - 30];
- Range-ottimale: [0 - 15].

Halstead volume per-function Il volume descrive la dimensione dell'implementazione di un algoritmo e si basa sul numero di operazioni eseguite e sugli operandi di una funzione. 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 esegue troppe operazioni.

$$V = N * \log_2 n$$

Parametri utilizzati:

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

Halstead effort per-function Lo sforzo per implementare o comprendere il significato di una funzione è proporzionale al volume a al suo livello di difficoltà.

$$E = V * D$$

Parametri utilizzati:

- Range-accettazione: [0 - 400];
- Range-ottimale: [0 - 300].

3 Gestione amministrativa della revisione

3.1 Comunicazione delle anomalie

Il processo di *Software Quality Managment*$_G$ è finalizzato alla ricerca dei difetti. L'identificazione delle anomalie ne permette la correzione e informa il *Responsabile* di progetto sullo stato del prodotto. Distinguere e catalogare le anomalie è utile per discutere, durante revisioni e riunioni, su che correzioni attuare e con quale priorità. Di seguito vengono elencate le definizioni di anomalie (IEEE 610.12-90) adottate dal gruppo:

- **Error**: differenza riscontrata tra risultato di una computazione e valore teorico atteso (e.g. uscita dal range di accettazione degli indici di misurazione);
- **Fault**: un passo, un processo o un dato definito in modo erroneo (e.g. violazioni di norme tipografiche da parte di un documento). Corrisponde a quanto viene definito come *bug*$_G$;
- **Failure**: il risultato di un *fault* (e.g. incongruenza del prodotto con funzionalità indicate nell'analisi dei requisiti, incongruenza del codice con il design del prodotto);
- **Mistake**: azione umana che produce un risultato errato (e.g. anomalie nel repository).

La catalogazione delle anomalie permette l'impostazione di metriche in grado di valutarne l'andamento e in alcuni casi di predirlo, in particolare è stata scelta la metrica che conteggia il numero di *bugs per lines of code*. Il *SCR*$_G$ (software change request) utilizzato dal gruppo è il sistema di task presente in *TeamworkPM*$_G$ secondo le modalità descritte in *Norme di Progetto v1.3.1*.

3.2 Procedure di controllo per la qualità di processo

Le procedure di controllo per la qualità di processo sono finalizzati a migliorare la qualità del prodotto e/o diminuire i costi e tempi di sviluppo. Esistono due approcci principali:

- A maturità di processo: riflette le buone pratiche di management e tecniche di sviluppo. L'obiettivo primario è la qualità del prodotto e la prevedibilità dei processi;
- Agile: sviluppo iterativo senza l'overhead della documentazione e di tutti gli aspetti predeterminabili. Ha come caratteristica la responsività ai cambiamenti dei requisti cliente e uno sviluppo rapido.

Verrà utilizzato il primo approccio in quanto più adatto ad un team inesperto. Con una visione proattiva si cerca di avere maggior controllo e previsione sulle attività da svolgere. Questa viene anche indicata come *best practice*$_G$ per gruppi poco esperti.
Il processo con maggiore influenza sulla qualità del sistema non è quello di sviluppo ma è la progettazione, è qui che le capacità e le esperienze dei singoli danno un contributo decisivo.
Il miglioramento dei processi è un processo ciclico composto da tre sotto processi:

- Misurazione del processo: misura gli attributi del progetto, punta ad allineare gli obiettivi con le misurazioni effettuate. Questo forma una *baseline*$_G$ che aiuta a capire se i miglioramenti hanno avuto effetto.
- Analisi del processo: vengono identificate le problematiche ed i colli di bottiglia dei processi.
- Modifiche del processo: i cambiamenti vengono proposti in risposta alle problematiche riscontrate.

Il gruppo procederà nel seguente modo:

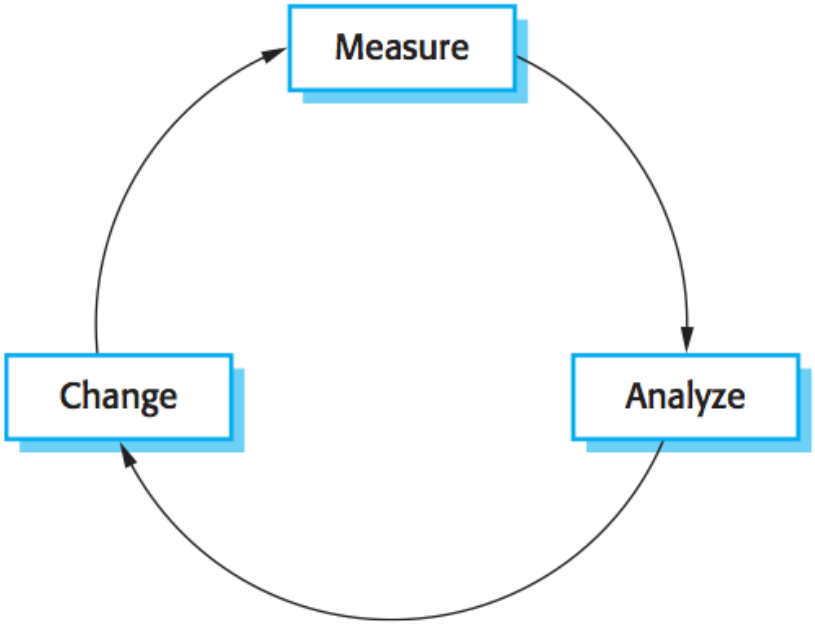

Figura 2: Il ciclo di miglioramento dei processi

- nella sezione *Dettaglio delle verifiche tramite analisi* (A.2) verranno inserite le misurazioni rilevate sulle le metriche descritte in *Misure e Metriche* (2.6);
- l'analisi viene effettuata i giorni precedenti alle consegne previste dal committente; il *Riassunto delle attività di verifica* (A.1) contiene l'analisi del processo, le relative considerazioni comprendenti le problematiche riscontrate;
- le modifiche al processo vengono attuate all'inizio del processo incrementale successivo. Queste attività sono programmate nel *Piano di Progetto v1.3.1*.

4 Pianificazione ed esecuzione del collaudo

Allo stato attuale non è possibile definire in dettaglio i collaudi in quanto non è stata affrontata la progettazione del prodotto, pertanto la pianificazione ed esecuzione dei collaudi saranno trattate nella prossima revisione.

A Resoconto delle attività di verifica

A.1 Riassunto delle attività di verifica

A.1.1 Revisione dei Requisiti

L'attività di verifica svolta dai *Verificatori* è avvenuta come determinato dal *Piano di Progetto v1.3.1* al termine della stesura di ogni documento previsto. La verifica svolta sui documenti è avvenuta seguendo le indicazioni delle *Norme di Progetto v1.3.1* e misurando le metriche indicate in 2.6.2. L'attività di *walkthrough* ha evidenziato una serie di anomalie, in questo modo è stato possibile stilare la lista di anomalie frequenti (2.5.1.2) che si potranno controllare tramite *Inspection*. Successivamente si è proceduto con le misurazioni delle metriche relative ai documenti. In questa revisione non è stato possibile valutare i processi poiché lo stato embrionale del team e impegni universitari sovrapposti non hanno permesso il rilevamento accurato di tutti i parametri necessari. Il gruppo ha in programma di colmare tale mancanza per la revisione successiva.

A.2 Dettaglio delle verifiche tramite analisi

A.2.1 Analisi

A.2.1.1 Documenti Vengono qui riportati i valori dell'indice Gulpease per ogni documento durante l'analisi e relativo esito basato sui range stabiliti in 2.6.2.1. Questa tabella verrà aggiornata in modo incrementale.

Documento	Valore indice	Esito
Analisi dei Requisiti v1.3.1	45	superato
Glossario v1.3.1	63	superato
Norme di Progetto v1.3.1	47	superato
Piano di Progetto v1.3.1	51	superato
Piano di Qualifica v1.3.1	53	superato
Studio di Fattibilità v1.3.1	43	superato

Tabella A.1: Esiti verifica documenti, Analisi

Il file *Analisi dei Requisiti v1.3.1* contiene numerose tabelle che possono facilmente falsare l'indice rilevato, il gruppo si riserva di analizzare tale problematica e aggiornare i relativi strumenti entro la prossima revisione.

B Qualità

La qualità perseguita nel presente documento si basa sugli standard ISO/IEC 15504 e ISO/IEC 9126 con l'obiettivo di approfondirne incrementalmente la copertura. Tutti i processi seguono il metodo $PDCA_G$ che prevede l'iterazione ripetuta tra i quattro stadi, incrementando di volta in volta quanto prodotto. La struttura di qualifica assicura un incremento della qualità ad ogni ciclo.

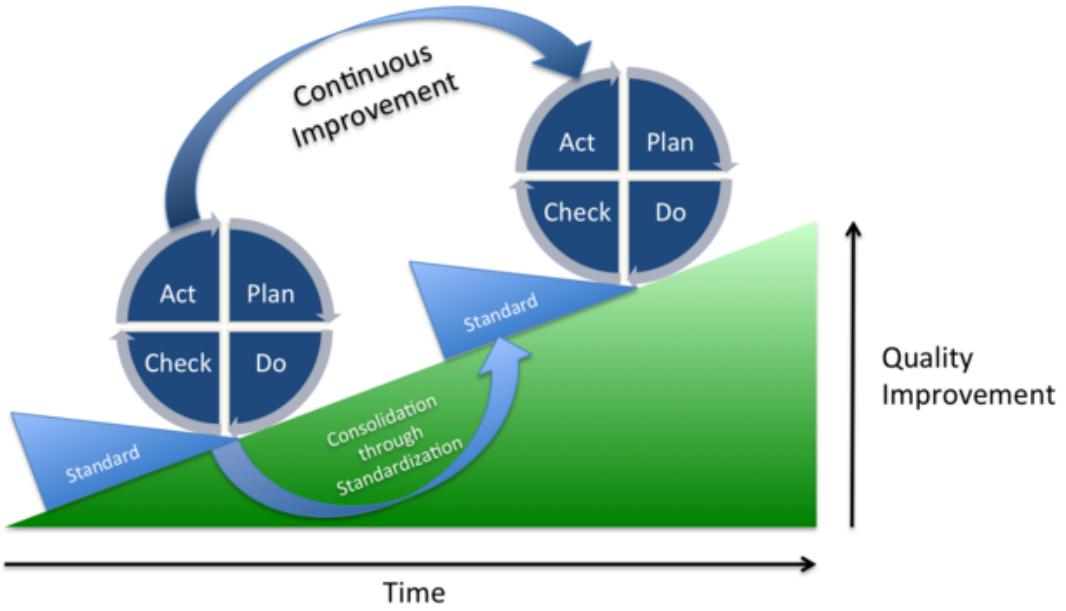

Figura 3: Continuous quality improvement with PDCA

1. **PLAN**: vengono stabiliti gli obiettivi e i processi necessari per raggiungere i risultati attesi;
2. **DO**: viene implementato il punto precedente, creando una parte del prodotto. In questo stadio vanno misurate le metriche;
3. **CHECK**: vengono studiate ed elaborate le metriche rilevate al punto precedente, un confronto con i risultati attesi sarà il riscontro se quanto operato va nella direzione giusta. Vanno considerate metriche come la *Schedule Variance* (vedi 2.6.1.1) e la completezza dei risultati attesi soddisfatti, vanno elaborati grafici e tabelle per avere una visione chiara di quanto rilevato;
4. **ACT**: vengono svolte le azioni correttive emerse dall'analisi del punto precedente e tutti i membri del gruppo vengono informati. Si potrà eseguire tramite riunioni o strumenti di messaggistica interna al gruppo. Terminato questo stadio si procederà con una nuova iterazione a partire dal punto 1.

B.1 Qualità dei processi

Definita in ISO/IEC 15504 come $SPICE_G$, specifica come la qualità è collegata alla maturazione dei processi, vengono individuati dei livelli di maturità al quale il fornitore può fare riferimento per determinare le proprie capacità organizzative. Vengono definiti:

- dei **Modelli di riferimento** su:
  - *dimensione del processo*;
  - *livelli di capacità dei processi*:
    6. ottimizzato
    5. predicibile
    4. stabilito
    3. gestito
    2. eseguito
    1. incompleto

    La capacità di un processo viene misurata tramite degli attributi che sono assimilabili alle metriche dei processi individuate in 2.6.1, in particolare la *Schedule Variance* permette di capire se un processo è incompleto o gestito; il gruppo giungerà a maturazione quando i processi diventeranno predicibili ossia quando la *Schedule Variance* subirà al più lievi oscillazioni.
- delle **Stime** che si concretizzano in una struttura per la misurazione composta da:
  - i *processi* di misurazione, indicati nel *Piano di Progetto v1.3.1*;
  - un *modello* per la misurazione identificabile in questo documento;
  - gli *strumenti* utilizzati, specificati nelle *Norme di Progetto v1.3.1*.
- le **Competenze e Qualifiche** di chi controlla; lo standard redige in modo rigoroso una serie di attività volte a formare chi opera l'attività di stesura del *Piano di Qualifica* e *Verifica*. Tali competenze sono assenti all'interno del gruppo e considerato che effettuare una formazione in linea con quanto specificato dallo standard sarebbe impossibile, tutti i membri si impegnano a studiare ed applicare al meglio quanto descritto in questo documento.

B.2 Qualità del prodotto software

Specificata in ISO/IEC 9126 si suddivide in:

- **Quality model**: classifica la qualità del software in un set di caratteristiche che verranno approfondite nel corso del progetto:
  - Functionality;
  - Reliability;
  - Usability;
  - Efficiency;
  - Maintainability;
  - Portability.

  Allo stato attuale, il gruppo non è in grado individuare un modello specifico per lo $stack_G$ tecnologico da utilizzare.
- **External metrics**: sono le metriche rilevate tramite analisi dinamica, verranno specificate con il concretizzarsi della *Specifica Tecnica*;
- **Internal metrics**: sono le metriche rilevate in analisi statica specificate in 2.6;
- **Quality in use metrics**: si tratta di metriche rilevabili allo stato di prodotto *usabile* in condizioni reali, si rimanda la definizione di tale aspetto a quando verranno trattate le considerazioni sull'usabilità del prodotto in uno scenario di utilizzo reale, questo deve avvenire non oltre la *Progettazione di Dettaglio e Codifica*.