TirocinioSmart

RTD - Regression Test Document

TirocinioSmart

Rosanna Coccaro

Data	Versione	Cambiamenti	Autori
14/08/2020	0.1	Prima stesura	RC
28/08/2020	1.0	Revisione finale	RC

Sommario

1. Introduzione

Con la parola "*regressione*", il dizionario intende: "il ritorno a uno stato precedente o meno sviluppato". Dunque, il regression testing viene eseguito proprio con l'obiettivo di catturare eventuali *regressioni* in una funzionalità già testata precedentemente.

Queste regressioni nel codice possono verificarsi a seguito di "correzioni di bug", "nuove funzionalità aggiunte al codice" o "modifica dei requisiti". L'obiettivo è testare tutto il codice che potrebbe essere influenzato dalle recenti modifiche per garantire che non vengano introdotti nuovi bug in una funzionalità già testata.

Dunque, il testing di regressione consiste nella ri-esecuzione selettiva di test che hanno come oggetto le componenti modificate al fine di verificare che le stesse modifiche non abbiano provocato effetti indesiderati.

Sottolineiamo che il test di regressione è il tipo più importante di test eseguito quando vengono apportato modifiche ad un software abbastanza stabile. Durante l'esecuzione del testing di regressione ci prefissiamo di tener sempre conto di alcuni punti chiave quali:

- Il test di regressione viene eseguito su caratteristiche e funzionalità stabili che vengono testate prima dell'introduzione di una nuova modifica in un'applicazione;
- Maggiore è il numero di modifiche introdotte in un'applicazione, maggiori sono le possibilità di trovare un nuovo bug in una funzionalità correlata;
- Test di regressione indica l'esecuzione di casi di test già creati per una funzionalità rilasciata;
- In una condizione ideale, una suite completa di casi di test dovrebbe essere eseguita dopo ogni modifica;
- Nel mondo reale, dove il tempo è sempre un vincolo, l'analisi dell'impatto viene eseguita per accertare le aree che possono essere influenzate da un cambiamento.

In questo documento, quindi, ci occupiamo di andare ad identificare le aree del codice precedentemente stabile che devono essere testate per individuare qualsiasi regressione o bug di nuova introduzione a causa delle modifiche e anche i livelli di test ai quali sarà effettuato il regression testing (unità, integrazione, sistema…).

A partire dalle porzioni di codice interessate dalla modifica, andremo a selezionare i casi di test da quelli già creati che testano il codice interessato, in base al tempo al tempo assegnato alla fase di testing stessa.

Infine, documenteremo anche la fase di esecuzione dei casi di test selezionati in base ai criteri che definiremo nel seguito. Segnaleremo eventuali problemi riscontrati durante l'esecuzione dei test di regressione.

2. Approccio al testing di regressione

Una suite di test di regressione è una raccolta di casi di test che devono essere eseguiti quando è necessario eseguire testing di regressione. Ma, poiché esistono diverse tecniche di regression testing adottate dall'azienda a seconda dei processi che seguono, esistono anche diversi tipi di test suite di regressione.

Se lo scopo, come nel nostro caso, è eseguire test di regressione parziale, i casi di test dovranno essere definiti in base al cambiamento dopo il quale è necessario eseguire il test di regressione.

Dunque, rifacendoci al nostro specifico problema, è bene **eseguire un insieme di casi di test selezionati**: a partire dall'impact analysis, andiamo ad individuare le aree interessate dalla modifica e di conseguenza i test case identificati ed eseguiti (la selezione del caso di test è basata sull'insieme dei test relativi alle funzionalità interessate).

Dunque, i casi di test vengono selezionati sulla base delle aree coinvolte nella modifica. In altre parole, andiamo a ridurre la test suite preservando la sua abilità nell'individuare i fault come nel caso della test suite originale, selezionando però solo i casi di test che sono coinvolti direttamente o indirettamente nelle modifiche al sistema.

In particolare, per il nostro progetto, i criteri di selezione e l'insieme dei casi di test eseguiti dipendono per la maggior parte dalla disponibilità di risorse, primo fra tutti il tempo. Infatti, ri-eseguire l'intera test suite porta con se delle limitazioni legate anche al numero di casi di test: più il sistema evolve più aumenta la size della test suite stessa. Quindi, eseguire l'intera test suite potrebbe richiedere molto tempo o anche molte risorse.

Inoltre, nel nostro specifico caso, il progetto è tale che i casi di test possano essere eseguiti rapidamente subito dopo le correzioni e quindi un test di regressione parziale è utile per risparmiare tempo.

Infine, dalle specifiche delle nostre change request si capisce che le modifiche da apportare sono localizzate (non hanno effetti sull'intero sistema) e pertanto non è necessario un test di regressione completo.

3. Livelli di test di regressione

Il testing di regressione può essere effettuato ad ognuno dei livelli di test (unitario, integrazione, sistema). Nel nostro specifico caso, il testing di regressione sarà effettuato solo a due livelli (unità e integrazione) selezionando:

- I casi di test di unità già eseguiti precedentemente e relativi alle componenti che hanno subito modifiche;
- I casi di test di integrazione già eseguiti precedentemente e relativi alle componenti che hanno subito modifiche.

Perché non viene considerato il livello di test di sistema? Per rispondere alla domanda è bene essere a conoscenza dei casi di test di sistema definiti all'interno del documento di test plan aggiornato all'ultima versione.

Consultando tale documento, e analizzando il paragrafo successivo (nel presente documento) in cui sono definite tutte le componenti impattate da modifiche, possiamo affermare che il livello di test di sistema è escluso poiché l'insieme di casi di test definiti ed eseguiti precedentemente (prima versione de software) fanno riferimento a funzionalità e componenti che rimarranno inalterate, e di conseguenza anche il loro relativo comportamento.

4. Criteri Pass/Failed

Il test ha successo se il comportamento osservato è diverso dal comportamento specificato nei requisiti funzionali. Ciò significa che raggiungiamo gli obiettivi che ci siamo posti durante questa fase se il test individuerà dei fault nel sistema.

In tal caso è facile individuare la componente coinvolta nell'errore poiché, come già specificato, i casi di test vengono eseguiti immediatamente dopo le migliorie apportate alla componente stessa.

Il testing fallirà se non saranno scovati errori causati dalle modifiche effettuate sulle componenti già esistenti nella precedente versione del software.

5. Materiale per il testing

L'esecuzione dei test necessita di un server correttamente configurato su cui siano installati Java e MySQL. La configurazione deve avvenire come da manuale d'installazione.

Il testing è condotto utilizzando alcuni dei framework più famosi ed efficaci in ambienti Java: JUnit, Mockito e Hamcrest. Ad essi viene affiancata tutta la suite di test relativa a Spring, framework adottato per la realizzazione della webapp.

In particolare, per l'esecuzione dell'integration testing si fa uso di JUnit e Hamcrest. Ovviamente, l'utilizzo di Junit è legato all'esigenza di scrivere casi di test che siano ripetibili. Per quanto riguarda Hamcrest, è utilizzato poiché ci consente la scrittura di oggetti cosidetti "matchers" e di definire le regole di "match" in modo dichiarativo.