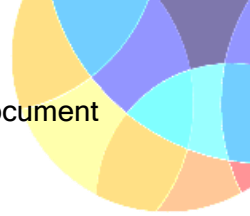
Test Plan Document

Versione 1.0

Partecipanti: Monaco Simone, Santella Pasquale, Perrino Francesco

Sommario



Introduzione

Il testing si propone di individuare le divergenze tra il comportamento definito nel modello di sistema e il comportamento reale del sistema implementato.

Pertanto, tale test servirà a provare il sistema e rilevare i problemi; quindi, a differenza delle altre attività svolte in precedenza (analisi, design e implementazione) volte a formare il sistema, il testing servirà a creare situazioni critiche all'interno dello stesso.

Per ogni feature del sistema verranno proposti dei Test Case: uno contenente input corretti e che soddisfi i requisiti dettati in questo documento, uno in cui i valori inseriti corrispondono a una classe di equivalenza non valida e uno in cui non si soddisfano i requisiti del Test Plan.

Relazioni con altri documenti

In questo paragrafo verranno descritte le relazioni con altri documenti già prodotti. Questo per far sì che si produca un testing coerente con le specifiche dedotte con l'analisi e il raffinamento dei requisiti e delle funzionalità.

[Documento 1] RAD- WarehouseConnect v3.pdf – Documento di Analisi dei Requisiti
[Documento 2] System Design Document.pdf – Documento di System Design

Relazioni con il documento di Analisi dei Requisiti

Il testing terrà conto delle specifiche espresse nel RAD. Innaniztutto, bisogna far presente che il testing terrà conto dei seguenti attori descritti nel suddetto documento:

- Utente (Commesso)
- Amministratore

Relazioni con il documento di System Design

Il testing dei componenti avverrà secondo la struttura di sistema specificata nel documento di System Design.

Panoramica del sistema

Il software Warehouse Connect è munito di un'architettura MVC:

- Model (Apache Derby DB) che gestisce i dati direttamente
- View (HTML, JSP) che visualizza i dati integrati in un'interfaccia familiare all'utente
- Controller (Java) che opera le transazioni tra le componenti precedenti, trasformando gli input delle viste in dati del modello

Possiamo dividere il sistema in sottosistemi definiti secondo le funzioni che ogni attore può svolgere. Il Model si estranea da tale suddivisione, in quanto conterrà la totalità dei dati e un insieme di operazioni CRUD volte alla lettura/scrittura sulla base di dati sottostante.

Il Test Plan si propone di:

- Specificare le operazioni di preparazione e di conduzione dei test
- Comunicare le ripartizioni delle attività che ciascuna parte responsabile è tenuta a svolgere per portare avanti il lavoro
- Definire le fonti necessarie per preparare la pianificazione

Il testing sarà guidato da alcuni principi elencati di seguito (che verranno seguite per quanto possibile) :

- ***Identificazione dello stato iniziale e finale del processo, per evitare che si cominci il processo senza tutte le informazioni necessarie o che si abbiano ritardi***
- ***Definire gli output attesi prima dei test, per evitare che certi risultati vengano assunti come corretti a priori***
- ***Il testing non sarà la verifica delle funzionalità, bensì la ricerca del maggior numero di errori***
- ***Definire una limitata quantità di test case di qualità, in modo da massimizzare il numero di errori minimizzando il lavoro***

Funzionalità testate

Il sistema permette agli utenti di interfacciarsi con un database; l'amministratore accede al sistema e gestisce il suddetto.

Tutte queste funzionalità richiedono meccanismi dinamici che permettono le diverse operazioni attraverso elaborazioni server. Tale dinamicità deve far parte del test tanto quanto tutto ciò che non richiede dinamicità non deve farne parte.

Feature da testare

Bisogna testare varie funzionalità a seconda dei componenti

View

Le funzionalità della View sono divise in sottolivelli riguardanti gli attori:

JSP Utente, di cui verranno testate:

- Funzione di accesso con livello superiore di credenziali (Login)
- Funzione di ricerca rapida (Menu)
- Funzione di ricerca avanzata tramite appositi campi
- Funzione di visualizzazione della pagina

JSP Amministratore, di cui verranno testate:

- Funzione di aggiunta e modifica del DB (Menu)
- Funzione di editing o eliminazione dal DB (Particolare della funzione di modifica)
- Funzione di ricerca avanzata tramite appositi campi

Controller

Le funzioni elencate per la View lavoreranno necessariamente e concorrentemente con le componenti del Controller. Di conseguenza verranno analizzate le medesime componenti dal punto di vista di quest'altro livello

Model

Verranno testate le funzioni di questo livello riguardanti la connessione e l'accesso alla base di dati

Pass/Fail Criteria

La fase di testing necessita di criteri formali per la determinazione del successo o dell'insuccesso di un determinato test. Un input supera il test se l'output ottenuto corrisponde ai risultati attesi

Pass criteria

I pass criteria determinano la correttezza del comportamento della componente testata. I pass criteria per la fase di testing del sistema sono raggruppati in due categorie principali

- Comportamento atteso
- Nessuna eccezione generata

Fail criteria

I fail criteria determinano la presenza di errori nel comportamento della componente testata. I fail criteria per la fase di testing del sistema sono raggruppati in due categorie principali

- Eccezione lanciata dalla componente
- Comportamento imprevisto

Approcci

Test di unità

Verifica di ogni modulo del programma. Si è usato il White Box Testing per le servlet.

Data la struttura complessa delle servlet, il White Box Testing permette di ricavare i casi di test in base all'implementazione ed alla struttura del programma. Si focalizza sulla struttura interna della componente. Indipendentemente dall'input, ogni stato nel modello dinamico dell'oggetto e ogni integrazione tra gli oggetti viene testata.

Vi sono 4 tipi di white-box testing:

- Statement Testing
    - Si testano i singoli statement
- Loop Testing:
    - Loop da eseguire esattamente una volta
    - Loop da eseguire più di una volta
- Path testing:
    - Assicurarsi che tutti i path nel programma siano eseguiti
- Branch Testing (Conditional Testing)
    - Assicurarsi che ogni possibile uscita da una condizione sia testata almeno una volta

Il criterio che ci indirizza alla scelta dei test case è il Path Testing.

- garanzia che tutti i cammini indipendenti entro un modulo, siano eseguiti almeno una volta
- esecuzione dei rami vero e falso per ogni decisione logica
- esecuzione di tutti i cicli
- esame della validità di tutte le strutture dati interne

Test d'integrazione

Si verifica la corretta integrazione tra un sottoinsieme dei moduli. Si è usato il Black Box Testing: l'obiettivo è quello di determinare se il programma fa quello che si suppone debba fare in base ai requisiti funzionali. Per la costruzione dei test ci si avvale solo delle specifiche dei requisiti, ignorando come sia stato realizzato il sistema al suo interno. Se ben condotto, riesce a determinare il manifestarsi di molti malfunzionamenti funzionali e può evidenziare il mancato rispetto di qualche requisito funzionale.

Sospensione e ripristino

La fase di testing può essere interrotta e ripresa più volte, se il fine ultimo è di rendere il sistema corretto ed ogni funzionalità completa. Di seguito sono riportati i criteri secondo cui è necessario sospendere la fase di testing e le modalità secondo cui deve poi essere ripresa.

Criteri di sospensione

La sospensione deve avvenire qualora un test abbia esito positivo, ovvero si è riscontrato un errore all'interno di una componente.

Criteri di ripristino

La ripresa della fase di testing avviene a partire dal test case che ne ha causato la sospensione, soltanto dopo che gli sviluppatori hanno corretto l'errore riscontrato.

Risorse

Per i test di unità e di integrazione si spera di adoperare JUnit. JUnit è un framework Open-Source. Fornisce un insieme di API che assistono gli sviluppatori nel creare in modo semplice e automatico dei test per il proprio software.

Permette di separare i test dal codice. Fornisce una struttura di asserzioni per confrontare i risultati attesi (oracolo) con quelli ottenuti. Fornisce un'interfaccia grafica molto elementare per la valutazione dei risultati dei test: barra verde se i test case in esecuzione falliscono, barra rossa se qualche test case della Test Suite fallisce.

- *"Test Case"*: singolo test su di una specifica funzionalità *(un metodo, una classe ecc.)*
- *"Test Suite"*: una collezione di test case che possono essere raggruppati in base a caratteristiche omogenee e possono essere eseguiti in blocco

Test case

I test case sono diretti a scoprire eventuali malfunzionamenti o comportamenti errati da parte del sistema. Per fare ciò è necessario testare il sistema su diversi istanze di input, ognuna diretta a testare comportamenti del sistema in determinate condizioni.

Il testing viene quindi strutturato in base alle funzionalità fornite dal sistema. Per ognuna di queste verranno fornite delle istanze di input che costituiscono i test case dei vari scenari. I test case potranno appartenere a tre diverse categorie di input in base al tipo di dati da cui sono costituiti relativamente allo use case in esame

- grammaticalmente non validi: fanno parte di questa categoria gli input che contengono caratteri non validi, suddivisi in classi di equivalenza
- non validi: fanno parte di questa categoria i dati che potrebbero essere grammaticalmente validi ma che nel contesto non lo sono, per esempio in un form che richiede l'inserimento di soli numeri, l'inserimento di stringhe non è contemplato
- validi: fanno parte di questa categoria i dati che sono grammaticalmente validi e hanno senso nel contesto in cui sono utilizzati

Login

Classi di equivalenza

INPUT	Login, Password	
CLASSI VALIDE	CE01	Stringa alfanumerica
CLASSI NON VALIDE	CE02	Stringa vuota

Considerazioni

- Sono presenti due account nel database: uno con login "admin" e password "admin", il secondo con login "admin2" e password "admin2"
- E' presente un sistema basico di sicurezza che divieta l'accesso a chi cerca di entrare nella schermata di gestione tramite la barra degli indirizzi

Amministratore

Classi di equivalenza (Aggiungi prodotto)

INPUT	Seriale, Nome, Descrizione breve, Descrizione lunga, Marca	
CLASSI VALIDE	CE03	Stringa alfanumerica
CLASSI NON VALIDE	CE04	Stringa vuota

INPUT	Tipologia (menu a tendina)	
CLASSI VALIDE	CE05	Selezione del tipo di prodotto
CLASSI NON VALIDE	CE06	-

INPUT	Prezzo	
CLASSI VALIDE	CE07	Numero double
CLASSI NON VALIDE	CE08	Campo vuoto, stringa alfanumerica, numero con virgola

INPUT	Quantità	
CLASSI VALIDE	CE09	Intero
CLASSI NON VALIDE	CE10	Stringa alfanumerica, campo vuoto, numero double o float

INPUT	Immagine	
CLASSI VALIDE	CE11	File immagine (JPG, PNG, BMP)
CLASSI NON VALIDE	CE12	File non contenente unicamente un’immagine

Classi di equivalenza (Modifica prodotto)

INPUT	Seriale, Nome, Descrizione breve, Descrizione lunga, Marca	
CLASSI VALIDE	CE13	Stringa alfanumerica
CLASSI NON VALIDE	CE14	Stringa vuota

INPUT	Tipologia (menu a tendina)	
CLASSI VALIDE	CE15	Selezione del tipo di prodotto
CLASSI NON VALIDE	CE16	-

INPUT	Prezzo	
CLASSI VALIDE	CE17	Numero double
CLASSI NON VALIDE	CE18	Campo vuoto, stringa alfanumerica, numero con virgola

INPUT	Quantità	
CLASSI VALIDE	CE19	Intero
CLASSI NON VALIDE	CE20	Stringa alfanumerica, campo vuoto, numero double o float

Classi di equivalenza (Ricerca per seriale)

INPUT	Seriale	
CLASSI VALIDE	CE21	Stringa alfanumerica
CLASSI NON VALIDE	CE22	Stringa vuota

Classi di equivalenza (Ricerca per nome e marca)

INPUT	Nome, Marca	
CLASSI VALIDE	CE23	Stringa alfanumerica
CLASSI NON VALIDE	CE24	Stringa vuota

Considerazioni

La funzione “Resoconto incassi” non può essere valutata se non con un software di terzi.
Sotto richiesta del committente del software, verrà predisposto il sistema con un sottosistema atto a comunicare con un software di terze parti specifico.

Utente (Commesso)

Classi di equivalenza (Ricerca per seriale)

INPUT	Seriale	
CLASSI VALIDE	CE21	Stringa alfanumerica
CLASSI NON VALIDE	CE22	Stringa vuota

Classi di equivalenza (Ricerca per nome e marca)

INPUT	Nome, Marca	
CLASSI VALIDE	CE23	Stringa alfanumerica
CLASSI NON VALIDE	CE24	Stringa vuota

Testing schedule

Di seguito sono elencate la gestione dei rischi, l'organizzazione e la schedulazione delle attività durante il periodo prestabilito per il testing.

Gestione dei rischi

I possibili rischi generati dalle attività di testing sono stati minimizzati diminuendo le componenti da testare. Inoltre, effettuando un testing di tipo funzionale viene limitato lo sviluppo di stub e driver per il testing delle singole componenti e quindi l'introduzione di nuovi errori nel nuovo codice di cui si sarebbero composti.

D'altro canto il testing di funzionalità rallenta l'individuazione di errori qualora un caso di test avesse esito positivo, poiché l'utilizzo di più componenti per il test di una singola funzionalità estranea dall'ipotesi di totale correttezza di ogni componente interessata.

Il risultato atteso è di riscontrare al più un errore per funzionalità: le componenti interagiscono fra loro e la funzionalità viene eseguita ma non in modo completamente corretto.

Qualora la fase di testing evidenziasse un numero di errori maggiore rispetto alla media attesa, viene pianificato un impegno maggiore dei membri del team sulle attività di testing e in casi estremi l'abbandono delle altre attività finché errori gravi (funzionalità non corretta, risultati errati, modifiche apportate in modo errato) non vengano risolti.

Organizzazione delle attività

Le attività di testing devono svolgersi sulle singole funzionalità divise nei livelli di suddivisione del sistema, rispettando le direttive indicate dal documento di system design.

Schedulazione delle attività

Le attività di testing sono state svolte in questo periodo:

- Inizio testing: 2 gennaio 2017
- Fine testing: 14 gennaio 2017