



MyTalk

Norme di Progetto

Informazioni sul documento

Nome file:	norme_di_progetto.3.0.pdf
Versione:	3.0
Data creazione:	2012-12-01
Data ultima modifica:	2013-02-15
Stato:	Approvato
Uso:	Interno
Lista di distribuzione:	Membri del Team
Redattori:	Riccardo Tresoldi
Approvato da:	Diego Beraldin
Verificatori:	Andrea Meneghinello

Storia delle modifiche

Versione	Descrizione intervento	Membro	Ruolo	Data
3.0	Approvazione documento	Diego Beraldin	Responsabile	2013-02-15
2.7	Correzione lessico ortografica del documento	Andrea Meneghinello	Verificatore	2013-02-14
2.6	Rese le sezioni del documento più procedurali	Riccardo Tresoldi	Amministratore	2013-02-14
2.5	Aggiunta e redatta sezione test	Riccardo Tresoldi	Amministratore	2013-02-14
2.4	Correzione e aggiornamento sezione strumenti	Riccardo Tresoldi	Amministratore	2013-02-13
2.3	Correzione e aggiornamento sezione verifica	Riccardo Tresoldi	Amministratore	2013-02-13
2.2	Aggiunta e redatta sezione Progettazione di dettaglio e Rapporti intrapersonali	Riccardo Tresoldi	Amministratore	2013-02-12
2.1	Aggiunta nota sulla rotazione dei redattori	Riccardo Tresoldi	Amministratore	2013-02-12
2.0	Approvazione documento	Riccardo Tresoldi	Responsabile	2013-01-23
1.6	Correzione errori rilevati dal verificatore	Diego Beraldin	Amministratore	2013-01-22
1.5	Validazione del documento	Stefano Farronato	Verificatore	2013-01-22
1.4	Aggiornata sezione software di tracciamento e verifica	Diego Beraldin	Amministratore	2013-01-17
1.3	Conclusa sezione progettazione	Diego Beraldin	Amministratore	2013-01-16
1.2	Aggiunta sezione progettazione	Diego Beraldin	Amministratore	2013-01-15
1.1	Correzione errori rilevati in RR	Diego Beraldin	Amministratore	2013-01-12
1.0	Approvazione documento	Stefano Farronato	Responsabile	2012-12-04
0.6	Validazione documento	Marco Schivo	Verificatore	2012-12-04
0.5	Fine stesura documento	Andrea Meneghinello	Amministratore	2012-12-03
0.4	Stesura sezione Milestones e Ticketing, Analisi dei requisiti	Andrea Meneghinello	Amministratore	2012-12-03
0.3	Stesura sezione ambiente documentale	Andrea Meneghinello	Amministratore	2012-12-03
0.2	Stesura sezione ambiente di lavoro	Andrea Meneghinello	Amministratore	2012-12-01
0.1	Stesura scheletro documento, sezione comunicazione	Andrea Meneghinello	Amministratore	2012-12-01

Indice

1	Introduzione	1
1.1	Scopo del prodotto	1
1.2	Scopo del documento	1
1.3	Ambiguità	1
2	Comunicazioni e relazioni intrapersonali	2
2.1	Comunicazione interna	2
2.2	Comunicazione esterna	2
2.3	Incontri Interni	2
2.4	Incontri Esterni	2
2.5	Considerazioni sui rapporti interpersonali	2
3	Ambiente di lavoro	4
3.1	Repository	4
3.1.1	Registrazione	4
3.1.2	Installazione	4
3.1.3	Struttura	4
3.1.4	Sistema di versionamento	5
3.2	Sistema di tracciamento dei requisiti	5
3.3	Sistema operativo	6
3.4	Diagrammi UML	6
3.5	Diagrammi di Gantt	6
4	Ambiente documentale	7
4.1	Software utilizzati	7
4.2	Nomi dei documenti	7
4.3	Impostazioni di base di un documento	8
4.4	Tipi di documento	9
4.5	Ciclo di vita di un documento	9
4.6	Template	11
4.7	Glossario	11
4.8	Norme tipografiche	11
4.8.1	Stili di testo	11
4.8.2	Punteggiatura	11
4.8.3	Composizione del testo	11
4.8.4	Formati ricorrenti	12
4.9	Immagini	12
4.9.1	Inserimento immagini	13
4.10	Inserimento tabelle	13
4.11	Inserimento link	13
4.12	Verifica dei documenti	13
5	Milestones e Ticketing	14
5.1	Milestones	14
5.1.1	Creazione nuova Milestone	14
5.1.2	Modifica Milestone	15
5.2	Ticketing	15
5.2.1	Creazione nuovo Ticket	15
5.2.2	Modifica Ticket	16
5.2.3	Ciclo di vita di un Ticket	16

6	Analisi dei Requisiti	19
6.1	Norme per i requisiti	19
6.2	Norme per i casi d'uso	20
6.3	Tracciamento	21
6.3.1	Casi d'Uso-Requisiti e viceversa	21
6.3.2	Requisiti-Test	21
6.3.3	Requisiti-Componenti e viceversa	22
6.3.4	Componenti-Design Pattern e viceversa	22
6.3.5	Componenti-Classi	22
7	Progettazione	22
7.1	Stile di progettazione	22
7.2	Design Pattern	23
7.3	Convenzioni sui diagrammi	23
7.4	Classi di verifica	24
7.5	Tracciamento	24
7.6	Struttura della Specifica Tecnica	24
8	Progettazione di dettaglio	24
8.1	Diagrammi	25
8.2	Specifica di una classe	25
8.3	Tracciamento	25
8.4	Struttura della Definizione di Prodotto	25
8.5	Convenzioni di scrittura	26
9	Codifica	27
9.1	Intestazione dei file	27
9.2	Convenzioni di codifica	27
9.2.1	Convenzioni sui nomi	27
9.2.2	Struttura delle classi	27
9.2.3	Struttura del codice	27
9.3	Verifica del codice	28
9.4	Norme sul codice Java	28
9.4.1	Manuale di Javadoc	29
9.5	Norme sul codice JavaScript	29
10	Verifica e Validazione	31
10.1	Strumenti di verifica	31
10.2	Tecniche di verifica	32
10.2.1	Analisi statica	32
10.2.2	Analisi dinamica	33
10.3	Test di unità, integrazione e sistema	34
10.3.1	JUnit e elemma	34
10.3.2	JUnit e JSCoverage	34

Elenco delle tabelle

1	Revisioni e Milestones	7
---	----------------------------------	---



Elenco delle figure

1	Ciclo di vita di un documento	10
2	Creazione di una nuova Milestone	14
3	Modifica di una Milestone	15
4	Creazione di un nuovo Ticket	16
5	Modifica di un ticket	17
6	Ciclo di vita di un ticket	18

1 Introduzione

1.1 Scopo del prodotto

Con il progetto “MyTalk” si intende un sistema software di comunicazione tra utenti mediante browser senza la necessità di installazione di plugin e/o software esterni. L’utente avrà la possibilità di interagire con un altro utente tramite una comunicazione audio - audio/video - testuale e, inoltre, ottenere delle statistiche sull’attività in tempo reale.

1.2 Scopo del documento

Questo documento viene redatto per definire le norme da adottare da parte di tutti i componenti del gruppo Software Synthesis durante il periodo di svolgimento del capitolato MyTalk, commissionato dall’azienda Zucchetti S.r.l. In particolare si andranno a definire le regole per:

- Relazioni interpersonali e comunicazione
- Definizione dell’ambiente di lavoro
- Redazione documenti
- Convenzioni e norme per l’analisi e la progettazione
- Convenzioni e norme per la verifica di file e documenti

1.3 Ambiguità

Al fine di evitare incomprensioni dovute all’uso di termini tecnici nei documenti, viene redatto e allegato il documento *glossario.3.0.pdf* dove vengono definiti e descritti tutti i termini marcati con una sottolineatura.

2 Comunicazioni e relazioni intrapersonali

2.1 Comunicazione interna

La comunicazione tra i vari componenti del team avverrà principalmente durante gli incontri che si terranno in un luogo fisico comune. Per evitare problemi dovuti alla mancata presenza da parte di un membro, le notifiche dell'attività svolta durante l'incontro verranno scritte in un calendario comune messo a disposizione.

Nei casi in cui il lavoro si svolgesse presso la propria abitazione, si potrà utilizzare *Skype* per avviare chat, chiamate e videoconferenze di gruppo.

Qualora si evinca la necessità di un ritrovo fisico comune ma vi fosse l'impossibilità da parte di alcuni membri di recarsi nel luogo prefissato, allora si procederà suddividendo il gruppo in due sottogruppi, ciascuno con un luogo di ritrovo prefissato. I due team di sviluppo quindi potranno comunicare con l'altro team mediante il software di comunicazione prestabilito.

2.2 Comunicazione esterna

I contatti verso l'esterno saranno a cura del responsabile di progetto. A tale scopo è stato creato un indirizzo di posta elettronica tramite il quale la persona incaricata tratterà a nome dell'intero gruppo Software Synthesis

L'email sopra descritta è *info@softwaresynthesis.org* e l'inoltro di risposte a tutti gli altri membri sarà sempre a carico del responsabile di progetto che dovrà utilizzare i metodi descritti nella sezione 2.1.

2.3 Incontri Interni

Saranno fissate delle riunioni formali decise dal responsabile di progetto che provvederà rendere noto a tutti luogo, data e ora della seduta mediante email personale con almeno tre giorni di anticipo.

Nell'email sarà contenuta anche la motivazione per cui si è reso necessario l'incontro. Ai membri è richiesta la conferma di partecipazione tramite risposta, sempre via email. Nel caso in cui un componente del team non potesse essere presente alla riunione decisa, dovrà specificarne il motivo nell'email di risposta al responsabile.

Ogni membro del gruppo ha la possibilità di richiedere un incontro interno: tale domanda dovrà venir indirizzata sempre al responsabile di progetto, il quale la visionerà e pianificherà un eventuale incontro.

Ad ogni riunione verrà redatto inoltre un verbale in cui verranno annotate tutte le decisioni prese e i punti salienti della discussione.

2.4 Incontri Esterni

Sarà il responsabile di progetto a prendere accordi per incontri con il committente o con i proponenti.

Ogni membro del gruppo può richiedere un incontro esterno al responsabile, presentando una motivazione. Questa necessità verrà presentata all'intero team e solo nel momento in cui ci saranno tre conferme, con relativa presenza all'incontro, il responsabile provvederà a prendere appuntamento con la parte esterna. In caso contrario la proposta sarà bocciata.

2.5 Considerazioni sui rapporti interpersonali

In questa sotto-sezione si intende riassumere i punti focali inerenti i rapporti tra i membri del team. Tali sono da intendere come suggerimenti da rispettare per il rispetto comune, al fine di non far insorgere litigi controproducenti allo sforzo comune.

- **Usare un linguaggio consono e rispettoso:** si invitano tutti i membri del team ad assumere un comportamento rispettoso verso il prossimo. Con ciò si fa riferimento all'uso di un linguaggio e gestualità civile (anche in virtù del fatto che molti luoghi di riunione sono pubblici).
- **Nessuno è superiore agli altri:** è evidente che tutti i membri del team hanno la medesima esperienza nel campo dell'ingegneria del software (da intendersi sia come conoscenze generali che come grado di "maturità" nel settore).

Pertanto in ogni dibattito è richiesto ad ogni membro del team di ricordare che le idee altrui non sono da criticare a prescindere, ma bensì vanno valutate e giudicate coscientemente, dando luogo ad un dibattito nel rispetto di ogni membro del team.

- **Mai mettere alla berlina chi sbaglia:** indubbiamente ogni membro del team, ha delle responsabilità che cambiano in virtù del ruolo che ricopre. Va però ricordato che in caso d'errore non si dovrà giudicare colpevole e conseguentemente screditare un componente del gruppo.

Ciò potrebbe si portare chi ha sbagliato a prestare una maggiore attenzione al proprio operato, ma il rischio di portare il suddetto membro del gruppo a non esporre le proprie idee per pregiudizi su quanto accaduto potrebbe nuocere alla qualità del prodotto finale.

Sarà quindi esclusiva responsabilità del responsabile comunicare in privato gli errori ai relativi "colpevoli". Così facendo il soggetto sarà corretto e si eviterà che questi si senta pubblicamente umiliato.

3 Ambiente di lavoro

3.1 Repository

Per un corretto svolgimento del progetto si è reso necessario adottare un luogo dove poter cercare e salvare tutti i documenti da redigere, nonché i file di codifica. Per questo si è deciso di adottare un *repository* ospitato da *GitHub*.

Il motivo principale di questa scelta sta nel fatto che *git*, rispetto ad altri sistemi quali *Sourceforge* ad esempio, è un sistema di controllo di versione distribuito.

Rientra infatti nei *repository* di tipo *Distributed Control Version System* (DVCS) mentre *Sourceforge* rientra nei *repository* di tipo *Centralized Control Version System*.

La differenza sta nel fatto che in *git* ognuno avrà in locale l'intera copia del *repository*, quindi tutti i vari *commit*, rami ecc. mentre con *Sourceforge* in locale si ha solo la situazione dell'ultimo *commit* e se si deve eseguire operazioni di *rollback* si ha bisogno della connessione di rete per potersi connettere al *server* centrale.

3.1.1 Registrazione

Per utilizzare *GitHub* occorre registrarsi presso:

<https://github.com>.

per far ciò basta inserire un *username*, un indirizzo *email* e una *password* nell'apposito form proposto nella pagina sopra indicata.

3.1.2 Installazione

Per aiutare i componenti del gruppo nella fase di installazione di *git*, è stata creata appositamente una guida che passo passo spiega come installare e configurare nel modo giusto il software.¹

3.1.3 Struttura

- **Progetto:** l'indirizzo web a cui fa riferimento l'intero progetto è:

<https://github.com/SoftwareSynthesis/SoftwareEngineeringProject>

- **Documentazione:** tutti i documenti saranno reperibili all'indirizzo

<https://github.com/SoftwareSynthesis/SoftwareEngineeringProject/Documents>

Ogni documento redatto dovrà essere contenuto in una sotto-cartella della directory *Documents* nominata come il nome del documento stesso: questa conterrà i file *L^AT_EX*.

È inoltre presente una cartella *Revisioni*, composta a sua volta da quattro sotto-cartelle (*RR*, *RP*, *RQ*, *RA*) che conterranno i documenti formali consegnati alle quattro revisioni del progetto ed una cartella *pics* che conterrà tutte le immagine utilizzate nei documenti.

- **Codice:** tutti i file sorgente saranno reperibili all'indirizzo

<https://github.com/SoftwareSynthesis/SoftwareEngineeringProject/Code>

- **Others:** per qualsiasi altro file di utilità per il progetto, è stata predisposta una cartella *Others* utilizzabile da tutti i membri.

¹La guida è reperibile all'indirizzo <https://github.com/SoftwareSynthesis/SoftwareEngineeringProject/blob/master/Manuals/GuidaGit/MasterDocument.pdf>.

3.1.4 Sistema di versionamento

Come sistema di controllo di versione è stato adottato *git*. Questa scelta è stata decisa dopo aver individuato diversi pregi, tra i quali:

- velocità;
- design semplice;
- forte supporto allo sviluppo non-lineare (possibilità di migliaia di rami paralleli);
- completamente distribuito;
- capacità di gestire, in modo efficiente (velocità e dimensione dei dati), grandi progetti come il kernel Linux.

3.2 Sistema di tracciamento dei requisiti

Al fine di rendere automatico e sistematico la gestione del tracciamento dei requisiti, è stato creato un sistema che si appoggia al sito aziendale. Il gestionale è stato nominato “SynthesisRequirementManager”. Ogni volta che un membro del gruppo necessita di interagire con tale sistema, si dovrà autenticare alla pagina

<http://www.softwaresynthesis.org/Login.php>

Il sistema di tracciamento offre le seguenti 5 opzioni:

- **Gestione requisiti:** permette l’inserimento, modifica e la cancellazione di un requisito;
- **Gestione fonti:** permette l’inserimento, modifica e la cancellazione di una fonte da intendersi come ad esempio il capitolato d’appalto;
- **Gestione casi d’uso:** permette l’inserimento di un caso d’uso con relativi scenari e requisiti associati;
- **Gestione attori:** permette l’inserimento, modifica e la cancellazione di un attore utilizzabile in seguito per lo sviluppo di un caso d’uso;
- **Gestione classi:** permette l’inserimento e la cancellazione di una classe rilevata nell’attività di progettazione;
- **Gestione componenti:** permette l’inserimento e la cancellazione di una componente rilevata nell’attività di progettazione;
- **Gestione design pattern:** permette l’inserimento e la cancellazione di un design pattern utilizzato nel progetto;
- **Stampa:** è suddivisa in due sezioni, la prima per la parte di analisi, la seconda per la parte di progettazione.

Il comportamento delle due stampe è analogo, crea in modo automatico un file di estensione *.tex* contenente nel primo caso la lista dei requisiti analizzati, i casi d’uso studiati e il tracciamento tra i requisiti-casi d’uso e viceversa, nel secondo caso viene stampato la lista di tracciamenti tra classi-componenti-design pattern (vedi sezione 7.5). Successivamente alla stampa tale file sarà da includere all’intero del documento “*analisi_dei_requisiti.tex*” o “*specifica_tecnica.tex*” a seconda.

Il sistema è basato su tecnologie MySQL con creazione del database sotto engine inno_db. Per sopperire ad alcune mancanze del sistema di tracciamento, lo sviluppatore si potrà appoggiare alla piattaforma phpMyAdmin fornita dal web host, al fine di modificare e gestire alcuni dati del database.

In particolare, se ne obbliga l'utilizzo nei seguenti casi:

- modifica di un caso d'uso;
- cancellazione di un caso d'uso.

Quest'obbligo di gestione è dovuto a mancanze temporali nella fase di sviluppo del sistema, nel periodo antecedente alla consegna dei capitolati, e anche in parte per la complessità di implementazione.

Ogni altra modifica attuata tramite *phpmyadmin* è assolutamente vietata per mansioni che non siano riportate al punto precedente.

3.3 Sistema operativo

L'intero progetto verrà portato avanti attraverso sistemi Unix e Windows, più in particolare Mac OSX 10.8.2, Fedora 17, Windows 7, Windows 8. Questa scelta deriva dal fatto che, essendo il progetto MyTalk un applicativo web, non si sono riscontrate dipendenze alcune da librerie di sistema.

3.4 Diagrammi UML

Per la produzione di diagrammi UML si adotterà il software ConceptDraw. Questa scelta è stata fatta perchè il *tool* si presenta molto semplice da usare, è molto potente e permette la creazione di moltissimi diagrammi UML adottando lo standard 2.0.

La qualità dei diagrammi prodotti inoltre è molto buona e le modalità con cui tali diagrammi verranno inseriti sono descritte nella sezione 4.9.

3.5 Diagrammi di Gantt

Come supporto alla pianificazione del progetto si è scelto di utilizzare Project Libre nella versione 1.5.2, software molto leggero e facile da installare. È uno strumento *opensource* e multi piattaforma. Permette la creazione di diagrammi di Gantt in modo molto semplice con possibilità di esportarli in formato jpg.

Per il download e l'installazione si rimanda al seguente link:

<http://sourceforge.net/projects/projectlibre/>.

4 Ambiente documentale

In questa sezione verranno illustrati i vari standard utilizzati dal team per la produzione di documenti durante l'intero progetto.

4.1 Software utilizzati

Tutti i documenti dovranno essere scritti in italiano con \LaTeX , un software free di composizione testuale che comprende una serie di caratteristiche atte alla produzione di documentazione tecnica e scientifica.

Per attuare questa scelta, l'azienda Software Synthesis ha deciso di utilizzare come editor \LaTeX *TeXMaker*, software disponibile sia per ambienti Unix che Windows. Si presenta come un software molto flessibile che include al suo interno la correzione ortografica automatica, l'autocompletamento e un visualizzatore di PDF integrato.

La versione da utilizzare è l'ultima disponibile in tale momento ed è la 3.5.2. Per il download seguire il seguente link:

<http://www.xmlmath.net/texmaker/download.html>.

Per l'installazione, una volta scaricato, basta seguire le indicazioni presenti nel file scaricato precedentemente.

4.2 Nomi dei documenti

Il nome dei documenti dovrà rappresentare in modo univoco la natura del file e sarà composto nel seguente modo:

nome_del_file.X.Y.estensione

dove:

- **nome_del_file**: il nome del file non dovrà contenere lettere maiuscole né caratteri speciali (compresi accenti). Nel caso sia composto da più parole, queste vanno separate con il carattere “*underscore*”.
- **Variabile X**: Indica il raggiungimento di uno stato formale rispetto ad una Milestone. Assumerà quindi principalmente 4 valori, dall'1 al 4, attribuiti come segue:

Milestone	X
Revisione dei Requisiti (RR)	1
Revisione dei Requisiti (RP)	2
Revisione dei Requisiti (RQ)	3
Revisione dei Requisiti (RA)	4

Tabella 1: Revisioni e Milestones

- **Variabile Y**: parte dal valore 0 e viene incrementata ogni qualvolta, senza limite superiore, si apportano modifiche sostanziali al documento, come ad esempio la stesura di una nuova sezione o correzione di errori.

4.3 Impostazioni di base di un documento

Ogni documento dovrà attenersi alle seguenti regole per la sua redazione:

- **Intestazione:** composta dalla sezione del documento nella parte sinistra e dal logo nella parte destra.
- **Piè di pagina:** composto dal nome del documento con relativa versione sulla sinistra e dal numero di pagina sulla destra.

La prima pagina di ogni documento dovrà contenere come prima cosa il logo dell'azienda Software Synthesis, il nome del documento e una tabella che riporta le seguenti informazioni sul file:

- nome del file;
- versione;
- data creazione;
- data ultima modifica;
- stato;
- uso;
- redattori;
- approvato da;
- verificatori.

I redattori presenti in tale pagina sono tutti e soli quelli che hanno operato sul documento nella fase temporale conclusasi al momento della consegna di un documento (ovvero tra due milestone progettuali), così come i verificatori designati. Nella seconda pagina invece dovrà essere presente la tabella delle modifiche apportate. Queste dovranno essere inserite in ordine cronologico dalla più recente alla creazione, specificando:

- versione del documento;
- descrizione dell'intervento;
- redattore;
- data modifica.

Seguirà su una nuova pagina l'indice dell'intero documento e in caso di presenza di tabelle e immagini sarà presente anche l'indice delle tabelle e l'indice delle immagini a seguire. La presenza di questi due indici va specificata impostando a *true* due flag presenti nel documento "Template" e ognuno di essi dovrà comparire in una nuova pagina.

Il documento dovrà essere suddiviso in sezioni con relative sottosezioni se presenti. Ogni sezione deve cominciare in una nuova pagina.

4.4 Tipi di documento

- **Informali:** verranno “etichettati” come informali tutti i documenti in fase di redazione, oppure completati ma non ancora approvati dal responsabile di progetto. Vanno utilizzati solo internamente e quindi non verranno distribuiti all'esterno.

Sono reperibili nella cartella **Documents** del *repository* e sono organizzati secondo quanto previsto nella sezione 3.1.3 “Struttura repository”.

- **Formali:** tutti i documenti approvati dal responsabile di progetto sono da considerarsi formali e quindi pronti alla distribuzione verso l'esterno. Sono reperibili nella cartella **Revisioni** del *repository* e sono organizzati secondo quanto previsto nella sezione 3.1.3 “Struttura repository”.
- **Verbali:** documenti redatti dal responsabile di progetto dopo un avvenuto incontro esterno. Servono come promemoria dell'incontro avvenuto e delle tematiche trattate. Dovranno essere denominati

verbale_incontro_{dataincontro}

dove con {dataincontro} intendiamo la data dell'incontro espressa nel formalismo descritto nella sezione 4.8.4 “Formati Ricorrenti”.

Ogni verbale dovrà contenere le seguenti informazioni riportate come seguono:

- **Data:** indica la data dell'incontro
- **Ora:** indica l'ora dell'incontro espressa in “HH:MM” dove HH sarà compresa tra 00 e 23 ed indicherà le ore mentre MM sarà compreso tra 00 e 59 ed indicherà i minuti.
- **Luogo:** indica il luogo dell'incontro e dovrà essere espresso nel seguente formalismo:

provincia, città, via, numero civico

- **Durata:** indica la durata dell'incontro espressa in minuti
- **Partecipanti:** indica tutti i partecipanti presenti alla riunione. Deve essere un elenco di nomi e cognomi suddivisi per membri interni ed esterni.
- **Oggetto:** indica l'argomento trattato all'incontro. Seguono i punti salienti e le decisioni prese.

Il verbale dovrà essere approvato da tutti i componenti firmandolo entro e non oltre il successivo incontro. L'approvazione è ufficializzata automaticamente nel momento in cui sono poste tutte le firme dei presenti.

In via del tutto eccezionale, nel qual caso un componente non possa approvare il documento nei termini prestabiliti, è compito del responsabile assicurarsi che ciò avvenga il prima possibile.

Nel caso in cui un membro del gruppo non abbia potuto partecipare alla riunione, esso dovrà comunque prendere visione del verbale e dovrà firmarlo per accertare la presa visione dello stesso.

4.5 Ciclo di vita di un documento

Come mostrato in figura 1, un documento seguirà i seguenti passi:

1. Il redattore di un nuovo documento dovrà redigere in locale la prima impostazione del documento e solo allora potrà caricarlo nel *repository* secondo le regole indicate nella sezione 3.1 “repository”.

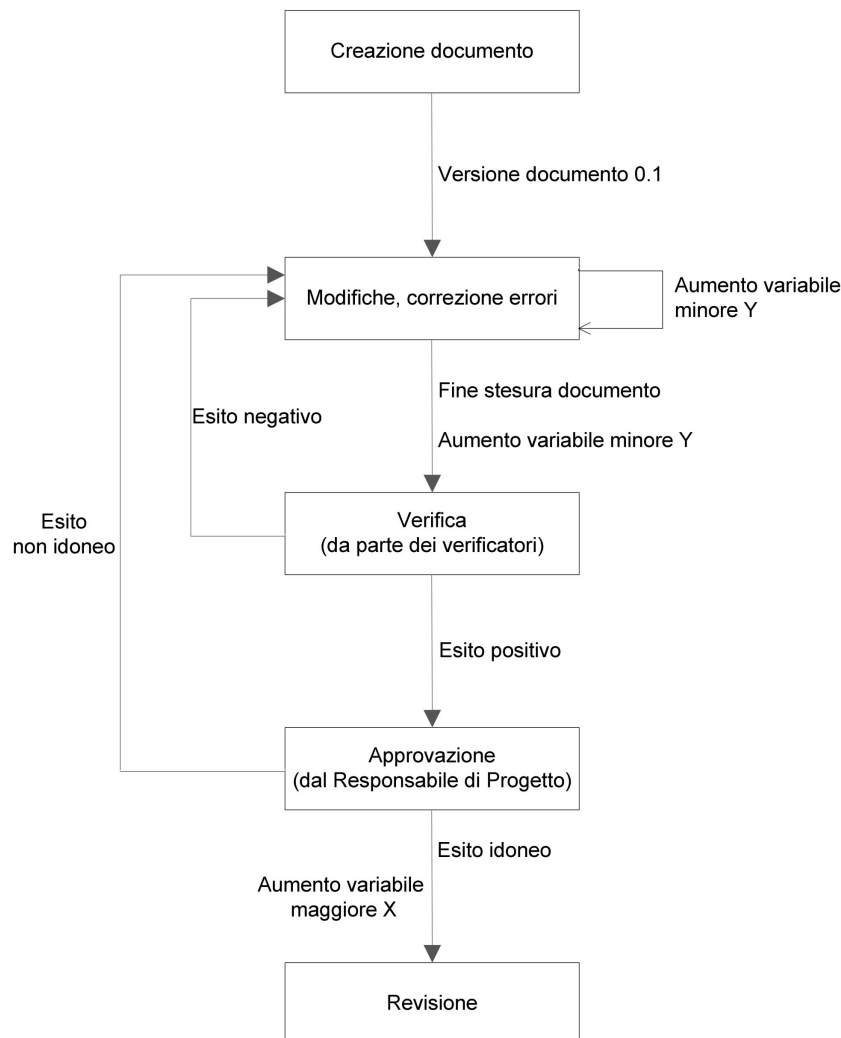


Figura 1: Ciclo di vita di un documento

2. Ad ogni sostanziale modifica verrà incrementato il contatore Y (vedi sezione 4.2 “Nomi di documenti”) e seguirà un *commit* nel *repository*.
3. Quando il redattore finisce la stesura o revisione del documento, questo verrà proposto al verificatore il quale avrà il compito di controllare che siano state seguite le norme riguardanti la stesura di documenti (vedi sezione 4.12 “Verifica dei documenti”). Il verificatore potrà dare due esiti, positivo o negativo.
4. In caso di esito negativo, il documento viene riproposto al redattore evidenziandone gli errori riscontrati. Se invece l'esito è positivo, il documento è stato redatto secondo le norme e non presenta errori, passerà quindi al responsabile di progetto che provvederà all'approvazione formale.
5. Anche il responsabile di progetto può dare due giudizi, idoneo o non idoneo. Nel caso di idoneità, il documento è da considerarsi formale e può essere inserito nella cartella corrispondente alla *milestone* presente nella directory **Revisioni**.

6. In caso di non idoneità, il documento verrà rimandato al redattore notificando il motivo di tale decisione e si ripartirà dal punto 3.

4.6 Template

Ogni documento dovrà essere generato includendo il *template* L^AT_EX presente nella cartella **Documents**.

Questo file è stato redatto prima dell'inizio di stesura di ogni altro documento sotto comune accordo. La modifica perciò di tale *template* deve essere richiesta all'amministratore di progetto che analizzerà l'esigenza e darà una risposta positiva o negativa.

In caso positivo, si procederà all'attuazione delle modifiche e verrà informato l'intero team al fine che ogni membro modifichi il documento su cui sta lavorando.

4.7 Glossario

Il glossario è un documento nel quale vengono riportate tutte le parole difficilmente comprensibili o dal significato ambiguo presenti nei documenti. È un documento un po' particolare in quanto non rispetta le regole stilistiche degli altri documenti.

Non sarà presente l'indice dei contenuti, ma solo una pagina iniziale contenente le informazioni del documento, la tabella delle modifiche, una breve introduzione e a seguire le voci ordinate alfabeticamente.

4.8 Norme tipografiche

Questa sezione descrive le norme da utilizzare per l'utilizzo dell'ortografia, tipografia e l'assunzione di uno stile uniforme nel redigere i documenti.

4.8.1 Stili di testo

- **Grassetto:** usato per evidenziare passaggi importanti. Si utilizza anche su elementi immediatamente seguenti agli elenchi per evidenziare l'oggetto trattato nel paragrafo.
- **Corsivo:** usato per dare enfasi a parole o frasi su cui prestare attenzione, nonché per indicare i forestierismi (ad es. i termini in inglese presenti in un documento).
- **Sottolineato:** le parole sottolineate sono riportate nel glossario. Al fine di facilitare la lettura, nei documenti dovrà essere sottolineata solo la prima occorrenza di ciascun termine la cui definizione è riportata nel glossario.
- **Maiuscolo:** deve essere utilizzato esclusivamente per sigle e acronimi.

4.8.2 Punteggiatura

Qualsiasi segno di punteggiatura non deve mai seguire uno spazio ma deve precederne uno. Dopo un punto fermo deve esserci uno spazio seguito da una lettera maiuscola.

Le parentesi vanno usate per raggruppare un periodo, non devono aprirsi con un carattere di spaziatura e non devono chiudersi con un carattere di punteggiatura o di spaziatura.

4.8.3 Composizione del testo

Gli elenchi, sia puntati che numerati, vanno usati per elencare una serie di elementi e solitamente hanno una breve lunghezza con la quale descrivono l'elemento in questione.

Per questo, ogni punto elenco deve finire con un punto e virgola (;) tranne l'ultimo elemento che finirà con un punto (.), mentre nel caso di punti elenco più discorsivi la frase finirà con un punto ogni volta.

Per quanto concerne le note a piè di pagina invece, ogni nota dovrà cominciare con la prima lettera della prima parola maiuscola, senza spaziatura davanti e finire con un punto.

4.8.4 Formati ricorrenti

- **Path:** per scrivere indirizzi web e url si adotterà il comando prestabilito da L^AT_EX `\url`.
- **Date:** per scrivere una data si dovrà utilizzare il formato AAAA-MM-GG (standard ISO 8601) dove:
 - AAAA: sta ad indicare l'anno;
 - MM: sta ad indicare il mese;
 - GG: sta ad indicare il giorno.
- **Sigle:** di seguito sono elencate una serie di sigle, da usare principalmente all'intero di diagrammi o tabelle per risparmiare spazio.
 - AR: documento "Analisi dei Requisiti"
 - PdP: documento "Piano di Progetto"
 - PdQ: documento "Piano di Qualifica"
 - NdP: documento "Norme di Progetto"

4.9 Immagini

Tutte le immagini utilizzate nei documenti devono avere estensione `.png`. Nel caso l'immagine abbia una grande dimensione, va compressa con il formato `.jpg`. Devono risiedere tutte nella cartella `Documents/pics`.²

Nel caso di immagini riguardanti diagrammi UML, si deve rispettare la seguente regola per il nome file:

$$tipologia\{x1-x2-...-xn\}$$

dove "tipologia" dovrà essere una di queste sigle:

- **UC:** diagrammi casi d'uso;
- **DC:** diagrammi delle classi;
- **DO:** diagrammi degli oggetti;
- **DS:** diagrammi di sequenza;
- **DA:** diagrammi di attività;
- **DP:** diagrammi di package.

e "x1-x2-...-xn" rappresenteranno la numerazione gerarchica dei diagrammi, dove il numero più a sinistra rappresenta il padre e quelli più a destra i figli.

Nel caso di immagine generiche, il loro nome dovrà rispecchiare il contenuto di tale immagine.

²Si veda <https://github.com/SoftwareSynthesis/SoftwareEngineeringProject/Documents/pics>.

4.9.1 Inserimento immagini

L'inserimento di immagini all'interno del documento verrà fatto tramite l'utilizzo del comando `\begin{figure}` e `\includegraphics[parametri]{riferimento}` dove:

- **parametri:** indica una serie di parametri da utilizzare per formattare al meglio l'immagine, come una scala di ridimensionamento (`[scale=0.8]` indica un ridimensionamento del 20%), l'altezza e larghezza (`[width=2cm, heigh=4cm]` indica un'immagine larga 2 cm e alta 4 cm) oppure parametri del tipo `[width=\textwidth]` per indicare che la figura corrisponda esattamente alla larghezza del testo;
- **riferimento:** va messo il nome dell'immagine presente nella cartella `pics` (vedi sezione 3.1.3) che si vuole inserire.

Prima della chiusura (`\end{figure}`) vanno inseriti anche i comandi `\caption{didascalia}` e `\label{riferimento}` per permettere l'indicizzazione e l'identificazione delle immagini tramite un codice.

4.10 Inserimento tabelle

L'inserimento di tabelle all'interno del documento verrà fatto tramite l'utilizzo del package `tabularx` che permette la creazione di colonne a larghezza dinamica.

Vanno inseriti prima della chiusura della tabella (quindi prima del comando `\end{tabularx}`) i comandi `\caption{didascalia}` e `\label{riferimento}` per permettere l'indicizzazione e l'identificazione delle tabelle tramite un codice.

4.11 Inserimento link

L'inserimento di riferimenti ipertestuali a pagine web avviene mediante il comando

```
\url{link ipertestuale}
```

dove con *link ipertestuale* indica l'indirizzo nella forma estesa

```
http://www.dominio./estensione/...
```

4.12 Verifica dei documenti

La verifica di un documento avviene nel momento in cui il redattore ha finito di redigere completamente il documento.

Tale processo si articolerà principalmente in tre fasi:

1. Si procederà con un'analisi ortografica, compreso il corretto uso della punteggiatura, degli accenti e degli apostrofi. Per far ciò il verificatore potrà aiutarsi con lo strumento "Verifica ortografia" presente in TexMaker raggiungibile dalla scheda "Modifica->Verifica Ortografia". Successivamente controllerà l'assenza di errori grammaticali, logici e sintattici nei periodi ed infine la coerenza del documento con le norme previste nella sezione 4 "Ambiente documentale".
2. Se è la prima volta che il documento viene verificato, il verificatore procederà con un metodo di tipo *walkthrough*, come descritto in sezione 10.2.1. Se invece il documento era già stato verificato e rimandato per via di errori, si adopererà un metodo di tipo *inspection*, come descritto in sezione 10.2.1.
3. Il verificatore prenderà nota in modo ordinato di tutti gli errori riscontrati e procederà con la creazione dei *tickets* relativi che saranno rivolti al redattore del documento come descritto nella sezione 5.2.3 "Ciclo di vita di un Ticket".

5 Milestones e Ticketing

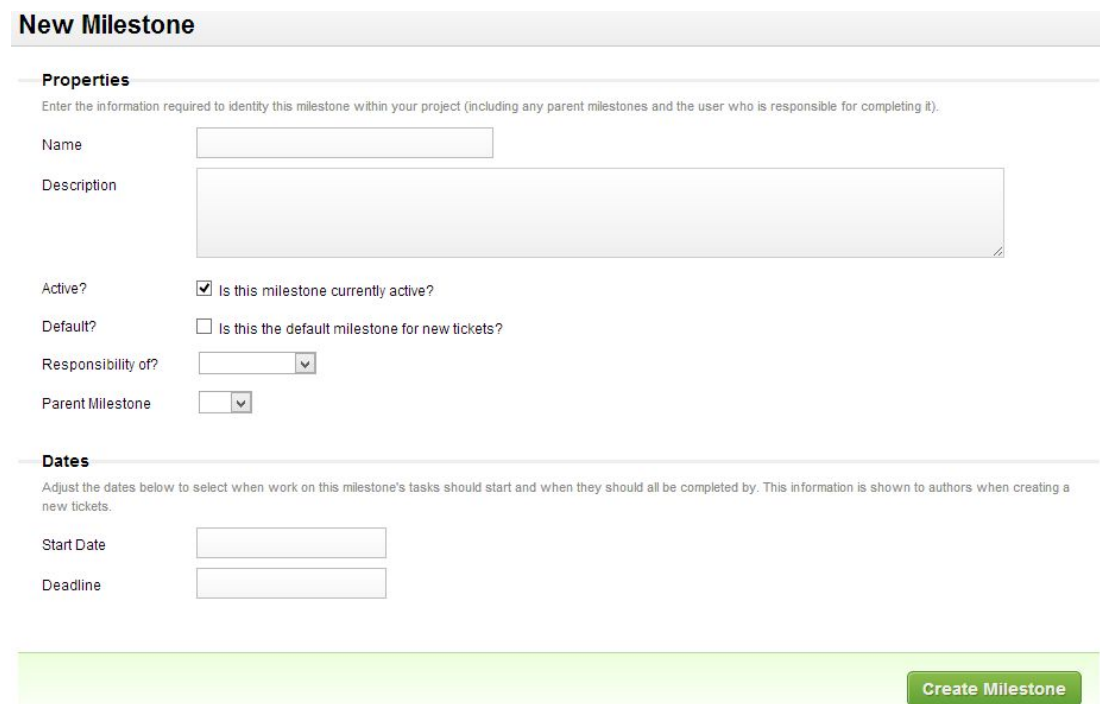
Allo scopo di agevolare lo svolgimento delle operazioni di assegnazione dei lavori e revisione durante l'intero ciclo di vita del progetto, l'azienda Software Synthesis userà il servizio Codebase. Questo permette la creazione di *milestone* e *tickets*, strumenti molto utili per tracciare il lavoro del team e anche per delineare una roadmap degli obiettivi a breve e lungo termine che l'azienda si prefigge di raggiungere.

5.1 Milestones

Come prima cosa, il responsabile di progetto dovrà inserire tutte le *milestone* previste dal progetto, ossia le quattro revisioni.

5.1.1 Creazione nuova Milestone

Per inserire una nuova *milestone* ci si dovrà posizionare nella pagina web dedicata alle *milestone* e cliccare su "New Milestone". Il form va completato nel seguente modo:



New Milestone

Properties

Enter the information required to identify this milestone within your project (including any parent milestones and the user who is responsible for completing it).

Name

Description

Active? ☒ Is this milestone currently active?

Default? ☐ Is this the default milestone for new tickets?

Responsibility of?

Parent Milestone

Dates

Adjust the dates below to select when work on this milestone's tasks should start and when they should all be completed by. This information is shown to authors when creating a new tickets.

Start Date

Deadline

Create Milestone

Figura 2: Creazione di una nuova Milestone

- **Name:** inserire il nome della *milestone*;
- **Description:** inserire una breve descrizione della *milestone*;
- **Active?:** flag da spuntare se si vuole che la *milestone* creata sia quella attiva;
- **Default?:** flag da spuntare se si vuole che i futuri *tickets* creati siano riferiti a questa *milestone*;
- **Responsibility of?:** selezionare la persona responsabile;

- **Parent *Milestone*:** selezionare se la *milestone* che si va a creare è figlia di qualche altra *milestone*;
- **Start Date:** inserire la data da cui far iniziare la *milestone*;
- **Deadline:** inserire la data di chiusura delle *milestone*.

Nel caso non si conosca qualche informazione, si può lasciare il relativo campo bianco e modificarlo successivamente (vedi paragrafo seguente) quando sarà resa pubblica l'informazione.

5.1.2 Modifica Milestone

Per modificare una *milestone*, ci si deve posizionare nella scheda "Edit *milestone*". Per raggiungerla, basta entrare nella *milestone* che si vuole modificare e cliccare sul bottone "Edit *milestone*" situato nella parte alta dello schermo a destra.



Figura 3: Modifica di una Milestone

Per la compilazione dei campi, valgono le stesse regole riportate nella sezione 5.1.1 "Creazione nuova *milestone*".

Da notare che si può eliminare anche una *milestone* da tale scheda, basta cliccare sul bottone "Delete Milestone" presente nella parte destra della pagina.

5.2 Ticketing

Un ticket rappresenta un compito da svolgere.

5.2.1 Creazione nuovo Ticket

Per inserire un nuovo ticket ci si dovrà posizionare nella pagina web dedicata ai *tickets* e cliccare su "New Ticket". Il form va completato compilando obbligatoriamente i seguenti campi:

- **Summary:** inserire il nome del compito da assegnare;
- **Status:** da impostare su "New";
- **Priority:** selezionare la priorità effettiva del compito;
- **Ticket Type:** da impostare su "Task";
- **Assigned User:** selezionare la persona che dovrà svolgere il compito;
- **Milestone:** selezionare la *milestone* corrispondente del compito.

Tutte le altre informazioni possono essere completate se si ritiene opportuno dare maggiori dettagli.

New Ticket

Summary — enter a brief description of this ticket (up to 250 characters)

Additional information — you may enter optional information for this ticket in the field below (this is optional)

Rich Text Editor: B I [link icon] [quote icon] [code icon] [table icon] [list icon] [H2 icon] [undo icon] [redo icon] [bold icon] [italic icon] [link icon] [unlink icon] [search icon]

Form Fields:

STATUS New ▼	PRIORITY Normal ▼	CATEGORY General ▼	TICKET TYPE Feature ▼
ASSIGNED USER [dropdown]	TAGS [text input]		

Navigation Tabs: Ticket Properties | Milestone & Dates | Attachments | Watchers

This ticket will be added to the **RR** milestone. To change this select the Milestone and Dates tab above.

Create Ticket

Figura 4: Creazione di un nuovo Ticket

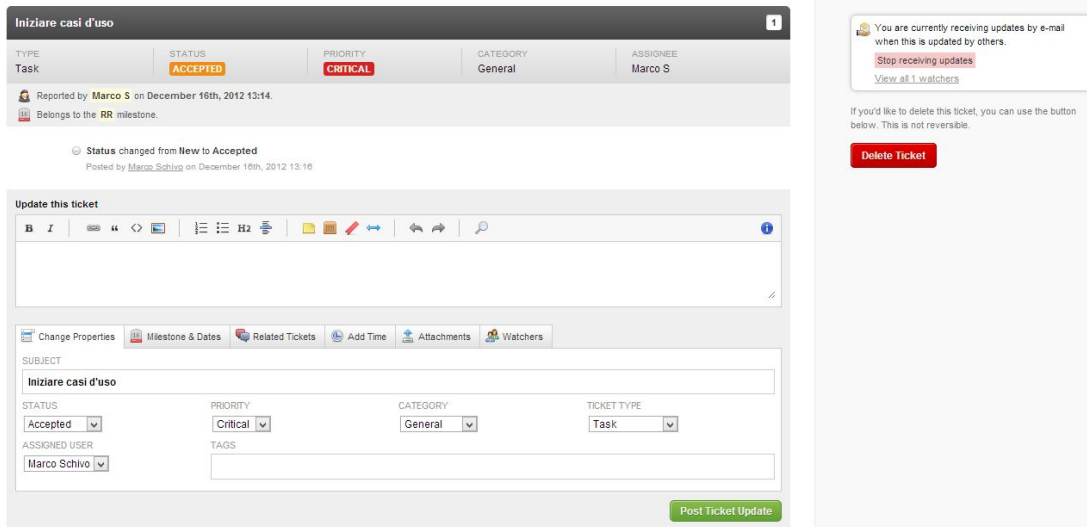
5.2.2 Modifica Ticket

Durante l'intero ciclo di vita di un ticket, questo viene modificato molte volte come avviene ad esempio quando dobbiamo modificare lo stato del ticket. Per far ciò, basta entrare nel ticket che si vuole modificare e appare subito la scheda di modifica.

5.2.3 Ciclo di vita di un Ticket

Come riportato in figura 6, il ciclo di vita di un ticket si articolerà nel seguente modo:

1. **Creazione:** come prima cosa, il ticket va creato come descritto nella sezione 5.2.1 “Creazione nuovo Ticket”.
2. **Esecuzione:** ogni membro dovrà prendere atto dei *tickets* a lui pervenuti e una volta visionati impostare lo stato ad “*Accepted*” o ad “*Invalid*”. Una volta accettati, lo stato va modificato in “*In Progress*” quando si inizia a lavorarci e “*Completed*” quando si è portato a termine il compito assegnato.
3. **Verifica:** una volta che il compito è stato completato, si procede nella creazione di un nuovo ticket di tipo “*Verification*” il quale verrà assegnato ad un verificatore. Il nuovo ticket va compilato nel seguente modo:
 - **Summary:** dovrà contenere il nome del documento o file da verificare;
 - **Status:** da impostare su “*New*”;
 - **Priority:** selezionare la priorità effettiva del compito;
 - **Ticket Type:** da impostare su “*Verification*”;
 - **Assigned User:** selezionare il verificatore che dovrà svolgere il compito;
 - **Milestone:** selezionare la *milestone* entro cui il documento dovrà essere verificato.
4. **Esecuzione verifica:** per ogni errore (esclusi quelli grammaticali) dovrà essere creato un nuovo ticket nel seguente modo:



The screenshot displays a web interface for managing tickets. At the top, a header bar shows the ticket title "Iniziare casi d'uso" and a count of 1. Below this, a summary section includes fields for TYPE (Task), STATUS (ACCEPTED), PRIORITY (CRITICAL), CATEGORY (General), and ASSIGNEE (Marco S). It also notes the ticket was reported by Marco S on December 16th, 2012 at 13:14 and belongs to the RR milestone. A status change log indicates the status was changed from New to Accepted by Marco Schivo on December 16th, 2012 at 13:16.

The main section is titled "Update this ticket" and contains a rich text editor with various formatting tools. Below the editor are tabs for "Change Properties", "Milestone & Dates", "Related Tickets", "Add Time", "Attachments", and "Watchers". The "SUBJECT" field contains "Iniziare casi d'uso". Below this are dropdown menus for STATUS (Accepted), PRIORITY (Critical), CATEGORY (General), and TICKET TYPE (Task). There is also a field for ASSIGNED USER (Marco Schivo) and a TAGS field. A "Post Ticket Update" button is located at the bottom right of the form.

On the right side of the interface, there is a notification box stating "You are currently receiving updates by e-mail when this is updated by others." with links to "Stop receiving updates" and "View all 1 watchers". Below this is a warning about deleting the ticket and a "Delete Ticket" button.

Figura 5: Modifica di un ticket

- **Summary:** dovrà contenere il nome del documento o file da modificare;
- **Additional information:** dovrà contenere la descrizione dell'errore presente nel documento;
- **Status:** da impostare su "New";
- **Priority:** selezionare la priorità effettiva del compito;
- **Ticket Type:** da impostare su "Bug";
- **Assigned User:** selezionare il Responsabile del progetto;
- **Milestone:** selezionare la *milestone* entro cui il documento dovrà essere verificato.

Sarà compito del Responsabile approvare o invalidare il ticket. In caso di non approvazione, il Responsabile dovrà decidere a chi far eseguire la modifica. Il membro selezionato dovrà quindi risolvere l'errore. Una volta completata la modifica il Responsabile dovrà riapplicare la procedura a partire dal punto 3.

Quando tutti i *tickets* di una *milestone* avranno stato "Completed", significa che tutte le attività sono state svolte e il Responsabile potrà procedere alla chiusura della *milestone*.

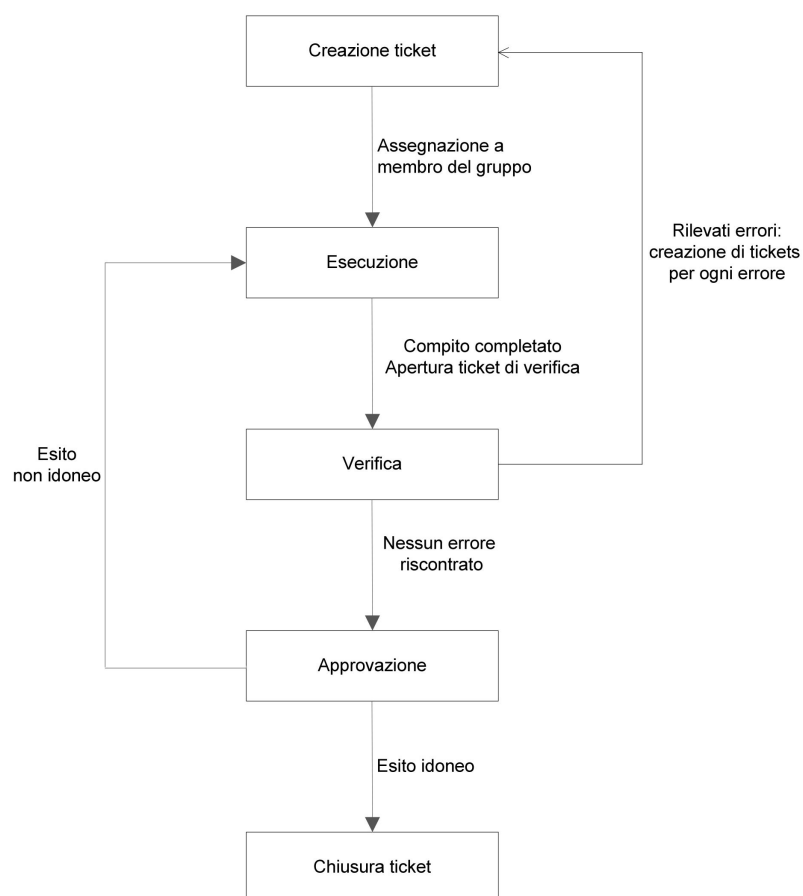


Figura 6: Ciclo di vita di un ticket

6 Analisi dei Requisiti

L'analisi dei requisiti dovrà essere redatta dagli analisti. Il documento si presenterà suddiviso principalmente in tre parti: una prima parte riguardante i casi d'uso, una seconda parte riguardante i requisiti e la parte finale riguardante il tracciamento dei due elementi appena elencati. Inoltre è presente una sezione dedicata ai test, che contiene l'indicazione dei test che verranno eseguiti rispetto ad ogni requisito.

6.1 Norme per i requisiti

La parte riguardante i requisiti dovrà elencare ordinatamente tutti i requisiti evinti dal capitolato d'appalto o dagli incontri con il proponente. Per far ciò, *ogni requisito avrà nome univoco* e dovrà essere indicato nel formalismo:

$$R\{\text{Ambito}\}\{\text{Tipo}\}\{\text{Priorità}\}\{\text{Gerarchia}\}$$

dove:

- **Ambito:** indicherà l'ambito del requisito e potrà assumere i seguenti valori
 - U per indicare “Utente”;
 - S per indicare “di Sistema”.
- **Tipologia:** indicherà il tipo del requisito e potrà assumere i seguenti valori:
 - F per indicare “Funzionale”, ossia requisiti che definiscono quali funzioni devono essere comprese. Tali requisiti si verificano per mezzo di test, dimostrazioni formali e revisioni.
 - Q per indicare “di Qualità”, ossia definisce i requisiti sui fattori di rilievo della qualità, usabilità, sicurezza, scalabilità. Per questi deve essere adoperata una verifica ad hoc.
 - P per indicare “Prestazionale”, ossia requisiti che stabiliscono vincoli su tempistiche e spazio, in relazione alla funzione da svolgere. Vengono verificati tramite misurazioni.
 - D per indicare “Dichiarativo”, ossia requisiti con vincoli spesso ambientali e culturali. Si verificano per mezzo di revisione.
- **Priorità:** indicherà la priorità del requisito e potrà assumere i seguenti valori
 - O per indicare “Obbligatorio”, ossia necessario per la realizzazione del progetto;
 - D per indicare “Desiderabile”, ossia non obbligatorio ma, se implementato, aumenta il valore del progetto;
 - F per indicare “Facoltativo”, ossia un requisito da implementare se avanzano risorse al termine del progetto.
- **Gerarchia:** indicherà il grado di parentela dei requisiti e andrà specificato nella sintassi

$$x.y.z$$

dove il numero più a sinistra rappresenta il padre e quelli più a destra i figli.

Ogni requisito dovrà essere presente nel documento attraverso una “scheda”, dove oltre i dettagli ricavabili dal nome, dovranno essere specificati anche una breve descrizione, la fonte (ossia la loro origine) e il motivo.

Esempio

“RUFO2.3.1” indica un requisito utente, funzionale obbligatorio con padre RUFO2.3.0 che a sua volta è figlio di RUFO2.0.0.

6.2 Norme per i casi d'uso

Il nome di ogni caso d'uso dovrà essere anch'esso, come per i requisiti, univoco e nella forma:

$$UC\{n1.n2...n3\}$$

dove con “n1.n2...n3” intendiamo la gerarchia, ossia il grado di parentela. Come per i requisiti, il numero più a sinistra rappresenta il padre e quelli più a destra i suoi figli.

Esempio

“UC1.2.3” indica un caso d'uso con padre UC1.2.0 che a sua volta è figlio di UC1.0.0.

Ogni caso d'uso al fine di essere descritto nel modo più esaustivo possibile conterrà i seguenti dettagli:

- **Nome:** composto dal codice di riferimento e una breve descrizione del caso d'uso;
- **Descrizione grafica:** creata mediante il formalismo di UML e implementata tramite creazione di specifici casi d'uso per formalizzare e facilitare la comprensione del requisito;
- **Descrizione didascalica:** viene correlata ad ogni diagramma di caso d'uso con una descrizione in cui compaiono le informazioni:
 - Attori principali;
 - Scopo e descrizione;
 - Precondizione;
 - Postcondizione;
 - Illustrazione scenario principale;
 - Illustrazione scenario alternativo.

6.3 Tracciamento

6.3.1 Casi d'Uso-Requisiti e viceversa

Durante l'intero ciclo di vita del progetto, sono previste misure per il tracciamento dei casi d'uso con i relativi requisiti. Questo avviene grazie all'utilizzo del sistema di tracciamento dei requisiti "SynthesisRequirementManager" descritto nella sezione 3.2 "Sistema di tracciamento dei requisiti". Tutta la gestione è automatica e in output sarà visualizzata una tabella con tre colonne: Codice UC, Nome UC, Requisito. Questa soluzione facilita notevolmente la gestione di grandi quantità di dati e un notevole risparmio di tempo. Avviene anche la stampa inversa, ossia Requisiti-Casi d'Uso.

6.3.2 Requisiti-Test

È prevista anche una tabella che traccia i requisiti e relativi test da effettuare per verificare il soddisfacimento di tutti i requisiti. Tale tabella avrà tre colonne, la prima riporta il codice del requisito, la seconda il codice del test e nella terza colonna una breve descrizione del test da effettuare.

6.3.3 Requisiti-Componenti e viceversa

Il documento *specifica_tecnica.pdf* conterrà come ultima sezione il tracciamento. La prima parte riguarda il tracciamento dei requisiti con i componenti individuati. Tale tracciamento ha lo scopo di verificare che tutti i requisiti abbiano almeno un componente che li soddisfa. Il tracciamento sarà organizzato in una tabella avente due colonne, la prima che riporta il codice del requisito e la seconda il componente. La seconda parte invece riguarda il tracciamento dei componenti con i requisiti. Tale tracciamento ha lo scopo di assicurare che ogni componente soddisfi almeno un requisito, altrimenti non avrebbe ragione di esistere.

6.3.4 Componenti-Design Pattern e viceversa

Il tracciamento del documento *specifica_tecnica.pdf* conterrà anche una sezione dedicata al tracciamento tra componenti e *design pattern*. La prima tabella è composta da una colonna dedicata ai componenti e un'altra dedicata ai *design pattern*. Tale tracciamento ha il solo scopo di chiarire quali *design-pattern* sono implementati da quali componenti. La seconda tabella invece riguarda il tracciamento dei *design-pattern* con i componenti. Tale tracciamento ha lo scopo di assicurare che ogni *design-pattern* sia presente in almeno un componente, altrimenti non avrebbe ragione di esistere.

6.3.5 Componenti-Classi

L'ultima parte del tracciamento del documento *specifica_tecnica.pdf* riguarda il tracciamento tra componenti e classi. Tale tracciamento ha lo scopo di garantire che ogni componente abbia almeno una classe che lo implementi. conterrà anche una sezione dedicata al tracciamento tra componenti e *design pattern*. La prima tabella è composta da una colonna dedicata ai componenti e un'altra dedicata ai *design pattern*. Tale tracciamento ha il solo scopo di chiarire quali *design-pattern* sono implementati da quali componenti. La seconda tabella invece riguarda il tracciamento dei *design-pattern* con i componenti. Tale tracciamento ha lo scopo di assicurare che ogni *design-pattern* sia presente in almeno un componente, altrimenti non avrebbe ragione di esistere.

7 Progettazione

Lo scopo principale di tale attività è di analizzare e produrre una visione generale del sistema garantendo solidità, efficienza e al tempo stesso il soddisfacimento dei requisiti emersi durante l'attività di analisi. Il documento prodotto al termine di questa attività, la Specifica Tecnica, descriverà l'organizzazione generale dell'intero sistema prodotto, specificando le scelte architetture in modo esauriente e giustificato. Si avranno inoltre definiti i vari test da eseguire per verificare la correttezza d'interazione tra le varie parti del sistema stesso.

In questa sezione verranno descritte le specifiche che i progettisti dovranno seguire durante la loro attività di progettazione architeturale.

7.1 Stile di progettazione

Al fine di semplificare la futura fase di verifica, nonché la leggibilità degli schemi e diminuire il livello di rischio dovuto ad una progettazione errata, si dovranno rispettare le seguenti regole e linee guida:

- **Information hiding:** si dovrà seguire il principio di *information hiding*, ossia nascondere dettagli organizzativi tramite attributi e metodi privati e/o protetti in modo tale da ridurre la complessità del contratto della classe. Ciò aumenta il grado di protezione e abbassa il rischio di errori logici.

- **Decomposizione:** dovrà essere eseguita una *decomposizione modulare* al fine di identificare fin da subito i componenti terminali che non richiedono ulteriore scomposizione, in modo tale da abbattere tempi e costi già dalle prime fasi.
- **Design pattern:** individuare i design pattern da poter usare e applicarli correttamente.
- **Livello di dettaglio:** raggiungere un livello di dettaglio in modo da permettere la parallelizzazione della codifica e non lasciare decisioni ai programmatori.
- **Annidamento di chiamate:** non dovranno essere presenti nell'applicativo chiamate di metodi annidate con profondità maggiore di dieci. Se è necessaria una modifica di tale limite il progettista dovrà proporlo, con relativa giustificazione, al responsabile di progetto che valuterà la richiesta.
- **Costrutti condizionali:** non dovranno essere presenti nell'applicativo flussi condizionali con profondità maggiore di cinque. Se è necessaria una modifica di tale limite il progettista dovrà proporlo, con relativa giustificazione, al responsabile di progetto che valuterà la richiesta.
- **Ricorsione:** la tecnica di ricorsione dovrà essere evitata il più possibile. Nel caso venga utilizzata, il progettista dovrà giustificarne la scelta e fornire inoltre la correttezza per dimostrarne la terminazione.
- **Parametri:** il numero di parametri passati ad una chiamata di metodo non dovrà superare il limite di sette.
- **Concorrenza:** se vengono utilizzati flussi di esecuzione concorrenti, dovrà essere fornito anche il relativo diagramma di flusso.

7.2 Design Pattern

Per ogni design *pattern* utilizzato, si dovrà dare una breve descrizione indicando:

- **Scopo:** verrà proposto lo scopo generico del pattern, al fine di evidenziare subito la sua utilità;
- **Componenti che lo implementano:** verranno elencati i componenti dell'architettura di sistema, che implementano il *pattern* descritto. Per ogni componente verrà inoltre associato un diagramma esplicativo che ne rappresenta la struttura. In tale sezione saranno evidenziati i vantaggi derivanti dall'adozione del *pattern* nel particolare contesto di applicazione.

7.3 Convenzioni sui diagrammi

Si dovrà utilizzare il linguaggio UML per definire:

- **Diagrammi dei package:** dovranno essere presenti per la parte server implementata con *Java*. Servono per definire i *package*, ossia un raggruppamento di classi.
- **Diagrammi delle classi:** Verranno usati per descrivere tipi di entità, le loro caratteristiche e le eventuali relazioni all'interno delle singole componenti.
- **Diagrammi di attività:** dovranno essere presenti per mostrare generici flussi di attività che un utente può compiere all'interno dell'applicativo.
- **Diagrammi di sequenza:** usati per definire le responsabilità dei vari package (o delle classi) nel portare a termine un determinato compito. Sarà necessario limitare la rappresentazione di cicli o percorsi alternativi ai soli casi per cui la loro omissione provochi difficoltà nella comprensione del funzionamento dell'applicativo.

7.4 Classi di verifica

Si dovranno sviluppare, quando possibile, delle classi fittizie da utilizzare in fase di verifica e come prototipo.

7.5 Tracciamento

Per il tracciamento generale di componenti, requisiti, classi, si utilizzeranno sei tabelle complessive con due colonne ognuna. Di seguito la lista di tali tabelle che devono comparire nel documento “specifica tecnica”:

- requisiti-componenti;
- componenti-requisiti;
- componenti-design pattern
- design pattern-componenti;
- componenti-classi
- classi-componenti.

7.6 Struttura della Specifica Tecnica

Il documento relativo alla Specifica Tecnica sarà strutturato nel seguente modo:

- una breve introduzione che spiega lo scopo del documento, i formalismi adottati e i riferimenti alle norme;
- un breve elenco degli strumenti utilizzati;
- breve trattazione dei design pattern utilizzati seguendo le norme indicate nel paragrafo 7.2;
- descrizione dell’architettura generale evidenziando i componenti rilevati in fase di progettazione indicando il tipo, la funzione e l’obiettivo;
- descrizione dell’architettura di dettaglio (quando disponibile);
- diagrammi UML;
- tracciamenti (come descritto in sezione 7.5).

8 Progettazione di dettaglio

Lo scopo di questa attività è di descrivere in modo molto particolareggiato quello che dovrà essere il sistema, estendendo e specificando nel dettaglio quanto individuato nella specifica tecnica.

Al termine di tale attività si avrà a disposizione il documento *Definizione di Prodotto*, contenente tutte le informazioni necessarie per la realizzazione pratica del sistema.

Così come nel caso della progettazione di alto livello il risultato dell’attività comprende altresì la definizione dei vari test da eseguirsi al fine di verificare la correttezza nel comportamento delle classi progettate.

I progettisti, analogamente a quanto stabilito relativamente all’attività precedente, dovranno attenersi più possibile alle specifiche elencate nei paragrafi seguenti.

8.1 Diagrammi

Come per la specifica tecnica si dovrà utilizzare il linguaggio UML per definire:

- *diagrammi delle classi*: al fine di rendere più chiare ed esplicite possibili le relazioni tra le varie classi all'interno delle componenti;
- *diagrammi di attività*: per definire in modo univoco i flussi delle attività svolte dai vari componenti;
- *diagrammi di sequenza*: per definire in modo preciso le responsabilità dei package (o classi) nello svolgimento dei propri compiti. Come nel caso della specifica tecnica si dovrà limitare la rappresentazione di cicli o condizioni alternative ai soli casi in cui la loro presenza risulti necessaria alla comprensione del funzionamento dell'applicativo.

Tali diagrammi, ovviamente, hanno lo scopo di approfondire ove presenti quanto già descritto nel documento di specifica tecnica e possono pertanto essere trascurati nei casi in cui la loro specializzazione non porti alcun ulteriore beneficio alla documentazione.

8.2 Specifica di una classe

Al fine di garantire una rappresentazione più chiara e dettagliata di ogni classe all'interno del documento *Definizione di Prodotto*, ogni classe dovrà essere descritta attenendosi al seguente schema:

- **Funzione**: la funzione svolta dalla classe stessa.
- **Relazione d'uso**: la relazione che la classe ha con altre classi (es. può *estendere* o *essere estesa*, *implementare* o *essere implementata*, da un'altra classe, ecc.)
- **Attributi**: gli attributi presenti all'interno della classe con una loro breve descrizione;
- **Metodi**: i metodi (pubblici) che la classe possiede e una relativa descrizione comportamentale. Al fine di renderne più chiara la suddetta descrizione è consigliato descrivere anche le eventuali eccezioni che ciascun metodo può sollevare.

8.3 Tracciamento

Come nel caso della specifica tecnica sarà presente in forma tabellare il tracciamento di ogni requisito con le varie classi al fine di garantire la soddisfacibilità del medesimo, in modo che ogni classe ha ragione di esistere e non ne esista nessuna senza alcun ruolo all'interno del sistema creato.

8.4 Struttura della Definizione di Prodotto

Il documento relativo alla Definizione di Prodotto che estenderà ed integrerà quello di Specifica Tecnica conterrà:

- i diagrammi delle classi per ogni componente evidenziato;
- le specifiche di *tutte* le classi.



8.5 Convenzioni di scrittura

Al fine di rendere quanto più agevole possibile la consultazione di questo documento da parte dei programmatori e del committente, è stata adottata una serie di accorgimenti. In primo luogo, sono stati inseriti riferimenti incrociati sui nomi delle classi aventi come destinazione la sottosezione che descrive la classe stessa.

In secondo luogo, per i membri di classi e interfacce, oltre a una famiglia di font a spaziatura fissa, sono state adottate le seguenti convenzioni cromatiche:

- **rosso** per i campi dati;
- **verde** per i metodi.

I differenti livelli di accessibilità sono contrassegnati, in maniera analoga a UML, come:

- + per la parte pubblica;
- # per la parte protetta;
- ~ per l'accessibilità di package;
- - per la parte privata.

Infine, i membri statici di classi e interfacce sono contrassegnati con una sottolineatura.

9 Codifica

Questa sezione andrà ampliata nel momento in cui si avvierà la vera attività di codifica. Vengono intanto fornite delle convenzioni di codifica generale.

9.1 Intestazione dei file

Tutti i file sorgente includeranno un'intestazione con le informazioni generiche relative al file medesimo quali:

- nome del progetto
- nome del team di sviluppo;
- nome del file;
- versione;
- data di creazione.

9.2 Convenzioni di codifica

Per rendere il codice più leggibile ed ordinato, per quanto riguarda l'uso del linguaggio Java, si dovranno rispettare le *Java Code Convention*.³

Tali convenzioni di codifica possono essere variate, ma tale modifica deve essere richiesta al responsabile di progetto che valuterà la reale ed effettiva necessità di tale modifica.

9.2.1 Convenzioni sui nomi

- **Classi e Interfacce:** il loro nome è un sostantivo e deve avere la prima lettera maiuscola.
- **Metodi:** devono avere il nome di un verbo e la prima lettera minuscola. Nel caso il nome sia composto da più parole, la prima lettera di ogni parola sarà maiuscola.
- **Variabili:** devono avere un nome che ne rende chiaro immediatamente il significato. Nel caso il nome sia composto da più parole, la prima lettera di ogni parola sarà maiuscola.
- **Costanti:** il nome va scritto tutto in maiuscolo. Nel caso il nome sia composto da più parole, vanno separate da un *underscore*.

9.2.2 Struttura delle classi

La classi dovranno essere composte riportando nel seguente ordine tali elementi:

1. variabili statiche e di istanza;
2. costruttori;
3. metodi.

9.2.3 Struttura del codice

Per la stesura di codice, si seguiranno le seguenti regole:

1. le dichiarazioni di variabili andranno riportate tutte all'inizio e si dovrà usare una riga per ogni variabile;
2. ogni blocco di codice (tipo definizione di un metodo) va separato con una linea bianca vuota;

³Disponibili all'indirizzo <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>.

3. le parentesi graffe che delineano un blocco di codice devono iniziare nella stessa riga della parola chiave a cui appartengono e finire in una nuova riga allineata con la parola d'inizio;
4. parole chiave e parentesi (ad esempio parentesi dovute a costrutti condizionali o ciclici) non devono essere separate da spazi.
5. l'annotazione `@override` sarà usata per assegnare automaticamente una descrizione definita per un metodo di una interfaccia/superclasse allo stesso metodo definito nella sottoclasse; serve inoltre al compilatore per controllare il corretto uso di *overloading* e *overriding*;

9.3 Verifica del codice

La verifica del codice verrà fatta, come prima cosa, durante lo sviluppo di ogni classe attraverso il proprio IDE che segnalerà gli errori più banali (errori di battitura o piccole mancanze).

Lo sviluppatore, prima di caricarlo nel *repository*, deve assicurarsi che il codice compili e non dia errori o *warning*.

Il verificatore successivamente dovrà assicurarsi che siano state rispettate le convenzioni descritte nella sezione 7 a pagina 22 e in caso di errori dovrà aprire un ticket (rispettando sempre le norme descritte in sezione 5.2.1 a pagina 15) rivolto allo sviluppatore che provvederà a risolvere gli errori.

9.4 Norme sul codice Java

Per la parte di progetto realizzato mediante codice java sarà usato Javadoc, un applicativo incluso nel Java Development Kit della Sun Microsystems, utilizzato per la generazione automatica della documentazione (in formato HTML) del codice sorgente scritto in linguaggio Java.

Il team ritiene pertanto che l'utilizzo di tale strumento sia non solo consigliato, ma bensì costituisca una *best practice*. I vantaggi che si prevede di ottenere utilizzando tale strumento sono:

- imporre uno standard nella compilazione della documentazione dei sorgenti;
- facilitare il lavoro di documentazione che potrà (in parte) svolgersi in concomitanza con l'attività di programmazione (ossia nel momento in cui il programmatore è più focalizzato sull'argomento trattato);
- automatizzare l'esportazione dei documenti in due possibili formati: html e pdf.
- nel documentare si coglie anche il vantaggio di fornire commenti utili sul codice stesso, in grado quindi di aiutare i programmatori nelle future attività di programmazione.

Si vuole sottolineare che, al fine di garantire un formato più standardizzato possibile e quindi incline alla comprensione collettiva, le norme esposte successivamente saranno da applicare con la massima precisione.

I *tag* Javadoc che verranno utilizzati sono i seguenti:

- `@version`: numero di versione di una classe o di un metodo;
- `@author`: nome dello sviluppatore assegnato;
- `@param`: definisce i parametri di un metodo;
- `@return`: indica i valori di ritorno di un metodo (da non usare ovviamente nei metodi o costruttori con valori di ritorno *void*);

- **@throws**: indica le eccezioni lanciate da un determinato metodo;
- **@see**: indica un'associazione ad un'altra classe o metodo.

Al fine di evitare la creazione di un tag Javadoc all'interno del codice in modo accidentale, sarà necessario utilizzare il codice HTML `@`, eliminando così ogni tipo di possibile ambiguità.

Definiamo in conclusione le convenzioni da utilizzare per la creazione di un commento Javadoc ben strutturato ed efficace :

- è sempre posto subito prima della dichiarazione della classe, interfaccia, metodo, attributo.
- è composto da due parti distinte: un tag e una descrizione. Deve iniziare con i caratteri `/**` e terminare con i caratteri `*/`, ogni riga del commento dovrà inoltre iniziare con il carattere `*`.
- avrà ragione d'esistere solo se la sua presenza sarà utile alla comprensione del codice, con un linguaggio sintetico, privo di ambiguità e più chiaro possibile.
- è pensato allo scopo per descrivere le funzionalità o i principi di una unità (sia essa un package, una classe, un'interfaccia o un metodo) e non per "spiegare" pezzi di codice; tali commenti, anche se formattati secondo lo standard, non saranno mai estratti dal comando `javadoc`;
- per ogni classe dovrà comprendere una descrizione delle sue funzioni, il tag **@author** con associato il nome dell'autore, il tag **@version** con il numero di versione di tale classe e la data di compilazione, infine il tag **@return** con il tipo di oggetto restituito.
- per ogni metodo, oltre ai tag già descritti e inseriti in una classe, dovrà contenere il tag **@throws** per ogni tipologia di eccezione che può sollevare e un tag **@param** per ogni parametro del metodo (con relativa descrizione).

9.4.1 Manuale di Javadoc

I programmatori del team di sviluppo possono inoltre avvalersi di ulteriori supporti per chiarire eventuali dubbi personali, come ad esempio al seguente link:

<http://wwwusers.di.uniroma1.it/~parisi/Risorse/IntroJavadoc.pdf>

Qualora un membro del team trovasse eventuali informazioni che ritiene coerenti e di potenziale valore sarà suo dovere avvisare l'amministratore designato che vaglierà la possibilità di introdurle in questa sezione delle norme.

9.5 Norme sul codice JavaScript

In fase di codifica della parte JavaScript i programmatori dovranno fare in modo che il codice prodotto sia mantenuto in file con estensione `.js` separati rispetto alla parte *html* e che in tutti i documenti *html* che utilizzeranno tale codice sia inserito al loro interno, nel tag *head*, il riferimento al file JavaScript esterno.

Per quanto riguarda l'assegnazione del nome a variabili, funzioni e classi:

- il nome delle classi potrà essere composta da una o più parole: nel primo caso solo la prima lettera del nome sarà maiuscola, nel secondo caso le parole non dovranno essere separate, bensì verranno distinte dalla prima lettera (maiuscola) che le compongono (es: `AbilitaCanale`);
- il nome delle variabili sarà scritto utilizzando solamente lettere minuscole;

- il nome delle funzioni come nel caso delle classi potrà essere composta da una o più parole, ma al contrario delle classi se composto da una singola parola verrà scritta con caratteri esclusivamente minuscoli, se al contrario sarà composto da più parole verrà usata la convenzione scelta per le classi, ma la prima lettera (della prima parola) resterà minuscola;
- il nome delle variabili, se composto da più parole, sarà creato dalle parole congiunte mediante *underscore*, dovranno inoltre essere utilizzate parole che identifichino il ruolo o lo scopo di tali variabili al fine di agevolare la comprensione del codice.

Infine, per quanto concerne le normative sul codice Javascript:

- al termine di ogni riga di codice dovrà essere presente il carattere “;” ;
- in ogni funzione la prima parentesi dovrà essere posta a capo rispetto al nome e alla massima sinistra così come l’ultima parentesi (che identifica il termine del codice della funzione).
- il codice di ogni funzione sarà anticipato da una pre-condizione che ne spieghi concisamente lo scopo.
- le parti di codice poco chiare o che fanno uso di funzioni specifiche dovranno essere correlate da commenti al fine di renderne comprensibile il funzionamento.

10 Verifica e Validazione

La verifica è un'attività molto importante svolta dal verificatore. Lo scopo è di accertare che i prodotti ottenuti, qualunque sia il loro tipo, siano conformi le norme e convenzioni riportate su questo documento.

Oltre a questo, il verificatore dovrà anche assicurare sotto la propria responsabilità che tali prodotti rispettino i requisiti di qualità descritti nel documento Piano di Qualifica (piano_di_qualifica.2.0.pdf).

10.1 Strumenti di verifica

Si riporta a continuazione un elenco dei principali strumenti per la verifica di cui il team si dovrà avvalere nell'arco dello sviluppo del progetto.

- **SynthesisRequirementManager**, il sistema di gestione dei requisiti (citato nel paragrafo 3.2 realizzato da Software Synthesis, avente lo scopo di rendere quanto più agevole gestire il tracciamento dei requisiti a tutti i livelli (requisiti-UC, requisiti-CI, ecc.) e, da un punto di vista prettamente qualitativo, assicurare la necessità e la sufficienza dei casi d'uso e delle componenti software;
- **lacheck** (≥ 1.26) per assicurare la correttezza sintattica e l'adozione delle *best practices* per i sorgenti \LaTeX nonché rilevare in modo semi-automatico le sviste non segnalate dal compilatore in quanto pur corrispondendo a codice ben formato nascondono errori tipografici sottostanti per es. bilanciamento delle virgolette, spaziature scorrette per frasi terminate da un acronimo prima di un punto, mancato utilizzo di spazi inescabibili, ecc. (www.ctan.org/pkg/lacheck);
- **hunspell** (≥ 1.3) come correttore ortografico e analizzatore morfologico in fase di redazione della documentazione, scelto per la sua portabilità ma anche perché alle sue librerie si appoggia l'applicativo **TexMaker** utilizzato per la stesura dei documenti \LaTeX (<http://hunspell.sourceforge.net>);
- le utilità che costituiscono la suite di **QA** del W3C al fine di verificare l'aderenza agli standard delle pagine web generate, in particolar modo gli strumenti online:
 - **Unicorn** in qualità di strumento di validazione unificato (<http://validator.w3.org/unicorn>);
 - **CSS Valitatom Service** come utilità di validazione per i fogli di stile a cascata (<http://jigsaw.w3.org/css-validator>);
- gli strumenti per sviluppatori integrati in **Google Chrome** (<https://developers.google.com/chrome-developer-tools>) e, in particolare:
 - la sezione Sources che rappresenta un'interfaccia al *debugger* per il motore JavaScript V8 e consente di impostare *breakpoint* (assoluti o condizionali) per seguire l'esecuzione del codice passo passo (con le consuete funzioni di “step over”, “step into” e “step out”) nonché arrestare temporaneamente l'esecuzione al sollevamento di un'eccezione (o di un'eccezione non controllata);
 - la sezione *Timeline* che permette di quantificare i tempi necessari al caricamento e all'esecuzione degli script, nonché di tracciare l'utilizzo della memoria e forzare l'invocazione del *garbage collector*;
 - gli strumenti di benchmark accessibili dalla sezione *Profiles*, vale a dire il *profiler* della CPU, che permette di ricostruire l'albero delle chiamate di funzione e la percentuale di utilizzo della CPU per ciascuna funzione, e il *profiler* dello *heap*, mediante il

quale è possibile ispezionare il contenuto dello *heap* e salvarne delle rappresentazioni istantanee;

- **FirebugLite**, un'estensione per Chrome che consente di ispezionare gli elementi HTML e la struttura del DOM nonché di modificare in tempo reale i valori delle proprietà dei CSS (<https://getfirebug.com/firebuglite>);
- **SpeedTracer** uno strumento che consente di identificare i problemi di prestazioni nelle applicazioni web visualizzando una serie di metriche in tempo reale grazie all'analisi dei dati resi disponibili a livello di motore di rendering del browser (<https://developers.google.com/web-toolkit/speedtracer>);
- **JSLint** analizzatore statico di codice JavaScript volto a rilevare e impedire l'adozione inconsapevole di "*worst practices*" in fase di codifica (<http://www.jshint.com>);
- **ApacheBench** per testare l'efficienza prestazionale dell'applicazione lato server mediante la simulazione di un numero arbitrario di richieste da parte dei client (<http://httpd.apache.org/docs/2.2/programs/ab.html>);
- **Eclipse IDE** multiplatforma e *cross-language*, scelto in particolare come ambiente di sviluppo per la parte server da realizzarsi in Java, che include al suo interno funzionalità di debugging (<http://www.eclipse.org>) utili ai fini della QA;
- plugin **Metrics** per Eclipse, estensione che permette di associare un valore su una scala di riferimento al soddisfacimento di una serie di parametri di qualità del codice sorgente tramite metriche prestabilite, come ad esempio la *complessità ciclomatica* dello stesso (<http://metrics.sourceforge.net>);
- il plugin **FindBugs** per Eclipse, al fine di effettuare analisi statica del codice a livello di bytecode alla ricerca di potenziali cause di malfunzionamento (*bug patterns*) o adozione inconsapevole di "*worst practices*" (<http://findbugs.sourceforge.net>);
- **JUnit** come framework per i test di unità da effettuarsi relativamente alla parte server dell'applicazione (<http://www.junit.org>);
- **QUnit** framework per i test di unità di codice *JavaScript*;
- **JSCoverage**, *tool* per la misurazione del *Code Coverage* di codice *JavaScript*.;
- **eclemma**, *tool* per la misurazione del *Code Coverage* di codice *Java*.

10.2 Tecniche di verifica

Responsabili delle attività di controllo interne al gruppo sono i verificatori, che si presuppone essere in ogni caso e senza alcuna deroga distinti dai realizzatori del prodotto soggetto a verifica (programmatore o redattori di documentazione). Al fine di garantire la qualità, i verificatori sono tenuti all'utilizzo di due tecniche di analisi: statica e dinamica.

10.2.1 Analisi statica

L'analisi statica è un tipo di controllo basato sulla non esecuzione del codice, ma in senso lato può essere applicato a qualsiasi tipo di prodotto anche non propriamente eseguibile (ad es. la documentazione di progetto).

Sono previste, in particolare, due forme di analisi statica: il controllo manuale (detto altrimenti "*desk check*") e il controllo assistito da strumenti automatici.

Per quanto concerne il **desk check**, cioè il controllo realizzato unicamente da parte di un agente umano, sono previsti due metodi formali:

- **walkthrough:** implica un esame ad ampio spettro del prodotto da verificare, che è preso in considerazione nella sua totalità in modo indiscriminato e senza alcuna assunzione previa sulla natura, la posizione e la frequenza degli errori da rilevare. Si tratta notoriamente di una tecnica molto onerosa in termini sia di tempo che di sforzo e può essere essa stessa per sua natura essere soggetta ad errori (in particolar modo falsi negativi). Tuttavia, almeno nelle fasi iniziali del lavoro, è l'unica scelta praticabile a causa della relativa inesperienza dei membri del gruppo nella realizzazione di prodotti complessi e articolati (sia software che documentazione). Allo scopo di ridurre il costo determinato dalla ripetizione di tale attività nell'arco di tutto il ciclo di vita è previsto che durante l'analisi in *walkthrough* sia stilata una lista di controllo relativa agli errori più frequenti e ai contesti in cui è più probabile che si producano errori in modo da collezionare una base di esperienza comune e consolidata destinata ad alimentare le attività di ispezione.
- **inspection:** prevede un controllo mirato avente obiettivi specifici, ristretti e stabiliti a priori *prima* che la verifica abbia luogo. Si tratta di un'attività meno dispendiosa in termini di risorse perché non presuppone l'analisi esaustiva del prodotto ma è focalizzata su determinate categorie di errori frequenti, enunciate in una lista di controllo (*checklist*) redatta sulla base dell'esperienza personale e delle attività di *walkthrough* precedentemente poste in essere.

La seconda forma di analisi statica prevede invece l'utilizzo di strumenti appositi denominati **analizzatori statici** e può essere svolta in modo semiautomatico senza richiedere necessariamente l'intervento di un umano. In particolare, come risulta dalla sezione 10.1 si è stabilito di utilizzare degli analizzatori statici tanto per la parte documentale del progetto (come il comando `lacheck`) quanto per la parte propriamente eseguibile (JSLint per la parte JavaScript e FindBugs per la parte Java).

10.2.2 Analisi dinamica

I controlli dinamici, altrimenti definiti test, prevedono l'esecuzione del software in un ambiente controllato e con dati di input specificatamente pensati per testarne le funzionalità e l'aderenza ai requisiti mettendo in luce l'eventuale presenza di malfunzionamenti dovuti alla presenza di difetti. Caratteristica fondamentale dei test è la loro *ripetibilità*, cioè dato lo stesso set di dati in ingresso e nello stesso contesto di esecuzione, l'output deve essere deterministico e univocamente determinato. Tale proprietà, unitamente all'auspicabile utilizzo di un *logger* che ha il compito di registrare le fasi dell'esecuzione del test, consente di individuare e riconoscere in maniera più agevole i difetti presenti nel prodotto.

In base al loro ambito di applicazione, i test possono essere suddivisi in:

- test di unità aventi come oggetto le singole unità e, oltre al modulo da verificare e ai dati d'esempio, possono coinvolgere anche componenti attive (*driver*) o passive (*stub*) che siano in grado di simulare le parti del sistema non ancora disponibili al momento in cui il test viene eseguito;
- test di integrazione atti a verificare la corretta interazione e integrazione fra le componenti che costituiscono le parti del sistema e hanno come risultato una *build*, vale a dire un sottosistema funzionante che può essere eseguito in modo indipendente;
- test di sistema, volti a testare il rispetto dei requisiti software individuati in fase di analisi dei requisiti da parte dell'intero sistema;
- test di regressione destinati a rilevare il caso indesiderabile in cui una modifica locale destabilizza il resto del sistema, si tratta del numero minimo di test necessario per scongiurare tale eventualità senza per questo dover ripetere *in toto* i test di unità e di integrazione;

- test di accettazione, o collaudo, realizzato sotto la supervisione del committente per verificare l'aderenza del prodotto ai requisiti utente di più alto livello.

10.3 Test di unità, integrazione e sistema

La validazione dei test di sistema verrà svolta basandosi sui test specificati durante l'attività di analisi dei requisiti. I risultati di tali test verranno riportati in forma tabellare, in modo da renderne più pratica ed efficace la lettura e la comprensione.

L'esecuzione dei test avverrà mediante l'utilizzo di *Selenium*, un *tool* che permette di impostare ed eseguire automaticamente i test per la dimostrazione delle funzionalità del sistema.

Per i rimanenti test al contrario la verifica dovrà basarsi sulle specifiche relative test di unità e di integrazione prodotte dalle attività di progettazione architetturale e di dettaglio, e verranno effettuati tramite i *framework* JUnit (per il codice java della parte server) e QUnit (per i test relativi al codice JavaScript).

Verrà successivamente redatto un documento riassuntivo contenente l'esito dei test in formato tabellare con specificati per ogni tipologia di test effettuati:

- ambiente in cui è stato svolto il test;
- tipologia del test;
- classi/Componenti testate nel test;
- descrizione del test effettuato;
- eventuali errori evidenziati dal test.

10.3.1 JUnit e eclemma

JUnit è una libreria Java utilizzata per la creazione delle classi delegate ai test funzionali di metodi (o unità) di una determinata classe da analizzare, *eclemma* invece ha il semplice scopo di calcolare il *Coverage* del codice java scritto. Per la creazione di tali test sarà necessario importare la libreria all'interno del progetto desiderato e successivamente creare un file che conterrà la classe con il codice effettivo del test. All'interno di tale classe è possibile creare diversi metodi, che saranno preceduti dalle annotazioni proprie di *JUnit*. Le più comuni sono:

- **@Test**: identifica i metodi utilizzati dalla classe per eseguire i test
- **@Before** e **@After**: un metodo preceduto da tali annotazioni viene eseguito prima (o dopo) ogni esecuzione di un metodo contrassegnato da *@Test*.
- **@Beforeclass** e **@Afterclass**: il metodo viene eseguito solo una volta prima (o dopo) l'esecuzione della classe testata.

E' possibile inoltre creare classi "suite" che contengono più classi di test allo scopo di eseguirle in sequenza senza la necessità di avviarle singolarmente.

Concludiamo citando i metodi contraddistinti dal prefisso *assert*, che hanno lo scopo di confrontare un valore preimpostato con il valore ritornato da un metodo restituendo *true* o *false* in base all'esito del test, risultando quindi molto utili per verificare la correttezza dello stesso.

10.3.2 QUnit e JSCoverage

Come accennato in precedenza, *QUnit* è un framework per i test di unità del codice *JavaScript*. La sua esecuzione richiede una pagina *HTML* all'interno della quale vengono inclusi i file *qunit.js*, *css.js* e i file *JavaScript* contenenti i test da eseguire.

Questi file dovranno riferire ad una specifica classe da testare e in particolare le funzioni “test”, realizzate in modo da verificare che il valore di ritorno sia corretto (e quindi atteso) con specificate:

- una stringa che indica il compito della funzione;
- la funzione che verrà richiamata all’avvio del test.

Le funzioni richiamate all’interno delle funzioni di *test* sono offerte da *QUnit* e specificano determinate azioni standard. Per una descrizione dettagliata di tali funzioni e del loro specifico compito si rimanda al sito di riferimento http://docs.jquery.com/Qunit#Using_QUnit.

JSCoverage al contrario è un *tool* per valutare il Code Coverage integrabile con *QUnit* in modo da ottenere contemporaneamente sia le informazioni relative al risultato dei test che quelle relative alla loro copertura in un unica pagina *HTML* denominata *jscoverage.html*.