Software testing

Introduzione

Obiettivi:

- Comprendere l'importanza e l'efficacia nell'utilizzo di fasi test all'interno dello sviluppo di un sistema software
- Adeguare propriamente metodiche di testing affinchè sia acquisibile un certo livello di confidenza con l'ambiente sviluppato

Parte cruciale all'interno di un qualsiasi processo di sviluppo software consiste nella creazione di *testing tools*, i quali provvedono a specificare il livello di *correttezza* e di *validità* del progetto implementato. Si sottolinea la sottile differenza che contraddistingue l'etimologia di *testing*; essa non rappresenta una metodica che specifica il grado di qualità della soluzione proposta, anzi essa è adoperata per ulteriori scopi, affini alla creazione di un concreto livello di *confidenzialità* capace di esprimere se la struttura implementata all'interno del sistema sia corretta o meno.

Rispetto a quanto detto si analizzano due tematiche inerenti alla breve descrizione precedente, in cui il termine *software testing* è accomunato da una duplice espressione, ossia:

Definizione informale
Il termine *software testing* promuove due caratterizzazioni suddivise in *validità* e *veridicità*; la prima esprime la valutazione attuata per definire la bontà dell'architettura del sistema software, mentre la corrisposta propone una stima della correttezza dell'implementazione adeguata fino ad ora.

Concludendo l'intento di un meccanismo simile consiste nella rivelazione del maggior numero possibile di diffetti, poichè il costo relativo alla correzione di situazioni errate è proporzionalmente diretto al tempo di risoluzione. Riassumendo le fasi di test sono fondamentali per veicolare il progetto verso un vantaggio competitivo, ma ciò può avvenire solamente se tali comportamenti siano immediatamente attuati e ripetitivi nel *lifecycle* del sistema.

Sintassi

Di seguito è riportato l'insieme di definizioni che contraddistinguono un contesto simile a *software testing*.

Definizione defect
Un *defect*, detto anche *bug*, indica un errore logico, che non sempre si traduce in un concreto malfunzionamento.

Per cui un difetto è una sequenza di istruzioni, che, quando eseguite con particolari dati in input, genera un *malfunzionamento*, ossia un comportamento software difforme dai requisiti espliciti. La mancata effettività del malfunzionamento avviene qualora non siano immessi i

dati in input tali da evidenziare l'errore all'interno del codice che contiene il diffetto.

Definizione failure
Un *failure* rappresenta il momento in cui viene riscontrato concretamente il *defect*.

Tradotto, un *malfunzionamento* indica un comportamento contrario all'aspettative, in cui, come già detto prima, si riscontra qualora programmatori o sviluppatori immettino un errore all'interno della soluzione provocando un *bug*, il quale diviene un *failure* solamente se poste certe condizioni.

Definizione test case
Test case rappresenta l'elenco che contiene una breve descrizione degli *outcome* attesi da singoli processi esecutivi, i quali includono *parametri*, *condizioni* e *risultati* attesi. Nel gergo l'insieme dei *test case* da vita a *test set*.

Testing levels

I test possono essere adeguati a differenti granularità, suddividendosi in *unit testing*, *integration testing* e *end-to-end testing*. Oltre al grado di precisione che contraddistingue ognuno di essi, sono caratterizzati da una specifica complessità, in cui è possibile porre al primo posto la tipologia *end-to-end*. Di seguito è proposta una visione più dettagliata di ogni caratterizzazione, illustrata come:

- *Unit testing* riferisce a un meccanismo di verifica che opera su specifiche sezioni del *code base*, solitamente relative a livelli funzionali e operativi. Tendenzialmente queste tipologie di test sono adoperate da sviluppatori in concomitanza alla stesura di funzionalità, affinchè possano assicurarsi che il metodo operi correttamente. *Unit test* non è in grado di analizzare la correttezza delle sezioni software prese in considerazione, ma stabilisce se i blocchi di codice possono svolgere in maniera indipendente le proprie funzionalità; si ricorda l'importanza delle *dipendenze* all'interno di uno sviluppo software, capaci di inibire l'intera struttura ideata qualora sia erroneamente gestita.
- *Integration testing* tenta di controllare attivamente il livello di astrazione posto tra classi legate alla logica algoritmica ed entità inclini ad interfacce utente. Per cui un meccanismo di controllo simile pone la propria attenzione sulla ricerca di *defects*, posti tra interazioni di elementi che compongono il modello analizzato.
- *End-to-end testing* rappresenta lo strumento di analisi più complesso e costoso in assoluto, poichè tenta di controllare l'intero sistema software attuato. Solitamente non sono riscontrabili test automatizzati simili, data l'elevata difficoltà nella creazione di un tool in grado di analizzare ogni singolo dettaglio che caratterizza il progetto osservato, a causa di questa principale ragione non risulta essere molto adoperato all'interno di team di sviluppo.

Testing approach

Sono presenti differenti approcci che possono essere adeguati in fasi di *software testing*, suddivisi in due famiglie *dynamic* e *static*. Le due metodologie illustrate sono strettamente correlate, in cui la prima citata è attuata affinchè sia possibile *eseguire* il *code base* pur

di visualizzare i *bug*, mentre la corrisposta è adoperata per *analizzare* il *code base* pur di individuare i *defects*. Entrambe condividono la stessa finalità anche se spesso sono valorizzati approcci *statici* che *dinamici*; la ragione è dovuta alla netta semplicità che condividino le prime metodologie rispetto alle seconde introdotte.

Data la netta supremazia degli approcci *statici*, di seguito sono proposti alcuni dei più diffusi, quali:

- *Model checking*, ...
- *Symbolic execution*, ...
- *Data-flow analysis*, ...
- *Abstract interpretation*, ...

Black-box testing

Black-box testing è un metodo di test del software che esamina la funzionalità di un sistema senza sapere come concretamente sia implementata la propria infrastruttura interna, da cui ne deriva la denominazione. Questa metodologia può essere implementata su un qualsiasi testing layer, la quale pone maggiore attenzione sulla bontà del risultato ottenuto piuttosto del procedimento attuato per risalire al dato finale.

Come detto in precedenza, non sono richieste conoscenze legate alle specifiche del codice, poichè il *tester* è formulato in modo che conosca *cosa* il sistema software dovrebbe fare, ma non su *come* sia il procedimento che porti al risultato tangibile.

Il termine *black-box testing* raffigura l'insieme di approcci simili, dove i più diffusi si suddividono in:

- *Boundary value analysis*, illustra una tecnica di *software testing* incline alla visualizzazione del solo risultato; come da denominazione, la sua implementazione pervede l'uso di limiti, i quali possono essere associati ad un'entità inferiore e superiore, in cui il risultato sperato dovrebbe rispettare tali vincoli valorizzativi. I vincoli sono definiti *punti di discontinuità*, da cui avviene la costruzione del test per osservare la correttezza del risultato ottenuto dal sistema software.