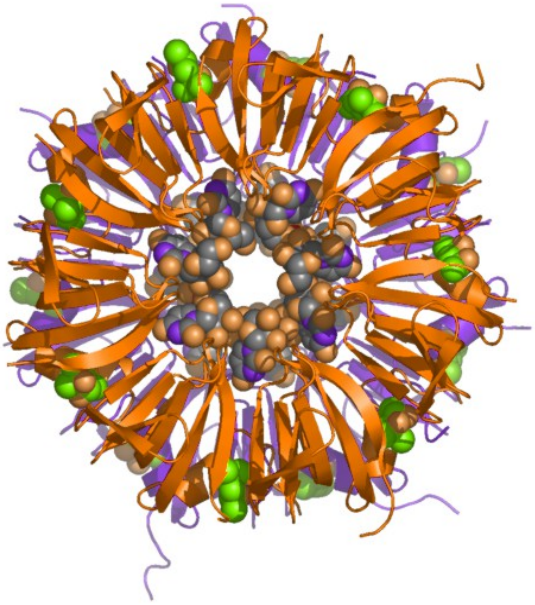

Matteo Cartuccia
Flavio Macciocchi

|FIXIT|

Documento di test

ID: 0-FIX-TST-v04-r00

Data ultima modifica: 18/01/2013

Progetto di Ingegneria del software A.A 2012/2013

1. Introduzione

L'attività di *testing* è una parte fondamentale durante lo *Unified Process* perché è compresa in quasi tutte le fasi di progettazione e ad ogni iterazione si pone l'obiettivo di verificare il corretto funzionamento dell'intero sistema.

In particolare, bisogna realizzare i seguenti punti:

1. Test di integrazione per ogni componente: si testano separatamente le varie parti del sistema (funzioni, oggetti componenti riutilizzabili).
2. Test finale di sistema: si testa il sistema completo.

L'obiettivo del punto (1) è quello di confermare che il sistema soddisfi i suoi requisiti funzionali e non funzionali, senza che si verifichino comportamenti imprevisti o anomali. Lo scopo di questa fase è duplice: deve sì dimostrare allo sviluppatore e agli utenti finali che il software soddisfa i requisiti sopra citati, ma anche rilevare possibili problemi e comportamenti fuori controllo.

Invece, il punto (2) si riferisce ai test che si occupano di verificare se le varie componenti (che separatamente non risultano errate o problematiche) unite tra loro continuino a lavorare correttamente, cioè secondo la progettazione.

Durante questi test si succedono altre due nuove fasi: la prima, chiamata anche *test dell'integrazion*e, si occupa di trovare difetti nel sistema avvalendosi del *debugging* del codice sorgente inerente alle componenti problematiche.

Nella seconda fase, identificata con il nome di *test della release*, le verifiche sono condotte a “scatola chiusa”, cioè senza sapere come è strutturato l'intero sistema. Si procede pertanto a controllare solamente se esso lavora a dovere oppure no; se poi viene rilevato un errore ci si limita a segnalarlo al team di sviluppo che provvederà a effettuare il *debugging* del programma.

2. Test di Prestazione

Questi test si riferiscono alle prestazioni (si deve assicurare che il sistema possa elaborare il carico di lavoro previsto) e all'affidabilità. Richiedono la pianificazione di test in cui il carico di lavoro è aumentato in modo costante finché le prestazioni del sistema diventano inaccettabili e portano a malfunzionamenti. Devono dimostrare che il sistema soddisfi i propri requisiti.

Esempi di test: nel nostro sistema si possono progettare dei test che verifichino la stabilità del software stesso, stressando l'utilizzo delle risorse e stabilendo con precisione di quanta RAM necessita il *puzzle-game* per non ritrovarsi nella fase di stallo.

3. Test delle Classi

Comprende, tra le varie cose:

- Il test di tutte le operazioni associate all'oggetto.
- L'impostazione e l'interrogazione di tutti gli attributi associati all'oggetto.
- La prova dell'oggetto in tutti i possibili stati.

Riferendoci alla classe checkUtente() vanno effettuati dei *test case* su *username* e su *email*. Il *test case* ha lo scopo di progettare gli input e gli output attesi. Ecco alcuni esempi di *test case*:

username: si deve simulare la registrazione di due utenti differenti e il relativo inserimento del nome. Si controlla che entrambi gli utenti non possano inserire lo stesso nome utente.

email: si effettuano due registrazioni e si procede controllando che gli indirizzi e-mail degli utenti siano distinti.

Per testare gli stati della classe si usa un modello a stati, tramite il quale si possono identificare sequenze di transizione di stato che devono essere testate.

Esempi di sequenze di stato da testare sono:

nomeUtente->indirizzoMail->nomeUtente.

nomeUtente->password->nomeUtente.

indirizzoMail->password->indirizzoMail.

4. Test delle Interfacce

Questo tipo di test si occupa di controllare l'interfaccia e i suoi comportamenti. I test per le interfacce dovrebbero:

- Esaminare il codice da testare ed elencarne ogni chiamata ad una componente esterna.
- Progettare test i cui valori dei parametri delle componenti esterne sono al limite del loro *range*.
- Se si utilizzano puntatori, testare l'interfaccia con i parametri dei puntatori nulli.
- Dove viene richiamata una componente tramite un'interfaccia procedurale, creare test che causino fallimenti della componente.
- Nel caso in cui molte componenti interagiscano tra loro usando memoria condivisa, creare dei test che variano l'ordine in cui le componenti sono attivate.

5. *Beta testing*

Un comunicato software in versione beta è una versione di un software incompleta: si da per scontato che il suddetto software funzioni come da stato finale, non garantendo però la sicurezza e la sua stabilità. Ci si affida, come molte volte avviene, all'utente finale che provvederà a segnalare malfunzionamenti e possibili bug inviando opportuni feedback.

E' sempre l'utente infatti a scovare errori, anche difficilmente individuabili da esperti che compiono un numero considerevole di cicli testing.

Per questo il progetto si avvarrà ancora una volta della potenza del web, distribuendo su larga scala il codice eseguibile. La forza della versione beta sta nel fatto che il produttore non è tenuto ad attendere ulteriore tempo per testare l'applicativo, e ciò si ripercuote anche sul risparmio finanziario che questo riceve.