Progetto SIGEOL

Piano di Qualifica

(Delta)

Redazione: Carlo Scortegagna, Barbiero Mattia

4 giugno 2009

quixoft.sol@gmail.com

Verifica:	Scarpa Davide
Approvazione:	Beggiato Andrea
Stato:	Formale
Uso:	Esterno
Distribuzione:	QuiXoft
	Rossi Francesca
	Vardanega Tullio
	Conte Renato

Sommario

Aggiornamento del Piano di Qualifica per il progetto "SIGEOL", contenente solamente le modifiche rispetto al documento consegnato alla Revisione di Qualifica.

Indice

1 Introduzione

1.1 Scopo del documento

Lo scopo del presente documento è di notificare gli aggiornamenti che sono stati apportati rispetto al documento consegnato alla precedente revisione.

Tali modifiche rappresentano il proseguimento delle attività di verifica e validazione del materiale prodotto e la descrizione degli ambienti di prova e di collaudo.

2 Resoconto delle attività di verifica

Ogni risultato ottenuto dalle attività di verifica, validazione e qualifica dovrà essere attentamente elencato in questa sezione, in modo da assicurare che tutti i problemi e le relative soluzioni siano tracciate per garantire la massima qualità possibile del prodotto finale.

2.1 Tracciamento componenti - requisiti

2.2 Dettaglio delle verifiche tramite test

Per l'esecuzione delle attività di verifica tramite test verranno utilizzati i moduli e le classi rese disponibili dal framework Rails, e più precisamente ogni test di unità dovrà estendere la classe ActiveSupport::TestCase ed ogni test funzionale dovrà estendere la classe ActionController::TestCase.

2.2.1 Functional Tests

Nello sviluppo di un'applicazione tramite il framework Rails, i test funzionali (functional tests) sono specifici per la verifica degli elementi appartenenti alla componente Controller. Dato che gli unit tests sono stati effettuati tramite istanze reali, il team QuiXoft ha scelto di utilizzare la strategia dei Mock objects per la verifica delle azioni presenti nei controller.

Ogni classe che implementa un insieme di test per un particolare controller dovrà essere denominata `NomeControllerTest` ed essere salvata su di un file chiamato `nome_controller_controller_test.rb` all'interno della directory `test/functional`.

Un esempio di functional test per il controller `sessions` è dato dalla seguente porzione di codice:
nel file `sessions_controller_test.rb`

```
class SessionsControllerTest < ActionController::TestCase

  test "Guest usa New" do
    get :new
```

```
    assert_template "new"
    assert_response :success
  end

  test "Immissione di email e password validi" do
    user = stub(:id => :an_id, :mail => "a_mail",
                :password => "a_password")
    User.stubs(:authenticate).returns(user)
    user.stubs(:active?).returns(true)
    post :create, :mail => user.mail, :password => user.password
    assert_equal session[:user_id], :an_id
    assert_redirected_to timetables_url
  end
end
```

2.3 Metriche

Il progetto 'Sigeol', al termine della fase di programmazione, è stato testato con numerosi strumenti dedicati a Ruby on Rails con lo scopo di generare delle metriche precise che indichino il livello qualitativo del codice prodotto dal team QuiXoft.

E' stato scelto di utilizzare Metric_Fu 1.0.2, raccolta di gemme per Ruby on Rails scaricabile da http://metric-fu.rubyforge.org/

Nei capitoli seguenti saranno analizzate una ad una tutte le metriche utilizzate:

2.3.1 Rcov

Lo strumento Rcov è stato usato per misurare la copertura dei test sul codice. Il team QuiXoft utilizzerà la metrica C0, ovvero sarà controllata la copertura di ogni istruzione, e quindi di ogni riga di codice dell'intero progetto.

La situazione di copertura dei test sul codice è riportato nel file *coverage.pdf* allegato al presente documento.

2.3.2 Flay

Qualsiasi progetto Rails dovrebbe seguire il più possibile il principio DRY: Don't Repeat Yourself. Anche se questo deve essere inteso come principio generale, dovremmo anche evitare di avere ripetizioni all'interno del medesimo file.

Flay permette di tenere sotto controllo le duplicazioni, analizzando le "similarità strutturali" (branch, cicli, etc) presenti nel codice: se due parti di codice sono simili allora potrebbero essere buone candidate per un refactoring.

Il risultato di tale metrica di analisi si può consultare nel file *flay.html* allegato al presente documento.

2.3.3 Flog

Lo strumento Flog applica una metrica ABC al codice del progetto 'Sigeol' al fine di misurarne la complessità. La metrica ABC misura la distanza euclidea dall'origine nello spazio tridimensionale formato da:

- Assignments
- Branch
- Condition

Più il codice risulta "lineare" minore sarà il valore di tale metrica ad esso applicato (e tenderà ad essere più gestibile). Al contrario, nel caso di codice ricco di cicli, sottocicli e diversi branch, mostrerebbe un valore elevato, indicando un codice, almeno in teoria, più soggetto a bachi.

Il risultato di tale metrica di analisi si può consultare nel file *flog.html* allegato al presente documento.

2.3.4 Saikuro

Saikuro è uno strumento per misurare la complessità ciclomatica, metrica strutturale relativa al flusso di controllo di un programma che rappresenta la sua complessità logica, cioè lo sforzo per realizzarlo e comprenderlo.

Il risultato di tale metrica di analisi si può consultare nel file *saikuro.html* allegato al presente documento.

2.3.5 Reek

Reek è uno strumeto di analisi del codice che ci dice se e dove compaiono pattern "sospetti", come ad esempio:

- metodi troppo lunghi
- classi troppo ampie
- nomi criptici
- liste di paramentri eccessivamente lunghe
- duplicazioni

Il risultato di tale metrica di analisi si può consultare nel file *reek.html* allegato al presente documento.

Al termine della fase di programmazione molte parti di codice del progetto 'Sigeol' sono state modificate e ottimizzate tenendo conto delle considerazioni nate consultando il risultato di Reek, ma molte delle segnalazioni ancora presenti nel file allegato sono state ritenute poco significative o addirittura in contraddizione con scelte fatte durante lo sviluppo: si è ritenuta quindi irrealizzabile l'idea di eliminare completamente tutti i problemi segnalati da questo strumento.

2.3.6 Roodi

Roodi è l'acronimo di "Ruby Object Oriented Design Inferometer". E' uno strumento che analizza il nostro codice relativamente a:

- nomi delle classi
- nomi dei metodi
- complessità ciclomatica (sia a livello di metodi che di blocchi)
- blocchi "rescue" vuoti
- cicli del tipo "for"
- lunghezza dei metodi

Il risultato di tale metrica di analisi si può consultare nel file *roodi.html* allegato al presente documento.

2.3.7 Churn

Churn tiene traccia dei file soggetti a maggiori modifiche. Un file il cui contenuto cambia spesso è un sintomo di probabile necessità di refactoring: forse è necessario rivedere le entità definite, oppure introdurre un diverso design pattern. Churn inferisce i cambiamenti attraverso l'analisi dei log dei Subversion.

Il risultato di tale metrica di analisi si può consultare nel file *churn.html* allegato al presente documento.

Diario delle modifiche

Data	Versione	Modifica
04-06-2009	Delta	Aggiunte le descrizioni dei tool di misurazione delle metriche