Piano di Qualifica

Informazioni sul documento	
Nome file:	piano di qualifica
Versione:	1.0
Data creazione:	05/12/2012
Data ultima modifica:	06/12/2012
Stato:	Non approvato
Uso:	Interno
Redattori:	Stefano Farronato Boh Qualcun'altro
Approvato da:	
Verificatori:	

Storia delle modifiche

Versione	Descrzione intervento	Redattore	Data
0.1	Stesura scheletro documento, introduzione	Stefano Farronato	12/12/2012

Indice

1 Organigramma

2 Introduzione

2.1 Scopo del prodotto

Con progetto MyTalk intendiamo un sistema software di comunicazione tra utenti mediante browser, utilizzando solo componenti standard, senza dover installare plugin o programmi esterni. L'utilizzatore dovrà poter chiamare un altro utente, iniziare la comunicazione sia audio che video, svolgere la chiamata e terminare la chiamata ottenendo delle statistiche sull'attività.

2.2 Scopo del documento

Il seguente documento ha lo scopo di presentare la strategia di verifica e di validazione complessiva che utilizzeranno i componenti del Team di lavoro di Software Synthesis a scopo di garantire la qualità richiesta nello svolgimento del progetto MyTalk regolarmente accettato dall'azienda appaltatrice Zucchetti s.r.l.
Durante lo svolgimento del suddetto progetto sarà possibile l'insorgere di modifiche a tale documento, dettate da eventuali scelte progettuali o da esplicite richieste del cliente committente stesso.

2.3 Glossario

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

3 Riferimenti

3.1 Normativi

VINCOLI DI ORGANIGRAMMA: Specificate dal Committente designato all'indirizzo
http://www.math.unipd.it/ tullio/IS-1/2012/Progetto/PD01b.html

NORME DI PROGETTO v1.0 allegato Verbale incontro con il proponente del 10/12/2012
Riferimenti a specifici estratti del testo *Software Engineering (8th edition) Isan Sommerville, Pearson Education | Addison-Wesley* quali:
ISO/IEC 9126:2001, Software engineering - Product quality - Part 1: Quality model
ISO/IEC 12207, Software Life Cycle Processes

3.2 Informativi

Analisi dei Requisiti 1.0
Piano di Progetto 1.0
Glosssario
CAPITOLATO D'APPALTO: MyTalk, v 1.0, redatto e rilasciato dal proponente Zucchetti s.r.l reperibile all'indirizzo: *http://www.math.unipd.it/ tullio/IS-1/2012/Progetto/C1.pdf*

Riferimenti a specifici estratti del testo *Software Engineering (8th edition) Isan Sommerville, Pearson Education | Addison-Wesley* quali:
Software Engineering - Part 5: Verification and Validation, Part 6: Management

4 Visione generale della strategia di verifica

4.1 Organizzazione, pianificazione strategica, pianificazione temporale e responsablità

Il processo di verifica inizierà quando il prodotto di un processo raggiungerà uno stadio che si potrà definire diverso da quello precedente. La verifica di tali cambiamenti sarà operata in modo mirato e ciscoscritta, grazie al registro delle modifiche che verrà complilato durante lo stilamento del documento stesso. Al termine della fase di verifica, i documenti saranno consegnati al responsabile di progetto, che provvederà ad approvarli.
Il team nel brainstorming successivo all'analisi generale del progetto MyTalk ha deciso di adottare un ciclo di vita incrementale (specificato nel Piano di Progetto).
Coerentemente a tale scelta, il processo di verifica adottato opererà nelle diverse fasi del progetto nel modo seguente:

Analisi dei Requisiti: Quando un documento uscirà dalla fase di redazione, verrà preso in esame ed effettuata una fase di revisione definitiva prima di essere presentato ufficialmente alla RR:

1. Verrà presa in esame la correttezza grammaticale.
2. Verrà presa in esame la correttezza lessicale mediante un accurata rilettura da parte del verificatore designato.
3. Verrà presa in esame la correttezza dei contenuti e la coerenza rispetto al documento mediante un accurata rilettura da parte del verificatore designato.
4. Verrà presa in esame la verifica che ogni tabella o figura sia corretta nel suo contenuto e disponga della rispettiva didascalia.
5. Verrà presa in esame la correttezza rispetto alle Norme di Progetto redatte, utilizzando gli strumenti più appropriati per la verifica.
6. Verrà presa in esame la corrispondenza tra ogni requisito e i casi d'uso, consultando e controllando le apposite tabelle di tracciamento, verificando inoltre la corretta gestione di entrambi mediante l'applicativo web creato appositamente da Software Synthesis.

Progettazione: Il processo di verifica garantirà la rintracciabilità nei componenti individuati durante la fase di Progettazione di ogni singolo requisito descritto nell'Analisi dei Requisiti.

Realizzazione: I programmatori svolgeranno le attività di codifica del prodotto e i test di unità per la verifica del codice realizzato nel modo più automatizzato possibile. I verificatori inoltre controlleranno successivamente la presenza di eventuali errori o anomalie.

Validazione: Alla fase di collaudo, il Team garantirà il corretto funzionamento del prodotto MyTalk. Successivi difetti riscontrati o eventuali caratteristiche non coerenti alle richieste dell'appaltatore saranno soggetti a modifica e correzione al fine di eliminare tali incongruenze.

4.2 Risorse necessarie e disponibili

L'utilizzo di risorse umane e tecnologiche è fondamentale per la verifica di qualità del prodotto e dei processi.

4.2.1 Umane

Software Synthesis si è imposta per garantire un elevato standard qualitativo un team di sviluppo comprendente i seguenti ruoli:

- Responsabile: responsabile della corretta realizzazione del prodotto secondo gli standard e le richieste commissionate, designato all'allocazione corretta delle risorse umane ai rispettivi compiti e stimolarne il coordinamento. Controlla inoltre la qualità dei processi interni mediante le attività di verifica da lui predisposte. Infine ha la facoltà di approvare o meno ogni proposta di correzione (migliorativa o di modifica generica) avanzata.

- Amministratore:
- Verificatore:
- Programmatore:

4.2.2 Software

A livello software risulteranno necessari strumenti per permettere l'automatizzazione nell'analisi statica del codice prodotto, ai fini di ricavarne il maggiorn numero possibile di informazioni.
Risulteranno altrettanto utili ai fini dei test di unità legati al linguaggio di programmazione scelto (HTML 5 e Java?) dei Frameworks specializzati, e degli strumenti per standardizzare (e automatizzare) i test sul prodotto finale nonchè produtte dei resoconti appropriati sulle eventuali anomalie riscontrate.
Infine il team ha deciso di produrre un semplice programma basato su interfaccia Web per il tracciamento e la gestione dei requisiti in modo da rendere più stardard e automatizzata possibile questa fase del progetto.

4.2.3 Hardware

Software Synthesis ha a disposizione oltre al materiale personale di ogni componente del team (Computer portatili e fissi) le strutture fisiche ed informatiche messe a disposizione dall'Università degli Studi di Padova per il dipartimento di Matematica Pura ed Applicata, quali laboratori didattici e le aule studio allocate negli stabili della Facoltà stessa.

5 Qualità

Al fine di garantire un elevato standard qualitativo, sia per ovvia scelta del team di sviluppo, sia per implicita richiesta da parte del cliente stesso, Software Synthesis ha preso come riferimento lo standard ISO/IEC 9126 le cui specifiche primarie sono consultabili al sito XXX.

5.1 Funzionalità

Il prodotto MyTalk deve soddisfare nelle sue funzionalità tutti i requisiti individuati nella fase di Analisi, garantendone il funzionamento e l'aderenza alle richieste specifiche del committente. Tutto questo sarà svolto nel modo meno oneroso sia dal punto di vista economico che di sfruttamento delle risorse disponibili. Per valutare il grado di funzionalità raggiunto dal prodotto si valuterà la quantità di requisiti che sono stati correttamente implementati all'interno del software finale. La soglia minima di soddisfacimento risulta essere l'assoluta copertura dei requisiti obbligatori imposti dal committente, tuttavia è parso chiaro che la fantasia del team in questa specifica area sarà ben valutata.

5.2 Portabilità

Il prodotto finale dovrà per vincoli di capitolato essere pianamente usufruibile mediante browser Chrome, prodotto da Google, su tutti i sistemi operativi sui quali questo browser risulta compatibile. A dimostrazione di tale soddisfacimento ci si affida alla dimostrazione del superamento della validazione del codice del front-end web (?) e dell'assoluta coerenza con lo standard WebRTC, proposta evolutiva di HTML5. All'approvazione del superamento di tale requisito si procederà con lo stesso metodo testando browser alternativi a quello imposto al fine di rendere MyTalk usufruibile da un bacino d'utenza più vasto possibile.

5.3 Usabilità

Il prodotto deve risultare facile ed intuitivo da parte dell'utenza che dispone di una conoscenza medio-bassa del web e dell'informatica generica. Utenti che hanno familiarità con programmi per la gestione di chiamate mediante VOIP (Skype, ...) non dovranno trovare alcuna difficoltà o iniziale disorientamento nell'utilizzo di MyTalk. Data l'aleatorietà di tale qualità di prodotto e la non oggettività nella misurazione di tale caratteristica si cercherà semplicemente di raggiungere tale risultato basandosi su esperienze personali o brevi test su specifici utenti selezionati.

5.4 Affidabilità

L'applicazione deve riuscire a stabilire e mantenere stabile una comunicazione tra due o più utenti, senza mostrare problemi di natura tecnica se non imputabili alla qualità della linea di cui dispongono gli utenti stessi. Deve dimostrarsi altresì robusta nella sua struttura e facile da ripristinare in caso di errori di varia natura. Al fine di garantire queste caratteristiche verrà utilizzata come unità di misura la quantità di interazioni tra utenti con esito positivo, tenendo conto di tutti i parametri che concorrono ad una corretta comunicazione (qualità audio, video, messaggi testuali correttamente inviati/ricevuti, etc.).

5.5 Efficienza

MyTalk si pone come obbiettivo oltre alla corretta ed appagante esperienza comunicativa, anche di non risultare particolarmente esosa dal punto di vista harware, sia dal punto di vista puramente componentistico dell'unità dalla quale si accede al prodotto, sia dal punto di vista dell'uso di banda a disposizione della rete. Verranno pertanto monitorate in fase di test sia le percentuali d'utilizzo di memoria e processore della macchina, sia la quantità di kb/s trasmessi e ricevuti durante l'esecuzione del programma. I test verranno eseguiti su varie tipologie di hardware e linee di diverse velocità, al fine di rendere il software usufruibile dalla più vasta fetta d'utenza possibile. I test risulteranno superati se nei momenti di massimi consumi di risorse il programma riuscirà a garantire un utilizzo fluido e una discreta navigabilità nel web dalla macchina soggetta al test. (ApacheBench citano 7Seeds per prove).

5.6 Manutenibilità

Il capitolato specifica esplicitamente che la modifica e la manutenibilità del software sono una caratteristica fondamentale dell'intero progetto, questa necessità nasce dal costante utilizzo di linguaggi non ancora qualificati come standard, pertanto soggetti a continua (ma fortunatamente non radicale) evoluzione. (Aggiungi-vedi 7Seeds)

5.7 Altro (?)

6 Strumenti, tecniche e metodi

6.1 Strumenti

Si riporta a continuazione un elenco dei principali strumenti per la verifica della qualità di cui intende avvalersi il team nell'arco dello sviluppo del progetto.[1]

- **SynthesisRequirementManager**, il sistema di gestione dei requisiti 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 (`www.softwaresynthesis.org`);
- **lacheck** ($\geq$ 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 insecabili, ecc. (`www.ctan.org/pkg/lacheck`);
- **hunspell** ($\geq$ 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;

[1] Si rimanda invece alle Norme di Progetto per un elenco degli strumenti utilizzati non strettamente in relazione con la verifica e il controllo qualitativo.

- **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 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.jslint.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 multipiattaforma 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 o metriche, per cui si rimanda alla sezione 6.3 (`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`);

6.2 Tecniche

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 (programmatori o redattori di documentazione). Al fine di garantire la qualità, i verificatori sono tenuti all'utilizzo di due tecniche di analisi: statica e dinamica.

6.2.1 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 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 6.1 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).

6.2.2 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.

6.3 Misure e metriche

Si riporta, senza pretesa di esaustività, un elenco delle principali metriche in riferimento alle quali il team di sviluppo si ripropone di valutare in modo univoco e quantificabile la qualità del prodotto relativamente alla parte di codifica:

- numero di righe di codice esclusi commenti e annotazioni, considerato nella sua totalità (TLOC, *Total lines of code*) o piuttosto come corpo dei soli metodi (MLOC, *Method lines of code*);
- numero di metodi (NOM, *Number of Methods*) e numero di campi dati di ciascuna classe (NOF, *Number of Fields*);
- profondità di una classe nell'albero di derivazione (DIT, *Depth of Inheritance Tree*);
- numero di metodi ridefiniti (NORM, *Number of OverRidden Methods*)
- indice di specializzazione (IS), definito come

$$IS := \frac{DIT \times NORM}{NOM}$$

- complessità ciclomatica (o complessità condizionale), un indice che misura all'interno di un metodo il numero di cammini distinti che il flusso di controllo può intraprendere nel codice sorgente incrementando un indice di 1 unità per ogni istruzione di branch (if, for, while, do case, catch, operatore condizionale ternario, operatori logici cortocircuitati);
- peso della classe (WMC, *Weighted Methods per Class*) definito come la somma della complessità ciclomatica di tutti i metodi membri di una classe;
- mancanza di coesione dei metodi di una classe (LCOM, *Lack of Cohesion of Methods*), un indice che misura quanto i metodi di una classe fanno riferimento ai campi dati della stessa, definito se $m(A)$ è il numero di metodi che riferiscono il campo dati A come

$$LCOM := \frac{\frac{1}{NOF}\left(\sum_{A \text{ attributo}} m(A)\right) - NOM}{1 - NOM}$$

- indice di utilità (Ca, *Afferent Coupling*) definito come il numero di classi esterne al package che dipendono da una determinata classe;
- indice di dipendenza (Ce, *Efferent Coupling*), definito come il numero delle classi interne al package che dipendono da una classe;
- instabilità (I) definita come

$$I := \frac{Ce}{Ca + Ce}$$

- astrattezza (a livello di progetto) dato dal rapporto fra il numero di classi astratte/interfacce e il numero totale di tipi;
- la metrica relativa a metodi e procedure $SFIN - SFOUT$, dove SFIN (*Structural fan-in*) è il numero di volte che tale metodo è invocato nel corpo di altri metodi e SFOUT è il numero di invocazioni di metodi esterni all'interno del metodo stesso (corrispondono a Ca e Ce rispettivamente).
- rapporto fra righe di commenti e righe di codice.

Altre metriche che saranno prese in considerazione sono la lunghezza dei metodi e il numero di parametri di ciascun metodo.

6.4 Metodi

Il processo di misurazione attraverso il quale è possibile valutare quantitativamente l'aderenza del prodotto agli standard di qualità è basato su un ciclo iterativo simile al PDCA. In particolare, in una fase preliminare prevede che sia stabilita l'importanza e l'ambito di applicazione della misurazione, la selezione di metriche da adottare come si è fatto nelle precedenti sezioni e la pianificazione del momento nel ciclo di sviluppo in cui le misurazioni dovranno essere effettuate.

La parte operativa, cioè la quantificazione dei valori delle metriche di qualità, avviene lungo tutto l'arco del ciclo di sviluppo, con particolare attenzione alle fasi di progettazione di dettaglio e di test effettuati contestualmente alla codifica e in fase di accettazione/collaudo.

Al fine di evitare che la verifica della qualità sia un onere troppo gravoso in termini di risorse umane e di tempo, anch'esso deve essere sottoposto a misurazione in quanto attività di progetto (cioè la parte 'check' del ciclo di Deming): il tempo di lavorazione impiegato per attività di verifica e QA dovrà dunque essere registrato in ogni momento, ed è prerogativa del responsabile di progetto adottare misure correttive qualora i tempi dovessero risultare eccessivi al punto da compromettere il rispetto delle scadenze stabilite nel piano di progetto.

In via preliminare, si prevede di utilizzare i seguenti metodi di analisi: **analisi del flusso di controllo** volta ad assicurare che il codice sia ben strutturato e non contenga punti non raggiungibili, **analisi del flusso dei dati** per scongiurare l'utilizzo inconsapevole di variabili non inizializzate e rilevare variabili inutilizzate o 'write-only', **analisi del flusso di informazione** al fine di assicurare l'assenza di dipendenze fra valori di variabili incoerenti con quanto previsto in fase di design architetturale (a livello di modulo, di componente o di sistema nella sua totalità).

7 Gestione amministrativa della revisione

7.1 Gestione anomalie e incongruenze

Un anomalia è una deviazione dalle aspettative definite sul prodotto, per la loro gestione è stato pianificato l'utilizzo di un sistema di ticketing. Lo strumento scelto dal team è Codebase, (`http://www.codebasehq.com/`) che permetterà ad ogni verificatore di aprire un ticket per ogni nuova anomalia riscontrata. I ticket sono strutturati nel modo seguente:

Titolo: comprende il nome del file da modificare e una descrizione stringata dell'errore;

Tipologia: indica la tipologia del ticket aperto, nel caso di anomalie di cartattere tecnico sarà impostato a Bug.

Categoria Errore: individua macroscopicamente il tipo di errore preso in esame.

Stato: tiene traccia dell'attuale stato del ticket ed è catalogato in cinque categorie:

- Nuovo: ticket appena creato dal Verificatore, rappresenta lo stato iniziale di ogni nuova pratica.
- Accettato: stato booleano che identifica l'approvazione da parte del Responsabile di Progetto e ne assegna la pratica ad uno specifico soggetto (vedi punto Responsabile Correzione).
- In gestione: il soggetto assegnato si sta occupando del problema e della relativa correzione dell'anomalia riscontrata.
- Corretto: l'anomalia è stata correttamente gestita e risolta, il Responsabile di Progetto può considerare il ticket chiuso.
- Non Valido: l'anomalia proposta dal verificatore viene respinta dal Responsabile di Progetto in quanto non sussiste o semplicemente è già trattata in un altro ticket.

Responsabile Correzione:

Note:

Tag:

7.2 Procedure di controllo di qualità di processo

8 Resoconto delle attività di verifica

9 Pianificazione ed esecuzione del collaudo (?)