Test e debug di un'applicazione rails

Ultimo articolo sul promettente framework MVC Ruby on Rails..

Test

Al contrario di quanto si potrebbe pensare, testare un'applicazione rails non e' un'operazione che si compie una volta ultimata (o quasi) la scrittura della stessa, anzi! Il test e' parte integrante della stesura del codice e addirittura si scrivono prima! Rails implementa gia' un suo framework di test, ma sempre piu' si sta diffondendo l'uso dell'ottimo tool **rspec**, uno strumento di **BDD** (*Behaviour Driven Development*). Anche questi tool introducono ovviamente dei **DSL** *(Domain Specific Language)* per assolvere ai propri compiti.

Una volta installato rspec con gem possiamo scrivere per esempio un test per il modello user. Che caratteristiche dovrebbe avere un utente? Deve avere un user name valido e univoco, una password di almeno tot caratteri, ecc... Traducendo in inglese le specifiche otteniamo i predicati del nostro test:

- A user should have a unique username

- A user should have a valid username

- A user should have a password long more than 5 characters

ecc...

Individuati i predicati, i test si realizzano scrivendo del codice ruby che verifichi quanto affermato. Un esempio di codice che possa verificare il primo predicato e':

```
describe User do

   it “should have a unique username” do

         User.find(:all, :conditions =>
[“username = pippo”]).size.should == 1

   end



   it “should have a valid username” do

      …

   end

end
```

I test li eseguiremo ovviamente mentre lavoriamo sul database di test che potremo appositamente costruire tramite le *fixture*, dei file yaml che specificano esempi di vari record da creare nel db. Lavorare con le fixtures fa si che si possa semplicemente *droppare* e *ricreare* il database in caso di necessita' e continuare a lavorare sui dati di test ai quali si e' abituati senza perdere anni a ricrearli. Le fixture le trovate nella cartella test/fixtures e ne trovate una per ogni modello.

I test ovviamente falliranno finche' non c'e' un codice che implementi i controlli necessari a far si ad esempio che l'utente rispetti le caratteristiche indicate. Sulla base quindi dei test individuati si puo' iniziare a scrivere l'applicazione e verificare, man mano che si completano i task, che i test diventino verificati.

Traducendo in codice applicazione i test che scriviamo puo' capitare di scrivere piu' codice del necessario, di implementare piu' funzioni di quelle che avevamo previsto. Come facciamo pero' a verificare che tutto il codice della nostra applicazione sia coperto dai test? Con dei tool di *code coverage*, cioe' dei test di copertura del codice: anche qui abbiamo a disposizione uno strumento validissimo, **rcov** che possiamo installare al solito dalle gem con:

```
# gem install rcov
```

Dopdiche':

```
$ rake spec:rcov
```

I risultati li troviamo in formato html nella cartella *coverage*.

Entrare nell'ottica del BDD non e' semplice e mi ha richiesto un bel po' prima che i concetti fossero del tutto assimilati, tuttavia e' un'esperienza da fare ed e' un metodo che puo' giovare notevolmente alla scrittura ordinata del codice ;)

Debug

Il debugging dell'applicazione Rails puo' essere effettuato tramite la libreria **ruby-debug**. Per utilizzarla e' necessario innanzitutto installarla con:

```
# gem install ruby-debug
```

Al termine dell'installazione e' sufficiente riavviare il server con l'apposita opzione per poter utilizzare subito il debugger:

```
$ script/server --debugger
```

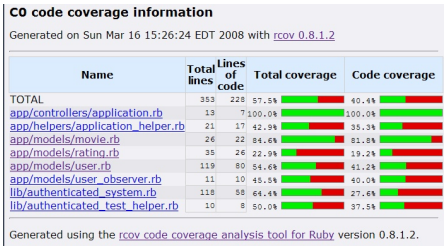
In alternativa, se non vogliamo riavviare il nostro server possiamo inserire all'interno del file da debuggare l'istruzione

```
require 'ruby-debug'
```

Possiamo inserire dei breakpoint all'interno del codice che fermino l'esecuzione dell'applicazione e aprano una shell dalla quale ispezionare lo stato delle variabili e delle nostre istanze. La shell avra' un prompt di questo tipo:

```
(rdb:n)
```

n e' il thread number. Precede il prompt una riga riportante la prossima istruzione da eseguire. Uno dei comandi importanti da imparare e' **list** che stampa a video le cinque righe che precedono e seguono l'istruzione attualmente in esecuzione. Digitare nuovamente list (o la sua abbreviazione *l*) fa si che venga stampato il seguito del codice. Per far avanzare l'esecuzione del programma utilizziamo **next**.



Co code coverage information

Generated on Sun Mar 16 15:26:24 EDT 2008 with rcov 0.8.1.2

Name	Total lines	Lines of code	Total coverage	Code coverage
TOTAL	353	228	57.5%	40.4%
app/controllers/application.rb	13	7	100.0%	100.0%
app/helpers/application_helper.rb	21	17	42.9%	35.3%
app/models/movie.rb	26	22	84.6%	81.8%
app/models/rating.rb	35	26	22.9%	19.2%
app/models/user.rb	119	80	54.6%	41.2%
app/models/user_observer.rb	11	10	45.5%	40.0%
lib/authenticated_system.rb	118	58	64.4%	27.6%
lib/authenticated_test_helper.rb	10	8	50.0%	37.5%

Generated using the rcov code coverage analysis tool for Ruby version 0.8.1.2.

Esempio dell'output di rcov

Link utili:
http://rubyforge.org/projects/rcov/
http://rspec.info/documentation/
http://rubyforge.org/projects/ruby-debug/