Eleworld2018

System Test Plan

Daniele Cioffi

Mario Consalvo

Indice

Coordinatori del progetto

Prof. Andrea DE LUCIA
Prof.ssa Rita FRANCESCE

Partecipanti

Daniele Cioffi – Matricola 0512103446
Mario Consalvo – Matricola 0512103500

Cronologia delle Revisioni

Data	Versione	Descrizione	Autore
22/01/2018	1.0	Prima Stesura	Cioffi Daniele Mario Consalvo
26/01/2018	1.1	Revisione prima stesura	Cioffi Daniele Mario Consalvo
29/01/2018	1.2	Aggiunta tabelle (test case)	Cioffi Daniele Mario Consalvo
06/02/2018	2.0	Stesura Finale	Cioffi Daniele Mario Consalvo

1. Introduzione

Questo documento è un piano per eseguire il test di sistema eleworld2018 prodotto dal team di test di sistema. Descrive la strategia di test e l'approccio ai test che il team utilizzerà per verificare che l'applicazione soddisfi i requisiti stabiliti prima del rilascio.

Obbiettivi:

- Soddisfa i requisiti, le specifiche e le regole aziendali.
- Supporta le funzioni aziendali previste e raggiunge gli standard software richiesti.
- Soddisfa i criteri di ammissione per il test di accettazione degli utenti.

2. Documenti Correlati

2.1 Relazioni con il Requirements Analisys Document (RAD)

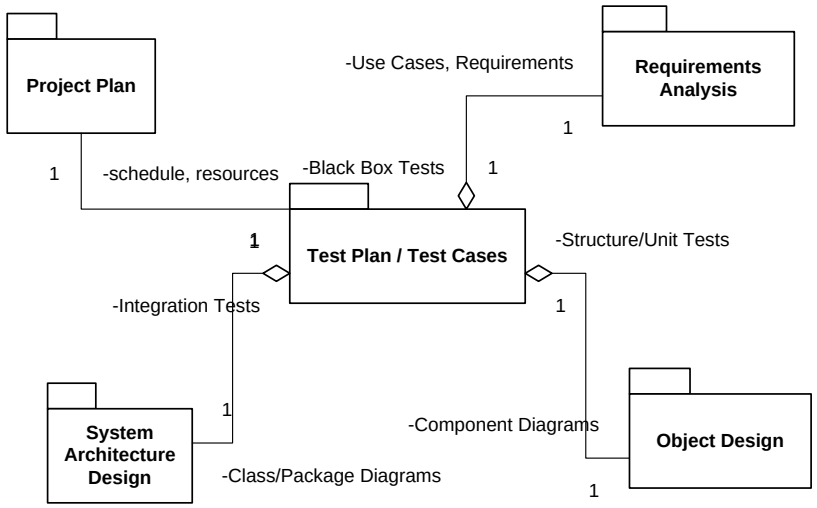
I test vengono eseguiti sui casi d'uso e sui diagrammi dei casi d'suo definiti nel RAD. Inoltre i test delle prestazioni, derivano dai requisiti non funzionali del RAD.

2.2 Relazioni con il System Design Document (SDD)

I test di integrazione vengono generati sulla base del sistem design document. I test di integrazione provengono generalmente dal diagramma del pacchetto che descrive l'architettura del sistema.

2.3 Relazioni con il Object Design Document (ODD)

I test strutturati(di unità) vengono generati sulla base dell'object design document. I test derivano generalmente dal diagramma delle componenti.



3. Panoramica del Sistema

Il progetto Eleworld2018 consiste nella creazione di un sito web e-commerce (Eleworld2018.it) completo di ogni aspetto e funzionalità. Il sito in questione, darà l'opportunità a gli utenti, di visionare e di acquistare, qualsiasi forma di elettrodomestico, in modo pratico e veloce. La finalità del progetto è proprio quella di riuscire a creare un sito web capace di raggiungere traguardi commerciali e di ottenere visibilità sulla rete, ricca di concorrenze.

4. Funzionalità da testare e non testare

Le seguenti features non saranno incluse nel test di sistema:

- Gestione del Carrello
- Gestione dei Pagamenti
- Gestione Inserti Pubblicitari

5. Criteria Pass/Failed

I test eseguiti sui componenti passano solo quando soddisfano le firme, i vincoli e le interfacce dettate dalla specifica di progettazione dell'oggetto per quel componente. Se un test mostra un difetto del prodotto nel soddisfare gli obiettivi delle specifiche di progettazione dell'oggetto, fallirà e un problema verrà segnalato nel sistema per tracciare la successiva fase di revisione. In particolare, per un dato di input

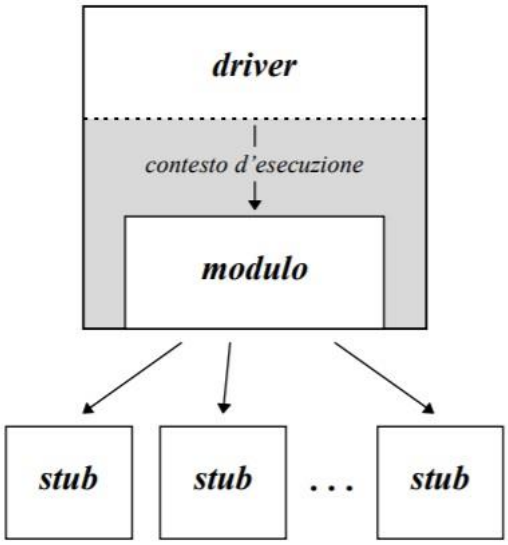
6. Approccio

6.1 Testing di unità (o dei moduli)

A questo livello, il test prende in considerazione il più piccolo elemento software previsto dall'architettura del sistema: il modulo. Il controllo su un modulo ha come obiettivo principale la correttezza funzionale delle operazioni esportate dal modulo a fronte della specifica definita nel progetto di dettaglio. In questo senso il controllo sui moduli si configura come una tipica attività di verifica. Per il controllo sui moduli sono in realtà usati metodi sia statici (ispezione e walkthrough) sia dinamici.

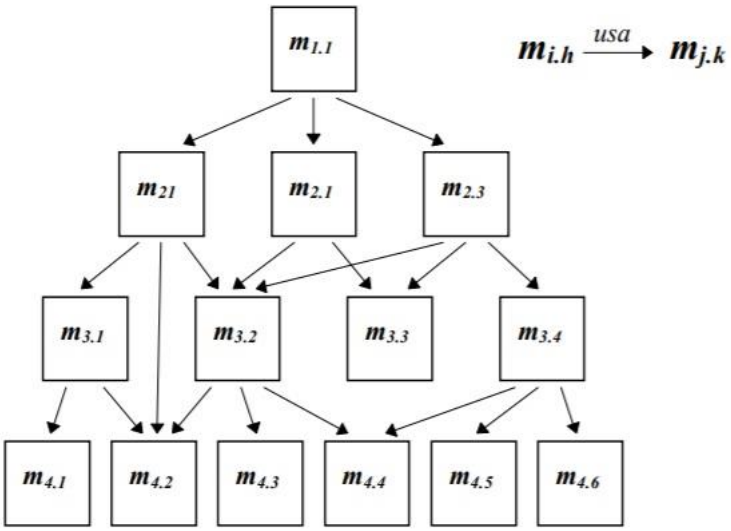
Fare test sui moduli significa dover effettuare i test su sistemi che sono incompleti. È necessario, in questa evenienza, che l'ambiente di test preveda dei componenti fittizi che simulino le parti mancanti del sistema.

La realizzazione di queste componenti – un costo di cui è necessario tener conto – deve essere semplice e diretta per poter un corretto comportamento ed evitare di inquinare i risultati dei controlli.



6.2 Testing di integrazione

È intuitivamente evidente la distanza che separa i controlli sui moduli dai controlli sul sistema completo. La pratica inoltre insegna che l'eliminazione di un difetto a partire da malfunzionamento rilevato da controlli sul sistema è più onerosa e in genere cade sempre in una fase del processo di sviluppo in cui le scadenze sono prossime e il rischio di ritardi elevato. A peggiorare questo quadro intervengono anche le statistiche: da esse scopriamo che circa il 40% dei malfunzionamenti riscontrati nei controlli di sistema sono riconducibili a difetti di mutuo interfacciamento tra i moduli. È quindi necessario colmare in modo efficace la lacuna che, dal punto di vista dei controlli, separa i moduli dal prodotto finito.
Le più recenti metodologie di sviluppo prevedono di condurre in modo progressivo l'integrazione di un sistema, intervallando i vari passi con sessioni di test. Obiettivi delle strategie d'integrazione sono la minimizzazione del lavoro e delle risorse necessarie all'integrazione e la massimizzazione del numero di difetti scoperti prima dei controlli sul sistema completo. Alternare passi di integrazione a passi di controllo significa dover effettuare i test su sistemi che sono incompleti. È necessario, anche in questo caso, che l'ambiente di test preveda stub e driver. Le strategie di integrazione descritte nel seguito nella loro formulazione fanno riferimento al grafo della relazione di uso dei moduli, una delle viste strutturali dell'architettura di un sistema software.



6.3 Testing di sistema

Il test di sistema è la più canonica delle attività di validazione che valuta ogni caratteristica di qualità del prodotto software nella sua completezza, avendo come documento di riscontro i requisiti dell'utente. Il sistema verrà testato nella sua completezza e in modo approfondito, prevedendo due fasi di testing:

- una fase di testing sull'utilizzo superficiale del sistema, ossia le funzionalità maggiormente utilizzate dagli utenti finali;
- una fase di testing sull'utilizzo approfondito del sistema, in cui verrà esaminato ogni singola funzionalità in modo da rilevare eventuali malfunzionamenti o scelte implementative errate.

Si prevede che il sistema sia in buona parte funzionante, forte dei testing fatti in precedenza(unità e integrazione).

Inoltre, il testing di sistema, verrà effettuao su più browser, per testare anche la compatibilità e consistenza di esso.

7.0 Sospensione e Ripresa

7.1 Criteri di Sospensione

Le circostanze che potrebbero indurre alla sospensione parziale o completa dei test sono:

- un malfunzionamento critico generato da un errore di funzionalità del sistema, che rende inutilizzabile l'apparato proposto;
- nel caso in cui eventuali funzionalità principali o set di funzionalità nel sistema integrato non siano implementate come previsto;
- il numero di fallimenti è troppo elevato e rende il test poco credibile. In tali circostanze, non ha senso continuare a testare e sprecare risorse e tempo preziosi;
- il tempo di inattività del sistema o tempo di inattività dell'ambiente.

7.2 Criteri di Ripresa

Le circostanze che portano alla ripresa del sistema sono essenzialmente:

- correzione di errori critici che impediscono il corretto funzionamento del sistema;
- correzione di errori minori che riescano a garantire almeno l 80% del successo dei test case;

8. Materiale per il testing

Per effettuare il testing, ci siamo serviti del framework di JUnit 4, della libreria Mockito e della libreria Java Bean Tester.

In informatica JUnit è un framework di unit testing per il linguaggio di programmazione Java. L'esperienza avuta con JUnit è stata importante nella crescita dell'idea di sviluppo guidato da test (in inglese Test Driven Development), ed è uno di una famiglia di framework di unit testing noti collettivamente come xUnit.

Mockito è un framework di testing open source per Java rilasciato sotto licenza MIT. Il framework consente la creazione di oggetti double test (oggetti simulati) in test di unità automatizzati allo scopo di Test-driven Development (TDD) o Behavior Driven Development (BDD).

Java Bean Tester è una libreria che utilizza la reflection per cercare tutti i metodi get/set in una classe, e genera un valore appropriato per ogni metodo set. Chiama il metodo e controlla se il valore restituito dal metodo get è lo stesso che era stato assegnato.

9. Test Case

Per sviluppare i test cases è stato utilizzato il metodo del category partition.

Il category partition test è un metodo per generare test funzionali da specifiche informali. Si compone di tre passi:

1. Decomporre le specifiche in feauture testabili indipendentemente : in questo passo, il test designer deve identificare le feauture che devono essere testate in modo separato, identificando i parametri e qualunque altro elemento dell'ambiente di esecuzione da cui dipende (ad esempio un database). Per ciascun parametro e elemento dell'ambiente si identificano le caratteristiche (elementari) del parametro, dette categorie. Nel caso in cui il numero di combinazioni per ogni valore del dominio all'interno di una categoria, cresca in modo esponenziale, è utile fissare un upper bound sul numero di combinazioni che si considerano.

2. Identificare Valori Rappresentativi : questo passo, prevedere che il test designer identifichi un'insieme di valori rappresentativi (classe di valori, detta choices) per ciascuna caratteristica di ogni parametro definito nella fase precedente. Questa attività è detta "partizionare le categorie in choices". I valori dovrebbero essere identificati per ciascuna categoria indipendentemente dai valori delle altre categorie. L'identificazione viene eseguita applicando delle regole di boundary value testingo erroneous condition testing. La regola di boundary value testing prevede la selezione di valori estremi all' interno di una classe (valori massimi o minimi), valori esterni ma molto vicini alla classe e valori interni (non estremi). I valori vicini al confine di una classe sono utili nel rilevare i casi d'errore del programma.

3. Generare Specifiche di Casi di Test : In questa fase si stabiliscono dei vincoli semantici sui valori per indicare le combinazioni non valide e ridurre il numero di quelle valide. Una specifica di caso di test per una feauture è data da una combinazione di classi di valori, una per ciascuna caratteristica dei parametri. Il metodo del category partition permette di eliminare alcune combinazioni indicando classi di valori che non hanno bisogno di esser combinate con tutti gli altri valori. L'etichetta[error] indica una classe di valori che può essere combinata una sola volta nelle combinazioni con valori non errati di altri parametri. I vincoli property e if-property sono utilizzati insieme; property raggruppa valori di una singola caratteristica di un parametro per identificare sottoinsiemi di valori con proprietà comuni. Il 10 vincolo è indicato con la stringa [property PropertyName]. Il vincolo if-property limita le combinazioni di un valore di una caratteristica di un parametro con particolari valori selezionati per una caratteristica di un parametro diverso. Il vincolo è indicato con la stringa [if PropertyName]. Infine il vincolo single si comporta come error, limitando il numero di occorrenze di un valore nella combinazione a 1.

La ripartizione in categorie, e i test case sviluppati sono riportati rispettivamente nel Test Plan (TP) e nel Test Case Specification (TCS).

L'esecuzione dei test cases sarà documentata nel Test Summary Report (TSR) e nell' Anomaly Report (AR).

10 . Pianificazione del testing

10.1 Determinazione dei ruoli

Il team dedicato al testing di sistema sarà composto da: Daniele Cioffi e Mario Consalvo. Le attività di testing (unità e integrazione) dei componenti saranno effettuati dagli stessi sviluppatori in quanto tutti i componenti del team hanno partecipato allo sviluppo del software.

10.2 Determinazione dei rischi

I rischi di un completo fallimento verranno minimizzati effettuando una pianificazione verticale delle attività di testing funzionale. Questo permtterà, in caso di ritardi dovuti ad una grande quantità di failure trovate, di rilasciare meno funzionalità del previsto ma completamente testate. Inoltre tale pianificazione ridurrà notevolmente la produzione di driver e stub, evitando l'introduzione di nuovi errori dovuti all'implementazione di tali componenti.

10.3 Decomposizione gerarchica del sistema