**Test Plan Template**

**Purpose**

**Audience**

**Template**

|  |  |
| --- | --- |
| **Outline** | **Description** |
| **1. Introduction** | *Section 1 of the test plan describes the objectives and extent of the tests. The goal is to provide a framework that can be used by managers and testers to plan and execute the necessary tests in a timely and cost-effective manner.* |
|  | |
| **2. Relationship to other documents** | *Section 2 explains the relationship of the test plan to the other documents produced during the development effort such as the RAD, SDD, and ODD (Object Design Document). It explains how all the tests are related to the functional and nonfunctional requirements, as well as to the system design stated in the respective documents. If necessary, this section introduces a naming scheme to establish the correspondence between requirements and tests.* |
|  | |
| **3. System overview** | *Section 3, focusing on the structural aspects of testing, provides an overview of the system in terms of the components that are tested during the unit test. The granularity of components and their dependencies are defined in this section.* |
|  | |
| **4. Features to be tested/not to be tested** | *Section 4, focusing on the functional aspects of testing, identifies all features and combinations of features to be tested. It also describes all those features that are not to be tested and the reasons for not testing them.* |
|  | |
| **5. Pass/Fail criteria** | *Section 5 specifies generic pass/fail criteria for the tests covered in this plan. They are supplemented by pass/fail criteria in the test design specification. Note that fail in the IEEE standard terminology means successful test in our terminology.* |
|  | |
| **6. Approach** | *Section 6 describes the general approach to the testing process. It discusses the reasons for the selected integration testing strategy. Different strategies are often needed to test different parts of the system. A UML class diagram can be used to illustrate the dependencies between the individual tests and their involvement in the integration tests.* |
|  | |
| **7. Suspension and resumption** | *Section 7 specifies the criteria for suspending the testing on the test items associated with the plan. It also specifies the test activities that must be repeated when testing is resumed.* |
|  | |
| **8. Testing materials (hardware/software requirements)** | *Section 8 identifies the resources that are needed for testing. This should include the physical characteristics of the facilities, including the hardware, software, special test tools, and other resources needed (office space, etc.) to support the tests.* |
|  | |
| **9. Test cases** | *Section 9, the core of the test plan, lists the test cases that are used during testing. Each test case is described in detail in a separate Test Case Specification document. Each execution of these tests will be documented in a Test Incident Report document. We describe these documents in more details later in this section.* |
|  | |
| **10. Testing schedule** | *Section 10 of the test plan covers responsibilities, staffing and training needs, risks and contingencies, and the test schedule.* |

Progetto di Ingegneria del software

Anno accademico 2018/2019



*Carcheck*



|  |  |  |
| --- | --- | --- |
| **Tabella Componenti** | | |
| **Capriglione** | **Francesco** |  |
| **D’Auria** | **Aldo** |  |
| **De Falco** | **Daniele** | 0512104666 |
| **Iacovazzo** | **Giovanni** | 0512104774 |

**Test Plan**

**1.Introduzione**

Questo documento pianifica il testing sul sistema. Qui vengono definiti una serie di concetti come la caratteristiche da testare/non testare oppure i criteri che portano a definire un successo/fallimento nell’attività di testing di un certo componente.

Il sistema verrà messo in relazione con le fasi precedenti e verrà data una descrizione del sistema che è stata progettato e implementato. Tale descrizione riguarderà la definizione dello stile architetturale utilizzato, dei vari packages e classi realizzate e delle strutture utilizzate.

**2. Relazioni ad altri documenti**

Il Test Plan ha molti riferimenti nei documenti precedenti. Tutto ciò che è stato definito sotto forma di requisiti, trasformato poi in progetto ed implementato deve essere testato. La pianificazione di tale testing è presente in questo documento che è in stretta relazione con il RAD, L’SDD e L’ODD

**3. Panoramica del sistema.**

Lo stile architetturale seguito da Carcheck è il MVC, acronimo che sta per Model-View-Control. All’interno del progetto sono stati definiti diversi package che permettono di definire tale struttura. Il view viene rappresentato dalle pagine JSP che mostrano all’utente il contenuto del sistema. Il model e il control sono contenuti nei package:

Control

* It.carcheck.control
* It.carcheck.control.exception
* It.carcheck.control.interfaces
* It.carcheck.control.request
* It.carcheck.control.service

Model

* It.carcheck.model
* It.carcheck.model.bean
* It.carcheck.model.bean.enums
* It.carcheck.model.interfaces

Per l’accesso ai dati persistenti, e quindi database, è stato introdotto un CRUD chiamato “FastCrud”. Si tratta di un ORM (Object Relational Mapping) capace di mappare le relazioni tra le classi e gli oggetti del database eseguendo operazioni di Insert, delete, update ecc…

Tale libreria permette di ottimizzare l’inserimento e gestione dei dati all’interno dei bean. Essa permette di gestire un database relazionale come se fosse un database ad oggetti, passando direttamente gli oggetti desiderati senza definire nessuna tabella o costrutto relazionale.

**4. Funzionalità da testare/non testare**

Nella definizione di cosa andare a testare/non testare abbiamo dovuto effettuare una serie di scelte.

Innanzitutto, i vari metodi getters() e setters() per ovvie ragioni non verranno testati.

Le principali componenti su cui si concentrerà il testing saranno i manager, ossia le classi responsabili dell’accesso ai dati utili all’applicazione.

**5. Criteri di successo/di fail**

Il lavoro svolto dal team sarà quello di andare a raggruppare tra loro dati omogenei.

Il testing avrà successo se si rileva una differenza tra l’output della componente testata e l’oracolo. In questi casi si andrà ad analizzare l’incident stabilendo le cause del failure e procedendo alle opportune correzioni.

**6. Approccio**

Per quanto riguarda l’unit testing, abbiamo deciso di utilizzare la strategia detta “Black-Box”. Essa consiste nel fornire un’input alla componente testata ed andare a confrontare l’output reale con l’oracolo ( l’output atteso ). L’input viene preso da un insieme specifico, spesso partizionato in sottoinsiemi significativi.

Questa tecnica non va ad analizzare la struttura interna della componente. Questo significa che non si ha interesse a conoscere il modo con cui essa è implementata, a conoscere i vari flussi di esecuzione (branch, condizioni if/else ecc…). L’unico suo interesse è quello di conoscere input e output.

Se l’output prodotto dalla componente si discosta dall’oracolo, allora il testing ha avuto successo ed è stato individuato un malfunzionamento. Questo malfunzionamento deve poi essere corretto e il testing viene ripetuto ancora.

**7. Criteri di sospensione e di ripresa**

Le attività di testing verranno “sospese” al raggiungimento di un code coverage di circa il 70%. Tale attività possono però essere sospese anche prima, in situazioni in cui ci si accorga di essere in ritardo con i tempi di consegna del progetto.

Le attività di testing potranno essere “riprese” a causa di modifiche al progetto ( modifica del codice, aggiunta di nuove funzionalità, modifica interfaccia grafica… ) oppure a causa di correzioni dovute alla scoperta precedente di malfunzionamenti nel sistema.

Il sistema o componente in questione verrà testata nuovamente tramite testing di regressione.

**8. Testing materials(Hardware/software)**

Per le attività di testing sono necessari:

**Hardware**

* Elaboratore su cui eseguire il software

**Software**

* Ecplise IDE
* MySql DBMS
* Selenium
* JUnit

**9.Test cases**

**10.Test schedule**

Il training necessario per tali attività consiste nell’apprendimento dei vari software da utilizzare ( JUnit,Selenium… ). Trattandosi di un progetto universitario non è presente un budget, ma tale training arricchisce la formazione e il bagaglio culturale dei componenti del gruppo.

Tra i rischi da considerare il principale è quello che il tempo per imparare ad utilizzare tali tools rallenti il lavoro effettivo al progetto, con conseguente aumento della probabilità di ritardare la consegna del progetto.