Sirius

Sequenziatore

Piano Di Qualifica

Versione 3.0.0

Ingegneria Del Software AA 2013-2014

Informazioni documento

Titolo documento:	Piano Di Qualifica
Data creazione:	21 Gennaio 2014
Versione attuale:	3.0.0
Utilizzo:	Interno
Nome file:	*PianoDiQualifica_v3.0.0.pdf*
Redazione:	Seresin Davide
Verifica:	Botter Marco
	Quaglio Davide
Approvazione:	Quaglio Davide
Distribuito da:	Sirius
Destinato a:	Prof. Vardanega Tullio
	Prof. Cardin Riccardo
	Zucchetti S.p.A.

Sommario

Il documento spiega in dettaglio la strategia di verifica adottata dal gruppo *Sirius* per lo sviluppo del progetto *Sequenziatore*.

Diario delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
3.0.0	2014-06-26	*Botter Marco*	*Responsabile*	Approvazione del documento
2.1.5	2014-06-26	*Seresin Davide*	*Amministratore*	Aggiornamento test di analisi statica
2.1.4	2014-06-25	*Quaglio Davide*	*Amministratore*	Aggiunta e rendicontazione dell'attività di verifica
2.1.3	2014-06-01	*Seresin Davide*	*Amministratore*	Definizione dei test di unità
2.1.2	2014-05-30	*Quaglio Davide*	*Amministratore*	Definizione dei test di integrazione
2.1.1	2014-05-29	*Seresin Davide*	*Amministratore*	Aggiornamento dello stato dei test di sistema
2.1.0	2014-05-10	*Marco Botter*	*Verificatore*	Verifica delle correzioni
2.0.2	2014-04-24	*Seresin Davide*	*Amministratore*	Aggiornamenti paragrafi per evitare discorsività
2.0.1	2014-04-20	*Quaglio Davide*	*Amministratore*	Aggiornamento titoli con contenuti
2.0.0	2014-03-23	*Quaglio Davide*	*Responsabile*	Approvazione del documento
1.2.0	2014-03-23	*Botter Marco*	*Verificatore*	Verifica della nuova definizione test e resoconto attività di verifica
1.1.2	2014-03-23	*Quaglio Davide*	*Amministratore*	Aggiunta del resoconto attività di verifica
1.1.1	2014-03-10	*Seresin Davide*	*Amministratore*	Modifica della sezione definizione obiettivi
1.1.0	2014-03-11	*Quaglio Davide*	*Verificatore*	Verifica delle modifiche
1.0.1	2014-03-10	*Seresin Davide*	*Amministratore*	Modifica dello scopo del documento e pianificazione dei test
1.0.0	2014-03-05	*Quaglio Davide*	*Amministratore*	Approvazione del documento
0.2.0	2014-03-05	*Botter Marco*	*Verificatore*	Verifica delle aggiunte su resoconto dell'attività di verifica

Versione	Data	Autore	Ruolo	Descrizione
0.1.1	2014-03-01	*Seresin Davide*	*Amministratore*	Aggiunta resoconto attività di verifica
0.1.0	2014-02-20	*Botter Marco*	*Verificatore*	Verifica del documento e appendice
0.0.3	2014-02-18	*Seresin Davide*	*Amministratore*	Aggiunta di informazioni dettagliate e appendice
0.0.2	2014-02-15	*Quaglio Davide*	*Verificatore*	Verificato scheletro e bozza documento
0.0.1	2014-02-12	*Seresin Davide*	*Amministratore*	Creato lo scheletro del documento

Indice

1 Introduzione

1.1 Scopo del Prodotto

Lo scopo del progetto *Sequenziatore*, è di fornire un servizio di gestione di processi definiti da una serie di passi da eseguirsi in sequenza o senza un ordine predefinito, utilizzabile da dispositivi mobili di tipo smaptphone o tablet.

1.2 Glossario

Al fine di rendere più leggibile e comprensibile i documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario_v3.0.0.pdf*.
Ogni occorrenza di vocaboli presenti nel *Glossario* deve essere seguita da una "G" maiuscola in pedice.

1.3 Riferimenti

1.3.1 Normativi

- ISO/IEC Standard 12207:1995;
- ISO/IEC 9126;
- IEEE Std 730TM-2002 (revision of IEEE Std 730-1998) - Standard for Software Quality Assurance Plans;
- **Norme di progetto:** Norme di progetto v3.0.0;
- Capitolato d'appalto C4: *sequenziatore*.

1.3.2 Informativi

- Informazioni sul sito del docente;
- Software Engineering (9th edition) Ian Sommerville Pearson Education Addison-Wesley;
- Ciclo di Deming - Estratto da Software Engineering (9th edition) Ian Sommerville Pearson Education Addison-Wesley;
- Capability Maturity Model Integration (CMMI) - Estratto da Software Engineering (9th edition) Ian Sommerville Pearson Education Addison-Wesley;
- SWEBOK cap.11 - *Software Quality*;
- **Piano di progetto:** Piano Di Progetto v3.0.0;

- Indice di Gulpease.

1.4 Scopo del documento

Il documento si prefigge di illustrare la strategia complessiva di verifica e validazione proposta dal team *Sirius* per pervenire al collaudo del sistema con la massima efficacia$_G$.In questo documento, inoltre, definiamo gli obiettivi di qualità intesa come il rispetto dei requisiti e prestazioni enunciati esplicitamente, la conformità agli standard di sviluppo esplicitamente documentati e le caratteristiche implicite che si aspetta da un prodotto software. Garantendo in particolar modo ed in modo macroscopico:

- La correttezza del prodotto;
- La verifica continua sulle attività svolte;
- Il soddisfacimento del cliente.

2 Visione generale della strategia di verifica

2.1 Organizzazione

Il team *Sirius* ha deciso di porre al centro di ogni periodo l'attività di verifica in quanto essa certifica la qualità del prodotto. L'attività di verifica sarà continua in tutte le fasi del progetto.
Il processo di qualifica accompagnerà tutte le fasi di ciclo di vita del software. Ogni procedura di verifica sarà schedulata attraverso appositi strumenti e i risultati saranno analizzati in questo documento. Tramite il diario delle modifiche è possibile tenere traccia dell'attività di verifica effettuata ed operare delle verifiche circoscritte ai soli cambiamenti. In particolare le operazioni di controllo verranno istanziate quando il prodotto da analizzare avrà raggiunto uno stato in cui presenti differenze sostanziali rispetto allo stato precedente. Lo schema che rappresenta l'organizzazione e la pianificazione delle attività di verifica è il **modello a V**(V-model). Il modello dimostra la relazione tra ogni periodo del ciclo di vita dello sviluppo del software e il suo periodo di *testing*.

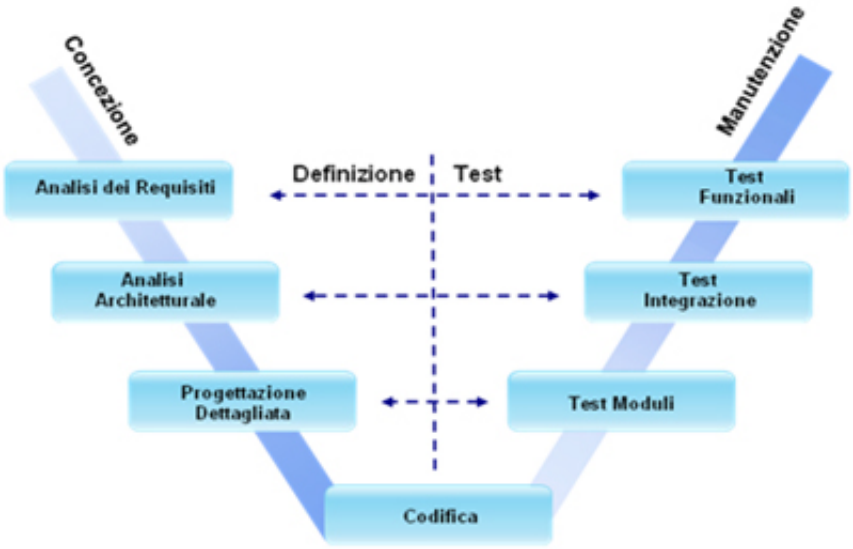

Figura 1: Modello a V

L'organizzazione della strategia di verifica prevede l'attività di verifica in tutti i periodi di avanzamento del prodotto, che sono paralleli alle scadenze definite in Piano Di Progetto v3.0.0.

- **Analisi**: in questa prima fase il compito del *Verificatore* è innanzitutto relativo alla documentazione e alla correttezza del tracciamento dei requisiti. Ogni documento che servirà per la consegna della RR, una volta ultimata la fase di redazione, verrà verificato in modo definitivo seguendo la procedura così definita:

1. Verrà controllata la correttezza dei contenuti rispetto alle aspettative del documento tramite una rilettura accurata;
2. Verrà controllata la correttezza grammaticale;
3. Verrà controllato che il documento rispetti le norme definite in Norme di progetto v3.0.0 tramite la lista di controllo presente in tale documento.
4. Verrà verificato che ogni requisito funzionale rilevato abbia una corrispondenza in almeno un caso d'uso e che questo sia tracciato tramite il software di tracciamento che *Sirius* ha deciso di utilizzare;
5. Verrà verificato che ogni requisito di vincolo e di qualità sia tracciato tramite il software di tracciamento che *Sirius* ha deciso di utilizzare.

- **Progettazione**: il *Verificatore* ha l'importante compito di controllare il soddisfacimento dei requisiti indicati in fase di analisi. Inoltre, si devono verificare che i processi che portano all'incremento dei documenti redatti nel precedente periodo siano conformi alle procedure e regole descritte in Norme di progetto v3.0.0;
- **Programmazione**: il *Verificatore* provvederà a controlli periodici e pianificati di porzioni di codice, inizialmente di tipo statico per poi passare a dei controlli di tipo dinamico per valutare la correttezza del software;
- **Collaudo**: in questa fase le verifiche saranno esclusivamente di tipo dinamico per garantire che il prodotto risponda a tutti i requisiti indicati e a tutte le richieste del committente: sia implicite che esplicite. Per le indicazioni precise circa la procedura di verifica adottata dal gruppo si fa riferimento a Norme di progetto v3.0.0.

Il processo di verifica verterà sulla parte di redazione di documenti, essendo questa un'attività predominante e costante durante tutto il progredire del progetto. Garantendo, altresì che il risultato software sia efficace$_G$ rispetto alla procedura analizzata, non perdendo di efficienza$_G$ contrattuale.

2.2 Pianificazione strategica temporale

Al fine di rispettare in modo ristretto le scadenze citate di seguito e spiegate in modo approfondito in Piano Di Progetto v3.0.0, *Sirius* ha deciso di pianificare in modo approfondito e sistematico l'attività di verifica. Facendo in modo di rilevare e risolvere nel più breve tempo possibile gli errori che vengono rilevati per evitare che questi possano creare maggiori problematiche nell' avanzamento del prodotto software.
Per questo si adottano delle specifiche tecniche in base all'avanzamento del progetto. Ogni attività di redazione dei documenti e di codifica saranno precedute da uno studio

preliminare sulla struttura e sui contenuti degli stessi. *Sirius*, conscio della poca esperienza nella pianificazione e gestione di progetti di questo tipo, ha deciso di inserire degli $slack_{\mathrm{G}}$ temporale durante la pianificazione delle attività. Tale scelta è approfondita in Norme di progetto v3.0.0 che ne definisce la quantità e in Piano Di Progetto v3.0.0 che ne analizza le motivazioni. L'aggiunta di slack temporali, oltre a portare al progetto una pianificazione più precisa, comporta un aumento dei costi che è però commisurato all'aumento della qualità finale.

2.3 Obiettivi

2.3.1 Qualità di processo

Al fine di garantire la qualità di prodotto è necessario ricercare la qualità dei processi che lo definiscono. Per questo *Sirius* ha deciso di adottare lo standard ISO/IEC 15504 denominato SPICE il quale fornisce le indicazioni necessarie a valutare l'idoneità dei processi attualmente in uso. Per applicare correttamente questo modello, ed adattarlo alla gestione attuata dal gruppo di lavoro, si è deciso di utilizzare il ciclo di Deming o PDCA: il quale definisce una metodologia di controllo dei processi durante il loro ciclo di vita permettendo, inoltre, di migliorare in modo continuo la qualità.

2.3.2 Qualità di prodotto

Al fine di aumentare il valore del prodotto e di garantire il corretto funzionamento dello stesso, è necessario fissare degli obiettivi e garantire che questi vengano effettivamente rispettati. Lo standard ISO/IEC 9126, descritto in appendice A.2, è stato redatto con lo scopo di definire obiettivi e di delineare metriche capaci di misurare il raggiungimento degli stesso.
Di seguito elenchiamo le caratteristiche che *Sirius* si impegna a garantire per il prodotto che andrà a realizzare. Oltre alla descrizione della caratteristica qui vengono definite le metriche, i parametri di accettazione.

- **Funzionalità** L'applicazione prodotta deve soddisfare tutti i requisiti obbligatori individuati in Analisi dei requisiti v3.0.0 nel modo più completo ed economico possibile, garantendo la sicurezza del prodotto e dei suoi componenti, e adeguandosi alle norme e alle prescrizioni imposte. Inoltre, data la natura del prodotto *Sequenziatore*, *Sirius* ha deciso di prestare particolare attenzione all'interoperabilità del codice: intesa come la capacità di agire con altri sistemi.
  - **misura:** percentuale di requisiti soddisfatti;
  - **metrica:** la soglia di sufficienza sia il soddisfacimento di tutti i requisiti obbligatori;

- **Affidabilità** L'applicazione deve dimostrarsi robusta, di facile ripristino e recupero in caso di errori, e aderire alle norme e alle prescrizioni stabilite.
  - **misura:** numero di esecuzioni dell'applicazione andate a buon fine;
  - **metrica:** le esecuzioni dovranno spaziare su tutta la gamma delle possibili casistiche. Il numero di esecuzioni andate a buon fine dovrà essere rapportato al numero totale delle casistiche considerate;
- **Usabilità** L'applicazione deve risultare comprensibile, facilmente apprendibile e soprattutto aderire a norme e prescrizioni per garantire facilità d'uso e soddisfacimento delle necessità dell'utente.
  - **misura:** data l'aleatorietà della qualità richiesta, non si riesce a definire un'unità di misura obiettiva;
  - **metrica:** non è stata definita una metrica di usabilità; ma *Sirius* cercherà di offrire la miglior esperienza di utilizzo per tutti coloro che usano il prodotto;
- **Efficienza** L'applicazione deve fornire tutte le funzionalità nel minor tempo possibile e con il minimo utilizzo di risorse.
  - **misura:** il tempo di latenza per ottenere una risposta dal programma; il tempo di latenza per ottenere una risposta simulando un sovraccarico della rete;
  - **metrica:** i tempi di latenza dovranno essere in linea con le tempistiche rilevate con l'utilizzo di architetture dello stesso tipo di quelle definite nel prodotto;
- **Manutenibilità** L'applicazione deve essere analizzabile, facilmente modificabile e verificabile; inoltre dovrà ridurre il rischio di comportamenti inaspettati al seguito dell'effettuazione di modifiche. Inoltre la documentazione prodotta deve essere chiara e comprensibile.
  - **misura:** le misurazioni per garantire questa caratteristica sono diverse e non esclusive e sono descritte in seguito;
  - **metrica:** le metriche da utilizzare sono descritte in seguito;
- **Portabilità** L'applicazione deve essere adattabile e compatibile con ambienti d'uso diversi, e con i quali dovrà coesistere condividendo risorse e anche per questo sarà validata da strumenti forniti dal W3C.
  - **misura:** L'applicazione dovrà essere eseguibile con i browser indicati in Analisi dei requisiti v3.0.0;

  - **metrica:** soddisfacimento dei requisiti di compatibilità e validazione tramite strumenti W3C;

Inoltre, *Sirius* ha definito delle altre caratteristi che andranno ricercate per il prodotto:

- **Semplicità**: realizzazione del prodotto nella maniera più semplice possibile, ma non semplicistica;
- **Incapsulamento**: il codice deve avere visibilità minima e permettere un utilizzo dall'esterno solamente mediante interfacce; ciò aumenta la manutenibilità e la possibilità di riuso del codice;
- **Coesione**: le funzionalità che concorrono allo stesso fine devono risiedere nello stesso componente; favorisce semplicità, manutenibilità, riusabilità e riduce l'indice di dipendenza.

La definizione delle metriche e gli strumenti utilizzati per la rilevazione delle stesse sono specificate in Norme di progetto v3.0.0. Di seguito per ogni metrica indichiamo le misure desiderabili: divise in range ottimale e range di accettazione. D.S indica che sarà definito in seguito.

Metrica	**Accettazione**	**Ottimale**
SV_1	>-(ore prev x5%)	>0
BV_2	>-(costo prev x10%)	>0
$Gulpease_3$	40-100	50-100
Complessità $ciclomatica_4$	1-15	1-10
Livelli di $annidamento_5$	1-6	1-3
Attributi per $classe_6$	0-16	3-8
Parametri per $metodo_7$	0-8	0-4
Linee di codice per linee di $commento_8$	>0,25	>0,30
Accoppiamento	D.S.	D.S.
Copertura	80%-100%	85%-100%

Tabella 1: parametri delle metriche adottate.

I parametri ottimali e di accettazione sono riferiti a:

1. Parametro calcolato da *Sirius* valutando l'inesperienza del gruppo;

2. Parametro calcolato da *Sirius* valutando l'inesperienza del gruppo;
3. Parametro calcolato in base all'utenza della documentazione;
4. Il valore 10 come massimo fu raccomandato da T.J.McCabe, l'inventore di tale metrica;
5. Valore ideale valutando la chiarezza di codifica;
6. Valore ideale valutando la chiarezza di codifica;
7. Valore ideale valutando la chiarezza di codifica;
8. Il valore 0,30 è ricavato dal rapporto 22/78. Valori ricavati dalle medie dichiarate da Ohlo(Open source network).

2.4 Procedure di controllo

2.4.1 Qualità di processo

Al fine di garantire la qualità di processo, *Sirius* adotterà il principio PDCA, descritto nella sezione A.1. Tramite tale tecnica, sarà garantito un miglioramento continuo dei processi e quindi della qualità degli stessi. Come diretta conseguenza si otterrà il miglioramento qualitativo del prodotto risultante. Per ottenere questo è necessario che il processo sia in controllo e quindi:

- Effettuare una dettagliata pianificazione dei processi;
- Pianificare il numero di risorse da utilizzare, e ripartire le stesse in modo chiaro;

Verranno utilizzate due metriche per monitorare la qualità di processo e per mantenere il processo in controllo, queste sono: BV che monitora il il consumo del budget nel tempo e SV che valuta lo stato di avanzamento temporale attuale rispetto a quello pianificato. Le specifiche di queste due metriche sono indicate in Norme di progetto v3.0.0.
Valori ottimali di BV ed SV indicano un elevato grado di conoscenza ed integrazione del processo come indicato nel CMMI nella sezione A.3.
La qualità dei processi viene monitorata inoltre tramite la qualità del prodotto: infatti un prodotto di bassa qualità indica un processo da migliorare.
Se su un processo non vengono rilevati problemi, è possibile apportare dei miglioramenti: riducendo il numero di cicli iterattivi, il tempo o le risorse ma garantendo che l'esecuzione del processo sia fedele al piano e soddisfi i requisiti. In questo modo si aumenta l'efficenza del processo e se ne determina un'adattamento positivo alla realtà del team di lavoro valutabile in efficacia.

2.4.2 Qualità di prodotto

Il controllo di qualità di prodotto verrà garantito da:

- *Quality Assurance*: insieme delle attività atte a garantire il raggiungimento degli obiettivi di qualità. Prevede tecniche di analisi statica e dinamica descritte in 2.8;
- *Verifica*: processo che determina se l'output è consistente, corretto e completo. Il processo di verifica sarà eseguito durante l'intera durata del progetto. I risultati delle attività di verifica saranno riportate in appendice B;
- *Validazione*: certificazione che attesta la conformità del sistema ai requisiti.

2.5 Risorse umane e responsabilità

Al fine di garantire l'efficacia e la sistematicità del processo di verifica vengono attribuite delle responsabilità a degli specifici ruoli di progetto. Per il processo di verifica le responsabilità sono attribuite a *Responsabile di Progetto* ed ai *Verificatori*. Mentre compito dell' *Amministratore* è quello di supporto a tutte le attività fornendo una solida infrastruttura software anche per il processo di verifica in ogni fase lavorativa. Per una descrizione più approfondita di ruoli e responsabilità si rimanda a Norme di progetto v3.0.0.

2.6 Risorse software

Al fine di effettuare la fase di verifica e validazione nel modo più sistematico possibile sono stati messi a disposizione di tutti i *Verificatori* dall' *Amministratore* un pacchetto di prodotti software il più specifico possibile rispetto alle esigenze del team. Inoltre, è sempre compito dell' *Amministratore* formare ogni verificatore all'utilizzo dei prodotti che permettano la verifica, evidenziando, se richiesto le funzionalità non utilizzate per ogni prodotto. *Sirius* ha deciso di adottare questo tipo di formazione per valorizzare il lavoro di chi effettivamente utilizza i prodotti di verifica, dando la possibilità che proprio da queste figure nascano idee e proposte di miglioramento che saranno poi valutate dal *Responsabile di progetto* congiuntamente all'*Amministratore*. Gli strumenti necessari al raggiungimento degli obiettivi definiti è indicato in Norme di progetto v3.0.0, che ne definiscono anche le specifiche tecniche e l'utilizzo.

2.7 Tecniche di analisi statica

Questa tecnica di analisi applicabile sia alla documentazione che al codice e permette di effettuare la verifica di quanto prodotto individuando errori ed anomalie. Le tecniche di

analisi statica utilizzate sono di tipo **inspection** per la documentazione, solo a seguito di una prima analisi di tipo **walkthrough**. Per il codice verrà adottata principalmente una tipologia di analisi **inspection** per gli errori che sono più ricorrenti o che sono stati rilevati nelle verifiche precedenti. Per la descrizione si faccia riferimento a Norme di progetto v3.0.0.

2.8 Tecniche di analisi dinamica

L'analisi dinamica si applica solamente al prodotto software e ciò consiste nell'esecuzione del codice mediante l'uso di test predisposti per verificarne il funzionamento o rilevare possibili difetti di implementazione eseguendo tutto o solo una parte del codice. Al fine di garantire l'utilità del test è necessario che il test sia *ripetibile*. Per *ripetibile* si intende che dato un certo input per la stessa porzione di codice, questo produca sempre lo stesso output sulla stesso ambiente. Per questo motivi saranno definiti a priori:

- **Ambiente**: si tratta sia del sistema hardware che di quello software sui quali è stato pianificato l'utilizzo del prodotto; di essi deve essere specificato lo stato iniziale dal quale poter far partire il test;
- **Specifica**: definire quali sono gli input e quali dovranno essere gli output attesi;
- **Procedure**: definire come sono svolti i test, l'ordine e come devono essere analizzati i risultati;

I test verteranno su test di unità per le porzioni di codice prodotte, test di integrazione per le componenti aggiunte in modo incrementale; test di sistema per verificare la corretta esecuzione del sistema; eventuali test di regressione per le modifiche apportate a componenti già testati; e infine test di accettazione per validare il prodotto finale.

2.8.1 Test di unità

Verifica di ogni singola unità del prodotto tramite l'utilizzo di stub, driver e logger.

2.8.2 Test di integrazione

Verifica dei componenti di sistema che andranno ad incrementare la parte già testata del prodotto; con l'obiettivo di testare la combinazione di più unità garantendone le funzionalità del risultato. Per l'esecuzione di tali test dovranno essere aggiunte delle componenti fittizie a sostituzione di quelle che non sono ancora state sviluppate, facendo in modo di non influenzare l'esito dell'analisi.

2.8.3 Test di sistema

Consiste nella validazione del prodotto software dal momento che lo si ritiene giunto ad una versione definitiva. Tale test ha lo scopo di verificare che la il prodotto copra tutti i requisiti stabiliti in fase di analisi.

2.8.4 Test di regressione

Si intende la nuova esecuzione di test a fronte di una modifica di alcune componenti software, per garantire la conformità del sistema post modifica.

2.8.5 Test di accettazione

Definito come il collaudo del prodotto. Questo test viene eseguito in presenza del proponente ed in caso di esito positivo si può procedere al rilascio ufficiale del prodotto sviluppato.

3 Gestione amministrativa della revisione

3.1 Comunicazione e risoluzione anomalie

Un'anomalia consiste in una deviazione del prodotto dalle aspettative prefissate. Per la gestione e risoluzione di anomalie ci si affida allo strumento di *ticketing* adottato da *Sirius*, e normato in Norme di progetto v3.0.0. Il *Verificatore*, per ogni anomalia riscontrata, dovrà aprire un nuovo ticket indirizzato al *Responsabile di Progetto*, il quale, dopo aver valutato l'impatto costi/benefici lo approverà e aprirà in ticket per il *Programmatore* che ha sviluppato quella parte di software o redatto il documento. Per la procedura di creazione del ticket si rimanda a Norme di progetto v3.0.0.

3.2 Trattamento delle discrepanze

Una discrepanza è un errore di coerenza tra il prodotto realizzato e quello atteso. *Sirius* interpreta la discrepanza come una forma di anomalia non grave, e per questo verrà trattata come tale.

4 Pianificazione dei test

Di seguito elenchiamo tutti i test di validazione, sistema ed integrazione previsti, prevedendo però ad un successivo aggiornamento per i test di unità. Per quanto riguarda le tempistiche di esecuzione dei test si faccia riferimento a Piano Di Progetto v3.0.0. Il valore **N.A** è da intendersi come non applicato, in quanto tali test saranno eseguiti successivamente nello svolgimento del progetto.

4.1 Test di sistema

In questa sezione vengono descritti i test di sistema che consentiranno a *Sirius* di verificare il comportamento dinamico del sistema rispetto ai requisiti descritti in Analisi dei requisiti v3.0.0. I test sotto riportati sono relativi ai requisiti software individuati e meritevoli di test.

4.1.1 Descrizione dei test di sistema

4.1.2 Ambito utente

Test	Requisito	Descrizione	Stato
TU1	FOBU 1	Verificare che il sistema permetta all'utente di registrarsi	Pianificato
TU1.1	FOBU 1.1	Verificare che lo *username* utente lo identifichi univocamente all'interno del sistema	Pianificato
TU1.1.1	FOBU 1.1.1	Verificare che lo *username* inserito dall'utente sia composto da almeno 6 caratteri	Pianificato
TU1.2	FOBU 1.2	Verificare che l'utente debba inserire una *password* d'accesso	Pianificato
TU1.2.1	FOBU 1.2.1	Verificare che la *password* sia composta almeno da 8 caratteri alfanumerici	Pianificato
TU1.5	FOBU 1.5	Verificare che il campo data di nascita sia un campo obbligatorio	Pianificato
TU1.5	FOBU 1.5.1	Verificare che la data di nascita inserita dall'utente sia antecedente alla data di iscrizione	Pianificato
TU1.6	FOBU 1.6	Verificare che la *email* utente sia un campo obbligatorio	Pianificato

TU1.6.1	FDEU 1.6.1	Verificare che la *email* inserita corrisponda ad un indirizzo di posta elettronica esistente	Pianificato
TU2	FOBU 2	Verificare che il sistema permetta all'utente di autenticarsi	Pianificato
TU2.1	FOBU 2.1	Verificare che il sistema non permetta il login con dati non presenti sul $server_G$	Pianificato
TU3.1	FOPL 3.1	Verificare che l'utente autenticato possa visualizzare le proprie credenziali	Pianificato
TU3.2	FOPL 3.2	Verificare che l'utente autenticato possa modificare i propri dati	Pianificato
TU4.1	FOBL 4.1	Verificare che l'utente autenticato possa scegliere un processo da una lista selezionata o da risultati di una ricerca	Pianificato
TU4.2	FOBL 4.2	Verificare che l'utente autenticato possa visualizzare la descrizione di un processo selezionato	Pianificato
TU4.3	FOBL 4.3	Verificare che un utente autenticato possa iscriversi a un processo precedentemente selezionato	Pianificato
TU4.4	FOBL 4.4	Verificare che un utente autenticato possa eseguire il processo scelto a cui è iscritto	Pianificato
TU4.4.1	FOBL 4.4.1	Verificare che all'utente autenticato sia concesso di visualizzare i criteri di terminazione di un processo	Pianificato
TU4.4.2	FOBL 4.4.2	Verificare che l'utente autenticato possa visualizzare le informazioni sullo stato corrente di avanzamento del processo selezionato	Pianificato
TU4.4.4	FOBL 4.4.4	Verificare che il sistema permetta all'utente autenticato di eseguire un passo del processo scelto	Pianificato
TU4.4.4.1	FOBL 4.4.4.1	Verificare che l'utente autenticato possa visualizzare le informazioni del passo in esecuzione	Pianificato

TU4.4.4.2	FOBL 4.4.4.2	Verificare che all'utente autenticato sia concesso visualizzare i vincoli da rispettare per superare il passo in esecuzione	Pianificato
TU4.4.4.3	FOBL 4.4.4.3	Verificare che il sistema permetta all'utente autenticato di inserire i dati richiesti per l'esecuzione del passo in corso	Pianificato
TU4.4.4.4	FOBL 4.4.4.4	Verificare che l'utente autenticato possa inviare al sistema i dati richiesti per l'esecuzione del passo in corso	Pianificato
TU4.4.4.4.6	FOPL 4.4.4.4.6	Verificare che il sistema permetta all'utente autenticato di raccogliere i dati in assenza di connessione e di inviarli a collegamento ripristinato	Pianificato
TU4.4.4.5	FOBL 4.4.4.5	Verificare che il sistema notifichi all'utente autenticato se i dati che ha inviato sono corretti, se non soddisfano i vincoli di superamento del passo o se sono in attesa di approvazione	Pianificato
TU4.4.4.6	FOBL 4.4.4.6	Verifica se il sistema permetta all'utente autenticato di concludere un passo del quale ha ricevuto l'approvazione sui dati da parte del sistema o dall'process owner	Pianificato
TU4.4.4.7	FOPL 4.4.4.7	Verifica che il sistema permetta all'utente autenticato di saltare il passo in esecuzione se facoltativo	Pianificato
TU4.4.5.1	FOPL 4.4.5.1	Verifica che all'utente autenticato sia concessa la creazione di un report finale su un processo terminato o del quale ha eseguito tutti i passi	Pianificato

TU4.4.5.2	FOBL 4.4.5.2	Verifica che all'utente autenticato sia permesso di eliminare un processo, un processo terminato o del quale ha eseguito tutti i passi, dalla lista dei processi gestiti	Pianificato
TU4.5	FOBL 4.5	Verificare che il sistema permetta all'utente autenticato di disiscriversi da un processo a cui è iscritto	Pianificato
TU5	FOBL 5	Verifica che l'utente possa terminare la propria sessione, diventando utente generico	Pianificato

Tabella 2: Tabella dei requisiti utente

4.1.3 Ambito process owner

Test	**Requisito**	**Descrizione**	**Stato**
TA1	FOBA 1	Verificare che al process owner sia consentita la creazione di processi	Pianificato
TA1.1	FOBA 1.1	Verificare che il process owner debba inserire un nome che identifichi univocamente il processo che vuole creare	Pianificato
TA1.2	FOBA 1.2	Verificare che il process owner debba inserire la descrizione del processo che vuole creare	Pianificato
T2.4.1	FOBA 2.4.1	Verificare che il sistema permetta al process owner di visualizzare i dati inviati dagli utenti che richiedono la sua approvazione	N.A
T1.3	FOBA 1.3	Verificare che il process owner debba definire i criteri di terminazione di un processo durante la sua creazione	Pianificato
T1.4	FOBA 1.4	Verificare che il sistema permetta al process owner di gestire i passi del processo in creazione	Pianificato

T1.4.1	FOBA 1.4.1	Verificare che il sistema permetta al process owner di creare un passo del processo in creazione	Pianificato
T1.4.2	FOBA 1.4.2	Verificare che il process owner possa visualizzare la lista dei passi creati durante la creazione di un nuovo processo	Pianificato
T1.4.3	FDEA 1.4.3	Verificare che il process owner, durante la creazione di un nuovo processo, potrà modificare un passo esistente	Pianificato
T1.4.4	FDEA 1.4.4	Verificare che il sistema permetta al process owner di eliminare un passo del processo in creazione	Pianificato
T1.5	FOBA 1.5	Verificare che il sistema permetta al process owner di avviare un processo in creazione che contiene almeno un passo	Pianificato
T2	FDEA 2	Verificare che il sistema permetta al process owner la gestione dei processi creati	Pianificato
T2.1	FDEA 2.1	Verificare che il process owner possa scegliere e gestire un processo avviato	Pianificato
T2.1.2	FOPA 2.1.2	Verificare che al process owner sia concesso di ricercare un processo inserendone il nome	Pianificato
T2.1.3	FDEA 2.1.3	Verificare che il sistema permetta al process owner di selezionare un processo da gestire	Pianificato
T2.2	FOPA 2.2	Verificare che il sistema permetta al process owner di selezionare gli utenti a cui permettere l'iscrizione al processo gestito	Pianificato

T2.4	FDEA 2.4 e FOBA 2.4.1	Verificare che al process owner sia concesso di controllare e visualizzare i dati inviati dagli utenti che richiedono la sua approvazione	Pianificato
T2.4.2	FDEA 2.4.2	Verificare che il process owner possa approvare i dati controllati	Pianificato
T2.4.3	FDEA 2.4.3	Verificare che il sistema permetta all'process owner di respingere i dati controllati	Pianificato
T2.4.4	FDEA 2.4.4	Verificare che il sistema invii l'esito del controllo agli utenti che hanno inviato dei dati di richiesta di approvazione	Pianificato
T2.5	FDEA 2.5	Verificare che il sistema permetta all'process owner di terminare un processo avviato	Pianificato
T2.6	FDEA 2.6	Verificare che al process owner sia concesso di eliminare un processo terminato dall'insieme dei processi creati	Pianificato

Tabella 3: Tabella dei requisiti process owner

4.2 Requisiti di vincolo

Test	**Requisito**	**Descrizione**	**Stato**
TV1	VOB1	Verificare che il sistema sia compatibile con il *browser*$_G$ *Google Chrome*$_G$ versione 27 e successive	Pianificato
TV5	VOB5	Verificare che il sistema sia compatibile con il *browser*$_G$ *mobile Android*$_G$ versione 2.3 e successive	Pianificato
TV6	VOB6	Verificare che il sistema sia compatibile con il *browser*$_G$ *Google Chrome per Android*$_G$ a partire dalla versione 18 inclusa	Pianificato
TV7	VOB7	Verificare che il sistema sia compatibile con il *browser*$_G$ *mobile Internet Explorer*$_G$ *per Windows Phone 8*$_G$ a partire dalla versione 10 inclusa	Pianificato

Tabella 4: Tabella dei requisiti di vincolo di compatibilità

4.3 Test di integrazione

In questa sezione verranno descritti i test di integrazione, da utilizzare per i vari componenti descritti nella progettazione ad alto livello, che permettono di verificare la corretta integrazione ed il corretto flusso dei dati all'interno del sistema. Si è deciso di utilizzare una strategia di integrazione incrementale bottom-up che permette di sviluppare e verificare le componenti in parallelo.
Assemblando le componenti in modo incrementale i difetti rilevati da un test sono da attribuirsi, con maggior probabilità, all'ultima parte aggiunta e si rende ogni passo di integrazione reversibile consentendo di retrocedere verso uno stato noto e sicuro. In questo modo i componenti base vengono testati più volte riducendo la possibile presenza di errori.
Nel diagramma

4.3.1 Descrizione dei test di integrazione

Di seguito sono elencati i test di integrazione relativo ai componenti indicati in Specifica Tecnica v3.0.0. I test di integrazione valutano l'integrazione dei vari componenti mano a mano che vengono aggregati al progetto, facendo in modo di non perdere le funzionalità acquisite fino a quel momento.

Test	**Descrizione**	**Componente**	**Test di integrazione**	**Stato**
TU1	Login process owner	Login	verifica con dati corretti verifica con dati errati	ok
TU2	Creazione processo	NuovoProcesso	creazione processo creazione passi	ok
TU3	Visualizzazione dati processo	VisualizzaProcesso	visualizzazione nome processo visualizzazione passi	ok
TU4	Approvazione passo	ApprovazionePasso	verifica approvazione verifica di non approvazione	ok

Tabella 5: Tabella dei test di integrazione process owner

Test	**Descrizione**	**Componente**	**Test di integrazione**	**Stato**

TU5	Iscrizione/disiscrizione	Registrazione	registrazione e login login e cancellazione	ok
TU6	Gestione dati	GestioneUser	controllo dati inseriti modifica dati utente	ok
TU7	Esecuzione passo	EsecuzionePasso	dati di tipo testuale dati di tipo numerico dati di tipo immagine dati di tipo geografici	ok
TU8	Visualizzazione report finale	Stampa	esecuzione stampa a video	ok

Tabella 6: Tabella dei test di integrazione user

4.4 Test di unità

I test di unità sono effettuati creando una nuova classe basata sulla precedente denominata *testnomeclasse* ed invocando i metodi di test con i rispettivi parametri. Per la definizione delle classi si faccia riferimento a Definizione Di Prodotto v1.0.0.

4.4.1 Test di unità per view

Per la *View*, essendo le classi prive di metodi, viene creato un metodo TestView() che riceve come parametro il tipo di dato richiesto e se ne verifica il risultato ottenuto confrontandolo con quello atteso.

4.4.1.1 TestLogin

- **Descrizione:** verifica che sia in grado di gestire le richieste di login;
- **Metodo:** TestView();
- **Esito:** Superato;

4.4.1.2 TestMainUser

- **Descrizione:** verifica del corretto passaggio di dati per l'utente *user*;
- **Metodo:** TestView(object userData);
- **Esito:** Superato;

4.4.1.3 TestRegister

- **Descrizione:** verifica del passaggio dati per la registrazione utente;
- **Metodo:** TestView();
- **Esito:** Superato;

4.4.1.4 TestUserData

- **Descrizione:** verifica correttezza dati e dell'eventuale aggiornamento in modo corretto;
- **Metodo:** TestView(object userdata);
- **Esito:** Superato;

4.4.1.5 TestOpenProcess

- **Descrizione:** verifica della corretta visualizzazione della ricerca e che da tale pagina sia permesso l'apertura di un processo tramite selezione o ricerca;
- **Metodo:** TestView(object processes);
- **Esito:** Superato;

4.4.1.6 TestManagementProcess

- **Descrizione:** visualizzazione corretta dello stato del processo e visualizzazione dei vincoli per la conclusione del passo;
- **Metodo:** TestView(object Process);
- **Esito:** Superato;

4.4.1.7 TestSendData

- **Descrizione:** verifica della possibilità di caricare informazioni per dichiarare l'esecuzione del passo;
- **Metodo:** TestView(object ProcessDataCollection);
- **Esito:** Superato;

4.4.1.8 TestSendText

- **Descrizione:** verifica della possibilità di caricare informazioni testuali per dichiarare l'esecuzione del passo;
- **Metodo:** TestView(object ProcessDataModel);
- **Esito:** Superato;

4.4.1.9 TestSendNumb

- **Descrizione:** verifica della possibilità di caricare informazioni numeriche per dichiarare l'esecuzione del passo;
- **Metodo:** TestView(object ProcessDataModel);
- **Esito:** Superato;

4.4.1.10 TestSendPosition

- **Descrizione:** verifica della possibilità di caricare informazioni di posizione per dichiarare l'esecuzione del passo;
- **Metodo:** TestView(object ProcessDataModel);
- **Esito:** Superato;

4.4.1.11 TestSendImage

- **Descrizione:** verifica della possibilità di caricare immagini per dichiarare l'esecuzione del passo;
- **Metodo:** TestView(object ProcessDataModel);
- **Esito:** Superato;

4.4.1.12 TestPrintProcess

- **Descrizione:** controllo della corretta esecuzione del report di terminazione passo;
- **Metodo:** TestView(object ProcessDataCollection);
- **Esito:** Superato;

4.4.1.13 TestMainProcessOwner

- **Descrizione:** verifica del corretto passaggio di dati per l'utente *user*;
- **Metodo:** TestView();
- **Esito:** Superato;

4.4.1.14 TestNewProcess

- **Descrizione:** verifica della corretta possibilità di inserimento dati per la creazione di un nuovo processo;
- **Metodo:** TestView(object ProcessCollection);
- **Esito:** Superato;

4.4.1.15 TestAddStep

- **Descrizione:** verifica della corretta possibilità di inserimento dati per la creazione di un nuovo passo del processo in creazione;
- **Metodo:** TestView(object StepModel);
- **Esito:** Superato;

4.4.1.16 TestOpenProcess

- **Descrizione:** verifica della possibilità di ricerca e apertura processo tramite ricerca o selezionandolo da una lista;
- **Metodo:** TestView(object ProcessCollection);
- **Esito:** Superato;

4.4.1.17 TestManageProcess

- **Descrizione:** verifica della corretta visualizzazione di *widget* per l'accesso ai dati inviati al$server_G$ dagli utenti;
- **Metodo:** TestView(object ProcessModel, object ProcessDataCollection);
- **Esito:** Superato;

4.4.1.18 TestCheckStep

- **Descrizione:** verifica della corretta realizzare i*widget* che consentono di gestire l'approvazione dei passi che richiedono intervento umano;
- **Metodo:** TestView(object ProcessModel, object ProcessDataCollection);
- **Esito:** Superato;

4.4.2 Test di unità per presenter

Per le classi *Presenter* vengono verificati i metodi della classe, per controllare che ritornino il valore atteso.

4.4.2.1 Router

- **Classe utilizzata per i test:** testRouter;
- **Descrizione:** Verificare che la classe permetta l'inizializzazione e la gestione delle pagine partendo dagli eventi;
- **Verifica dei metodi:**
  - `+ void home():`
    gestisce l'evento di $routing_G$ *home*;
  - `+ void processes():`
    gestisce l'evento di $routing_G$ *processes*;
  - `+ void newProcess():`
    gestisce l'evento di $routing_G$ *newProcess*;
  - `+ void checkStep():`
    gestisce l'evento di $routing_G$ *checkStep*;
  - `+ void process():`
    gestisce l'evento di $routing_G$ *process*;
  - `+ void register():`
    gestisce l'evento di $routing_G$ *register*;
  - `+ void user():`
    gestisce l'evento di $routing_G$ *user*;
  - `+ bool checkSession(String pageId):`
    ritorna `true` solo se l'utente è autenticato; in caso contrario crea e renderizza la pagina di *login*;

  - `+ void load(String resource, String pageId):`
    crea e aggiunge una vista di tipo *resource* al campo dati `this.views`, all'indice *pageId*;
  - `+ void changePage(String pageId):`
    imposta la pagina con id *pageId* come attiva, ed esegue la transizione di cambio pagina.

- Esito: Superato;

4.4.2.2 Login

- **Classe utilizzata per i test:** testLogin;
- **Descrizione:** Verifica della corretta gestione delle richieste di autenticazione al sistema;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ void login(Event event):`
    effettua una richiesta di *login*, utilizzando il campo dati com.si/ri/us.se/quen/zia/to/re.cli/ent.mo/del per comunicare con il $server_G$.

- Esito: Superato;

4.4.2.3 MainUser

- **Classe utilizzata per i test:** testMainUser;
- **Descrizione:** verifica della gestione generale della logica delle funzionalità utente;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;

  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe.

- Esito: Superato;

4.4.2.4 Register

- **Classe utilizzata per i test:** testRegister;
- **Descrizione:** verifica della richieste di registrazione da parte dell'utente;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ void register(Event event):`
    verifica della richiesta di registrazione, utilizzando il campo dati com.si/ri/us.se/quen/zia/to/re.cli/ent.mo/del per comunicare con il $server_G$.
- Esito: Superato;

4.4.2.5 UserData

- **Classe utilizzata per i test:** testUserData;
- **Descrizione:** verifica della corretta gestione della visualizzazione e la modifica dei dati dell'utente;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;

- `+ void editData():`
  verifica del corretto salvataggio dei dati modificati dall'utente nel $server_G$.

- Esito: Superato;

4.4.2.6 OpenProcess

- **Classe utilizzata per i test:** testOpenProcess;
- **Descrizione:** Classe che ha il compito di selezionare, ricercare e aprire un processo fra quelli eseguibili;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ void update():`
    verifica del corretto aggiornamento del campo dati `collection` e relativa comunicazione con il $server_G$.
- Esito: Superato;

4.4.2.7 ManagementProcess

- **Classe utilizzata per i test:** testManagementProcess;
- **Descrizione:** Classe che ha il compito di gestire e accedere alle informazioni relative allo stato del processo selezionato.
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ void update():`
    verifica del corretto aggiornamento dei campi dati `process` e `processData` relativa comunicazione con il $server_G$;

- `+ String getParam(String param):`
  ritorna il valore del parametro *param* se presente nella URL_G.

- Esito: Superato;

4.4.2.8 PrintReport

- **Classe utilizzata per i test:** testPrintReport;
- **Descrizione:** verifica della corretta creazione del report di fine processo;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
- Esito: Superato;

4.4.2.9 SendData

- **Classe utilizzata per i test:** testSendData;
- **Descrizione:** verifica della corretta gestione, inserimento e invio di dati da parte degli utenti, per completare il passo corrente;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ bool getData():`
    controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge alla collezione `processData`, altrimenti ritorna `false`;
  - `+ bool saveData():`
    utilizza metodi del campo dati `processData`, per inviare i dati raccolti al $server_G$.
- Esito: Superato;

4.4.2.10 SendText

- **Classe utilizzata per i test:** testSendText;
- **Descrizione:** Classe che permette l'inserimento e il controllo di dati testuali inseriti dagli utenti;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ bool getData(ProcessDataModel data):`
    controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.
- Esito: Superato;

4.4.2.11 SendNumb

- **Classe utilizzata per i test:** testSendNumb;
- **Descrizione:** Classe che ha il compito di permettere l'inserimento e il controllo di dati numerici inseriti dagli utenti;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ bool getData(ProcessDataModel data):`
    controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.
- Esito: Superato;

4.4.2.12 SendImage

- **Classe utilizzata per i test:** testSendImage;
- **Descrizione:** Classe che gestisce l'inserimento e il controllo di immagini inserite dagli degli utenti;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ bool getData(ProcessDataModel data):`
    controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.
- Esito: Superato;

4.4.2.13 SendPosition

- **Classe utilizzata per i test:** testSendPosition;
- **Descrizione:** verifica della corretta gestione della posizione geografica dell'utente;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ bool getData(ProcessDataModel data):`
    controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.
- Esito: Superato;

4.4.2.14 MainProcessOwner

- **Classe utilizzata per i test:** testMainProcessOwner;
- **Descrizione:** verifica della gestione generale della logica delle funzionalità *Process Owner*$_G$;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina *HTML*$_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina *HTML*$_G$ del *template* campo dati della classe;
- Esito: Superato;

4.4.2.15 OpenProcess

- **Classe utilizzata per i test:** testOpenProcess;
- **Descrizione:** Classe che ha il compito di gestire la ricerca e la selezione di un processo;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina *HTML*$_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina *HTML*$_G$ del *template* campo dati della classe;
  - `+ void update():`
    aggiorna il campo dati `collection` comunicando con il *server*$_G$.
- Esito: Superato;

4.4.2.16 NewProcess

- **Classe utilizzata per i test:** testNewProcess;
- **Descrizione:** Classe che ha il compito di gestire la logica della definizione di un nuovo processo;

- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe, e che questo sia compilato con gli eventuali errori(effettuare due prove);
  - `+ void newStep():`
    utilizza la classe `com.si/ri/us.se/quen/zia/to/re.cli/ent.pre/sen/-ter.pro/cess/ow/ner.Add/Step` per definire e aggiungere un nuovo passo al processo `model`;
  - `+ bool getData():`
    controlla se i dati inseriti dal *process owner*$_G$ sono corretti: se lo sono ritorna `true` e li aggiunge al processo `model`, altrimenti ritorna `false`;
  - `+ bool saveProcess():`
    utilizza metodi del campo dati `collection`, per inviare il processo `model` al *server*$_G$.
- **Esito:** Superato;

4.4.2.17 AddStep

- **Classe utilizzata per i test:** testAddStep;
- **Descrizione:** Classe che ha il compito di gestire la logica di definizione dei passi di un processo;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe, e che questo sia compilato con gli eventuali errori(effettuare due prove);
  - `+ bool getData():`
    controlla se i dati inseriti dal *process owner*$_G$ sono corretti: se lo sono ritorna `true` e li aggiunge al passo `model`, altrimenti ritorna `false`.

- **Esito:** Superato;

4.4.2.18 ManageProcess

- **Classe utilizzata per i test:** testManageProcess;
- **Descrizione:** Classe che ha il compito di gestire e accedere alle informazioni relative allo stato dei processi e ai dati inviati dagli utenti. Le operazioni di gestione dello stato comprendono la terminazione e l'eliminazione di un processo;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ void update():`
    aggiorna i campi dati `process` e `processData` comunicando con il $server_G$;
  - `+ String getParam(String param):`
    ritorna il valore del parametro *param* se presente nella URL_G;
- **Esito:** Superato;

4.4.2.19 CheckStep

- **Classe utilizzata per i test:** testCheckStep;
- **Descrizione:** Verifica della logica per l'approvazione di un passo che necessita dell'intervento dell'utente;
- **Verifica dei metodi:**
  - `+ void initialize():`
    verifica della corretta aggiunta di una pagina $HTML_G$ associata al componente;
  - `+ void render():`
    verifica della corretta aggiunta alla pagina $HTML_G$ del *template* campo dati della classe;
  - `+ void update():`
    aggiorna il campo dati `processData` comunicando con il $server_G$;

  - `+ String getParam(String param):`
    ritorna il valore del parametro *param* se presente nella URL_G;
  - `+ void approveData():`
    salva nel *server* lo stato approvato ai dati della collezione *processData* dei quali il *process* $owner_G$ ha richiesto l'approvazione;
  - `+ void rejectData():`
    salva nel *server* lo stato approvato ai dati della collezione *processData* che il *process* $owner_G$ ha respinto;

- **Esito:** Superato;

4.4.3 Test di unità per model

Per le classi *Model* vengono verificati i metodi della classe, per controllare che ritornino il valore atteso.

4.4.3.1 UserModel

- **Classe utilizzata per i test:** testUserModel;
- **Descrizione:** Classe che permette di gestire i dati di una sessione di un utente autenticato o di un *Process Owner*$_G$;
- **Verifica dei metodi:**
  - `+ void login(String username, String password):`
    delega al server il controllo delle credenzili e, al completamento della richiesta, salva i dati di sessione in caso di successo;
  - `+ void logout():`
    cancella di dati di sessione dell'utente;
  - `+ void signup():`
    effettua una richiesta di registrazione al $server_G$ inviando i dati della classe.
- **Esito:** Superato;

4.4.3.2 ProcessModel

- **Classe utilizzata per i test:** testProcessModel;
- **Descrizione:** Classe che permette di gestire i dati di un processo, e di salvarli o recuperarli dal $server_G$;
- **Verifica dei metodi:**

- `+ void fetchProcess():`
  recupera dal $server_G$ i dati del processo, e i dati dei passi che assegna alla collezione `steps`, sincronizzando le operazioni.

- **Esito:** Superato;

4.4.3.3 ProcessDataModel

- **Classe utilizzata per i test:** testProcessDataModel;
- **Descrizione:** verifica del corretto invio di dati da un utente relativi ad un processo, e e il relativo salvataggio e recupero dal $server_G$;
- **Verifica dei metodi:**
  - `+ void subscribe(bool subscription):`
    effettua una richiesta di iscrizione o disiscrizione al $server_G$ a seconda del valore del parametro *subscription*, riguardante il processo con id `idProcesso`;
  - `+ void sendData(int nextStep):`
    invia al $server_G$ i dati della classe e l'id del prossimo passo da eseguire, che identifica una condizione del processo con id `idProcesso`.
- **Esito:** Superato;

4.4.3.4 ProcessCollection

- **Classe utilizzata per i test:** testProcessCollection;
- **Descrizione:** Classe che permette di gestire un insieme di dati inviati da un utente relativi ad un processo;
- **Verifica dei metodi:**
  - `+ void fetchProcesses():`
    verifica del corretto ritorno dal server della lista dei processi a cui l'utente identificato dai dati di sessione può accedere;
  - `+ void saveProcess(ProcessModel process):`
    aggiunge il processo `process` alla collezione dei processi nel $serve_G$.
- **Esito:** Superato;

4.4.3.5 ProcessDataCollection

- **Classe utilizzata per i test:** testProcessDataCollection;
- **Descrizione:** Classe che permette di gestire un insieme di dati inviati dagli utenti;
- **Verifica dei metodi:**
  - `+ void fetchProcessData(int stepId):`
    richiede al $server_G$ la lista dei dati inviati riguardanti il passo con id `stepId`, ai quali l'utente identificato dai dati di sessione può accedere;
  - `+ void fetchStepData(int processId):`
    richiede al $server_G$ la lista dei dati inviati riguardanti il processo con id `processId`, ai quali l'utente identificato dai dati di sessione può accedere;
  - `+ void fetchWaitingData():`
    richiede al $server_G$ la lista dei dati inviati che richiedono controllo umano;
  - `+ void approveData(int stepId, String username):`
    invia al $server_G$ la richiesta di approvazione dei dati riguardanti il passo con id `stepId` e l'utente con username `username`.
  - `+ void rejectData(int stepId, String username):`
    invia al $server_G$ l'esito negativo del controllo dei dati riguardanti il passo con id `stepId` e l'utente con username `username`.
- **Esito:** Superato;

4.4.3.6 IDataAcessObject

- **Classe utilizzata per i test:** testIDataAcessObject;
- **Descrizione:** Interfaccia che permette di gestire la comunicazione e l'interrogazione con il *database*.
- **Verifica dei metodi:**
  - `+ void setJdbcTemplate(JdbcTemplate jdbcTemplate):`
    imposta i parametri per l'accesso alla sorgente dei dati;
  - `+ ITransferObject getAll():`
    ritorna tutti i dati di competenza della classe che estende questa interfaccia.
- **Esito:** Superato;

4.4.3.7 UserDao

- **Classe utilizzata per i test:** testUserDao;
- **Descrizione:** classe che si occupa delle interrogazioni del *database* relative agli utenti del sistema.
- **Verifica dei metodi:**
  - `+ User getUser(String userName):`
    ritorna l'utente con il nome utente specificato;
  - `+ List<User> getAllUser():`
    ritorna tutti gli utenti;
  - `+ boolean insertUser(User user) :`
    aggiunge l'utente passato come parametro;
  - `+ public boolean updateUser(User user) :`
    verifica del corretto aggiornamento dei dati dell'utente con il nome utente corrispondente a quello dell'utente passato, con i dati dell'utente passato.
- **Esito:** Superato;

4.4.3.8 ProcessDao

- **Classe utilizzata per i test:** testProcessDao;
- **Descrizione:** classe che si occupa delle interrogazioni del *database* relative ai processi.
- **Verifica dei metodi:**
  - `+ Process getProcess(int id)):`
    ritorna il processo con l'`id` specificato;
  - `+ List<Process> getAllProcess():`
    ritorna tutti i processi;
  - `+ boolean insertProcess(Process process) :`
    aggiunge il processo passato come parametro;
  - `+ public boolean updateProcess(Process process) :`
    aggiorna i dati del processo con lo stesso `id` di quello del processo passato, con i dati del processo passato.
- **Esito:** Superato;

4.4.3.9 ProcessOwnerDao

- **Classe utilizzata per i test:** testProcessOwnerDao;
- **Descrizione:** classe che si occupa delle interrogazioni del *database* relative all'autenticazione del *ProcessOwner*.
- **Verifica dei metodi:**
  - `+ Process getProcessOwner(int id)):`
    ritorna l'oggetto rappresentante il *ProcessOwner*.
- **Esito:** Superato;

4.4.3.10 StepDao

- **Classe utilizzata per i test:** testStepDao;
- **Descrizione:** classe che si occupa delle interrogazioni del *database* relative a tutte le operazioni sui passi dei processi.
- **Verifica dei metodi:**
  - `+ Step getStep(int id):`
    Ritorna il passo con l'`id` specificato;
  - `+ List<Step> getAllStep():`
    Ritorna tutti i passi;
  - `+ List<Step> getStepOf(int ProcessId):`
    Ritorna tutti i passi appartenenti al processo di cui si è passato l'`id`;
  - `+ boolean insertStep(Step step) :`
    Aggiunge il passo passato come parametro;
  - `+ public boolean updateStep(Step step) :`
    Aggiorna i dati del passo con l'`id` corrispondente a quello del passo passato, con i dati del passo passato;
  - `+ List<UserStep> userStep(String userName)`
    Ritorna una lista di oggetti informativi sullo stato dei passi in corso da parte dell'utente di cui si è passato il nome utente;
  - `+ List<UserStep> userProcessStep(String userName, processId)`
    Ritorna una lista di oggetti informativi sullo stato dei passi in corso appartenenti al processo di cui si è passato l'`id` da parte dell'utente di cui si è passato il nome utente;

  - `+ List<DataSent> getData(Step step)`
    Ritorna tutti i dati da tutti gli utenti relativi al passo passato;
  - `+ List<DataSent> getData(String userName, Step step)`
    Ritorna tutti i dati inviati dall'utente di cui si è passato il nome utente relativi al passo passato;
  - `+ boolean completeStep(String userName, Step step, List<DataSent> data, Step next)`
    Notifica e aggiorna nel *database* lo stato dell'utente quando completa o tenta di completare un passo.

- **Esito:** Superato;

4.4.3.11 User

- **Classe utilizzata per i test:** testUser;
- **Descrizione:** verifica della corretta gestione gli utenti del sistema e corretta interazione con il database.
- **Verifica dei metodi:**
  - `+ String getUserName():`
    Ritorna il nome utente;
  - `+ void setUserName(String userName):`
    Imposta il nome utente;
  - `+ String getPassword():`
    Ritorna la password dell'utente;
  - `+ void setPassword(String password):`
    Imposta la password dell'utente;
  - `+ String getName():`
    Ritorna il nome anagrafico dell'utente;
  - `+ void setName(String name):`
    Imposta il nome anagrafico dell'utente;
  - `+ String getSurName():`
    Ritorna il cognome dell'utente;
  - `+ void setSurName(String surName):`
    Imposta il cognome dell'utente;
  - `+ Date getDateOfBirth():`
    Ritorna la data di nascita dell'utente;

- `+ void setDateOfBirth(Date dateOfBirth):`
  Imposta la data di nascita dell'utente;
- `+ String getEmail():`
  Ritorna l'indirizzo di posta elettronica dell'utente;
- `+ void setEmail(String email):`
  Imposta l'indirizzo di posta elettronica dell'utente;
- `+ int getId():`
  Ritorna il codice `id` associato all'utente;
- `+ void setId(int id):`
  Imposta il codice `id` associato all'utente.

- **Esito:** Superato;

4.4.3.12 Process

- **Classe utilizzata per i test:** testProcess;
- **Descrizione:** verifica della corretta gestione dei processi.
- **Verifica dei metodi:**
  - `+ String getName():`
    Ritorna il nome del processo;
  - `+ void setName(String name):`
    Imposta il nome del processo;
  - `+ String getDescription():`
    Ritorna la descrizione del processo;
  - `+ void setDescription(String description):`
    Imposta la descrizione del processo;
  - `+ int getCompletionsMax():`
    Restituisce il numero massimo di completamenti del processo;
  - `+ void setCompletionsMax(int completionsMax):`
    Imposta il numero massimo di completamenti del processo;
  - `+ Date getDateOfTermination():`
    Ritorna data di terminazione del processo;
  - `+ void setDateOfTermination(Date dateOfTermination):`
    Imposta la data di terminazione del processo;
  - `+ boolean isTerminated():`
    Ritorna vero se il processo è terminato;

  - `+ void setTerminated(boolean terminated):`
    Imposta vero se il processo è terminato;
  - `+ int getMaxTree():`
    Ritorna il massimo alberi del processo;
  - `+ void setMaxtree(int maxTree):`
    Imposta il massimo alberi del processo;
  - `+ List<Integer> getStepsId():`
    Ritorna lista di codici `id` relativi ai passi del processo;
  - `+ void setStepsId(List<Integer> stepsId):`
    Imposta lista di codi `id` relativi ai passi del processo;
  - `+int getId():`
    Ritorna codice identificativo `id` associato al processo;
  - `+void setId(int id):`
    Imposta codice identificativo `id` associato al processo.

- **Esito:** Superato;

4.4.3.13 Step

- **Classe utilizzata per i test:** testStep;
- **Descrizione:** Verifica della corretta gestione dei passi.
- **Verifica dei metodi:**
  - `+ int getId():`
    Ritorna codice identificativo `id` associato al passo;
  - `+ void setId(int id):`
    Imposta codice identificativo `id` associato al passo;
  - `+ String getDescription():`
    Ritorna descrizione del passo;
  - `+ void setDescription(String description):`
    Imposta descrizione del passo;
  - `+ List<Data> getData():`
    Ritorna lista con i campi dato del passo;
  - `+ void setData(List<Data> data):`
    Imposta lista con i campi dato del passo;
  - `+ List<Condition> getConditions():`
    Ritorna lista delle condizioni di avanzamento del passo;

  - `+ void setConditions(List<Condition conditions):`
    Imposta lista delle condizioni di avanzamento del passo;
  - `+ int getProcessId():`
    Ritorna codice `id` associato al processo padre;
  - `+ void setProcessId(int processId):`
    Imposta codice `id` associato al processo padre;
  - `+ boolean isFirst():`
    Ritorna vero se il passo è primo per il processo padre;
  - `+ void setFirst():`
    Imposta vero se il passo è primo per il processo padre.

- **Esito:** Superato;

4.4.3.14 Data

- **Classe utilizzata per i test:** testData;
- **Descrizione:** verifica della corretta gestione dei dati;
- **Verifica dei metodi:**
  - `+ String getName():`
    Ritorna il nome del campo dati;
  - `+ void setName(String name):`
    Imposta il nome del campo dati;
  - `+ DataType getType():`
    Ritorna il tipo di dato richiesto dal campo;
  - `+ void setType(DataType type):`
    Imposta il tipo di dato richiesto dal campo;
  - `+ int getId():`
    Ritorna il codice `id` associato al campo dati;
  - `+ void setId(int id):`
    Imposta il codice `id` associato al campo dati.
- **Esito:** Superato;

4.4.3.15 Condition

- **Classe utilizzata per i test:** testCondition;

- **Descrizione:** verifica della corretta gestione delle condizioni di terminazione passo;
- **Verifica dei metodi:**
  - `+ int getId():`
    Ritorna il codice `id` associato alla condizione di avanzamento;
  - `+ void setId(int id):`
    Imposta il codice `id` associato alla condizione di avanzamento;
  - `+ boolean isRequiresApproval():`
    Ritorna vero se è richiesta l'approvazione del *Process Owner*;
  - `+ void setRequiresApproval(boolean requiresApproval):`
    Imposta vero se è richiesta l'approvazione del *Process Owner*;
  - `+ List<Constraint> getConstraints():`
    ritorna lista di vincoli che soddisfano la condizione di avanzamento;
  - `+ void setConstraints(List<Constraint> constraints):`
    Imposta lista di vincoli che soddisfano la condizione di avanzamento;
  - `+ boolean isOptional():`
    Ritorna vero se la condizione è opzionale per l'avanzamento;
  - `+ void setOptional(boolean optional):`
    Imposta vero se la condizione è opzionale per l'avanzamento;
  - `+ int getNextStepId():`
    Ritorna codice `id` del passo successivo;
  - `+ void setNextStepId(int nextStepId):`
    Imposta codice `id` del passo succesivo.
- **Esito:** Superato;

4.4.3.16 Constraint

- **Classe utilizzata per i test:** testConstraint;
- **Descrizione:** verifica della corretta gestione dei vincoli di passo;
- **Verifica dei metodi:**
  - `+ Data getAssociatedData():`
    Ritorna il campo dati su cui è posto il vincolo;
  - `+ void setAssociatedData(Data associatedData):`
    Imposta il campo dati sui cui è posto il vincolo.
- **Esito:** Superato;

4.4.3.17 NumericConstraint

- **Classe utilizzata per i test:** testNumeriConstraint;
- **Descrizione:** corretta gestione dei vincoli numerici.
- **Verifica dei metodi:**
  - `+ int getId():`
    Ritorna codice `id` del vincolo;
  - `+ void setId(int id):`
    Imposta codice `id` del vincolo;
  - `+ int getMinDigits():`
    Ritorna minimo numero di cifre;
  - `+ void setMinDigits(int minDigits):`
    Imposta minimo numero di cifre;
  - `+ int getMaxDigits():`
    Ritorna massimo numero di cifre;
  - `+ void setMaxDigits(int maxDigits):`
    Imposta massimo numero di cifre;
  - `+ boolean isDecimal():`
    Ritorna vero se atteso un decimale;
  - `+ void setDecimal(boolean decimal):`
    Imposta vero se atteso un decimale;
  - `+ double getMinValue():`
    Ritorna valore minimo;
  - `+ void setMinValue(double minValue):`
    Imposta valore minimo;
  - `+ double getMaxValue():`
    Ritorna valore massimo;
  - `+ void setMaxValue(double maxValue):`
    Imposta valore massimo;
- **Esito:** Superato;

4.4.3.18 TemporalConstraint

- **Classe utilizzata per i test:** testTemporalContraint;
- **Descrizione:** verifica corretta gestione dei vincoli temporali;

- **Verifica dei metodi:**
  - `+ int getId():`
    Ritorna codice `id` del vincolo;
  - `+ void setId(int id):`
    Imposta codice `id` del vincolo;
  - `+ Date getBegin():`
    Ritorna inizio arco temporale valido;
  - `+ void setBegin(Date begin):`
    Imposta inizio arco temporale valido;
  - `+ Date getEnd():`
    Ritorna fine arco temporale valido;
  - `+ void setEnd(Date end):` Imposta fine arco temporale valido.
- **Esito:** Superato;

4.4.3.19 GeographicConstraint

- **Classe utilizzata per i test:** testGeographicContraint;
- **Descrizione:** verifica della corretta gestione dei vincoli di posizione.
- **Verifica dei metodi:**
  - `+ int getId():`
    Ritorna codice `id` del vincolo;
  - `+ void setId(int id):`
    Imposta codice `id` del vincolo;
  - `+ double getLatitude():`
    Ritorna latitudine richiesta;
  - `+ void setLatitude(double latitude):`
    Imposta latitudine richiesta;
  - `+ double getLongitude():`
    Ritorna longitudine richiesta;
  - `+ void setLongitude(double longitude):`
    Imposta longitudine richiesta;
  - `+ double getAltitude():`
    Ritorna altitudine richiesta;
  - `+ void setAltitude(double altitude):`
    Imposta altitudine richiesta;

  - `+ double getRadius():`
    Ritorna raggio di tolleranza;
  - `+ void setRadius(double radius):`
    Imposta raggio di tolleranza.

- **Esito:** Superato;

4.4.3.20 DataSent

- **Classe utilizzata per i test:** testDataSent;
- **Descrizione:** Verifica della corretta gestione dei dati inseriti dagli utenti;
- **Verifica dei metodi:**
  - `+ String getUser():`
    Ritorna nome utente dell'utente che ha inviato il dato;
  - `+ void setUser(String user):`
    Imposta nome utente dell'utente che ha inviato il dato;
  - `+ DataType getType():`
    Ritorna il tipo del dato inviato;
  - `+ void setType(DataType type):`
    Imposta il tipo del dato inviato;
  - `+ IDataValue getValue():`
    Ritorna oggetto con il valore del dato;
  - `+ void setValue(IDataValue value):`
    Imposta oggetto con il valore del dato;
  - `+ int getStepId():`
    Ritorna codice `id` del passo richiedente il dato;
  - `+ void setStepId(int stepId):`
    Imposta codice `id` del passo richiedente il dato.
- **Esito:** Superato;

4.4.3.21 IDataValue

- **Classe utilizzata per i test:** testIDataValue;
- **Descrizione:** verifica del corretto ritorno per i dati ricevuti dagli utenti;
- **Verifica dei metodi:**

  - `+ int getId():`
    Ritorna codice `id` associato al valore;
  - `+ void setId(int id):`
    Imposta codice `id` associato al valore.

- **Esito:** Superato;

4.4.3.22 TextualValue

- **Classe utilizzata per i test:** testTextualValue;
- **Descrizione:** verifica del corretto ritorno e importazione dei dati di tipo testuale;
- **Verifica dei metodi:**
  - `+ String getValue():`
    Ritorna valore testuale;
  - `+ void setValue(String value):`
    Imposta valore testuale.
- **Esito:** Superato;

4.4.3.23 NumericValue

- **Classe utilizzata per i test:** testNumericValue;
- **Descrizione:** verifica del corretto ritorno e importazione dei dati di tipo numerico;
- **Verifica dei metodi:**
  - `+ double getValue():`
    Ritorna valore numerico;
  - `+ void setValue(double value):`
    Imposta valore numerico.
- **Esito:** Superato;

4.4.3.24 ImageValue

- **Classe utilizzata per i test:** testImageValue;
- **Descrizione:** verifica del corretto ritorno e importazione dei dati di tipo immagine;
- **Verifica dei metodi:**
  - `+ String getImageUrl():`
    Ritorna percorso *URL* dell'immagine;
  - `+ void setImageUrl(String imageUrl):`
    Imposta percorso *URL* dell'immagine.
- **Esito:** Superato;

4.4.3.25 GeographicValue

- **Classe utilizzata per i test:** testGeographicValue;
- **Descrizione:** verifica del corretto ritorno e importazione dei dati di tipo geografico;
- **Verifica dei metodi:**
  - `+ double getLatitude():`
    Ritorna latitudine;
  - `+ void setLatitude(double latitude):`
    Imposta latitudine;
  - `+ double getLongitude():`
    Ritorna longitudine;
  - `+ void setLongitude(double longitude):`
    Imposta longitudine;
  - `+ double getAltitude():`
    Ritorna altitudine;
  - `+ void setAltitude(double altitude):`
    Imposta altitudine.

4.4.3.26 UserStep

- **Classe utilizzata per i test:** testUserStep;
- **Descrizione:** verifica della corretta gestione dei passi che sono in corso;
- **Verifica dei metodi:**
  - `+ int getCurrentStepId():`
    Ritorna il codice `id` del passo attuale;
  - `+ void setCurrentStepId(int currentStepId):`
    Imposta il codice `id` del passo attuale;
  - `+ stepStates getState():`
    Ritorna stato avanzamento;
  - `+ void setStates(stepStates state):`
    Imposta stato avanzamento;
  - `+ String getUser():`
    Restituisce nome utente dell'utente in caso;
  - `+ void setUser(String user):`
    Imposta nome utente dell'utente in caso.
- **Esito:** Superato;

4.4.3.27 ProcessOwner

- **Classe utilizzata per i test:** testProcessOwner;
- **Descrizione:** verifica della corretta gestione dell'ambito *process owner*;
- **Verifica dei metodi:**
  - `+ String getUserName():`
    Ritorna il nome utente del *Process Owner*;
  - `+ void setUserName(String userName):`
    Imposta il nome utente del *Process Owner*;
  - `+ String getPassword():`
    Ritorna la password del *Process Owner*;
  - `+ void setPassword(String password):`
    Imposta la password del *Process Owner*.
- **Esito:** Superato;

4.4.4 Descrizione dei test di validazione

I test di validazione saranno definiti in seguito.

A Appendice

A.1 Ciclo di Deming

Alla luce delle informazioni sopra citate il team ha deciso di adottare la politica del ciclo PDCA per le attività da svolgere. Lo stesso, oltre a fornire supporto nella pianificazione garantisce un elevato standard qualitativo tramite il *Miglioramento continuo*, che è alla base del ciclo di Deming.

Figura 2: Ciclo di Deming

- ***Plan***: pianificazione che prevede la definizione di procedure, risorse, scadenze e responsabilità ;
- ***Do***: esecuzione delle attività pianificate;
- ***Check***: controllo dei risultati ottenuti e confronto con quelli pianificati;
- ***Act***: Analisi dei risultati ottenuti e modifica o definizione di nuove procedure che permettano di evitare gli aspetti critici dei processi in esame.

L'adozione del PDCA garantisce un continuo arricchimento dei processi tramite dei cambiamenti e delle riorganizzazioni. Alla base di questo, ci deve essere una conoscenza specifica delle Norme di progetto v3.0.0 da parte di tutti i componenti del team. Inoltre, queste migliorie aumentano i costi di gestione e per questo devono essere valutati dal *Responsabile di progetto*.

A.2 ISO/IEC 9126

Lo standard ISO/IEC 9126 descrive gli obiettivi qualitativi di prodotto e delinea in generale le metriche per misurare il raggiungimento di tale obiettivo (figura 3). In questo standard i criteri sono divisi in 3 aree diverse:

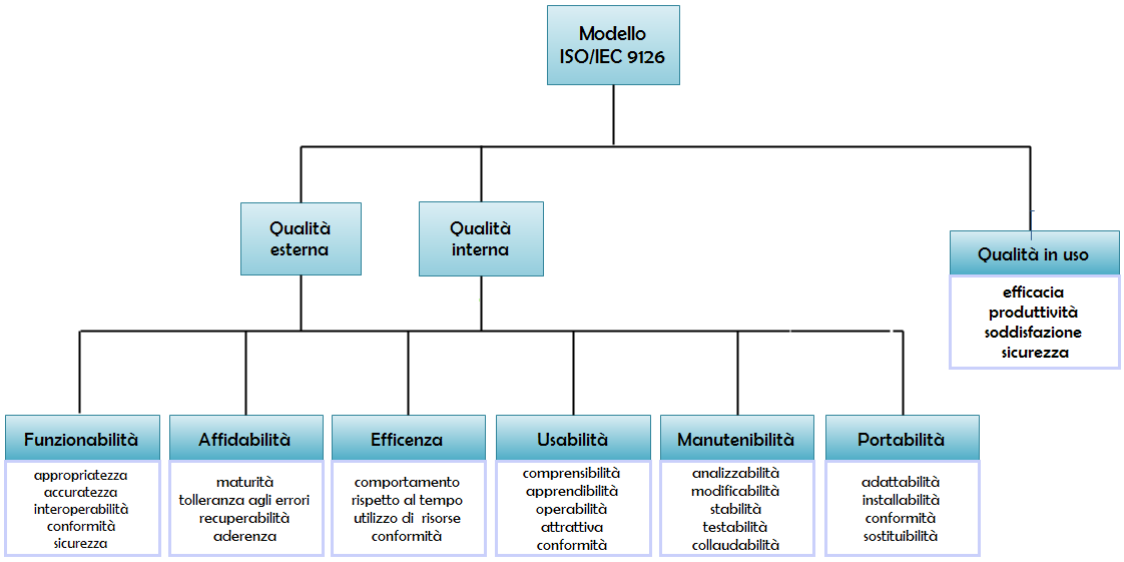

Figura 3: Caratteristiche qualitative definite dal modello ISO/IEC 9126

- **Qualità in uso**:è la qualità del software dal punto di vista dell'utilizzatore;
- **Qualità esterna**:è la qualità del software dal punto di vista esterno nel momento in cui esso viene eseguito e testato in ambiente di prova;
- **Qualità interna**:è la qualità del software vista dall'interno e quindi sono le caratteristiche implementative del software quali architettura e codice che ne deriva.

Non avendo modo di verificare la qualità in uso, *Sirius*ha deciso di lavorare su qualità interna ed esterna definendo apposite metriche.

- **Funzionalità** è la capacità di un prodotto software di fornire funzioni che soddisfano esigenze stabilite, necessarie per operare sotto condizioni specifiche.
  - Appropriatezza: rappresenta la capacità del prodotto software di fornire un appropriato insieme di funzioni per gli specificati compiti ed obiettivi prefissati all'utente.
  - Accuratezza: la capacità del prodotto software di fornire i risultati concordati o i precisi effetti richiesti;
  - Interoperabilità: è la capacità del prodotto software di interagire ed operare con uno o più sistemi specificati;
  - Conformità: la capacità del prodotto software di aderire a standard, convenzioni e regolamentazioni rilevanti al settore operativo a cui vengono applicate;

  - Sicurezza: la capacità del prodotto software di proteggere informazioni e dati negando in ogni modo che persone o sistemi non autorizzati possano accedervi o modificarli, e che a persone o sistemi effettivamente autorizzati non sia negato l'accesso ad essi.

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

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

- **Efficienza**: è la capacità di fornire appropriate prestazioni relativamente alla quantità di risorse usate.
  - Comportamento rispetto al tempo: è la capacità di fornire adeguati tempi di risposta, elaborazione e velocità di attraversamento, sotto condizioni determinate;

  - Utilizzo delle risorse: è la capacità di utilizzo di quantità e tipo di risorse in maniera adeguata.
  - Conformità: è la capacità di aderire a standard e specifiche sull'efficienza$_G$.

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

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

A.3 Capability Maturity Model Integration (CMMI)

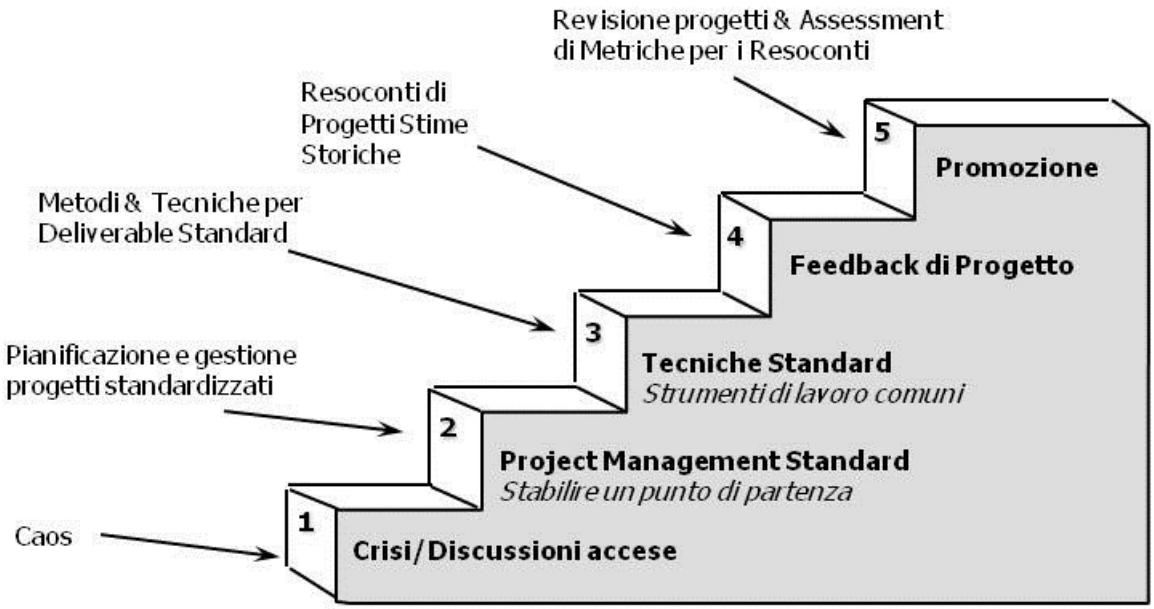

Figura 4: Livello di maturità delle procedure

Il modello identifica cinque livelli di maturità dei processi all'interno di un'organizzazione: dal Livello 1, il processo$_G$ più immaturo, o caotico, al Livello 5, il processo$_G$ più maturo, o di qualità

- **Livello di maturità 1**
  Partendo dall'assunzione che una pratica non può essere migliorata se non è ripetibile, il livello di maturità iniziale vede l'organizzazione effettuare la gestione delle persone tramite procedure ad hoc, spesso informali e non ripetibili se non sporadicamente. Un esempio tipico è data dall'impossibilità, da parte delle persone, di assicurare la data di rilascio del software, indipendentemente dalle tecnologie utilizzate o dalla preparazione delle persone. Un'altra conseguenza tipica è la gestione incontrollata delle modifiche ai requisiti con conseguenze negative sui piani di lavoro. L'attività principale da compiere in questo periodo è quella di aiutare l'organizzazione a rimuovere ogni impedimento alla ripetibilità delle pratiche;

- **Livello di maturità 2**
  Al livello di maturità 2, l'organizzazione stabilisce una politica per divulgare presso tutti i gruppi di lavoro i processi stabiliti. Prima di pensare ad ogni miglioramento, l'organizzazione deve assicurare un ambiente di lavoro stabile in cui eseguire in maniera ripetibile i propri processi. Finché si opera in una modalità non strutturata, il management è troppo occupato nel controllo quotidiano

delle operazione per poter pensare a qualsivoglia cambiamento in ottica di miglioramento. L'obiettivo principale del livello 2 è quindi quello di permettere alle persone di svolgere il proprio lavoro in maniera ripetibile, in base a quanto già fatto in passato ed in base all'esperienza maturata. A questo livello il management lascia ai responsabili dei singoli gruppi il compito di controllare il lavoro quotidiano, dedicandosi a sua volta al controllo dei risultati finali e della baseline (ed alle rispettive modifiche). Solo quando le pratiche stabilite saranno eseguite con naturalezza dall'intero gruppo, questo potrà iniziare il periodo successivo di utilizzo di processi comuni a tutto il team;

- **Livello di maturità 3**
  Al livello di maturità 3, l'organizzazione seleziona le migliori pratiche e le include in un processo comune. Operando tutti con le stesse pratiche definite, l'organizzazione sarà in grado di valutare le pratiche con migliori performance nell'ambiente comune. Documentate nell'ambito del processo comune le pratiche, queste diventano anche lo strumento di apprendimento per le nuove persone. Le misure effettuate sulle pratiche di maggiore criticità sono registrate in un archivio ed utilizzare per effettuarne l'analisi. In tale modo si è creato il fondamento per una cultura di base comune all'organizzazione: un processo comune conosciuto ed applicato da tutti. E' il fondamento della cultura professionale di base dell'organizzazione;

- **Livello di maturità 4**
  Al livello di maturità 4, l'organizzazione inizia a gestire i processi in base ai risultati utilizzando l'analisi delle misure effettuate. Le attività sono svolte secondo i processi comuni definiti ed i risultati sono quindi più controllabili in base all'esperienza storica. Le deviazioni dai risultati attesi sono analizzate, le cause delle deviazioni individuate e le azioni correttive prese di conseguenza. I processi sono quindi gestiti quantitativamente ed i risultati sono prevedibili con maggiore cura. I risultati del business sono controllati da valori e non più dalle $milestone_G$ come prima. Si crea quindi la cultura per un vero miglioramento dei processi e quindi delle performance reali;

- **Livello di maturità 5**
  Al livello di maturità 5, l'organizzazione opera utilizzando in maniera ripetitiva i propri processi, ne valuta le performance quantitativamente ed opera per migliorarli di continuo. Gli eventuali difetti sono analizzati e le cause che li generano sono rimosse per evitare il loro ripetersi. Le persone sono culturalmente abituate ad eseguire i processi conosciuti ed il management a gestirli quantitativamente ed a migliorarli. Si crea anche la cultura dell'accettazione del cambiamento. L'organizzazione entra in un circolo virtuoso di miglioramento continuo;

B Resoconto attività di verifica

B.1 Riassunto dell'attività di verifica su RR

Sirius, ha deciso di valorizzare sopratutto i requisiti desiderabili per fare in modo di tenere alto l'indice di efficienza$_G$. Valutando, durante lo stato di avanzamento, quali di questi requisiti saranno successivamente sviluppati nella realizzazione del prodotto software. Questo tipo di scelte risultano infatti difficoltose allo stato attuale in quanto una pianificazione approfondita necessiterebbe di *lead time* $_G$ precisi che non sono al momento tra le conoscenze dei componenti del gruppo. Nel periodo di tempo che ha portato *Sirius* alla consegna di questa revisione sono stati verificati i **documenti** ed i **processi**.
I **documenti** sono stati verificati anche durante le operazioni di redazione per portare a conoscenza dei contenuti tutti i componenti del gruppo di lavoro. L'analisi statica, in primo luogo utilizzando la tecnica del *walkthrough*, ha portato alla redazione di una lista di controllo che verrà poi incrementata ed utilizzata nell'analisi finale del documento prima di procedere alla consegna. Una volta rilevati gli errori questi sono stati notificati al redattore che ha proceduto alla correzione, evidenziando gli errori frequenti che sono stati utilizzati per migliorare il processo di verifica. *Sirius* adotta il ciclo PDCA per rendere più efficiente$_G$ ed efficace$_G$ nel tempo il processo di verifica.
L'attività di verifica, inoltre, utilizzando la tecnica *inspection* è stata utilizzata principalmente per la verifica dei grafici dei casi d'uso. Per verificare la correttezza dei requisiti richiesti e la successiva completezza ci si è affidati ad un particolare strumento di tracciamento definito in Norme di progetto v3.0.0. L'avanzamento dei processi, dettato dal Piano Di Progetto v3.0.0, è stato mantenuto in controllo tramite una costante verifica delle metriche definite in questo documento e di cui troviamo una rappresentazione grafica in seguito.

B.2 Dettaglio dell'attività di verifica su RR

B.2.1 Documenti

Indice di *Gulpease* per i documenti redatti:

Documento	**Valore di accettazione**	**Esito**
Piano Di Progetto v3.0.0	>40	*Positivo*
Norme di progetto v3.0.0	>40	*Positivo*
Analisi dei requisiti v3.0.0	>40	*Positivo*
Piano Di Qualifica v3.0.0	>40	*Positivo*

Studio Di Fattibilità v2.0.0	>40	*Positivo*

Tabella 7: Esito del calcolo indice di Gulpease per ogni documento per RR

B.2.2 Processi

Sirius ha condotto l'attività di verifica per i processi. In questo primo periodo il processo di documentazione è predominante nella pianificazione delle risorse. Di seguito viene riportato l'indice **SV (*schedule variance*)** per le attività eseguite e i risultati sono i seguenti:

Attività	**Ore pianificate**	**Ore rilevate**	**SV rilevato**	**SV accettazione**
Norme di Progetto	17 H	17 H	0 H	> -1 H
Studio di Fattibilità	8 H	14 H	**-6 H**	> **-1 H**
Analisi dei Requisiti	70 H	68 H	2 H	> -4 H
Piano di Progetto	37 H	35 H	2 H	> -2 H
Piano di Qualifica	26 H	22 H	4 H	> -2 H

Tabella 8: Indice SV per le attività per RR

Da una prima analisi, si denota che *Sirius* ha pianificato in modo preciso le attività. L'attività di Studio di Fattibilità, essendo stato uno dei primi documenti che *Sirius* ha redatto, la pianificazione non è stata precisa questo ha portato ad un SV dell'attività fuori dal range di accettazione. Le cause di questo problema sono da ricercare anche nella poca confidenza con gli strumenti di *editor* testi e con gli strumenti di condivisione. La singola occorrenza del problema, non è quindi indice di allarme per gli altri processi che saranno pianificati nell'avanzamento del prodotto.

- SV-totale = 2 H;

SV-totale maggiore di zero denota che *Sirius* stà producendo più velocemente rispetto a quanto pianificato. Questo può essere una diretta conseguenza dell'aggiunta di uno *slack* temporale nella pianificazione delle attività. Il team ha valutato la possibilità di ridurre il tempo di *slack*, per fare in modo che la pianificazione corrisponda alla realtà; ma data la variabilità delle attività che *Sirius* intende svolgere nel *Sirius*ha effettuato l'analisi dei documenti e ha rilevato la conformità controllando l'indice di Gulpease di tutti i documenti prodotti o modificati in questo periodo di tempo.proseguo del progetto e la poca esperienza, è stato deciso di non modificare tale valore.
Di seguito viene riportato l'indice **BV (*budget variance*)** per le attività eseguite e i risultati sono i seguenti:

Attività	**Costo pianificato**	**Costo consuntivo**	**BV rilevato**	**BV limite**
Norme di Progetto	325.00 €	325.00 €	0.00 €	> -32.50 €
Studio di Fattibilità	180.00 €	310.00 €	**-130.00 €**	> **-18.00 €**
Analisi dei Requisiti	1630.00 €	1600.00 €	30.00 €	> -16.30 €
Piano di Progetto	1005.00 €	945.00 €	60.00 €	> -10.05 €
Piano di Qualifica	490.00 €	420.00 €	70.00 €	> -49.00 €

Tabella 9: Indice BV per le attività per RR

Come descritto sopra per SV, anche BV denota che il preventivo di costo previsto per le attività svolte è stato corretto. In particolare nell'attività di Studio di Fattibilità il costo a consuntivo è stato maggiore rispetto a quello preventivato. Questo è da collegare al costo orario dell'amministratore e non meno alle cause elencate sopra per l'indice SV che ne è strettamente collegato. Complessivamente *Sirius* ha ottenuto:

- BV-totale = 30.00 €.

il risultato ottenuto è una diretta conseguenza di un preventivo appropriato, e quindi ad un piccolo margine di guadagno nel budget di spesa dell'intero progetto.

B.3 Riassunto dell'attività di verifica su RP

Il processo di progettazione è per *Sirius* un importante strumento per giungere ai periodi successivi con maggiore sicurezza del lavoro da fare e con una stima precisa delle risorse e degli obiettivi che si è deciso di perseguire. Nonostante le tempistiche ridotte, l'organizzazione del lavoro ha voluto che prima di procedere alla progettazione vera e propria si correggessero le opportunità di miglioramento riscontrate in RR, con l'obiettivo di consolidare il grado di maturità dei processi esistenti prima di aggiungerne di nuovi. Nella progettazione, descritta nella Specifica Tecnica v3.0.0, si è provveduto a delineare la struttura del software provando a pianificare la tipologia di test che *Sirius* andrà ad eseguire nei periodi a seguire.

B.4 Dettaglio dell'attività di verifica su RP

B.4.1 Documenti

Indice di *Gulpease* per i documenti redatti:

Documento	**Valore di accettazione**	**Esito**

Piano Di Progetto v3.0.0	>40	*Positivo*
Norme di progetto v3.0.0	>40	*Positivo*
Analisi dei requisiti v3.0.0	>40	*Positivo*
Piano Di Qualifica v3.0.0	>40	*Positivo*
Studio Di Fattibilità v2.0.0	>40	*Positivo*
Specifica Tecnica v3.0.0	>40	*Positivo*

Tabella 10: Esito del calcolo indice di Gulpease per ogni documento per RP

B.4.2 Processi

Sirius ha condotto l'attività di verifica per i processi. In questo primo periodo il processo di documentazione è predominante nella pianificazione delle risorse. Di seguito viene riportato l'indice **SV (*schedule variance*)** per le attività eseguite e i risultati sono i seguenti:

Attività	**Ore pianificate**	**Ore rilevate**	**SV rilevato**	**SV accettazione**
Norme di Progetto	5 H	9 H	**-4 H**	> -1 H
Analisi dei Requisiti	53 H	25 H	27 H	> -3 H
Piano di Progetto	42 H	25 H	17 H	> -2 H
Piano di Qualifica	7 H	23 H	**-16 H**	> -1 H
Specifica Tecnica	74 H	90 H	**-32 H**	> -4 H

Tabella 11: Indice SV per le attività per RP

Da una prima analisi, si denota che *Sirius* ha pianificato in modo preciso le attività. Questo ha fatto in modo che la progettazione architetturale sia stata svolta più velocemente rispetto a quanto pianificato. La causa principale può essere ricercata nella velocità di accordo tra i progettisti di *Sirius*, che hanno definito in modo rapido la progettazione del prodotto. Questo ha portato al risultato positivo.

- SV-totale = 2 H;

SV-totale maggiore di zero denota che *Sirius* stà producendo più velocemente rispetto a quanto pianificato. Questo può essere una diretta conseguenza dell'aggiunta di uno *slack* temporale nella pianificazione delle attività. Il team ha valutato la possibilità di ridurre il tempo di *slack*, per fare in modo che la pianificazione corrisponda alla

realtà; ma data la variabilità delle attività che *Sirius* intende svolgere nel proseguo del progetto e la poca esperienza, è stato deciso di non modificare tale valore.
Di seguito viene riportato l'indice **BV (*budget variance*)** per le attività eseguite e i risultati sono i seguenti:

Attività	**Costo pianificato**	**Costo consuntivo**	**BV rilevato**	**BV limite**
Norme di Progetto	61.00 €	105.00 €	**-44.00 €**	> -6.10 €
Analisi dei Requisiti	1253.00 €	578.00 €	675.00 €	> -125.30 €
Piano di Progetto	1182.00 €	705.00 €	477.00 €	> -118.20 €
Piano di Qualifica	71.00 €	582.00 €	**-511.00 €**	> -7.10 €
Specifica Tecnica	1584.00 €	2132.00 €	**-548.00 €**	> -158.40 €

Tabella 12: Indice BV per le attività per RP

Come descritto sopra per SV, anche BV denota che il preventivo di costo previsto per le attività svolte è stato corretto. In particolare la stesura della specifica tecnica ha richiesto più budget di quello preventivato, le altre attività non hanno però consumato del tutto il proprio budget. Questo nonostante le correzioni da effettuare sulla documentazione prodotta in RR. Complessivamente *Sirius* ha ottenuto:

- BV-totale = 49.00 €.

il risultato ottenuto è una diretta conseguenza di un preventivo appropriato, e quindi ad un piccolo margine di guadagno nel budget di spesa dell'intero progetto.

B.5 Riassunto dell'attività di verifica su RQ

Sirius, ha deciso di valorizzare sopratutto i requisiti desiderabili per fare in modo di tenere alto l'indice di efficienza$_G$. Valutando, durante lo stato di avanzamento, quali di questi requisiti saranno successivamente sviluppati nella realizzazione del prodotto software. Questo tipo di scelte risultano infatti difficoltose allo stato attuale in quanto una pianificazione approfondita necessiterebbe di *lead time* $_G$ precisi che non sono al momento tra le conoscenze dei componenti del gruppo. Nel periodo di tempo che ha portato *Sirius* alla consegna di questa revisione sono stati verificati i **documenti** ed i **processi**.
I **documenti** sono stati verificati anche durante le operazioni di redazione per portare a conoscenza dei contenuti tutti i componenti del gruppo di lavoro. L'analisi statica, in primo luogo utilizzando la tecnica del *walkthrough*, ha portato alla redazione di una lista di controllo che verrà poi incrementata ed utilizzata nell'analisi finale del documento

prima di procedere alla consegna. Una volta rilevati gli errori questi sono stati notificati al redattore che ha proceduto alla correzione, evidenziando gli errori frequenti che sono stati utilizzati per migliorare il processo di verifica. *Sirius* adotta il ciclo PDCA per rendere più efficiente$_G$ ed efficace$_G$ nel tempo il processo di verifica.
L'attività di verifica, inoltre, utilizzando la tecnica *inspection* è stata utilizzata principalmente per la verifica dei grafici dei casi d'uso. Per verificare la correttezza dei requisiti richiesti e la successiva completezza ci si è affidati ad un particolare strumento di tracciamento definito in Norme di progetto v3.0.0. L'avanzamento dei processi, dettato dal Piano Di Progetto v3.0.0, è stato mantenuto in controllo tramite una costante verifica delle metriche definite in questo documento e di cui troviamo una rappresentazione grafica in seguito.

B.6 Dettaglio dell'attività di verifica su RQ

B.6.1 Documenti

Indice di *Gulpease* per i documenti redatti:

Documento	**Valore di accettazione**	**Esito**
Piano Di Progetto v3.0.0	>40	*Positivo*
Norme di progetto v3.0.0	>40	*Positivo*
Analisi dei requisiti v3.0.0	>40	*Positivo*
Piano Di Qualifica v3.0.0	>40	*Positivo*
Studio Di Fattibilità v2.0.0	>40	*Positivo*
Definizione Di Prodotto v1.0.0	>40	*Positivo*
Manuale utente v1.0.0	>40	*Positivo*
Manuale process owner v1.0.0	>40	*Positivo*
Specifica Tecnica v3.0.0	>40	*Positivo*

Tabella 13: Esito del calcolo indice di Gulpease per ogni documento per RQ

B.6.2 Processi

Sirius ha condotto l'attività di verifica per i processi. Di seguito è riportato l'indice **SV (*schedule variance*)** per le attività eseguite.

Attività	**Ore pianificate**	**Ore rilevate**	**SV rilevato**	**SV accettazione**

Norme di Progetto	8 H	6 H	2 H	> -1 H
Piano di Progetto	13 H	6 H	7 H	> -1 H
Piano di Qualifica	19 H	27 H	**-8 H**	> **-1 H**
Specifica Tecnica	7 H	56 H	**-49 H**	> **-1 H**
Definizione di Prodotto	78 H	84 H	**-6 H**	> **-4 H**
Codifica	51 H	48 H	3 H	> -3 H
Manuale Utente	17 H	15 H	2 H	> -1 H
Attività di Verifica	93 H	40 H	53 H	> -5 H

Tabella 14: Indice SV per le attività per RQ.

Dall'analisi della tabella sopra si denota come la pianificazione del periodo sia stata poco efficace. Questo è dovuto ad una attività di progettazione svolta in modo non preciso, che ha portato ad una correzione e nuova stesura di alcuni documenti. Inoltre, si vede come la pianificazione dei test e la progettazione di dettaglio siano state sottovalutate nella pianificazione del periodo. Del resto l'attività di verifica e di codifica, sulla quale si era pianificato un notevole monte ore, sono risultate essere attività sopravalutate.

- SV-totale = 4 H;

Di seguito viene riportato l'indice **BV (*budget variance*)** per le attività eseguite e i risultati sono i seguenti:

Attività	**Costo pianificato**	**Costo consuntivo**	**BV rilevato**	**BV limite**
Norme di Progetto	160.00 €	120.00 €	40.00 €	> -16.00 €
Piano di Progetto	390.00 €	180.00 €	210.00 €	> -39.00 €
Piano di Qualifica	475.00 €	675.00 €	**-200.00 €**	> -47.00 €
Specifica Tecnica	154.00 €	1232.00 €	**-1078.00 €**	> **-15.00 €**
Definizione di Prodotto	1759.00 €	1812.00 €	**-53.00 €**	> -176.00 €
Codifica	765.00 €	720.00 €	45.00 €	> -77.00 €
Manuale Utente	374.00 €	330.00 €	44.00 €	> -37.00 €
Verifica	1395.00 €	600.00 €	795.00 €	> -140.00 €

Tabella 15: Indice BV per le attività per RQ

Complessivamente *Sirius* ha ottenuto:

- BV-totale = -197.00 €.

B.6.3 Risultati delle misurazioni sul codice

Di seguito i risultati dei test di analisi statica effettuati sul codice fin'ora prodotto. Si è deciso di non effettuare i test sulla parte HTML in quanto statica e priva di metodi da analizzare. Per ogni metrica viene indicata la media e il valore massimo, rilevati dall'analisi sul codice prodotto, sia parte server che parte client. Il livello di accettabilità è valutato sul valore medio della metrica.
I valori fuori range saranno commentati di seguito.

Metrica	**Media**	**Massimo**	**Accettazione**
Complessità ciclomatica	4	13	Acc.
Livelli di annidamento	1,46	3	Acc.
Attributi per classe	2,3	8	Acc.
Parametri per metodo	3	7	Acc.
Linee di codice per linee commento	32%	44%	Acc.
Copertura	53%		**N. Acc.**

Tabella 16: Risultati delle misurazioni sul codice

La copertura non è risultata accettabile in quanto si dovrebbe stabilizzare sopra al 80%, questo sarà l'obiettivo dei antecedente i test di validazione aumentare la copertura dei test per fare in modo di limitare il codice non testato.