



SIRIUS

SEQUENZIATORE

Norme di progetto

Versione 3.0.0

Ingegneria Del Software AA 2013-2014

Informazioni documento

Titolo documento:	Norme di progetto
Data creazione:	2014-02-05
Versione attuale:	3.0.0
Utilizzo:	Interno
Nome file:	<i>NormeDiProgetto_v3.0.0.pdf</i>
Redazione:	Santangelo Davide
Verifica:	Marcomin Gabriele Giacchin Vanni
Approvazione:	Quaglio Davide
Distribuito da:	Sirius

Sommario

Il presente documento descrive le norme adottate dal gruppo *Sirius* durante la realizzazione del prodotto software *Sequenziatore*.

Diario delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
4.0.0	2014-08-22	<i>Giachin Vanni</i>	<i>Responsabile</i>	Approvazione del documento
3.1.0	2014-08-20	<i>Marcomin Gabriele</i>	<i>Verificatore</i>	Approvate le modifiche
3.0.1	2014-08-16	<i>Santangelo Davide</i>	<i>Amministratore</i>	Apportate delle modifiche alla struttura
3.0.0	2014-05-16	<i>Giachin Vanni</i>	<i>Responsabile</i>	Approvazione del documento
2.2.0	2014-05-13	<i>Marcomin Gabriele</i>	<i>Verificatore</i>	Verificato documento
2.1.1	2014-05-12	<i>Quaglio Davide</i>	<i>Amministratore</i>	Aggiunto capitolo: definizione di prodotto, sistemati errori rilevati
2.1.0	2014-05-08	<i>Marcomin Gabriele</i>	<i>Verificatore</i>	Verificato documento
2.0.2	2014-05-06	<i>Quaglio Davide</i>	<i>Amministratore</i>	Aggiunte alla sezione: Codifica
2.0.1	2014-05-05	<i>Quaglio Davide</i>	<i>Amministratore</i>	Ristrutturazione dello scheletro documento
2.0.0	2014-02-29	<i>Quaglio Davide</i>	<i>Responsabile</i>	Approvazione del documento
1.1.0	2014-02-29	<i>Marcomin Gabriele</i>	<i>Verificatore</i>	Verificato documento
1.0.2	2014-02-28	<i>Santangelo Davide</i>	<i>Amministratore</i>	Aggiunto norme relativi a test e al tracciamento dei package e classi
1.0.1	2014-03-20	<i>Santangelo Davide</i>	<i>Amministratore</i>	Corretti errori notificati in sede di Revisione dei Requisiti
1.0.0	2014-02-09	<i>Quaglio Davide</i>	<i>Responsabile</i>	Approvazione del documento
0.1.0	2014-02-09	<i>Giachin Vanni</i>	<i>Verificatore</i>	Verifica del documento
0.0.2	2014-02-07	<i>Santangelo Davide</i>	<i>Amministratore</i>	Stesura del documento
0.0.1	2014-02-05	<i>Seresin Davide</i>	<i>Amministratore</i>	Creato lo scheletro del documento

Indice

1	Introduzione	1
1.1	Scopo del documento	1
1.2	Ruoli di progetto	1
1.2.1	Responsabile di progetto	1
1.2.2	Amministratore	2
1.2.3	Analista	2
1.2.4	Verificatore	3
1.2.5	Progettista	3
1.2.6	Programmatore	3
1.3	Ref. Glossario	4
1.4	Riferimenti	4
1.4.1	Normativi	4
1.4.2	Informativi	4
2	Processi primari	5
2.1	Acquisizione	5
2.1.1	Analisi dei requisiti	5
2.2	Fornitura	8
2.2.1	Pianificazione	8
2.3	Sviluppo	14
2.3.1	Progettazione	14
2.3.2	Specifiche tecniche	14
2.3.3	Definizione di prodotto	17
2.3.4	Norme sulla codifica	17
3	Processi di supporto	20
3.1	Comunicazioni	20
3.1.1	Comunicazioni interne	20
3.1.2	Riunioni interne	20
3.1.3	Comunicazioni Esterne	21
3.1.4	Incontri esterni	21
3.1.5	Strumenti per le comunicazioni	21
3.2	Documentazione	21
3.2.1	Template	21
3.2.2	Classificazione documenti	22
3.2.3	Versionamento documenti	22
3.2.4	Struttura Documentazione	23
3.2.5	Norme tipografiche	24

3.2.6	Calcolo indice di Gulpease	25
3.2.7	Glossario	26
3.2.8	Strumenti per la documentazione	26
3.3	Gestione configurazione	28
3.3.1	Strumento di versionamento	28
3.4	Verifica	30
3.4.1	Metriche	30
3.4.2	Procedure di verifica	33
3.4.3	Strumenti per la verifica	35

1 Introduzione

1.1 Scopo del documento

Questo documento è stato redatto al fine di definire e standardizzare tutte le norme che ogni componente del team *Sirius* dovrà adottare durante il periodo di svolgimento del capitolato *Sequenziatore*, commissionato dalla *Zucchetti S.P.A.G.* In particolare si andranno di seguito ad elencare convenzioni e norme per:

- Le comunicazioni e le relazioni inter-personali;
- Il processo di sviluppo dell'applicazione;
- Il processo di pianificazione del progetto;
- Il processo di *Analisi*;
- La redazione e verifica della correttezza dei documenti;
- L'utilizzo degli strumenti dell'ambiente di lavoro.

Ogni membro del gruppo *Sirius* ha l'obbligo di visionare questo documento. Tutti i componenti del gruppo sono tenuti a sottostare alle norme qui descritte di modo da garantire un corretto e coerente insieme di file prodotti, assicurando conseguentemente un continuo miglioramento dell'efficienza nello sviluppo. Infine, qualora lo si ritenesse necessario, ogni membro del team potrà contattare l'*Amministratore* per discutere norme esistenti o per valutarne l'adozione di nuove.

1.2 Ruoli di progetto

1.2.1 Responsabile di progetto

Il *responsabile di progetto* è colui che ha potere decisionale sul progetto, l'entità quindi che ha l'incarico di approvare le scelte effettuate ed il lavoro svolto. Questo potere decisionale si ripercuote sulla responsabilità che detiene nel presentare il risultato del prodotto creato al proponente. Il responsabile di progetto deve inoltre occuparsi della gestione (assegnazione, modifica) dei *ticket*, ed assicurarsi che i verificatori seguano sistematicamente le regole imposte dalle *Norme di progetto*. Riassumendo, tale figura ha responsabilità sotto il profilo di:

- Approvazione dell'offerta economica;
- Approvazione dei documenti;
- Analisi e gestione delle risorse;
- Analisi e gestione dei rischi;

- Pianificazione del piano di lavoro;
- Coordinazione del team;
- Controllo del regolare svolgimento delle attività;
- Evitare conflitti di interesse tra redattori e verificatori.

Per ciò che concerne la documentazione, il responsabile di progetto è colui che deve redigere il *Piano di progetto* e collaborare nella stesura del *Piano di qualifica*.

1.2.2 Amministratore

La figura dell'*Amministratore* è responsabile per tutto ciò che è pertinente all'efficienza ed operatività dell'ambiente di lavoro. I suoi compiti di primaria importanza sono qui di seguito riportati:

- Gestione del versionamento del prodotto;
- Ricerca degli strumenti più adatti allo svolgimento del progetto;
- Automatizzazione delle attività che possono essere risolte tramite strumenti e procedure automatiche, in modo da snellire il carico di lavoro ad personam;
- Creare o ricercare strumenti che possano controllare/monitorare la qualità del prodotto;
- Normare le procedure standard di pianificazione del progetto, coordinazione del team, redazione dei documenti, produzione del codice.

L'amministratore deve redigere interamente le *Norme di progetto* nonché collaborare nel *Piano di qualifica*.

1.2.3 Analista

L'*Analista* è competente nell'attività di analisi ed astrazione dei requisiti di progetto. Di seguito vengono riportate le sue mansioni principali:

- Astrarre i requisiti dal problema in modo da creare una specifica di progetto comprensibile dal progettista, dal proponente e dal committente;
- Comprendere i requisiti meno espliciti del problema da affrontare.

I documenti: *Analisi dei Requisiti* e *Studio di Fattibilità* devono essere stilati dall'*Analista*. Nel *Piano di qualifica* dovrà illustrare il livello di qualità richiesta e le procedure da attuare per raggiungerla.

1.2.4 Verificatore

Il verificatore è colui che deve effettuare l'attività di verifica. Questa figura deve attenersi alle regole imposte dalle *Norme di progetto*, e tramite l'ausilio degli strumenti illustrati nel *Piano di qualifica* avrà il compito di assicurare che:

- Le attività svolte siano coerenti agli standard adottati.
- La documentazione sia conforme alle *Norme di progetto*;

1.2.5 Progettista

Il progettista è colui che ha la responsabilità sull'attività di progettazione. I suoi compiti possono essere riassunti come segue:

- Agire in modo che il progetto sia sviluppato tramite tecnologie al più possibile stabili e note;
- Agire in modo che il progetto sia sviluppato seguendo soluzioni (come ad esempio soluzioni progettuali) ottimizzate e note;
- Creare una soluzione progettuale adeguata, ossia comprensibile ed attuabile;
- Agire sulle scelte progettuali in modo da sviluppare un prodotto facilmente manutenibile.

Tale ruolo avrà il compito di redigere la *Specifica Tecnica*, la *Definizione di Prodotto* e la parte inerente alla metrica di verifica nel *Piano di Qualifica*.

1.2.6 Programmatore

Come deducibile è la figura che si occuperà dell'attività di codifica. Le principali mansioni di questo ruolo si riassumono nei seguenti punti:

- Implementare le soluzioni progettuali specificate dal *Progettista*, senza discuterle o senza prendere alcuna iniziativa personale.
- Scrivere codice manutenibile rispettando gli standard imposti per la codifica;
- Documentare tutto il codice generato, in modo chiaro e conciso;
- Implementare i test per la verifica e validazione del codice.

Il *Programmatore* si occuperà infine di scrivere il *Manuale Utente*.

1.3 Ref. 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*.

Tutte le prime occorrenze di vocaboli presenti nel *Glossario* devono essere seguite da una “G” maiuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- **Amministrazione di progetto** <http://www.math.unipd.it/~tullio/IS-1/2013/Dispense/P05.pdf>

1.4.2 Informativi

- **Software Engineering-** Ian Sommerville;
- **Calcolo indice di gulpease;**
 - *Indice di leggibilità di un testo* <http://xoomer.virgilio.it/roberto-ricci/variabilialeatorie/>
- **Strumenti di collaborazione;**
 - *Collaborative development eviroments* <http://www.math.unipd.it/~tullio/IS-1/2010/Approfondimenti/A12.pdf>
- **Strumenti di gestione del ciclo di vita del software**<http://www.math.unipd.it/~tullio/IS-1/2008/Materiale/P0a.pdf>

2 Processi primari

2.1 Acquisizione

Il processo di acquisizione comincia con la ricerca e l'identificazione delle necessità espresse dal cliente riguardo il prodotto che vuole ottenere, e tale processo si conclude una volta che queste necessità, opportunamente catalogate e identificate, vengono approvate da quest'ultimo. L'acquisizione prevede quindi di formulare e catalogare i requisiti, impliciti e non, del progetto in questione.

2.1.1 Analisi dei requisiti

2.1.1.1 Procedura di classificazione requisiti Ogni requisito deve essere definito tramite una descrizione testuale e un codice identificativo, classificante e univoco avente la seguente forma:

$$\{\text{Tipologia}\}\{\text{Importanza}\}\{\text{Categoria}\}\{\text{Identificatore}\}$$

- Tipologia:
 - **F:** requisito funzionale;
 - **Q:** requisito di qualità;
 - **V:** requisito di vincolo;
- Importanza:
 - **OB:** requisito obbligatorio;
 - **DE:** requisito desiderabile;
 - **OP:** requisito opzionale;
- Categoria:
 - **U:** requisito funzionale riguardante la parte utente;
 - **L:** requisito funzionale riguardante la parte utente autenticato;
 - **A:** requisito funzionale riguardante la parte process owner;
 - vuoto se il requisito è di qualità oppure di vincolo;
- Identificatore è un codice gerarchico composto da uno o più numeri separati da un punto, in cui l'ultimo numero è un identificatore incrementale intero.
La rimanente parte di codice viene utilizzata quando il requisito da definire è sotto-requisito di un altro, e identifica il requisito gerarchicamente superiore.

2.1.1.2 Casi d'uso Per ciascun caso d'uso deve essere fornito un codice identificativo, una descrizione testuale e un diagramma UML.

Il codice identificativo deve rispettare la seguente forma:

$$UC\{Tipologia\}\{Identificatore\}$$

Tipologia può essere U, L o A, che stanno rispettivamente per utente non autenticato, utente autenticato e process owner.

Identificatore è un codice gerarchico composto da uno o più numeri separati da un punto, in cui l'ultimo numero è un identificatore incrementale intero. La rimanente parte di codice viene utilizzata quando il caso d'uso da definire è una specifica o estensione di un altro, e identifica il caso d'uso gerarchicamente superiore.

La descrizione deve contenere i seguenti dettagli:

- Descrizione del caso d'uso;
- Attori coinvolti;
- Precondizione;
- Scenario principale dello svolgersi degli eventi;
- Senari alternativi;
- Post-condizione.

Il diagramma deve rispettare le regole della notazione UML 2.x_G.

2.1.1.3 Strumenti per l'analisi

2.1.1.3.1 Piattaforma per il tracciamento Per il tracciamento è stato sviluppato un semplice programma denominato *Sirius RTg*. Questo programma è stato sviluppato utilizzando PHP_G, CSS_G ed HTML_G. Sirius RTg è attualmente un programma il cui sviluppo non è terminato, principalmente per permettere una aggiunta di funzionalità in caso di necessità. Sirius RTg, alla versione 2.0.0, fornisce le seguenti funzionalità:

- Inserimento requisito e relativo tracciamento;
- Visualizzazione dello script per la tabella dei requisiti e relativo tracciamento.

2.1.1.3.2 Inserimento requisito e relativo tracciamento Questa funzionalità è fornita all'esterno attraverso un'interfaccia scritta in HTML e CSS.

L'interfaccia è costituita da uno semplice *form*, in cui è possibile inserire:

- Codice requisito;
- Descrizione del requisito;
- Categoria;
- Importanza;
- Tipo;
- Relativi casi d'uso;
- Relative fonti.

Ogni requisito deve avere obbligatoriamente definito il Tipo, l'Importanza, ed il Codice requisito. Il codice deve necessariamente identificare univocamente il requisito, altrimenti un uso ridondante di codici verrà notificato all'utente. Obbligatoria la categoria, in caso il requisiti sia della parte utente.

2.1.1.3.3 Visualizza script Questa funzionalità serve per la stampa a video dei vari script in \LaTeX per i vari requisiti e il relativo tracciamento.

Visualizza script è composto dalle seguenti funzionalità:

- Visualizzazione script per Requisiti di tipo utente amministratore;
- Visualizzazione script per Requisiti di tipo utente-autentificato;
- Visualizzazione script per Requisiti di vincolo;
- Visualizzazione script per Requisiti di qualità;
- Visualizzazione script tracciamento Requisiti-uc;
- Visualizzazione script tracciamento Uc-requisiti.

Ogni script stampato su video, segue le regole definite nelle *Norme di Progetto*.

Anche se Visualizzazione script utente e Visualizzazione script utente autentificato sono due sotto-funzionalità distinte di Visualizza script, devono essere utilizzate assieme per produrre la tabella dei requisiti di tipo utente; infatti Visualizzazione script utente stampa l'intestazione della tabella e la parte dei requisiti utente, mentre Visualizzazione script utente autentificato stampa la parte dei requisiti utente autentificato e i comandi necessari per chiudere la tabella.

Tutti gli altri Script, invece possono essere usati singolarmente e a video, oltre al contenuto comparirà la relativa intestazione e chiusura della tabella.

2.1.1.3.4 Inserimento componente e relativo tracciamento Questa funzionalità è stata inserita al fine di poter tracciare le componenti del sistema, permette di inserire tramite un form:

- Nome del componente
- Requisiti associati al componente

Sono state inoltre implementati due script che stampano a video il codice \LaTeX del tracciamento componente-requisito e requisito-componente, il codice \LaTeX generato è conforme alle *NormeDiProgetto_v3.0.0.pdf*.

2.1.1.3.5 Script glossario Anche se non inerente al tracciamento dei requisiti, lo strumento *Sirius RTg* alla versione 2.0 prevede la funzionalità aggiuntiva di poter inserire automaticamente i pedici $_G$ (per le parole contenute nel *Glossario_v3.0.0.pdf*) ai documenti del team.

2.2 Fornitura

Il processo di fornitura agglomera le attività di: gestione delle pianificazione, gestione delle risorse, gestione dei rischi. Tale processo quindi ha come obiettivo quello di fornire un **piano** per la gestione del progetto che imponga delle direttive in merito alla:

- pianificazione delle attività;
- pianificazione delle milestone;
- pianificazione delle scadenze.

Il responsabile di tutto questo, durante il corso dell'intero progetto, è il *Responsabile di progetto*.

2.2.1 Pianificazione

Il *responsabile di progetto* per ogni attività indicata nel documento *Piano di Progetto* dovrà creare un nuovo progetto seguendo la procedura qui descritta:

1. Inserire una milestone $_G$;
2. Inserire le attività da svolgere;
3. Inserire le rispettive sotto-attività;
4. Calcolare ed inserire i periodi di slack $_G$ qualora fosse necessario;
5. Creare le risorse;

6. Assegnare le risorse create ad ogni attività;

7. Salvare la baseline_G.

Sarà decisa a discrezione del *Responsabile di Progetto* per ogni attività la possibilità di assegnare un surplus di ore, queste ore supplementari verranno scelte basandosi sulla criticità dell'attività considerata.

- Per le attività non critiche non è previsto alcun surplus di ore;
- Per le attività di media criticità il surplus di ore potrà essere del 15%;
- Per le attività di criticità massima il surplus di ore potrà essere del 30%.

2.2.1.1 Procedura utilizzo del ticketing Ogni membro del team *Sirius* avrà accesso al sistema di *ticketing*. Le figure che invece potranno assegnare ticket sono le seguenti:

- Il *Responsabile di Progetto* assegnerà i *ticket* di massima importanza cioè quelli correlati allo sviluppo delle attività necessarie all' avanzamento del progetto;
- Il *Verificatore* potrà assegnare *ticket* allo scopo di segnalare errori di grave entità rilevati durante l'attività di verifica.

Di conseguenza i *ticket* sono suddivisi in due macro-categorie:

- *Ticket* di pianificazione, i quali rappresentano le attività che devono essere svolte per procedere con l'avanzamento del progetto, sono suddivisi in 4 sotto-categorie:
 - *Documento*: che rappresenta una *task* inerente alla redazione di un documento;
 - *Codice*: che rappresenta una *task* inerente alla stesura di codice;
 - *Verifica*: che rappresenta una *task* inerente all'attività di verifica di un'attività;
 - *Generali*: che rappresenta *tasks* i cui scopi sono svariati ed in genere non ad alta priorità, come ad esempio la ricerca di un determinato *software*.

lo svolgimento dell'insieme di tutti i ticket di una *task-list* non porterà alla conclusione della *task-list* stessa, questo poiché è prevista la possibilità di aggiungere durante l'avanzamento del progetto ulteriori *task*, fino a quando il *responsabile di progetto* non ne dichiarerà la conclusione;

- Ticket di verifica, contenenti gli errori identificati dai *verificatori* a seguito dell'analisi del lavoro svolto da qualche membro del *team*.

Add a new milestone

MARCH
3
Monday

Give this milestone a name

+ Add Description...

When is it due?
03/03/2014

Who's responsible?
Davide Santangelo (Me)
+ Multiple People

Privacy
Everybody on project

Reminder?
(An email reminder will not be sent to the person responsible)

Notify now?
(The person responsible will not be notified)

Add this milestone or Cancel

Figura 1: Creazione di una milestone.

Ogni membro del *team* sarà tenuto ad utilizzare la barra di avanzamento di stato del *ticket* fornita dall'interfaccia di *TeamWorkPM*, evitando così superflue norme aggiuntive atte a determinare lo stato del *ticket*.

2.2.1.2 Procedura creazione di una milestone Il *Responsabile di Progetto* dovrà creare una *milestone*, essa indica la data della revisione a cui il gruppo *Sirius* intende presentarsi, è possibile visualizzare lo stato di avanzamento che tiene conto del numero di *ticket* completati rispetto al numero di *ticket* complessivi. Per la creazione di una nuova *milestone* il *Responsabile di Progetto* dovrà seguire i seguenti passi:

1. Aprire il progetto dall'interfaccia *web* di *TeamWorkPM*;
2. Posizionarsi sull'opzione: *Milestones* ed accedervi;
3. Cliccare sull'opzione: *Add a new milestone*.

Completati questi passaggi apparirà il seguente *form_G* (fig 1) che dovrà essere compilato per concludere la creazione della una *milestone*.

2.2.1.3 Procedura di creazione ticket Il *Responsabile di Progetto* dovrà attenersi alla seguente procedura per la creazione di una nuova task-list, ovvero la con-

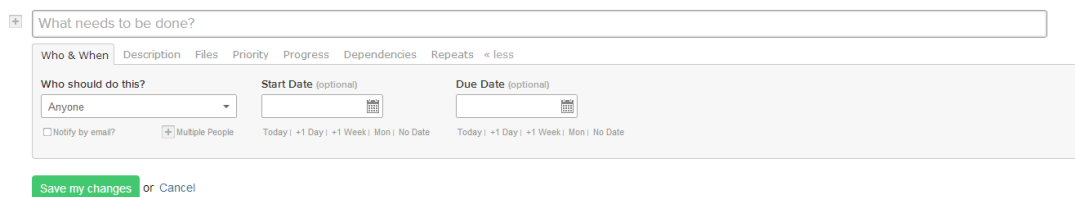


Figura 2: Creazione di un ticket.

creazione di un macro-attività e delle sue relative task (ticket). Si ricorda che TeamWorkPM prevede la possibilità di indicare interdipendenze tra task-list.

1. Dall'interfaccia *web* accedere al progetto *Sequenziatore*, e selezionare dal *menù* principale il comando: *Task*;
2. Procedere, se necessario, con la creazione di una nuova *task-list* tramite il comando: *Add task list*;
3. Una volta creata la *task-list* sarà possibile creare i *ticket* (*Task* nel contesto di *TeamWorkPM*) inerenti alla *task-list* scelta.

La struttura di un *ticket* è visualizzabile nella figura 2 di questa sezione.

Nella task è necessario specificare **obbligatoriamente**:

- Il **Titolo** del *ticket*, dovrà contenere tra parentesi quadre la categoria (per i *ticket* di pianificazione anche la sotto-categoria) di *ticket* di cui si tratta;
- Il **Destinatario** del *ticket*, cioè colui a cui è stato assegnato;
- Le **Date** di inizio e scadenza del *ticket*;
- Le **Dipendenze** del *ticket*, che specificano l' eventuale necessità di attendere la terminazione di un insieme di *task* prima di poter svolgere quel determinato compito;
- Una **Descrizione** la quale dovrà essere breve e concisa, ma spiegare efficacemente il lavoro assegnato;
- La **Priorità** del *ticket* suddivisa in tre categorie: bassa, media, alta.

Compilati i seguenti campi, il *ticket* sarà creato ed inviato regolamentarmene.

2.2.1.4 Procedura di terminazione ticket Se un *ticket* sarà completato è necessario applicare questa procedura di accertamento:

1. Il membro del *team* a cui è stato assegnato il *ticket* dovrà spuntare la casella di terminazione su *TeamWorkPM*;

2. Durante il controllo giornaliero il *Responsabile di Progetto* controllerà quanto necessario a determinare che il lavoro sia stato effettivamente svolto;
3. Se il lavoro è stato effettivamente svolto, sarà avviato un *ticket* di *Pianificazione* a scopo di verificare il lavoro;
4. Nel caso di irregolare svolgimento del *ticket* o problemi di grave entità, il *Responsabile di Progetto* dovrà applicare la procedura di modifica o riassegnazione del *ticket* presente nel prossimo paragrafo;
5. Nel caso di esito positivo (cioè con regolare svolgimento) il *ticket* sarà concluso ed archiviato, mentre al contempo, qualora fosse necessario, saranno avviati dei *ticket* di *Verifica* per la correzione degli errori non gravi rilevati durante la *Verifica*;

2.2.1.5 Procedura per la modifica o riassegnazione ticket Durante il suo ciclo di vita un *ticket* per varie ragioni può andare in contro a modifiche, è necessario quindi normare la seguente procedura:

1. Aprire il progetto dall'interfaccia *web* di *TeamWorkPM*;
2. Selezionare il *ticket* di interesse;
3. Selezionare il comando: *Edit Task*;
4. Aggiungere una descrizione riguardo la modifica effettuata;
5. Avvertire l'interessato che è stata effettuata una modifica inserendo: (MOD) sul titolo del *ticket*, e qualora fosse necessario reimpostandone la sua priorità.

2.2.1.6 Strumento per la gestione del progetto Al fine di gestire rigorosamente lo sviluppo del progetto il team *Sirius* ha adottato l'utilizzo del software *TeamWorkPM* (www.teamwork.com), tale strumento fornisce le seguenti funzionalità:

- Creazione di *ticket_G*, *milestone_G* e liste di attività;
- Creazione ed assegnazione di attività;
- Calendario di progetto;
- *Report* automatico giornaliero delle attività svolte ed in ritardo inviato tramite *e-mail*;
- Gestione dei ruoli;
- Generazione di grafici *Gantt_G* a partire dalle *task-list*;
- Monitoraggio dei tempi;

- Registro dei rischi.

Sono stati valutati altri software come ad esempio *Redmine*, il quale fu ritenuto quasi altrettanto completo ed intuitivo. Tuttavia si è optato per *TeamWorkPM* data la sua estrema semplicità di utilizzo.

Il *Responsabile di Progetto* per garantire il regolare svolgimento delle attività dovrà necessariamente verificare con cadenza giornaliera se sono presenti *ticket* scaduti e non ancora completati, nel caso citato infatti sarà obbligato a richiedere informazioni (o in caso di ritardi gravi convocare l'interessato) circa la causa del ritardo.

Infine, il *Responsabile di Progetto* dovrà tenere nota che l'assegnazione di *ticket* la cui scadenza è prevista per il giorno successivo può avvenire solo se il lavoro da svolgere non supera le 2 ore lavorative.

2.2.1.7 Strumenti per la pianificazione

2.2.1.7.1 GanttProject Per la pianificazione del progetto nonché gestione delle risorse è stato adottato il software GanttProject, software open source basato su piattaforma Java. Qui di seguito vengono elencate le principali caratteristiche che hanno portato alla scelta di questo strumento:

- Portabilità, essendo un software basato su Java;
- Open-source_G;
- Compatibile con MicrosoftProject;
- Può generare grafici Work Breakdown Structure (WBS_G);
- Fornisce la possibilità di creare grafici di Gantt_G;
- Può generare grafici Program Evaluation and Review Technique (PERT_G);
- In grado di gestire e generare grafici delle risorse assegnate.

2.2.1.7.2 Team Work PM Al fine di gestire rigorosamente lo sviluppo del progetto il team *Sirius* ha adottato l'utilizzo del software *TeamWorkPM* (www.teamwork.com), tale strumento fornisce le seguenti funzionalità:

- Creazione di *ticket_G*, *milestone_G* e liste di attività;
- Creazione ed assegnazione di attività;
- Calendario di progetto;
- *Report* automatico giornaliero delle attività svolte ed in ritardo inviato tramite *e-mail*;

- Gestione dei ruoli;
- Generazione di grafici *Gantt_G* a partire dalle *task-list*;
- Monitoraggio dei tempi;
- Registro dei rischi.

Sono stati valutati altri software come ad esempio *Redmine*, il quale fu ritenuto quasi altrettanto completo ed intuitivo. Tuttavia si è optato per *TeamWorkPM* data la sua estrema semplicità di utilizzo.

Il *Responsabile di Progetto* per garantire il regolare svolgimento delle attività dovrà necessariamente verificare con cadenza giornaliera se sono presenti *ticket* scaduti e non ancora completati, nel caso citato infatti sarà obbligato a richiedere informazioni (o in caso di ritardi gravi convocare l'interessato) circa la causa del ritardo.

Infine, il *Responsabile di Progetto* dovrà tenere nota che l'assegnazione di *ticket* la cui scadenza è prevista per il giorno successivo può avvenire solo se il lavoro da svolgere non supera le 2 ore lavorative.

2.3 Sviluppo

2.3.1 Progettazione

Questa sezione descrive le norme cui i progettisti dovranno attenersi durante la stesura del documento: *Specifica tecnica* ed il documento **definizione di prodotto**. Queste norme sono dettate al fine di poter redigere un documento il più possibile formale e senza ambiguità.

2.3.2 Specifica tecnica

2.3.2.1 Diagrammi Per quanto concerne i diagrammi sarà adottato il linguaggio *Unified Modelling Language* (UML) 2.0, tramite questo linguaggio si andranno a definire:

- **Diagrammi dei package:** ossia elementi di raggruppamento di classi. Tali elementi dovranno figurare durante la progettazione generale e dovranno essere identificati univocamente al fine di stabilire come i suddetti interagiscono tra di loro.
- **Diagrammi di sequenza:** I quali andranno a descrivere come un gruppo di oggetti andranno a collaborare per implementare collettivamente un comportamento;
- **Diagrammi di attività:** Atti a mostrare i flussi di attività che gli utenti potranno percorrere durante l'uso dell'applicazione;

- **Diagrammi delle classi:** I quali consentono di descrivere tipi di entità con le loro caratteristiche.

2.3.2.2 Design pattern Per ogni design pattern utilizzato sarà necessario andare a definire i seguenti punti:

- Una **descrizione generale** che riporta la struttura del design pattern scelto;
- Una **motivazione** che descriva i vantaggi che ne comporta il suo uso;
- Il **contesto applicativo** che associa ai design pattern utilizzati il contesto dove sono stati adottati.

2.3.2.3 Nomenclatura delle classi Ogni classe descritta nel documento specifica tecnica seguirà il seguente schema:

- **Nome:** enuncia il nome della classe che andrà descritta, devono essere obbligatoriamente in inglese e devono comparire con l'iniziale in maiuscolo.
- **Package:** enuncia il pacchetto, ed i relativi sotto-pacchetti, all'interno dei quali è contenuta la classe di interesse. Ogni package deve seguire la seguente notazione: Pack1::Pack2::...::Pack-n; ove con:
 - Pack1, si intende il package principale;
 - Pack(x)::Pack(y) indica che y è sotto package di x;
- **Descrizione:** deve contenere una breve ma significativa descrizione testuale riguardante l'utilizzo della classe;
- **Relazione con altri componenti:** viene specificato se la classe di interesse ha relazioni con le classi di altri componenti.

2.3.2.4 Tracciamento Ogni componente sarà tracciato seguendo tali regole:

AmbitoUtenteCodice

ove **Ambito** rappresenta con:

- **V:** view;
- **CP:** client presenter;
- **SP:** server presenter;
- **CM:** client model;

- **SM**: server model.

ove **Ambito** rappresenta:

- **A**: process owner;
- **U**: utente;

Per quanto concerne il codice, si adotteranno i codici del tracciamento degli UC.

2.3.2.5 Test I progettisti avranno il compito di definire delle classi e dei test fittizi con lo scopo di valutare il lavoro svolto. I test saranno suddivisi in:

- Test di integrazione;
- Test di unità;

2.3.2.6 Tracciamento requisiti-componenti

- *Requisito*: contiene il codice univoco e classificante del requisito;
- *Descrizione requisito*: contiene la descrizione del requisito;
- *Nome componente*: contiene il codice identificativo del componente.

2.3.2.7 Tracciamento componenti-requisiti La struttura della tabella sarà la seguente:

- *Requisito*: contenente il codice univoco e classificante del requisito;
- *Componente*: contenente il nome del package ed il nome della classe.

2.3.2.8 Test di sistema ed integrazione Il tracciamento dei test sarà inserito in forma tabellare nel documento *PianoDiQualifica_v3.0.0.pdf*, e dovrà rispettare il seguente stampo:

- **Codice test**: il quale dovrà essere obbligatoriamente univoco ed atto ad identificare il test;
- **Descrizione**: la quale specifica lo scopo del test;
- **Requisito annesso**: che specifica il requisito cui il test fa riferimento;
- **Stato**: che riporta se il test è stato effettuato.

2.3.3 Definizione di prodotto

Redigere il documento di *definizione di prodotto* sarà compito dei *progettisti*. La *definizione di prodotto* è un documento che si pone l'obiettivo di definire dettagliatamente ogni singola unità di cui è composto il sistema. Al fine di non lasciare libertà di iniziativa durante l'attività di codifica sarà necessario quindi specificare i metodi e gli attributi di ogni classe. Questo documento presenterà una versione più raffinata, rispetto alla specifica tecnica, dei seguenti diagrammi:

- **Diagrammi delle classi;**
- **Diagrammi di sequenza.**

2.3.3.1 Formalismo delle classi Ogni classe deve essere definita attenendosi alle seguenti convenzioni:

- **attributi:** dovranno essere indicati specificando il grado di accessibilità come previsto da UML 2.4. Infine dovrà seguire (riportata sotto il diagramma) la descrizione dell'attributo stesso.
- **metodi:** come per gli attributi si dovrà specificare il grado di accessibilità dei suddetti, a seguito dovrà figurare il nome del metodo stesso. Tra parentesi tonde si dovranno inserire i suoi parametri. Infine si inserirà il tipo di ritorno del metodo.
- **parametri:** i parametri, racchiusi da parentesi tonde, devono presentare il loro nome seguito da due punti ed il loro tipo.

2.3.4 Norme sulla codifica

2.3.4.1 Convenzioni di codifica Al fine di produrre codice ordinato e leggibile, in modo da semplificare il più possibile l'attività di manutenzione, per quanto concerne la programmazione *Java_G*, si adotteranno le norme imposte dalla *Java code conventions*, reperibili all'indirizzo:

<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Variazioni e modifiche a queste convenzioni possono essere richieste all'*Amministratore*, allegandone la motivazione. Se l'*Amministratore* riterrà opportune le variazioni presentate, sarà tenuto a notificarlo al team seguendo le convenzioni imposte dalle *Norme di Progetto*.

2.3.4.2 Nomi Sarà adottata la notazione *CamelCase* al fine di identificare facilmente il nome di variabili, classi, metodi, funzioni, interfacce. Più specificatamente bisognerà rispettare le seguenti regole:

- **variabili, metodi, funzioni, interfacce:** dovranno avere la prima lettera minuscola;
- **classi:** dovranno avere la prima lettera maiuscola.

Inoltre sarà obbligatorio utilizzare la lingua **inglese** per assegnare i suddetti nomi. Inoltre è opportuno ma non obbligatorio che i nomi (di variabili, metodi, etc..) siano esplicativi rispetto al ruolo che assumono nel contesto di applicazione.

2.3.4.3 Commenti ed intestazioni Ogni file contenente codice dovrà essere provvisto di un'intestazione che rispetta la seguente forma:

file: che riporta il nome del file;
author: che riporta il nome dell'autore;
date: che riporta la data di creazione;
lastModified: che riporta la data di ultima modifica;
brief: che descrive brevemente lo scopo e contenuto del file.

Le classi dovranno obbligatoriamente essere provviste di commenti che contengono:

class: nome della classe;
brief: breve descrizione della classe.

Per quanto concerne i metodi, essi dovranno essere provvisti di commenti che si adeguano alla seguente forma:

brief: breve descrizione del compito del metodo;
param: che comparirà tante volte quanti sono i parametri in input al metodo, e ne riporterà il nome ed il tipo;
return: nome e tipo del valore ritornato dalla funzione.

Infine per le interfacce la forma sarà la seguente:

brief: breve descrizione dello scopo dell'interfaccia.

Qualora fosse impossibile utilizzare la tecnica del *refactoring_G* per ristrutturare il codice, nel qual caso si trattasse di codice particolarmente difficile da comprendere, è possibile dedicare un commento aggiuntivo:

spiegazione approfondita: ...

al fine di facilitarne la comprensione.

2.3.4.4 Strumenti per la codifica Il team *Sirius* durante lo sviluppo dell'applicazione utilizzerà diversi framework e librerie quali:

- Spring;
- Backbone.js;
- RequireJS.

Per non incorrere in problemi di incompatibilità del codice sviluppato, si è deciso di uniformare le versioni di tali *framework* e librerie. I programmatori sono quindi **tenuti** a sviluppare tramite software aggiornato alla seguente *release*:

- *Spring*: 4.0.3;
- *Backbone.js*: 1.1.2;
- *RequireJS*: 1.1.

2.3.4.5 IDE Per la stesura del codice dell'applicazione *Sequenziatore* il team *Sirius* sarà tenuto ad utilizzare gli ide_G ammessi nella lista di seguito riportata. Nel caso il team di sviluppo lo ritenesse necessario, sarà possibile richiedere all'*amministratore* di modificare tale lista aggiungendo nuovi ambienti di sviluppo che vengono ritenuti più adatti per la stesura del codice.

Spring tool suite (versione 3.5.1).

Spring tool suite è un ide basato su Eclipse che integra tutto il necessario per lavorare ad un progetto che necessita l'utilizzo del *framework Spring*.

Infine è possibile utilizzare l'ide *Eclipse* in quanto si tratta di un ambiente di sviluppo *open source* multiplatforma e multilinguaggio piuttosto completo, con possibilità di installare plug-in per espanderne le sue funzionalità. La versione da adottare di questo software è la 4.3.2 (Kepler SR2).

3 Processi di supporto

In questa sezione verranno trattati i processi di supporto allo sviluppo del progetto.

3.1 Comunicazioni

3.1.1 Comunicazioni interne

L'apparato di comunicazioni interne ufficiali sarà gestito tramite una *mailing list*_G basata su *Google Groups*.

Il nome della *mailing list* è Sirius esattamente come il nome del gruppo (*Sirius*).

Vige l'obbligo di utilizzare la *mailing list* solamente per le comunicazioni interne ufficiali, così da evitare intasamenti superflui che graverebbero sul lavoro di verbalizzazione delle comunicazioni di rilievo, in quanto renderebbero più complessa ed inutilmente lunga l'estrazione delle informazioni utili. Tuttavia è preferibile che la comunicazione tra i vari componenti del team avvenga principalmente durante gli incontri che si terranno in un luogo fisico comune (sezione 2.2, Riunioni interne).

Al fine di facilitare anche le comunicazioni informali, sono stati adottati due strumenti di *instant messaging*_G e videoconferenza quali Skype e Google Plus ed un strumento di *web storage* per lo scambio di dati ufficiosi (sezione 8.1, Gestione condivisione file).

3.1.2 Riunioni interne

Le riunioni del gruppo *Sirius* avranno una frequenza almeno settimanale. Il giorno della settimana, il luogo e l'ora in cui riunirsi ufficialmente sarà deciso dal *Responsabile di Progetto* su consultazione degli altri membri del team e sarà comunicato tramite il gruppo *Google Groups* Sirius. Chiunque non abbia la possibilità di essere presente fisicamente nel luogo della riunione, dovrà possibilmente restare in contatto con il nucleo dei membri mediante videoconferenza.

Qualunque membro del team può richiedere al *Responsabile di Progetto* di indire una riunione allegandone l'argomento di discussione ed una sua breve descrizione, a seguito della comunicazione il *Responsabile di Progetto* deciderà se indire la suddetta riunione generale, cioè con obbligatoria presenza di tutti i membri del gruppo. Qualsiasi riunione surplus a quella settimanale deve essere indetta con almeno 2 giorni di anticipo, in modo da verificare la disponibilità del gruppo.

Se dovessero essere necessarie riunioni che non richiedono la presenza del gruppo nella sua totalità, ogni membro potrà presentare la richiesta di ritrovo tramite l'apposita *mailing list* (sezione 2.1, Comunicazioni interne), richiedendo la disponibilità degli specifici membri del team che riterrà necessari, questo poiché è auspicabile che alcune figure come ad esempio *Progettista* ed *Analista* collaborino tra di loro frequentemente. Le riunioni che non coinvolgono interamente il team non necessitano dell'approvazione

del *Responsabile di Progetto* in modo da ridurre il suo carico di lavoro, nonostante queste le decisioni effettuate durante queste discussioni inter-membri dovranno comunque essere verbalizzate.

3.1.3 Comunicazioni Esterne

Per le comunicazioni esterne di ogni tipo è stato creato in indirizzo *e-mail* del team:

swesirius@gmail.com

Il *Responsabile di Progetto* rappresenta il team stesso, sarà quindi incaricato di mantenere i contatti con proponente, committente, ed eventuali altre figure non facenti parte del nucleo del *team*, tramite questo indirizzo *e-mail*, inoltre sarà sempre parte del suo compito aggiornare i membri del gruppo stesso riguardo le corrispondenze pervenute attenendosi alle istruzioni della sezione 2.1, Comunicazioni interne.

3.1.4 Incontri esterni

Il *Responsabile di Progetto* ha inoltre l'onere di organizzare eventuali incontri esterni (per chiarificazioni o quant'altro) con *Proponente* o *Committente/i*. Ogni membro del gruppo può richiedere un incontro esterno al *Responsabile di Progetto*, presentando una motivazione valida. Infine, il *Responsabile di Progetto* dopo aver valutato personalmente la proposta, dovrà presentarla al gruppo (con allegata la motivazione ed il nome di chi l'ha richiesta) quindi, per l'approvazione definitiva di quest'ultima, almeno due membri escluso l'artefice dovranno dare ulteriore conferma e disponibilità, in caso contrario la proposta sarà bocciata.

3.1.5 Strumenti per le comunicazioni

Oltre agli strumenti precedentemente citati quali: Skype, Google Hangout, mailing list Google, per gestire efficientemente la condivisione dei file intra-gruppo è stato scelto l'utilizzo di: Google Drive, un servizio *web* di *storage* e sincronizzazione *online* che dovrebbe facilitare la condivisione e fornire una base d'appoggio secondaria ed informale per alcuni file che non necessitano versionamento. L'utilizzo di *Google Drive* è limitato ai documenti che:

- Non necessitano di versionamento;
- Necessitano di essere acceduti velocemente tramite *web*;

3.2 Documentazione

3.2.1 Template

Ogni documento dovrà essere generato includendo il *template* L^AT_EX presente nella cartella Modello. Questo modello è stato creato prima dell'inizio della redazione di

ogni altro documento del team *Sirius*, la sua modifica può avvenire solo presentando all'*Amministratore* una richiesta formale, allegandone la motivazione ed il tipo di modifica richiesta. Se l'*Amministratore* riterrà opportuno effettuare il cambiamento, prima di apportare la modifica dovrà avvertire l'intero team al fine di evitare disguidi.

3.2.2 Classificazione documenti

3.2.2.1 Documenti formali Sono catalogati come formali tutti i documenti approvati dal *Responsabile di Progetto*, ovvero i documenti ritenuti pronti per essere visionati dal committente. Tali documenti, prima di raggiungere l'approvazione dovranno aver superato con successo la procedura di verifica e validazione riportata nel *Piano di Qualifica*.

3.2.2.2 Documenti informali Tutti i documenti che non sono stati approvati dal *Responsabile di Progetto* sono da ritenersi informali, e di utilizzo esclusivamente interno. Tutti i documenti non versionati sono da ritenersi non ufficiali.

3.2.3 Versionamento documenti

Il versionamento di tutta la documentazione del gruppo *Sirius* è stato organizzato secondo le seguenti convenzioni:

- Il numero di versionamento deve essere nella forma:

X, Y, Z

con **X, Y, Z** numeri interi non negativi;

- Tutti gli elementi devono salire di una sola unità alla volta.

Di seguito vengono inoltre riportati i significati che possono assumere le variazioni della versione del documento:

- La **X** rappresenta il numero di uscite formali del documento, ogni qual volta un documento verrà pubblicato il valore della cifra **Y** e della cifra **Z** verrà azzerato. Riportando quanto detto più precisamente:
 1. X assumerà il valore: 1, alla **revisione dei requisiti**;
 2. X assumerà il valore: 2, alla **revisione di progettazione**;
 3. X assumerà il valore: 3, alla **revisione di qualifica**;
 4. X assumerà il valore: 4, alla **revisione di accettazione**.

- La **Y** rappresenta il numero di *push* effettuati sul *branch_G master* in *GitHub* (sezione 8.2.1, *GitHub*), ossia il numero di volte in cui sono state compiute importanti modifiche al documento. Ogni qual volta aumenterà l'indice **Y** si azzererà l'indice **Z**.
- La **Z** rappresenta il numero di modifiche minori apportate al documento durante il suo sviluppo. Aumenta al termine di ogni sessione di lavoro sul documento.

Ogni documento formale riporterà un diario delle modifiche contenente le trasformazioni più rilevanti che ha attraversato sotto forma tabellare.

3.2.4 Struttura Documentazione

3.2.4.1 Header Ogni pagina esclusa la prima presenta un *header* raffigurante il logo del gruppo sulla sinistra, mentre sulla destra il nome del team ed il nome del progetto.

3.2.4.2 Footer Ogni pagina esclusa la prima presenta un *footer* riportante il nome del documento corredato della versione sulla sinistra, mentre sulla destra il numero della pagina. Per il numero di pagina delle prime quattro facciate saranno utilizzati i numeri romani, a seguire invece verranno utilizzati i numeri occidentali.

3.2.4.3 Prima pagina La prima pagina di ogni documento conterrà:

- Il logo del *team*, riportante la scritta *Sirius*;
- Il titolo del progetto;
- Il nome del documento e la sua versione;
- Il nome del corso;
- L'anno di sviluppo del progetto;

3.2.4.4 Seconda pagina La seconda pagina di ogni documento conterrà:

- Informazioni sul documento come segue:
 - Titolo del documento;
 - Data di creazione;
 - Versione attuale;
 - Utilizzo, che specifica se il documento è per utilizzo interno o esterno;
 - Nome file;
 - Redazione;

- Revisione;
 - Approvazione;
 - Distribuito da, a cui seguirà il nome del gruppo.
- Un sommario riportante una breve descrizione;

3.2.4.5 Terza pagina La terza pagina di ogni documento conterrà:

- Un diario delle modifiche apportate al documento, dall'inizio fino alla versione corrente.

3.2.4.6 Quarta pagina La quarta pagina di ogni documento ne riporterà l'indice, è possibile che l'indice si estenda per più di una singola pagina.

3.2.5 Norme tipografiche

3.2.5.1 Generali

- Ogni documento deve essere in lingua italiana, altre lingue possono essere utilizzate per riferirsi a termini tecnici informatici o in situazioni che lo richiedono strettamente;
- Ogni documento deve essere grammaticalmente, sintatticamente e semanticamente corretto, cercando di essere meno verboso possibile;
- Utilizzare il più possibile elenchi puntati invece di lunghe frasi.

3.2.5.2 Punteggiatura

- Non si usa mai un punto alla fine di un titolo: di capitolo, di paragrafo, di sotto-paragrafo;
- Ogni elemento di un elenco puntato termina con un punto e virgola, se è l'ultimo elemento con un punto;
- Prima di ogni segno di punteggiatura non va mai messo uno spazio bianco, dopo invece lo spazio bianco va messo sempre;
- Il testo racchiuso tra parentesi non deve aprirsi o chiudersi con un carattere di spaziatura né terminare con un carattere di punteggiatura.

3.2.5.3 Ortografia

- Le lettere maiuscole vanno poste solo all'inizio di ogni elemento di un elenco puntato e dove lo prevede l'ortografia italiana (all'inizio di un periodo o dopo un segno di punteggiatura forte, cioè dopo il punto fermo, i puntini di sospensione, il punto esclamativo ed il punto interrogativo). È inoltre utilizzata l'iniziale maiuscola nel nome del team, del progetto, dei documenti, dei ruoli di progetto.

3.2.5.4 Stile

- Se si devono elencare delle di istruzioni in serie o una divisione in paragrafi e sotto-paragrafi è necessario utilizzare un elenco numerato, altrimenti è preferibile un elenco puntato;
- Il primo livello di profondità degli elenchi puntati è contrassegnato da un pallino nero pieno, il secondo da un trattino, il terzo da un asterisco;
- Le date dovranno essere espresse nella forma **aaaa-mm-gg** secondo lo standard **ISO G 8601:2004**;
- Gli orari dovranno essere espressi nella forma **hh:mm** secondo lo standard **ISO G 8601:2004**;
- **URL** ed indirizzi mail dovranno essere preceduto dal comando $\text{\LaTeX}\ \text{\url}$;
- Ogni prima (e possibilmente anche successiva) occorrenza di una parola presente sul *Glossario* sarà seguita da pedice **G**.
- Stile di testo:
 - Il corsivo deve essere utilizzato obbligatoriamente nelle citazioni, per il nome delle figure di rilievo (es. *committente*, *Responsabile di Progetto*) e per il nome dei documenti (es. *Analisi dei requisiti*), mentre a discrezione del redattore per termini stranieri in modo da evidenziarli;
 - il grassetto deve essere utilizzato per evidenziare (se si reputa necessario) le parole chiave ed i passaggi particolarmente rilevanti.

3.2.6 Calcolo indice di Gulpease

In ogni documento redatto il verificatore dovrà calcolare l'indice di Gulpease, ossia il valore di leggibilità del documento. Per raggiungere il seguente scopo è disponibile uno *script online*, reperibile al sito:

<http://www.xoomer.virgilio.it/roberto-ricci/variabiliatore/esperimenti/leggibilita.htm>

Questo *script* già esistente è stato verificato prima di essere adottato, in modo da scongiurare il rischio di incompatibilità tra i documenti redatti e la forma che doveva avere l'input per lo *script*. Se l'indice risultante di un documento si troverà in un range compreso tra lo 0 ed il 40, sarà necessario ricercare nel testo frasi troppo lunghe e complesse per reimpostarle.

3.2.7 Glossario

Durante la stesura di un documento, ogni qual volta il redattore riterrà necessario chiarire il significato di un termine utilizzato sarà tenuto ad aggiungerlo nel *glossario*. Il *glossario* sarà strutturato seguendo questo schema:

- Nel file \LaTeX ogni parola sarà contenuta nel **tag**: elemento;
- A seguire, andando a capo-riga, sarà riportata la descrizione del termine.

Il *glossario* sarà inoltre suddiviso in due sezioni:

- Termini;
- Acronimi.

Termini ed acronimi dovranno essere necessariamente elencati in ordine alfabetico, la definizione dovrà essere breve ed esplicativa, inoltre sempre la definizione non potrà iniziare con una **E accentata**.

3.2.8 Strumenti per la documentazione

3.2.8.1 LaTeX Per la stesura dei documenti il team *Sirius* ha deciso di adottare il linguaggio di markup \LaTeX la scelta è stata effettuata prevalentemente per le seguenti ragioni:

- Facilità di separazione tra contenuto e formattazione;
- Possibilità di definire macro ed incorporare scripts;
- Software open source;
- Grande quantità di pacchetti disponibili, possibilità quindi di implementare semplicemente le funzionalità comuni.

Gli altri software valutati (Open office, Microsoft Office, Google Docs) non erano in grado di fornire il più delle sopracitate funzionalità, di conseguenza sono stati scartati. Inoltre come editor è consigliato ma non obbligato l'uso di TeXstudio.

3.2.8.2 Macro Al fine di velocizzare il lavoro di stesura documenti, il team *Sirius* ha deciso di creare delle apposite macro, qui vengono riportate le principali assieme ad una breve spiegazione delle loro funzionalità:

- `\gruppo` riporta il nome del team *Sirius*;
- `\progetto` riporta il nome del progetto: *Sequenziatore*;
- `\lastversion+sigla-del-documento` riporta il nome del documento che appare in sigla (NDP, AR, etc..) aggiornato alla versione più recente.

3.2.8.3 Scripts Al fine di implementare una funzionalità quale l'inserimento automatico dei pedici in tutte le parole dei documenti formali che comparivano anche nel glossario, è stato creato uno script apposito in Python_G. Tale script può essere eseguito solamente con una versione di Python non superiore alla 2.7.6. Per il corretto funzionamento dello script il glossario è stato organizzato tramite tag `\LaTeXelementoparola` del glossario, la definizione è riportata a capo-riga rispetto alla suddetta parola.

3.2.8.4 Correttezza

3.2.8.5 Correttezza ortografica Per evitare di compiere errori di tipo ortografico devono essere adottate due precauzioni:

- Verifica delle parole durante la stesura stessa del documento tramite lo spell checker_G di TeXstudio;
- Verifica finale tramite lo spell checker Aspell.

Lo spell checker di TeXstudio è una sua feature_G molto utile che sfrutta dizionari Open Office per sottolineare eventuali parole scorrette, dizionari sufficientemente completi che assicurano quindi un grado piuttosto elevato di correttezza già durante la stesura del testo.

Al fine poi di assicurarsi il massimo grado possibile di correttezza viene effettuata una verifica ulteriore tramite il software open source GNU Aspell (www.aspell.net).

3.2.8.6 Lista controllo errori Il team ha stilato una lista di controllo al fine di riassumere gli errori più ricorrenti in ogni documento, i suddetti saranno catalogati e descritti nella seguente sezione.

3.2.8.7 Errori stilistici e di punteggiatura I principali errori rilevati sono i seguenti:

- Le figure di rilievo non vengono scritte in corsivo;

- Negli elenchi puntati la prima parola non compare con la prima lettera maiuscola;
- Negli elenchi puntati alcuni elementi centrali non terminano con un punto e virgola ma con un punto fermo;
- Alcune date vengono erroneamente scritte senza seguire lo standard **ISO G 8601:2004**;
- La parola LaTeX compare senza l'utilizzo del comando `\LaTeX` (`LATEX`).

3.2.8.8 Errori ortografici e di sintassi

- La è accentata compare (erroneamente) come una e apostrofata;
- Utilizzando le seguenti macro `\gruppo` e `\progetto`, le quali scrivono testualmente e rispettivamente il nome del team ed il nome del capitolato, non compaiono separate dalla parola successiva, anche se la spaziatura è presente;
- Non viene utilizzata (erroneamente) la terza persona per la stesura dei documenti.

3.2.8.9 UML Per la modellazione dei diagrammi User Case (UC_G) sono stati presi in considerazione tre editor: Dia, Microsoft Visio, Astah. Infine il team ha optato per adottare Astah come strumento definitivo in quanto si tratta di un software open source, con supporto di Unified Modeling Language (UML_G) 2.x e secondo l'analisi del team dotato di un interfaccia più responsiva ed intuitiva degli altri software. Per lo sviluppo dei diagrammi di sequenza, dopo aver effettuato i test necessari, è stato permesso l'utilizzo del software *VisualParadigm*, questa modifica è stata attuata dopo aver verificato che il software *Astah* forniva un iterfaccia di creazione per diagrammi di sequenza poco intuitiva e difficoltosa da utilizzare. *VisualParadigm* è adeguato per lo sviluppo di diagrammi UML 2.x.

3.3 Gestione configurazione

La gestione di configurazione deve comprendere tutte le attività necessarie a rendere affidabile l'evoluzione del progetto, tenendo traccia delle sue versioni.

3.3.1 Strumento di versionamento

Di pressoché fondamentale importanza è stata la definizione di un ambiente ordinato in cui organizzare e mantenere tutti i *file* che attraversano il ciclo di vita, per questa ragione è stato scelto di avvalersi di un *repository*.

Corretto macro.tex per AnalisiRequisiti		
Vgiachin authored 13 hours ago latest commit c0b080d72a		
AnalisiDeiRequisiti	Aggiornata Analisi dei Requisiti	13 hours ago
EsempioDocumento	aggiornato esempio documento	21 days ago
Glossario	ultima modifica alla grafica del Diario	22 days ago
NormeDiProgetto	Update Sviluppo.tex	4 days ago
PianoDiProgetto	ultima modifica alla grafica del Diario	22 days ago
PianoDiQualifica	bozza piano di qualifica	21 days ago
StudioDiFattibilita	terza persona nello studio fattibilità	16 days ago
Verbali	creata copia di modello dentro Verbale	17 days ago
modello	aggiornamento modello da master	13 hours ago
.gitignore	tolto .gitingore~	21 days ago

Figura 3: Struttura master branch.

3.3.1.1 GitHub Come sistema di controllo di versione è stato adottato il software *GitHub_G*, i pregi di questo strumento vengono qui di seguito riportati:

- Molto reattivo;
- Design semplice;
- *Software* gratuito.

3.3.1.2 Struttura repository L'indirizzo di root_G del *repository* contenente tutta la documentazione è:

<https://github.com/Dquaglio/Sirius>

Ogni documento presente sarà contenuto in una sotto cartella del *master branch_G* nominata come il nome del documento stesso. All'interno della cartella potranno essere contenuti solamente *file.tex*, per la visualizzazione del relativo *pdf_G* sarà necessario scaricarli e compilarli, assicurandosi di avere l'ultima versione del modello disponibile.

modello.git: <https://github.com/Dquaglio/Sirius/tree/master/modello.git>

conterrà il *template* $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, le *macro* e gli script aggiornati all'ultima versione disponibile.

Il *master branch* è stato quindi suddiviso seguendo questa struttura: A scopo puramente dimostrativo è stato creato l'esempio di un documento formale del gruppo *Sirius*, contenuto nella cartella: EsempioDocumento, questa scelta è stata fatta per illustrare

la struttura generale che deve preservare qualsiasi documento (sezione 5.4, Struttura documentazione). Per sfruttare il parallelismo nello sviluppo di uno stesso documento sono stati creati appositamente dei *branch* denominati con il nome dei membri del gruppo, i documenti *baseline_G* invece saranno contenuti solamente nel *master branch*. Il *merge_G* con il ramo *master* avviene quindi solamente dopo la terminazione dell'attività di verifica di un documento.

3.4 Verifica

3.4.1 Metriche

I dati rilevati durante l'attività di verifica devono essere analizzati tramite precise metriche. Con questo termine si intende l'insieme di parametri misurabili su un processo. Qualora le metriche definite in questo documento siano approssimative e/o ambigue, queste dovranno essere ridefinite in modo specifico e seguiranno in modo incrementale il ciclo di vita del prodotto. Di seguito sono riportate le metriche adottate dal team *Sirius*; gli obiettivi qualitativi che invece definiscono il grado di accettazione/ottimalità verranno riportati nel *PianoDiQualifica_v3.0.0.pdf*.

3.4.1.1 Metriche per i processi Le metriche dei processi ne stabiliscono la qualità, definita come connubio tra *capability_G*, *maturity_G* e i miglioramenti. Queste caratteristiche di qualità si possono individuare in tre classi di misure di processo:

- **Tempo:** il tempo richiesto per il completamento di un particolare processo;
- **Risorse:** le risorse richieste per un particolare processo, in genere vengono definite risorse-uomo, per le risorse software si fa riferimento a Norme di progetto v3.0.0;
- **Occorrenze:** il numero di volte che capita un particolare evento, che può essere il numero di difetti scoperti durante l'attività di verifica.

Per rilevare questi dati *Sirius* ha deciso di utilizzare, indici che valutano i tempi e i costi del processo. La scelta di queste metriche è dettata anche dal loro possibile utilizzo durante lo svolgimento del processo, per capire in modo semplice se lo stato del processo è conforme a quanto pianificato, mantenendo quindi il processo in controllo. In Piano Di Progetto v3.0.0 viene specificato come sono stati pianificati questi indici nello stato di avanzamento.

3.4.1.2 (SV) Schedule Variance Indica se si è in linea, in anticipo o in ritardo rispetto alla pianificazione temporale delle attività citata in Piano Di Progetto v3.0.0. È un indicatore di efficacia temporale e per questo *Sirius* ha deciso di esprimerlo in ore. Se $SV > 0$ significa che il gruppo di lavoro sta producendo con maggior velocità rispetto

a quanto pianificato, viceversa se negativo.

3.4.1.3 (BV) Budget Variance Indica se allo stato attuale si è speso più o meno rispetto a quanto pianificato. È un indicatore che ha valore contabile e finanziario per questo è espresso in euro. Se $BV > 0$ significa che l'attuazione del progetto sta consumando il proprio budget con minor velocità rispetto a quanto pianificato, viceversa se negativo.

3.4.1.4 Metriche per i documenti Come metrica per la verifica dei documenti *Siriusha* ha deciso di utilizzare l'indice di leggibilità. Vi sono a disposizione molti indici di leggibilità, ma i più importanti sono per la lingua inglese. Si è deciso quindi di adottare un indice di leggibilità per la lingua italiana. L'indice *Gulpease* è un indice di leggibilità di un testo tarato sulla lingua italiana. Rispetto ad altri indici, esso ha il vantaggio di utilizzare la lunghezza delle parole in lettere anziché in sillabe, semplificandone il calcolo automatico. Permette di misurare la complessità dello stile di scrittura di un documento. L'indice viene calcolato utilizzando la formula citata nelle Norme di progetto v3.0.0. I risultati sono compresi tra 0 e 100, dove il valore 100 indica la leggibilità più alta e 0 la leggibilità più bassa. In generale risulta che testi con un indice:

- Inferiore a 80 sono difficili da leggere per chi ha la licenza elementare;
- Inferiore a 60 sono difficili da leggere per chi ha la licenza media;
- Inferiore a 40 sono difficili da leggere per chi ha un diploma superiore.

3.4.1.5 Metriche per il software Al fine di perseguire gli obiettivi qualitativi dichiarati nel *PianoDiQualifica.v3.0.0.pdf* è necessario definire delle metriche, queste metriche hanno quindi l'obiettivo di rendere quantificabile il lavoro svolto. Questa sezione, però, è da intendersi come modificabile nell'arco dello svolgimento del progetto.

3.4.1.6 Complessità ciclomatica Pensata da T.J. McCabe è utilizzata per misurare la complessità per funzioni, moduli, metodi o classi di un programma. Misura direttamente il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. Alti valori di complessità ciclomatica indicano una ridotta manutenibilità del codice. Al contrario, valori bassi potrebbero determinare una scarsa efficienza dei metodi. Questo parametro è inoltre un indice del carico di lavoro richiesto dal *testing*. Indicativamente un modulo con complessità ciclomatica più bassa richiede meno test di uno con complessità più elevata.

Il valore 10 come massimo di complessità ciclomatica fu raccomandato da T.J.McCabe, l'inventore di tale metrica.

3.4.1.7 Numero livelli di annidamento Rappresenta il numero di livelli di annidamento, quindi l'inserimento di una struttura di controllo all'interno di un'altra. Un elevato valore comporta un'alta complessità e un basso livello di astrazione del codice.

3.4.1.8 Attributi per classe Un elevato numero di attributi per classe può rappresentare la necessità di suddividere la classe in più classi, possibilmente utilizzando la tecnica dell'incapsulamento, e può inoltre rappresentare un possibile errore di progettazione.

3.4.1.9 Numero di parametri per metodo Un elevato numero di parametri potrebbe richiedere di ridurre le funzionalità del metodo o provvedere ad una nuova progettazione dello stesso.

3.4.1.10 Linee di codice per linee di commento Indica il rapporto tra linee di codice e linee di commento: questo parametro è fondamentale per valutare la manutenibilità del codice prodotto, nonché del possibile riuso.

3.4.1.11 Accoppiamento

- **Accoppiamento afferente:** indica il numero di classi esterne al package_G che dipendono da classi interne ad esso. Un alto valore indica che è presente un alto grado di dipendenza del resto del software dal package. Questo non indica necessariamente una progettazione errata o di bassa qualità, ma possono rappresentare una criticità del package, che quindi perderebbe di robustezza. Al contrario un valore troppo basso potrebbe segnalare che il package analizzato fornisce poche funzionalità e quindi potrebbe risultare scarsamente utile. **Parametri utilizzati:** I valori di range di tale indice verranno definiti durante la progettazione di dettaglio.
- **Accoppiamento efferente:** indica il numero di classi interne al package che dipendono da classi esterne ad esso. Mantenendo un basso valore di questo indice, è possibile mantenere il package in grado di garantire funzionalità di base indipendentemente dal resto del sistema.

3.4.1.12 Copertura del codice Indica la percentuale di istruzione che vengono eseguite durante i test. Maggiore è la percentuale e più probabilità si hanno di rilevare minori errori nel prodotto. Tale valore può essere abbassato tramite l'utilizzo di metodi

molto semplici che non richiedono test.

3.4.2 Procedure di verifica

3.4.2.1 Tecniche di analisi statica L'analisi statica è una tecnica di analisi applicabile sia alla documentazione che al codice e permette di effettuare la verifica di quanto prodotto individuando errori ed anomalie. Essa può essere svolta in due modi diversi ma complementari tra di loro in quanto per utilizzare *inspection* bisogna prima aver effettuato *walkthrough*

3.4.2.1.1 Inspection Questa tecnica, di analisi statica, consiste nella verifica di sezioni ben definite di un documento o del codice. Questo tipo di controlli per i documenti sono usualmente definiti tramite una lista di controllo (checklist) redatta anticipatamente rispetto all'attività di verifica da intraprendere. Per la verifica dei documenti, la lista di controllo è stata elaborata a seguito di analisi eseguite tramite *walkthrough*, ed evidenziando gli errori più ricorrenti riscontrati. *Inspection* è una strategia rapida in quanto permette l'analisi di alcune parti ritenute critiche nella checklist senza bisogno di una lettura integrale di documento o di tutto il codice in oggetto.

3.4.2.1.2 Walkthrough *Walkthrough* è una tecnica di analisi statica che consiste nella lettura critica a largo raggio di tutto il documento. In questa tipologia di analisi il *Verificatore* utilizza molto tempo per la lettura e correzione del documento o codice. Questa tecnica viene di solito utilizzata nella prima parte dello sviluppo di progetti in quanto, la poca esperienza del *Verificatore* non permette un'altro tipo di verifica. Al termine di questo primo set di analisi *walkthrough* viene usualmente definita una lista di controllo che permetta di ricercare in primo luogo gli errori più ricorrenti, e maggiormente riscontrati. *Walkthrough* è un'attività onerosa e collaborativa che richiede l'intervento di più persone per essere efficiente ed efficace

3.4.2.2 Tecniche di analisi dinamica L'analisi dinamica si applica solamente al prodotto software e 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. La **ripetibilità** del test è una caratteristica fondamentale per questo tipo di test, in quanto dichiara che il codice con un certo *input* produce sempre lo stesso *output* su uno specifico ambiente. In questo modo si è in grado di riscontrare problemi e verificare la correttezza del prodotto. Per questo *Sirius* ha deciso di definire a priori le seguenti caratteristiche:

- **Ambiente:** sistema *hardware* e quello *software* sui quali è stato pianificato l'utilizzo del prodotto, di essi si deve definire uno stato iniziale dal quale poter iniziare ad eseguire i test;
- **Specifica di *input*:** definire quali sono gli *input* e quali devono essere gli *output* attesi;
- **Procedure:** definire quali devono essere i test ed in che ordine devono essere analizzati i risultati ottenuti.

Di seguito sono definiti cinque diversi tipi di test.

3.4.2.2.1 Test di unità Per test di unità si intende la verifica di ogni singola unità di prodotto software tramite l'utilizzo di stub_G , driver_G e logger_G . Per unità si intende la più piccola porzione di codice che è utile verificare singolarmente e che viene prodotta da un unico programmatore. Tramite questo tipo di test si vogliono testare i vari le unità per rilevare errori di implementazione da parte dei programmatori.

3.4.2.2.2 Test di integrazione I test di integrazione prevedono la verifica dei componenti del sistema che vengono aggiunti incrementando il prodotto di origine e si prefigge quindi di analizzare la combinazione di due o più unità software che hanno quindi superato i test di unità. Questa tecnica di verifica serve ad individuare errori residui nella programmazione dei singoli moduli: come modifiche delle interfacce e comportamenti inaspettati di componenti software di parti terze e che pregiudicherebbero la validità del prodotto. Per effettuare tali test può essere necessario l'aggiunta di componenti software fittizie e non ancora implementate al fine di non pregiudicare negativamente l'esito dell'analisi.

3.4.2.2.3 Test di sistema Consiste nella validazione del sistema attraverso la verifica della copertura di tutti i requisiti obbligatori individuati in Analisi dei requisiti v3.0.0, e tracciati grazie allo strumento messo a punto da *Sirius*;

3.4.2.2.4 Test di regressione I test di regressione vengono eseguiti quando si apportano delle modifiche a parte del software e questi consistono nella riesecuzione dei test riguardanti le i componenti che hanno subito modifiche e che precedentemente non erano soggetti ad errori. Tale operazione viene aiutata dal tracciamento, che permette di individuare e ripetere facilmente i test di unità, integrazione ed eventualmente di sistema che sono stati potenzialmente influenzati dalle modifiche.

3.4.2.2.5 Test di accettazione Si tratta del collaudo del prodotto software sotto il controllo del proponente. Se il collaudo viene superato in modo positivo, il sistema viene rilasciato e la commessa si conclude.

3.4.3 Strumenti per la verifica

3.4.3.1 Strumenti per testing Per quanto concerne gli strumenti utilizzati per la stesura dei test, i test lato server si baseranno sull'ausiglio di:

- *Eclipse metrics*, che fornisce un tool di testing per codice *Java* in grado di evidenziare la copertura fornita dai test;
- *JUnit*, framework per effettuare test *Java*;
- *Mockito*.

mentre lo sviluppo di test lato client si baserà su:

- *Jasmine* un framework per effettuare test *JavaScript*.

Per quanto concerne la copertura del codice *JavaScript* si utilizzerà *Istanbul*.