Obelix Group
obelixswe@gmail.com

Piano di Qualifica

Versione	*v4_0_0*
Data creazione	2017-02-25
Redattori	Nicolò Rigato
	Tomas Mali
	Federica Schifano
Verificatori	Silvio Meneguzzo
	Tomas Mali
Approvazione	Nicolò Rigato
Stato	Approvato
Uso	esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Red Babel
	Gruppo Obelix

Sommario

Documento contenente le strategie adottate dal gruppo Obelix per raggiungere gli obiettivi qualitativi richiesti per il prodotto Monolith.

Diario delle revisioni

Versione	Modifica	Autore e Ruolo	Data
4.0.0	Approvazione documento	Nicolò Rigato Responsabile	2017-08-20
3.2.0	Verifica sezione 5 "Pianificazione dei test"	Tomas Mali Verificatore	2017-08-18
3.1.2	Aggiornate tabelle sezione 5 "Pianificazione dei test"	Federica Schifano Verificatore	2017-08-18
3.1.1	Aggiunta sezione 5 "Pianificazione dei test"	Federica Schifano Verificatore	2017-07-15
3.1.0	Verifica sezione 2 "Visione generale della strategia di gestione della qualità"	Silvio Meneguzzo Verificatore	2017-07-01
3.0.4	Aggiunto paragrafo "Copertura dei test" alla sottosezione 3.3.2 "Metriche per i prodotti"	Tomas Mali Verificatore	2017-06-30
3.0.3	Modifica sottosezione 3.3.2 "Metriche per i prodotti"	Nicolò Rigato Verificatore	2017-06-29
3.0.2	Modifica sottosezione 2.1.2 "Qualità di prodotto"	Nicolò Rigato Verificatore	2017-06-29
3.0.1	Rinominata sezione 2 "Strategie di Verifica" in "Visione generale della strategia di gestione della qualità"	Tomas Mali Verificatore	2017-06-28
3.0.0	Approvazione documento	Riccardo Saggese Responsabile	2017-06-19
2.1.0	Verifica sezione "Resoconto delle attività di verifica"	Emanuele Crespan Verificatore	2017-06-19
2.0.1	Modifica sezione "Resoconto delle attività di verifica"	Nicolò Rigato Verificatore	2017-06-18
2.0.0	Approvazione documento	Nicolò Rigato Responsabile	2017-04-30
1.1.0	Verifica documento	Federica Schifano Verificatore	2017-04-29
1.0.3	Aggiunta sezione "Standard di Qualità" all'appendice	Silvio Meneguzzo Verificatore	2017-04-25

Versione	Modifica	Autore e Ruolo	Data
1.0.2	Aggiunto paragrafo "Inspection"	Silvio Meneguzzo Verificatore	2017-04-23
1.0.1	Aggiunto paragrafo "Walkthrough"	Silvio Meneguzzo Verificatore	2017-04-23
1.0.0	Approvazione documento	Nicolò Rigato Responsabile	2017-03-09
0.1.0	Verifica del documento	Silvio Meneguzzo Verificatore	2017-03-09
0.0.6	Stesura sezione Resoconto delle attività di verifica	Tomas Mali Verificatore	2017-03-05
0.0.5	Stesura sezione Gestione amministrativa della revisione	Riccardo Saggese Verificatore	2017-03-04
0.0.4	Stesura sezione Strategie di Verifica	Tomas Mali Verificatore	2017-02-28
0.0.3	Stesura sezione Definizione obiettivi di qualità	Riccardo Saggese Verificatore	2017-02-27
0.0.2	Stesura sezione Introduzione	Tomas Mali Verificatore	2017-02-26
0.0.1	Creazione template	Nicolò Rigato Responsabile	2017-02-25

Indice

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Lo scopo principale di questo documento è di ottenere una buona qualità, intesa non solo come qualità di prodotto, ma anche come qualità di processo. Per raggiungere tale obiettivo è necessario un lavoro intellettuale e arduo poiché la qualità del software è diverso dagli altri prodotti manifatturieri ed è difficile misurarla in maniera oggettiva. Per rilevare e successivamente correggere le anomalie in modo efficace è necessaria un'attività costante di verifica e validazione da parte del $team_{|G|}$ Obelix.

1.2 Scopo del prodotto

Lo scopo del prodotto è quello di creare un $SDK_{|G|}$ che permetta la realizzazione di cosiddette "bolle interattive" di diversi tipi in base alle richieste di utenti (che nel nostro caso saranno gli sviluppatori). Il prodotto si completerà con la realizzazione di una demo presentabile mediante una web app con obiettivo di testare e provare il corretto funzionamento del SDK.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario v3.0.0*

1.4 Riferimenti

1.4.1 Normativi

- **Norme di progetto**: *Norme di Progetto v4.0.0*
- **Capitolato d'appalto C5**: *RedBabel, Monolith* `http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C5.pdf/`

1.4.2 Informativi

- **Piano di Progetto**: *Piano di progetto v4.0.0*
- **PDCA (Plan-Do-Check-Act):** `http://it.wikipedia.org/wiki/PCDA`
- **Standard ISO/IEC 12207:2008-IEEE Std 12207-2008**: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4475822`
- **Standard ISO/IEC 15504**: `http://en.wikipedia.org/wiki/ISO/IEC_15504/`
- **Standard ISO/IEC 9126**: `http://it.wikipedia.org/wiki/ISO/IEC_9126`
- **Indice di Gulpease**: `http://it.wikipedia.org/wiki/Indice_Gulpease`

2 Visione generale della strategia di gestione della qualità

2.1 Obiettivi di qualità

2.1.1 Qualità di processo

La qualità di prodotto non può essere garantita a meno che non sia garantita la qualità del processo che lo realizza. Nello standard ISO/IEC 15504 (SPICE) viene definita la qualità di processo e vengono fornite linee guida per la sua stima. Per maggiori informazioni sullo standard si veda l'appendice A.1. Adottando lo standard il gruppo Obelix sarà in grado di stimare il livello di maturità raggiunto dal suo lavoro.

Nelle revisioni di progettazione il committente fornisce una valutazione precisa del lavoro svolto fino a quel momento. Grazie a questo meccanismo e a valutazioni interne al gruppo è possibile migliorare il metodo di lavoro.

É fondamentale inoltre rispettare i tempi stabiliti nel piano di progetto vxxx. Per garantire una corretta stima è necessario misuare eventuali discostamenti dalla pianificazione.

Miglioramento costante Tra una revisione di progettazione e la successiva è possibile mettere a frutto quanto appreso nel periodo precedente. Viene adottato il metodo manageriale PDCA (Plan,Do,Check,Act) che permette l'uso di un approccio ingegneristico in un'attività generica. In particolare si prescrive la misurazione di grandezze ritenute rilevanti nello svolgimento delle attività al fine di averne una stima oggettiva. In seguito le informazioni così raccolte possono essere utilizzate per individuare cambiamenti desiderabili nello svolgimento delle attività. Per una descrizione più dettagliata del metodo PDCA si faccia riferimento all'appendice *B.1*

Rispetto della pianificazione Per capire se le attività di un processo sono in ritardo rispetto a quanto pianificato all'interno del documento Piano di progetto v4.0.0 viene utilizzata la metrica Schedule Variance. Si desidera che il ritardo accumulato sia minore del 5% rispetto al totale pianificato. Sarebbe invece ottimale essere esattamente in linea con quanto prevede il Piano di progetto v4.0.0, o essere addirittura in anticipo.

Rispetto del budget Viene utilizzata la metrica Budget Variance Per capire se i costi di un processo rientrano nel budget previsto dal Piano di progetto v4.0.0. L'obiettivo minimo è quello di avere dei costi che non superano il budget a disposizione per più del 10%. Sarebbe invece ottimale che i costi fossero esattamente in linea con il preventivo o addirittura minori.

2.1.2 Qualità di prodotto

Lo svolgimento del progetto prevede la realizzazione di due prodotti: i documenti e il software Monolith. I documenti devono essere precisi e corretti nella forma e nei contenuti in modo che possano essere un supporto efficace allo svolgimento del lavoro.

Per garantire la qualità del software il team Obelix cercherà di aderire al meglio allo standard di qualità ISO/IEC 9126 che evidenzia quelli che possono essere gli aspetti rilevanti nella valutazione della qualità di un software.

Per garantire la qualità del prodotto esistono i seguenti processi:

- **SQA: Software Quality Assurance** l'insieme delle attività realizzare al fine di garantire il raggiungimento degli obiettivi di qualità. É importante che tale processo sia preventivo e non correttivo
- **Verifica** assicura che l'esecuzione delle attività dei processi svolti non introduca errori nel prodotto. Durante l'intera durata del progetto verranno svolte attività di verifica sugli output dei processi, accertando che esso sia corretto, completo e rispetti regole, convenzioni e procedure
- **Validazione** la conferma oggettiva che assicura che i prodotti finali soddisfino i requisiti e le aspettative attese

Qualità dei documenti Gli obiettivi della qualità dei documenti che il team Obelix desidera raggiungere sono i seguenti:

- i documenti devono essere il più possibile comprensibili
- i documenti devono essere corretti a livello ortografico
- i documenti non devono contenere concetti errati

Leggibilità e comprensibilità Per stimare la leggibilità e la comprensibilità dei documenti viene utilizzato l'indice Gulpease. É desiderabile che i documenti abbiano costantemente un indice maggiore di 40 come soglia accettabile e 60 come ottimale. Per una descrizione dettagliata della metrica utilizzata si faccia riferimento alla metrica "Indice di Gulpease" alla sezione *2.8.2*.

Qualità del software Al fine di garantire la qualità del prodotto software, il gruppo ha deciso di adottare lo standard ISO/IEC 9126, esso classifica la qualità del software e definisce delle metriche utili alla sua misurazione. In particolare il gruppo si prefigge di garantire le seguenti qualità per il prodotto software:

- deve possedere le funzionalità descritte dai requisiti obbligatori
- deve possedere le funzionalità descritte dai requisiti desiderabili
- il codice deve risultare manutenibile e facilmente comprensibile
- deve risultare affidabile e robusto
- deve essere testato in ogni sua parte per garantirne il funzionamento

Funzionalità obbligatorie Il prodotto deve implementare tutte le funzionalità descritte dai requisiti obbligatori. Per monitorare lo stato di implementazione di tali funzionalità si rapportano i requisiti obbligatori completati con quelli ancora da completare. A tal fine viene realizzato il tracciamento dei requisiti sui componenti software[1].

[1] presente nlla Definizione di Prodotto vxxx

Funzionalità desiderabili Il prodotto deve implementare la maggior parte possibile delle funzionalità descritte dai requisiti desiderabili. Per monitorare lo stato di implementazione di tali funzionalità si rapportano i requisiti desiderabili completati con quelli ancora da completare.

Caratteristiche misurabili del software La qualità del software si misura attraverso l'adozione di metriche scelte perchè ritenute rilevanti per descrivere le proprietà critiche del software. Oltre alle proprietà intrinseche del prodotto viene anche misurata la qualità dei test, ovvero quanto i test effettivamente verifichino l'assenza di anomalie nel software.

2.2 Organizzazione temporale

2.2.1 Analisi

In questo periodo verrà verificata la corrispondenza tra i casi d'uso e i requisiti. Verrà inoltre controllato il rispetto dei processi e della documentazione prodotta.

2.2.2 Progettazione Architetturale

In questo periodo avviene la verifica dei processi relativi all'analisi e ai nuovi documenti di progettazione. Inoltre si verifica che i test siano adeguatamente pianificati come descritti ed eseguiti nel documento *Norme di Progetto vxxx*

2.2.3 Progettazione di Dettaglio e Codifica

In questo periodo avviene la verifica dei processi relativi alla progettazione insieme alla verifica delle attività di codifica tramite tecniche di analisi statica e dinamica.

2.3 Tecniche di analisi

2.3.1 Analisi statica

L'analisi statica è una tecnica di analisi che si applica sia alla documentazione che al codice e permette di individuare errori ed anomalie. Essa si può svolgere in due modi distinti che sono Walkthrough ed Inspection.

Walkthrough É una tecnica che viene utilizzata soprattutto nelle prime attività del progetto quando ancora non è presente una adeguata esperienza 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 nel lavoro futuro. Walkthrough 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 L'inspection consiste nell'analisi mirata di alcune parti dei documenti o del codice ritenute maggior fonte di errore. Deve essere seguita una lista di controllo per svolgere efficacemente questa attività; tale lista deve essere redatta anticipatamente ed è sostanzialmente frutto dell'esperienza maturata dai membri del team con tecniche di walkthrough. L'inspection è dunque più rapida del walkthrough, in quanto il prodotto viene analizzato solo in alcune sue parti e con una lista di controllo ben precisa.

2.3.2 Analisi dinamica

L'analisi dinamica si applica solamente al prodotto software e viene svolta durante l'esecuzione del codice mediante l'uso di test progettati per rilevare la presenza di difetti. L'obiettivo del test del software è infatti, quello di realizzare un prodotto il più possibile esente da errori. Il principale ostacolo alla fase di test è sintetizzato nella tesi di Dijkstra, la quale afferma che il test può indicare la presenza di errori, ma non ne può garantire l'assenza. Affinchè tale attività sia utile e generi risultati attendibili è necessario che i test effettuati siano ripetibili: dato un certo input deve essere prodotto sempre uno stesso output in uno specifico ambiente. Di conseguenza, i tre elementi fondamentali di un test sono:

- **Ambiente**: sistema hardware e software sui quali è stato pianificato l'utilizzo del prodotto software sviluppato. Su di essi deve essere specificato uno stato iniziale dal quale poter eseguire il test
- **Specifica**: definizione di input e output
- **Procedure**: definizione di come devono essere svolti i test, in che ordine devono essere eseguiti e come devono essere analizzati i risultati

3 Strategia di gestione della qualità nel dettaglio

Il *Piano di progetto v4.0.0*fissa una serie di scadenze improrogabili e quindi risulta necessario definire con chiarezza una strategia di qualifica efficace. Gli incrementi della documentazione o del codice possono essere di natura programmata, quindi prefissati nel calendario, oppure possono insorgere inaspettatamente. In questo caso sarà necessario programmare le dovute modifiche. La qualità di ogni incremento è basata sul rispetto delle Norme di Progetto vxxx in quanto esse derivano da considerazioni di qualità. Parte significativa del lavoro verrà svolto con l'aiuto di automatismi che segnaleranno le problematiche rilevate in modo da permettere una rapida correzione. L'utilizzo di software apposito permette di eseguire controlli mirati minimizzando l'utilizzo di risorse umane. L'implementazione di tali controlli viene descritta nelle *Norme di Progetto v4.0.0*.

3.1 Responsabilità

La responsabilità delle verifiche è attribuita al *Responsabile di progetto* e ai *Verificatori* . All'interno del *Piano di Progetto vxxx* sono definiti i compiti e le modalità di attuazione.

3.2 Risorse

Il funzionamento del processo di verifica è garantito grazie al consumo di risorse, distinguibili nelle categorie a seguire.

3.2.1 Risorse umane

Hanno maggiore responsabilità per l'attività di verifica e validazione il *Responsabile del progetto* e il *Verificatore*. Per una dettagliata descrizione dei ruoli e delle loro responsabilità bisogna fare riferimento alle *Norme di Progetto vxxx*. Per una dettagliata descrizione dell'impiego delle risorse umane nell'arco del progetto bisogna riferimento al Piano di progetto vxxx.

3.2.2 Risorse tecnologiche

Per risorse tecnologiche si intendono tutti gli strumenti software e hardware che il gruppo Obelix intende utilizzare per attuare le attività di verifica su processi e prodotti. Per una dettagliata e accurata descrizione di tali strumenti si faccia riferimento nelle Norme di Progetto v4.0.0.

3.3 Misure e metriche

Con lo scopo di poter monitorare in modo consapevole l'andamento dei processi e la qualità del prodotto si utilizzano delle metriche per rendere misurabili e valutabili in modo oggettivo alcune caratteristiche di documenti, processi e software.

3.3.1 Metriche per i processi

Le metriche per i processi hanno lo scopo di monitorare e rendere prevedibile l'andamento delle variabili di maggior criticità del progetto: tempo e costo. Le metriche sono utilizzate in modo consultivo per consentire 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*.

Schedule Variance Permette di calcolare le tempistiche rispetto l'organizzazione delle attività pianificate alla data corrente. É un indicatore di efficacia soprattutto a beneficio del cliente.

$$SV = BCWP - BCWS$$

Dove:

- **BCWP**: indica il valore delle attività realizzate alla data corrente
- **BCWS**: indica il costo pianificato per realizzare le attività di progetto alla data corrente

Quindi con:

- $SV > 0$: il lavoro prodotto è in anticipo rispetto quanto pianificato
- $SV < 0$: il lavoro è in ritardo
- $SV = 0$: il lavoro è in linea con quanto stabilito

Budget Variance Permette di calcolare i costi rispetto alla data corrente. É un indicatore che ha un valore unicamente contabile e finanziario.

$$BV = BCWS - ACWP$$

Dove:

- **BCWS**: indica il costo pianificato per realizzare le attività di progetto alla data corrente
- **ACWP**: indica il costo effettivamente sostenuto per realizzare le attività di progetto alla data corrente

Quindi:

- $BV > 0$: il budget speso è minore di quanto pianificato
- $BV < 0$: il budget speso è maggiore di quanto pianificato
- $BV = 0$: il budget speso è in linea con quanto stabilito

Produttività

Produttività di documentazione Indica la produttività media nel redigere i documenti.

$$\text{Produttività di documentazione} = \frac{\text{Parole}}{\text{Ore persona}}$$

Dove:

- **Parole**: indica il numero di parole presente nei documenti
- **Ore persona**: indica il numero di ore produttive impiegate per realizzare tali documenti

Parametri utilizzati

- Range-ottimale:[≥ 100]

Produttività di codifica Indica la produttività media nelle attività di codifica.

$$\text{Produttività di codifica} = \frac{\text{LOCs}}{\text{Ore persona}}$$

Dove:

- **LOCs**: indica il numero di linee di codice prodotte (Lines Of Code)
- **Ore persona**: indica il numero di ore produttive impiegate per produrre tale codice

Parametri utilizzati

- Range-ottimale:[≥ 100]

Copertura del test Indica la percentuale di casi coperti dai test eseguiti.

$$\text{Copertura del test} = \frac{\text{Numero di funzioni testate} \times 100}{\text{Numero totale di funzioni disponibili}}$$

Parametri utilizzati

- Range-accettazione: $[70 - 100]$
- Range-ottimale: $[80 - 100]$

3.3.2 Metriche per i prodotti

Metriche per i documenti I documenti sono di qualità se sono leggibili. La leggibilità è misurabile tramite un indice calcolabile sulle caratteristiche del testo.

Indice Gulpease L'indice Gulpease misura la leggibilità di un testo in Italiano utilizzando variabili linguistiche come la lunghezza delle parole e delle frasi.

$$G = 89 + \frac{300 \times \text{numero delle frasi} - 10 \times \text{numero delle lettere}}{\text{numero delle parole}}$$

Il valore risultante è un numero compreso tra 0 e 100, dove 100 indica la leggibilità massima e 0 la leggibilità minima. I documenti sono da considerare secondo le seguenti fasce:

- $G < 80$ sono considerati difficili per chi ha la sola licenza elementare
- $G < 60$ sono considerati difficili per chi ha la sola licenza media
- $G < 40$ sono considerati difficili per chi ha un diploma di scuola superiore

Parametri utilizzati

- Range di accettazione [40-90]
- Range ottimale [50-90]

Metriche per software Di seguito vengono elencate le metriche per il software prodotto e le relative caratteristiche di qualità che intendono valutare.

Metriche scelte	Caratteristiche di qualità
Complessità ciclomatica	Manutenibilità
Numero di metodi per package	Manutenibilità
Variabili non utilizzate e non definite	Manutenibilità
Numero di parametri per metodo	Manutenibilità
Metriche di Halstead	Manutenibilità
Maintainability index	Manutenibilità
Statement coverage	Affidabilità
Branch coverage	Affidabilità
Copertura dei test	Affidabilità

Tabella 2: Scopo delle metriche adottate

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. Tale indice può essere applicato indistintamente a singole funzioni, moduli, metodi o $package_{|G|}$ di un programma. Si vuole utilizzare tale metrica per limitare la complessità durante le attività di sviluppo del prodotto software. Può rivelarsi utile durante il testing per determinare il numero di test necessari, essendo l'indice di complessità un limite superiore al numero di test necessari per raggiungere la copertura completa.

Parametri utilizzati:

- Range-accettazione: $[0 - 25]$
- Range-ottimale: $[0 - 10]$

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 scomposizione del package.

Parametri utilizzati:

- Range-accettazione: $[3 - 15]$
- Range-ottimale: $[3 - 7]$

Variabili non utilizzate e non definite La presenza di variabili non utilizzate viene considerata pollution, pertanto non viene tollerata. Tali occorrenze vengono rilevate analizzando l'Abstract syntax tree eseguendo un confronto 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:

- Range-accettazione: $[0 - 0]$
- Range-ottimale: $[0 - 0]$

Numero parametri per metodo Un numero elevato di parametri per un metodo potrebbe evidenziare un metodo troppo complesso. Non c'è una regola forte per il numero di parametri possibili in un metodo, ma citando Robert Martin in Clean Code[2] :

[2]Robert Martin, Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall (2008)

> The ideal number of arguments for a function is zero (niladic). Next comes one (monadic), followed closely by two (dyadic). Three arguments (triadic) should be avoided where possible. More than three (polyadic) requires very special justification – and then shouldn't be used anyway.

Vengono quindi seguite le linee guida dei seguenti parametri:

- Range-accettazione: $[0 - 8]$
- Range-ottimale: $[0 - 4]$

Halstead La metrica di $Halstead_{|G|}$ oltre ad essere un indice di complessità, permette di identificare le proprietà misurabili del software e le relative relazioni. Si basa sulla necessità che una metrica sia indipendente dalla specifica piattaforma e dal linguaggio di programmazione. Sono identificati i seguenti dati all'interno di un problema astratto:

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

Da cui si ottiene:

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

Data la scarsa disponibilità in rete di valori di riferimento per il Javascript, i range saranno da valutare in un momento successivo alla RR.

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

$$D = (\frac{n_1}{2}) \times (\frac{N_2}{n_2})$$

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.

$$V = N \times \log_2 n$$

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 e al suo livello di difficoltà.

$$E = V \times D$$

Parametri utilizzati:

- Range-accettazione: $[0 - 400]$
- Range-ottimale: $[0 - 300]$

Maintainability index Questa metrica[3] è una scala logaritmica da $-\infty$ a 171, calcolata sulla base delle linee di codice logiche, della complessità ciclomatica e dall'indice Halstead effort. Un valore alto indica una maggiore manutenibilità.

Parametri utilizzati:

- Range-accettazione: $[> 70]$
- Range-ottimale: $[> 90]$

Statement Coverage Permette di calcolare quante linee di codice di ciascun modulo delle unità sono eseguite almeno una volta nell'esecuzione dei test. Tale metrica è espressa in percentuale.

Parametri utilizzati:

- Range-accettazione: $[70 - 100]$
- Range-ottimale: $[85 - 100]$

Branch Coverage Permette di calcolare quanti rami della logica di flusso sono attraversati almeno una volta durante l'esecuzione dei test. Tale metrica è espressa in percentuale.

Parametri utilizzati:

- Range-accettazione: $[70 - 100]$
- Range-ottimale: $[85 - 100]$

[3] Definita nel 1991 da Paul Oman e Jack Hagemeister alla University of Idaho.

Copertura dei test Questa metrica esamina la percentuale di successo dei test ricavati dai requisiti e dalle relative funzionalità che il software dovrà ottenere. Indica infatti la percentuale dei test eseguiti con successo.

$$\text{Copertura dei test} = \frac{\text{Numero test superati} \times 100}{\text{Numero test pianificati}}$$

4 Gestione amministrativa della revisione

4.1 Comunicazione delle anomalie

Identificare le anomalie permette la correzione dei difetti ricercati dal processo di Software Quality Management e informa il *Responsabile* di progetto sullo stato del prodotto. Analizzare e catalogare le anomalie è utile per discutere, durante revisioni e riunioni, su quali modifiche e correzioni applicare e con quale priorità. Di seguito è presente la lista delle definizioni di anomalie (IEEE 610.12-90) adottate dal gruppo:

- **Error**: differenza riscontrata tra il risultato di una computazione e il valore teorico atteso
- **Fault**: un passo, un processo o un dato definito in modo erroneo. Corrisponde a quanto viene comunemente definito come bug
- **Failure**: il risultato di un fault
- **Mistake**: azione umana che produce un risultato errato

4.2 Procedure di controllo per la qualità di processo

Le procedure di controllo per la qualità di processo hanno il fine di migliorare la qualità del processo e 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.

Il team adotterà il primo approccio, essendo più adatto ad un gruppo inesperto. Con una visione proattiva si cerca di avere maggior controllo e previsione sulle attività da svolgere. Questa viene anche indicata come best practice per gruppi poco esperti. Il processo con maggiore influenza sulla qualità del sistema non è quello di sviluppo ma quello di 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 team procederà nel seguente modo:

- Nell'appendice del seguente documento verranno inserite le misurazioni rilevate sulle metriche descritte nella sezione *2.8.1*
- Le modifiche al processo vengono attuate all'inizio del processo incrementale successivo. Queste attività sono programmate nel Piano di progetto v4.0.0

5 Pianificazione dei Test

Vengono riportati e descritti in questa sezione i test da implementare per garantire che il software prodotto rispecchi le funzionalità a fronte dei risultati attesi. Tutte le attività di testing prodotte devono poter essere ripetibili e devono essere deterministiche, al fine di poter fornire delle informazioni utili a intraprendere azioni di correzione nel caso in cui i risultati ottenuti siano diversi da quelli attesi. Ogni test è identificato da un codice univoco la cui sintassi viene descritta nel documento *Norme di Progetto v4.0.0*

5.1 Test di Unità

Tale tipologia di test serve per testare il corretto funzionamento delle singole unità, ossia delle più piccole componenti software singolarmente verificabili. Solitamente l'unità trova corrispondenza in un metodo di una classe tra quelli descritti nel documento *Definizione di prodotto v2.0.0*. Per ogni test viene specificato il proprio codice univoco, la descrizione, lo stato di implementazione attuale e il risultato (Superato/Non Superato).

Test	Descrizione	Stato
TU1	Verificare che la classe CheckBoxList sia instanziabile.	Implementato
TU2	Verificare che la classe CheckButton sia instanziabile.	Implementato
TU3	Verificare che la classe ComboBox sia instanziabile.	Implementato
TU4	Verificare che la classe Image sia instanziabile.	Implementato
TU5	Verificare che la classe ImageButton sia instanziabile.	Implementato
TU6	Verificare che la classe LineEdit sia instanziabile.	Implementato
TU7	Verificare che la classe LineEditComboBox sia instanziabile.	Implementato
TU8	Verificare che la classe LineEditPushButton sia instanziabile.	Implementato

Test	Descrizione	Stato
TU9	Verificare che la classe PushButton sia instanziabile.	Implementato
TU10	Verificare che la classe RadioButtonGroup sia instanziabile.	Implementato
TU11	Verificare che la classe TextAreaButton sia instanziabile.	Implementato
TU12	Verificare che la classe TextAreaComboBox sia instanziabile.	Implementato
TU13	Verificare che un componente LineEdit passi al componente genitore che lo contiene l'input testuale nel modo corretto.	Implementato
TU14	Verificare che in un componente PushButton venga impostato il nome nel modo corretto.	Implementato
TU15	Verificare che al click di un componente PushButton vengano eseguiti gli eventi associati alla pressione.	Implementato
TU16	Verificare che in un componente ComboBox vengano impostate le opzioni (mandate dal componente genitore) nel modo corretto.	Implementato
TU17	Verificare che un componente ComboBox passi al componente genitore che lo contiene l'opzione selezionata nel modo corretto.	Implementato
TU18	Verificare che in un componente CheckButton venga cambiato in modo corretto lo stato da "cliccato" a "non cliccato" e viceversa.	Implementato
TU19	Verificare che ad un componente CheckButton venga associata la label scelta nel modo corretto.	Implementato
TU20	Verificare che in un componente Image l'immagine venga aggiunta in modo corretto.	Implementato
TU21	Verificare che in un componente Image la didascalia venga aggiunta in modo corretto.	Implementato
TU22	Verificare che in un componente Image le dimensioni (larghezza e altezza) vengano impostate in modo corretto.	Implementato
TU23	Verificare che in un componente ImageButton l'immagine venga aggiunta in modo corretto.	Implementato
TU24	Verificare che in un componente ImageButton la didascalia venga aggiunta in modo corretto.	Implementato
TU25	Verificare che in un componente ImageButton le dimensioni (larghezza e altezza) vengano impostate in modo corretto.	Implementato
TU26	Verificare che al click di un componente ImageButton vengano eseguiti gli eventi associati alla pressione.	Implementato
TU27	Verificare che al click del PushButton associato al componente LineEditPushButton venga passato al componente genitore l'input testuale del LineEdit nel modo corretto.	Implementato

Test	Descrizione	Stato
TU28	Verificare che in un componente TextAreaButton le dimensioni (larghezza e altezza) vengano impostate nel modo corretto.	Implementato
TU29	Verificare che al click del PushButton associato al componente TextAreaButton venga passato al componente genitore l'input testuale della textarea nel modo corretto.	Implementato
TU30	Verificare che in un componente CheckBoxList il numero di opzioni presenti sia corretto.	Implementato
TU31	Verificare che in un componente RadioButtonGroup il numero di opzioni presenti sia corretto.	Implementato
TU32	Verificare che un componente RadioButtonGroup passi al componente genitore che lo contiene l'opzione selezionata nel modo corretto.	Implementato
TU33	Verificare che la classe Check sia instanziabile.	Implementato
TU34	Verificare che la classe BubbleDatabase sia instanziabile.	Implementato
TU35	Verificare che la classe BubbleMenu sia instanziabile.	Implementato
TU36	Verificare che la classe ConfigArea sia instanziabile.	Implementato
TU37	Verificare che la classe SentBubbles sia instanziabile.	Implementato
TU38	Verificare che la classe Sidearea1 sia instanziabile.	Implementato
TU39	Verificare che la classe ReceivedBubble sia instanziabile.	Implementato
TU40	Verificare che la classe Sidearea2 sia instanziabile.	Implementato
TU41	Verificare che la classe ContainedElement sia instanziabile.	Implementato
TU42	Verificare che la classe HorizontalLayout sia instanziabile.	Implementato
TU43	Verificare che la classe VerticalLayout sia instanziabile.	Implementato
TU44	Verificare che la classe AbsBubble NON sia instanziabile.	Implementato
TU45	Verificare che la classe AbsBubbleConfig NON sia instanziabile.	Implementato
TU46	Verificare che la classe AbsButton NON sia instanziabile.	Implementato
TU47	Verificare che la classe BubbleCreator NON sia instanziabile.	Implementato
TU48	Verificare che ogni componente grafico abbia l'attributo key.	Implementato

Test	Descrizione	Stato
TU49	Verificare che il metodo validate della classe Check effettui la validazione dell'oggetto in modo corretto.	Implementato
TU50	Verificare che il metodo insertBubble effettui l'inserimento dei dati nel database correttamente.	Implementato
TU51	Verificare che il metodo updateBubble effettui la modifica dei dati nel database correttamente.	Implementato
TU52	Verificare che il metodo removeBubble effettui la rimozione dei dati dal database correttamente.	Implementato

Tabella 4: Test di unità

5.2 Test di Integrazione

Tale tipologia di test serve per verificare che le varie componenti del sistema software interagiscano tra loro nel modo atteso. Per ogni test viene specificato il proprio codice univoco, la descrizione, il componente cui si fa riferimento, lo stato di implementazione attuale e il risultato (Superato/Non Superato).

Test	Descrizione	Stato
TI1	**Monolith::UI::UI-Layouts**: Verificare che VerticalLayout disponga i componenti uno sotto l'altro.	Implementato
TI2	**Monolith::UI::UI-Layouts**: Verificare che HorizontalLayout disponga i componenti uno accanto all'altro.	Implementato
TI3	**Monolith::UI::Bubbles**: Verificare che BubbleDiscriminator assegni correttamente al tipo di BubbleCreator il nome scelto.	Implementato
TI4	**Monolith::UI::Bubbles**: Verificare che BubbleDiscriminator selezioni il sottotipo di BubbleCreator corretto per la creazione del menu di configurazione del tipo di bolla selezionato.	Implementato
TI5	**Monolith::UI::Bubbles**: Verificare che BubbleDiscriminator selezioni il sottotipo di BubbleCreator corretto per la creazione di un'istanza del tipo di bolla inviata.	Implementato
TI6	**Monolith::UI::Bubbles**: Verificare che BubbleDiscriminator selezioni il sottotipo di BubbleCreator corretto per la creazione di un'istanza del tipo di bolla ricevuta.	Implementato
TI7	**Monolith::UI::Bubbles**: Verificare che BubbleDiscriminator selezioni il sottotipo di BubbleCreator corretto per la creazione del pulsante relativo al tipo di bolla selezionato.	Implementato

Test	Descrizione	Stato
TI8	**Monolith::UI::SideAreas:: SideArea1_pkg**: Verificare che SentBubbles invochi correttamente useDoMakeBubbleSender di BubbleDiscriminator per la creazione dello storico delle bolle inviate.	Implementato
TI9	**Monolith::UI::SideAreas:: SideArea1_pkg**: Verificare che BubbleMenu invochi correttamente useDoMakeButton di BubbleDiscriminator per la creazione del menu contenente i pulsanti relativi ai diversi tipi di bolle.	Implementato
TI10	**Monolith::UI::SideAreas:: SideArea1_pkg**: Verificare che SideArea1 contenga BubbleMenu, ConfigArea e SentBubbles.	Implementato
TI11	**Monolith::UI::SideAreas:: SideArea2_pkg**: Verificare che ReceivedBubble invochi correttamente useDoMakeBubbleReceiver di BubbleDiscriminator per la creazione dello storico delle bolle ricevute.	Implementato
TI12	**Monolith::UI::SideAreas:: SideArea2_pkg**: Verificare che SideArea2 contenga ReceivedBubble.	Implementato
TI13	**CurrencyBubble**: Verificare che CurrencyCreator crei la corretta istanza della bolla di tipo convertitore inviata.	Implementato
TI14	**CurrencyBubble**: Verificare che CurrencyCreator crei la corretta istanza della bolla di tipo convertitore ricevuta.	Implementato
TI15	**CurrencyBubble**: Verificare che CurrencyCreator crei correttamente il menu di configurazione per la bolla di tipo convertitore.	Implementato
TI16	**CurrencyBubble**: Verificare che CurrencyCreator crei correttamente il pulsante relativo al tipo di bolla convertitore.	Implementato
TI17	**ListBubble**: Verificare che ListCreator crei la corretta istanza della bolla di tipo lista inviata.	Implementato
TI18	**ListBubble**: Verificare che ListCreator crei la corretta istanza della bolla di tipo lista ricevuta.	Implementato
TI19	**ListBubble**: Verificare che ListCreator crei correttamente il menu di configurazione per la bolla di tipo lista.	Implementato

Test	Descrizione	Stato
TI20	**ListBubble**: Verificare che ListCreator crei correttamente il pulsante relativo al tipo di bolla lista.	Implementato
TI21	**PollBubble**: Verificare che PollCreator crei la corretta istanza della bolla di tipo sondaggio inviata.	Implementato
TI22	**PollBubble**: Verificare che PollCreator crei la corretta istanza della bolla di tipo sondaggio ricevuta.	Implementato
TI23	**PollBubble**: Verificare che PollCreator crei correttamente il menu di configurazione per la bolla di tipo sondaggio.	Implementato
TI24	**PollBubble**: Verificare che PollCreator crei correttamente il pulsante relativo al tipo di bolla sondaggio.	Implementato
TI25	**RandBubble**: Verificare che RandCreator crei la corretta istanza della bolla di tipo dado inviata.	Implementato
TI26	**RandBubble**: Verificare che RandCreator crei la corretta istanza della bolla di tipo dado ricevuta.	Implementato
TI27	**RandBubble**: Verificare che RandCreator crei correttamente il menu di configurazione per la bolla di tipo dado.	Implementato
TI28	**RandBubble**: Verificare che RandCreator crei correttamente il pulsante relativo al tipo di bolla dado.	Implementato
TI29	**Monolith::Status::checks**: Verificare che registerCheck assegni correttamente il nome al controllo (Check) selezionato.	Implementato
TI30	**Monolith::Status::checks**: Verificare che execCheck effettui il controllo (Check) sull'oggetto passato in modo corretto.	Implementato
TI31	**Monolith::Status::checks**: Verificare che execCheck effettui correttamente il controllo (Check) anche in presenza di più tipologie di controllo.	Implementato
TI32	**Monolith::Database:: databaseInitialization**: Verificare che l'inizializzazione del database avvenga in modo corretto.	Implementato
TI33	**Monolith::Database**: Verificare che l'inserimento dei dati nel database avvenga in modo corretto.	Implementato
TI34	**Monolith::Database**: Verificare che la modifica dei dati nel database avvenga in modo corretto.	Implementato

Test	Descrizione	Stato
TI35	**Monolith::Database**: Verificare che la rimozione dei dati dal database avvenga in modo corretto.	Implementato

Tabella 6: Test di integrazione

5.3 Test di Sistema

Tale tipologia di test serve per verificare che il comportamento dinamico complessivo dell'intero sistema sia conforme ai requisiti definiti nel documento *Analisi dei Requisiti v3.0.0*. Per ogni test viene specificato il proprio codice univoco, la descrizione,lo stato di implementazione attuale e il risultato (Superato/Non Superato).

Test	Descrizione	Stato
TSObFu10	Verificare che l'utente possa accedere alle bolle dall'interfaccia di Rocket.Chat tramite l'utilizzo di due pulsanti aggiunti alla tabbar laterale.	Implementato
TSObFu10.2	Verificare che l'utente possa utilizzare la SideArea1 per creare, inviare, visualizzare e modificare le bolle inviate.	Implementato
TSObFu10.2.5.1	Verificare che l'utente visualizzi un messaggio appropriato nel caso non ci siano bolle nello storico in uscita.	Implementato
TSObFu10.2.7	Verificare che l'utente visualizzi un messaggio di errore nel caso in cui la configurazione della bolla non fosse andata a buon fine.	Non Implementato
TSObFu10.3	Verificare che l'utente possa utilizzare la SideArea2 per visualizzare e interagire con le bolle ricevute.	Implementato
TSObFu10.3.1.1	Verificare che l'utente visualizzi un messaggio appropriato nel caso in cui non ci siano bolle nello storico in ingresso.	Implementato
TSObFu11.2.1.1	Verificare che lo sviluppatore possa aggiungere ad una bolla un componente grafico tra quelli presenti nel SDK.	Implementato
TSObFu11.2.1.2	Verificare che l'utente possa aggiungere ad una bolla un layout tra quelli presenti nel SDK.	Implementato
TSObFu11.2.1.3	Verificare che lo sviluppatore possa modificare le proprietà di un componente grafico.	Implementato

Test	Descrizione	Stato
TSObFu11.2.1.6	Verificare che lo sviluppatore possa inserire il menu di configurazione per la bolla.	Implementato
TSObFu11.2.3	Verificare che lo sviluppatore possa gestire la persistenza dei dati della bolla.	Implementato
TSObFu11.2.3.4	Verificare che lo sviluppatore possa specificare i parametri di accettazione dell'input.	Non Implementato
TSDeFu24	Verificare che l'utente riceva notifiche per eventi di rilievo.	Non Implementato
TSObFu01-cv	Verificare che l'utente possa utilizzare la bolla di tipo convertitore di valuta.	Implementato
TSObFu01.4-cv	Verificare che il mittente e il ricevente possano visualizzare gli importi convertiti.	Implementato
TSObFu01.7-cv	Verificare che l'utente non possa inviare la bolla nel caso in cui non siano state selezionate le valute in ingresso e in uscita o l'importo da convertire.	Implementato
TSOpFu02-cv	Verificare che l'utente possa convertire importi da valori di pacchetti azionari.	Non Implementato
TSObFu01-dd	Verificare che l'utente possa utilizzare la bolla di tipo estrazione di numero casuale.	Implementato
TSObFu01.2-dd	Verificare che il mittente e il ricevente visualizzino il numero casuale generato.	Implementato
TSOpFu01.2.1-dd	Verificare che il mittente e il ricevente visualizzino il numero casuale sotto forma di immagine.	Non Implementato
TSObFu01.6-dd	Verificare che l'utente non possa inviare la bolla senza aver impostato il range.	Implementato
TSObFu01-ls	Verificare che l'utente possa utilizzare la bolla di tipo lista.	Implementato
TSDeFu08-ls	Verificare che l'utente riceva una notifica al completamento di una lista.	Non Implementato
TSOpFu01-mt	Verificare che l'utente possa utilizzare la bolla di tipo meteo.	Non Implementato
TSObFu01-sd	Verificare che l'utente possa utilizzare la bolla di tipo sondaggio.	Implementato
TSObFu04-sd	Verificare che il mittente e il ricevente visualizzino i risultati del sondaggio.	Implementato

Test	Descrizione	Stato
TSObFu07-sd	Verificare che l'utente non possa inviare la bolla nel caso in cui non siano stati inseriti il titolo e almeno due opzioni.	Implementato
TSDeFu08-sd	Verificare che l'utente riceva una notifica quando tutti i partecipanti hanno espresso il voto.	Non Implementato
TSOpFu01-tr	Verificare che l'utente possa utilizzare la bolla di tipo traduttore.	Non Implementato

Tabella 8: Test di sistema

5.4 Test di Validazione

Tale tipologia di test viene utilizzata durante l'attività di collaudo del prodotto finale, per accertare che il prodotto sia conforme alle attese del committente. Per ogni test viene specificato il proprio codice univoco, la descrizione, il requisito cui si fa riferimento, lo stato di implementazione attuale e il risultato (Superato/Non Superato).

Test	Descrizione	Stato
TVObDi01	Verificare che l'SDK possa essere installata correttamente in un'istanza di Rocket.Chat.	Implementato
TVObDi02	Verificare che l'SDK sia utilizzabile da alcune bolle predefinite.	Implementato
TVObDi03	Verificare che sia incluso nell'SDK un set di API per lo sviluppo di bolle.	Implementato
TVObDi09	Verificare che la demo sia installata su Heroku.	Non Implementato
TVObFu10	Verificare che l'utente possa accedere alle bolle dall'interfaccia di Rocket.Chat.	Implementato
TVObFu10.2.1	Verificare che l'utente possa visualizzare i tipi di bolla disponibili.	Implementato
TVObFu10.2.2	Verificare che l'utente possa selezionare il tipo di bolla da inviare.	Implementato
TVObFu10.2.3	Verificare che l'utente possa configurare la bolla tramite l'apposito menu.	Implementato
TVObFu10.2.4	Verificare che l'utente possa inviare la bolla che ha configurato.	Implementato
TVObFu10.2.5	Verificare che l'utente possa visualizzare lo storico delle bolle inviate.	Implementato

Test	Descrizione	Stato
TVObFu10.3.1	Verificare che l'utente possa visualizzare lo storico delle bolle ricevute.	Implementato
TVObFu11.1	Verificare che lo sviluppatore possa creare le proprie bolle a partire dalla bolla vuota.	Implementato
TVObFu11.2.1.1	Verificare che lo sviluppatore possa inserire un componente grafico.	Implementato
TVObFu11.2.1.2	Verificare che lo sviluppatore possa inserire nella bolla un contenitore.	Implementato
TVObFu11.2.1.2.1	Verificare che lo sviluppatore possa inserire nella bolla uno dei layout presenti nel SDK.	Implementato
TVObFu11.2.2	Verificare che sia possibile distinguere il mittente di una bolla da tutti gli altri utenti.	Implementato
TVObDi13	Verificare che Monolith sia supportato dai browser in cui è eseguibile Rochet.Chat.	Non Implementato
TVObFu21	Verificare che ogni client possa ricevere solo i dati che riguardano tutte le bolle presenti in rooms in cui sia presente anche l'utente.	Implementato
TVDeFu24	Verificare che l'utente riceva notifiche per eventi di rilievo.	Non Implementato
TVObFu01-ls	Verificare che l'utente possa creare una lista con Checklist e inviarla all'interno di un canale Rocket.Chat.	Implementato

Tabella 10: Test di validazione

A Standard di qualità

Vengono riportati gli obbiettivi di qualità che il team Obelix dovrà raggiungere durante lo svolgimento del progetto. Il team adotta standard, modelli e metriche per verificare che ciascun obiettivo sia stato raggiunto.

A.1 Standard ISO/IEC 15504

La qualità di processo è definita dallo standard ISO/IEC 15504 chiamato anche SPICE. La qualità viene descritta in base a 6 possibili livelli di maturità ciascuno dei quali

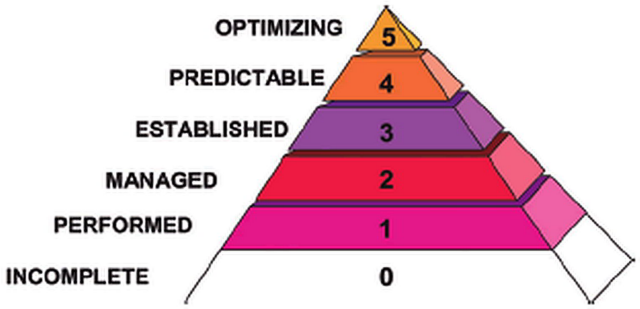

Figura 1: Standar ISO/IEC 15504

- **Livello 0 - Incomplete process**
  Il processo non è implementato o non riesce a raggiungere i suoi obiettivi
- **Livello 1 - Performed process**
  Il processo viene messo in atto e raggiunge i suoi scopi
- **Livello 2 - Managed process**
  Il processo viene eseguito sulla base di obiettivi ben definiti
- **Livello 3 - Established process**
  Il processo viene eseguito in base ai principi dell'ingegneria del software
- **Livello 4 - Predictable process**
  Il processo è attuato all'interno di limiti ben definiti
- **Livello 5 - Optimizing process**
  Il processo è predicibile ed è in grado di adattarsi per raggiungere obiettivi specifici e rilevanti

Per valutare il livello di maturità di un processo si utilizzano le metriche descritte nella sezione *2.8.1*, in particolare lo schedule variance aiuta a capire quando un processo raggiunge uno stato accettbile se il valore subisce al massimo lievi oscillazioni da quanto previsto.

A.2 Ciclo di Deming - PDCA

Il ciclo di Deming (o PDCA) è un approccio manageriale per il controllo e il miglioramento continuo dei processi e dei prodoti ed è parte integrante della gestione della qualità. Il ciclo di Deming è suddiviso in quattro fasi:

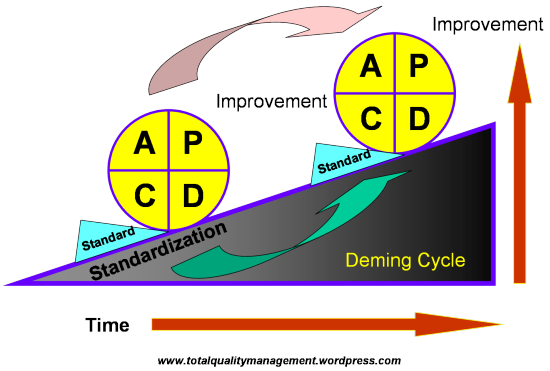

Figura 2: Ciclo di Deming

- **Plan** - stabilire gli obiettivi e i processi necessari per fornire risultati in accordo con i risultati attesi, attraverso la creazione di attese di produzione, di completezza e accuratezza delle specifiche scelte. Quando possibile, avvio su piccola scala, per verificare i possibili effetti.
- **Do** - Esecuzione del programma, dapprima in contesti circoscritti. Attuare il piano, eseguire il processo, creare il prodotto. Raccogliere i dati per la creazione di grafici e analisi da destinare alla fase di "Check" e "Act".
- **Check** - Test e controllo, studio e raccolta dei risultati e dei riscontri. Studiare i risultati, misurati e raccolti nella fase del "Do" confrontandoli con i risultati attesi, obiettivi del "Plan", per verificarne le eventuali differenze. Cercare le deviazioni nell'attuazione del piano e focalizzarsi sulla sua adeguatezza e completezza per consentirne l'esecuzione. I grafici dei dati possono rendere questo molto più facile, in quanto è possibile vedere le tendenze di più cicli PDCA, convertendo i dati raccolti in informazioni. L'informazione è utile per realizzare il passo successivo : "Act".
- **Act** - Azione per rendere definitivo e/o migliorare il processo (estendere quanto testato dapprima in contesti circoscritti all'intera organizzazione). Richiede azioni correttive sulle differenze significative tra i risultati effettivi e previsti. Analizza le differenze per determinarne le cause e dove applicare le modifiche per ottenere il miglioramento del processo o del prodotto. Quando un procedimento, attraverso questi quattro passaggi, non comporta la necessità di migliorare la portata a cui è applicato, il ciclo PDCA può essere raffinato per pianificare e migliorare con maggiore dettaglio la successiva iterazione, oppure l'attenzione deve essere posta in una diversa fase del processo.

A.3 Standard ISO/IEC 9126

Aderendo a questo standard il team si impegna a garantire nel prodotto le seguenti qualità (definite con relativa metrica, misura e strumenti di controllo):

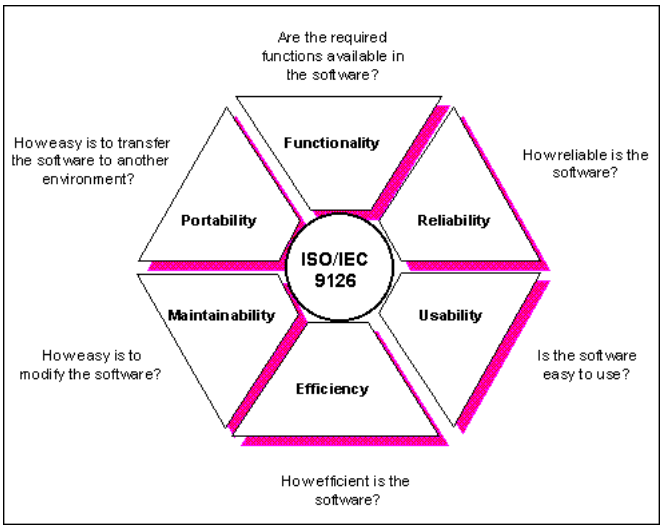

Figura 3: Standar ISO/IEC 9126

1. **Funzionalità**: il sistema prodotto deve garantire tutte le funzionalità indicate nel documento Analisi dei Requisiti v3.0.0 L'implementazione dei requisiti deve essere più completa possibile

   - **Misura**: l'unità di misura usata sarà la quantità di requisiti mappati sulle componenti del sistema create e funzionanti
   - **Metrica**: la sufficienza è stabilita nel soddisfacimento di tutti i requisiti obbligatori
   - **Strumenti**: per soddisfare questa qualità il sistema deve superare tutti i test previsti. Per informazioni dettagliate sugli strumenti si veda Norme di Progetto v4.0.0

2. **Affidabilità**: il sistema deve dimostrarsi il più possibile robusto e di facile ripristino in caso di errori

   - **Misura**: l'unità di misura utilizzata sarà la quantità di esecuzioni del sistema andate a buon fine
   - **Metrica**: le esecuzioni dovranno spaziare il più possibile sulle varie parti del codice
   - **Strumenti**: si vedano le metriche adottate per la copertura a

3. **Usabilità**: il sistema prodotto deve risultare di facile utilizzo per la classe di utenti a cui è destinato. Tale sistema deve essere facilmente apprendibile e allo stesso tempo deve soddisfare tutte le necessità dell'utente.

- **Misura**: l'unità di misura usata sarà una valutazione soggettiva dell'usabilità. Questo è dovuto all'inesistenza di una metrica oggettiva adatta allo scopo
- **Metrica**: non esiste una metrica adeguata che determinerà la sufficienza su questa qualità
- **Strumenti**: si vedano le Norme di Progetto v4.0.0

4. **Efficienza**: il sistema deve fornire tutte le funzionalità nel più breve tempo possibile, riducendo al minimo l'utilizzo di risorse
   - **Misura**: il tempo che lo sviluppatore impiegherà per la creazione di una bolla interattiva
   - **Metrica**: non è possibile definire una soglia oggettiva di sufficienza perché non è possibile valutare ogni possibile casistica di utilizzo
   - **Strumenti**: si vedano le Norme di Progetto v4.0.0
5. **Manutenibilità**: il sistema deve essere comprensibile ed estensibile in modo facile e verificabile
   - **Misura**: le metriche sul codice descritte nella sezione *2.8.2* costituiscono l'unità di misura
   - **Metrica**: il software avrà le caratteristiche di manutenibilità descritte. Questo sarà possibile se il prodotto avrà la sufficienza in tutte le metriche
   - **Strumenti**: si vedano le Norme di Progetto v4.0.0
6. **Portabilità**: il sistema deve essere il più portabile possibile, in particolare dentro a Rocket.chat
   - **Misura**: il front end deve rispettare gli standard W3C
   - **Metrica**: il software avrà le caratteristiche di portabilità descritte. Questo sarà possibile se il prodotto avrà la sufficienza in tutte le metriche, descritte nella sezione *2.8.2*
   - **Strumenti**: si vedano le *Norme del Progetto vxxx*

B Resoconto delle attività di verifica

B.1 Revisione dei Requisiti

L'attività di verifica svolta dai *Verificatori* è avvenuta come determinato dal Piano di progetto v4.0.0 al termine della stesura di ogni documento previsto. La verifica svolta sui documenti e sui processi è avvenuta seguendo le indicazioni delle Norme di Progetto v4.0.0 e misurando le metriche indicate nella sezione *2.8.2*

B.1.1 Verifiche sui Processi

In questo periodo è stata svolta un' attività di walkthrough non avendo gli elementi per effettuare l'attività di inspection. Nella verifica dei documenti sono stati riscontrati soprattutto errori grammaticali e di battitura dovuti a disattenzioni durante la stesura.

B.1.2 Documenti

Vengono qui riportati i valori dell'indice Gulpease per ogni documento durante l'analisi e relativo esito basato sui range stabiliti nella sezione *2.8.2*

Documento	Valore indice	Esito
Analisi dei Requisiti v1.0.0	70	superato
Norme di Progetto v1.1.0	58	superato
Piano di Progetto v1.1.0	65	superato
Piano di Qualifica v1.0.0	65	superato
Studio di Fattibilità v1.0.0	67	superato

Tabella 11: Tabella risultati test Gulpease

B.2 Revisione di Progettazione

In questo periodo (antecedente la consegna di tale revisione) sono stati verificati i documenti ed i processi.

- Sono stati verificati i documenti applicando la procedura descritta nel documento *Norme di progetto vxxx*
- É stata applicata *l'analisi statica* secondo i criteri e le modalità indicata alla sezione *2.7.1.*
- Si sono calcolate per i documenti le metriche descritte nella sezione *2.8.*
- Il tracciamento (requisiti - componenti) è stato effettuato tramite un database MySQL creato ad uso interno.
- L'avanzamento dei processi è stato controllato e verificato secondo le metodiche descritte nelle *Norme di Progetto vxxx.*

B.2.1 Verifiche sui processi

Vengono qui riportati gli esiti delle verifiche sui processi produttivi.

Budget Variance / Schedule Variance

Attività	SV	BV
Analisi dei Requisiti 2.0.0	0€	0€
Norme di Progetto 3.0.0	+10€	+70€
Piano di Progetto 3.0.0	0€	0€
Definizione di prodotto 1.0.0	+25€	-110€
Piano di qualifica 3.0.0	-10€	0€
Glossario 2.0.0	0€	0€

Tabella 12: Tabella risultati Budget Variance / Schedule Variance

Complessivamente nel periodo di Revisione di Progettazione vengono misurate:

- Schedule Variance = 25€

- Budget Variance = -40€

Da valori di tali indici possiamo dedurre che:

- La data di fine della Revisione di Progettazione si è dimostrata essere un giorno prima di quella pianificata nel Piano di Progetto v3.0.0 .
- Il limite inferiore di accettabilità del Budget Variance è pari a -436€. Il valore ottenuto risulta essere quindi accettabile.

B.2.2 Verifica dei Documenti

Vengono qui riportati i valori dell'indice Gulpease per ogni documento durante l'analisi e relativo esito basato sui range stabiliti nella sezione *2.8.2*

Documento	Valore indice	Esito
Analisi dei Requisiti 2.0.0	60.1	superato
Norme di Progetto 3.0.0	56.4	superato
Piano di Progetto 3.0.0	67.5	superato
Piano di Qualifica 3.0.0	61.4	superato
Definizione di Prodotto 1.0.0	76.9	superato

Tabella 13: Tabella risultati test Gulpease

B.3 Revisione di Qualifica

In questo periodo (antecedente la consegna di tale revisione) sono stati verificati i documenti ed i processi.

- Sono stati verificati i documenti applicando la procedura descritta nel documento *Norme di progetto vxxx*
- É stata applicata *l'analisi statica* secondo i criteri e le modalità indicata alla sezione *2.7.1.*
- Si sono calcolate per i documenti le metriche descritte nella sezione *2.8.*
- L'avanzamento dei processi è stato controllato e verificato secondo le metodiche descritte nelle *Norme di Progetto vxxx.*

B.3.1 Verifiche sui processi

Vengono qui riportati gli esiti delle verifiche sui processi produttivi.

Budget Variance / Schedule Variance

Metrica	Unità di misura	Risultato	Risultato massimo	Esito	Valore
Schedule Variance	Attività	0	/	Superato	Ottimale
Budget Variance	Euro	-687	/	Non superato	/

Tabella 14: RQ- BV e SV

- La data di fine della Revisione di Progettazione si è dimostrata essere il giorno stabilito nel Piano di Progetto v4.0.0 .
- Il limite inferiore di accettabilità del Budget Variance è pari a -396€. Il valore ottenuto risulta essere quindi non accettabile.

B.3.2 Produttività

Metrica	Unità di misura	Risultato	Risultato massimo	Esito	Valore
Produttività di documentazione	n° parole/h	116	/	Superato	Ottimale
Produttività di codifica	n° linee codice/h	86	/	Superato	Accettabile
Copertura del test	Percentuale	73%	/	Superato	Accettabile

Tabella 15: RQ-Produttività

B.3.3 Verifica dei Documenti

Vengono qui riportati i valori dell'indice Gulpease per ogni documento durante l'analisi e relativo esito basato sui range stabiliti nella sezione *2.8.2*

Documento	Valore indice	Esito
Analisi dei Requisiti 2.0.0	60.1	superato
Norme di Progetto 3.0.0	56.4	superato
Piano di Progetto 3.0.0	67.5	superato
Piano di Qualifica 3.0.0	61.4	superato
Definizione di Prodotto 1.0.0	76.9	superato

Tabella 16: Tabella risultati test Gulpease

B.3.4 Metriche per software

Si noti che alcune componenti e i test a loro correlati non sono ancora state sviluppate, e che la loro assenza peserà nelle misurazioni.

Metrica	Unità di misura	Risultato	Risultato massimo	Esito	Valore
Complessità Ciclomatica media	Cammini	2.1	/	Superato	Ottimale
Numero di metodi per package (max)	Metodi	/	14	Superato	Accettabile
Variabili non utilizzate e non definite	Variabili	/	0	Superato	Ottimale
Numero parametri per metodo (max)	Parametri	/	5	Superato	Accettabile

Metrica	Unità di misura	Risultato	Risultato massimo	Esito	Valore
Halstead difficulty per-function (media)	indice complessità	3.42	/	Superato	Ottimale
Halstead volume per-function (media)	indice complessità	100.76	/	Superato	Ottimale
Halstead effort per-function	indice complessità	344,59	/	Superato	Accettabile
Maintainability index	Valore	71.98	/	Superato	Accettabile
Statement Coverage	Percentuale	77.4%	/	Superato	Accettabile
Branch Coverage	Percentuale	83%	/	Superato	Accettabile
Copertura dei test	Percentuale	???%	/	Superato	Ottimale

Tabella 17: RQ-Metriche per Software