

Test Plan

Pr. D. - Programmazione Didattica

|  |  |
| --- | --- |
| **Riferimento** |  |
| **Versione** | 1.1 |
| **Data** |  |
| **Destinatario** | Top Management |
| **Presentato da** | Carpentiero Edoardo, Della Greca Attilio, Prisco Antonio |
| **Approvato da** |  |

Sommario

[1. Introduzione 2](#_Toc48661699)

[2. Relazione con altri documenti 2](#_Toc48661700)

[3. Panoramica del sistema 2](#_Toc48661701)

[4. Funzionalità da testare 3](#_Toc48661702)

[5. Criteri Pass/Failed 4](#_Toc48661703)

[6. Approccio 4](#_Toc48661704)

[6.1 Testing di unità 4](#_Toc48661705)

[6.2 Testing di integrazione 5](#_Toc48661706)

[6.3 Testing di sistema 5](#_Toc48661707)

[6.4 Testing di regressione 5](#_Toc48661708)

[7. Testing pre-modifica 5](#_Toc48661709)

[8. Sospensione e ripristino 6](#_Toc48661710)

[8. Strumenti per il testing 6](#_Toc48661711)

# Introduzione

Il **testing** è una tecnica di “fault detection” il cui scopo è appunto quello di rilevare errori in maniera pianificata all'interno del codice prodotto. L'obiettivo è, quindi, di evitare che questi errori si presentino in fase di utilizzo del sistema da parte dell'utente finale.

L’attività di testing è una componente fondamentale nello sviluppo di un prodotto software efficiente e funzionale perché permette di rilevare possibili errori all’interno del sistema, che potrebbero verificarsi in seguito alla scrittura del codice sorgente.

In questo documento vengono pianificate e descritte in modo dettagliato le attività di testing che il team si prepara ad effettuare riguardo alle attività di gestione del carico didattico.

# Relazione con altri documenti

I test cases sono basati sulle funzionalità individuate nel documento *Master Document* **Prd.D.\_MD\_Vers1.2.** La combinazione e il numero dei test case è riportato nel documento **Prd.D.\_CP\_Vers1.0.** I test case individuati per ogni funzionalità sono riportati nel documento Test Case Specification- **Prd.D.\_TCS\_Vers1.3**. I log di esecuzione dei test case sono riportati nel documento *Test Execution Report* - **Prd.D.\_TER\_v1.1 -** e gli eventuali errori riscontrati durante il testing sono riportati nel documento *Test Incident Report*.

# Panoramica del sistema

Il sistema risulta essere suddiviso in tre sottosistemi in cui ognuno di essi è composto da moduli che si occupano di svolgere determinate attività:

● **Sottosistema Presidente:**

➢ Gestione autenticazione

➢ Gestione docenti

➢ Gestione insegnamento

➢ Gestione ordinamento

➢ Gestione regolamento

➢ Gestione programmazione didattica

➢ **Gestione carico didattico (new)**

➢ Gestione account Docenti

● **Sottosistema Docente:**

➢ Visualizza docenti

➢ Visualizza insegnamento

➢ Visualizza ordinamento

➢ Visualizza regolamento

➢ Visualizza programmazione didattica

➢ Visualizza carico didattico personale

➢  **Gestione carico didattico (new)**

➢ Gestione account personale

● **Sottosistema Utente Generico**

➢ Visualizza docenti

➢ Visualizza insegnamento

➢ Visualizza ordinamento

➢ Visualizza regolamento

➢ Visualizza programmazione didattica

# Funzionalità da testare

Le funzionalità considerate nel testing riguardano la gestione del carico didattico presenti nel sottosistema Presidente e Docente. Le funzionalità da testare sono le seguenti:

|  |  |
| --- | --- |
| FUNZIONALITÀ’ | DESCRIZIONE |
| **Visualizza insegnamenti proposti** | Consente all’utente di visualizzare gli insegnamenti proposti e i relativi stati |
| **Visualizza insegnamenti disponibili** | Consente all’utente di visualizzare gli insegnamenti disponibili |
| **Proponi Insegnamento disponibile** | Consente all’utente di proporre un insegnamento disponibile |
| **Valuta insegnamento proposto** | Consente all’utente di cambiare lo stato dell’insegnamento proposto |
| **Visualizza status carico didattico dei docenti** | Consente all’utente di visualizzare lo status del carico didattico dei docenti |
| **Visualizza informazioni insegnamento proposto o disponibile** | Consente all’utente di visualizzare le informazioni dell’insegnamento proposto o disponibile |

# Criteri Pass/Failed

Lo scopo del testing è quello di dimostrare la presenza di faults (errori), ovvero comportamenti non ottimali da parte del sistema.

Il test avrà *successo* se dato un input restituirà i risultati attesi, cioè quelli specificati dall’oracolo nel test case. Con l’attività di testing infatti, si cercherà di “rompere” il sistema realizzato. Quando invece la funzionalità in esame accetterà un input previsto e restituirà anche l’output previsto, allora il test sarà *fallito*.

# Approccio

La tecnica di testing utilizzata si basa sull’approccio di tipo BlackBox.

Sarà praticato uno **unit testing**, per verificare il corretto funzionamento di tutte le singole unità di codice implementate in determinate condizioni. Sarà quindi testata ogni nuova classe introdotta ed ogni metodo appartenente ad essa.

Dopo la fase di unit testing seguirà l’**integration testing**, con l’obiettivo di collaudare l’interazione tra le nuove componenti e quelle pre-esistenti. Dopodiché verrà effettuato il **system testing** per dimostrare che il sistema soddisfa i requisiti definiti dalla change request, e infine il **regression testing** per verificare se le funzionalità preesistenti funzionano correttamente e che i nuovi cambiamenti non abbiano introdotto nuovi bug. Al fine di minimizzare il numero dei test cases, è stato utilizzato il *category partition*.

## 6.1 Testing di unità

Durante questa fase l’attenzione viene focalizzata sulle nuove componenti implementate relative alla gestione del carico didattico. In particolare, vengono ricercate condizioni di fallimento isolando le componenti tramite utilizzo di **moduli Stub** (implementazione completa o parziale di una componente dalla quale dipende la componente sotto testing) e **moduli Driver** (implementazione completa o parziale di una componente che fa uso della componente sotto testing). Tali test vengono applicati dopo aver isolato ogni singola funzionalità della gestione del carico didattico da testare in modo da verificare una completa validità delle componenti e tener traccia dei possibili errori che vengono riscontrati. Se si verifica un errore con dei risultati inattesi si interviene in maniera tempestiva sulla componente in modo da renderla correttamente funzionante e procedere con le fasi di testing successive.

## 6.2 Testing di integrazione

La modalità utilizzata nel testing di integrazione si basa sul testing incrementale, dove le componenti che precedentemente erano state isolate e testate singolarmente vengono integrate man mano. La tecnica di approccio a questo tipo di operazione prevede l’uso combinato di strategie Bottom-Up e Top-Down, in particolare una strategia Sandwich Testing. Questo approccio mira principalmente a focalizzare la ricerca di errori nelle interfacce di comunicazione tra sottosistemi.

## 6.3 Testing di sistema

Tale testing permette di verificare la corretta adesione dei requisiti dettati dalla change request con il resto del sistema. L’approccio utilizzato sarà di tipo Blackbox. Sarà verificata la qualità e l’affidabilità del prodotto software, in modo che possa essere utilizzato dall’utente finale senza margini di errori.

## 6.4 Testing di regressione

Lo scopo è quello verificare se le funzionalità presistenti del sistema mantengano lo stesso comportamento anche dopo che la change request è stata implementata. Al fine di effettuare tale verifica bisogna rieseguire i test case su tali funzionalità in modo da consentire se quest’ultime presentano o meno malfunzionamenti prima della modifica.

# Testing pre-modifica

Al fine di garantire il corretto funzionamento dell’intera piattaforma in cui viene integrato il modulo richiesto dalla change request, sono stati effettuati dei test di sistema tenendo in considerazione i test log presenti nel *Test Execution Report.* Data l’impossibilità di rieseguire in modo automatico i test case documentati e di evitare di sviluppare i rispettivi test case code oriented, per verificare il funzionamento pre e post implementazione della change request è stato deciso effettuare un test di sistema automatizzato, attraverso *Selenium,* sulle componenti considerate nell’impact analysis prima di effettuare attività di manutenzione. Maggiore attenzione sarà dedicata alle funzionalità relative al modulo **Gestione Programmazione Didattica**, la quale è risultata essere la principale componente ad essere impattata dalla change request.

# Sospensione e ripristino

Il testing sarà **sospeso** quando tutti i test case individuati sono stati eseguiti con successo. Nel momento in cui il testing rivelerà un errore si dovrà passare alla fase di correzione che dovrà essere eseguita da una ripetizione dell’intero processo di testing per rilevare la presenza di eventuali errori introdotti dalle correzioni stesse. Tutto ciò deve essere in accordo con i tempi di sviluppo previsti, tenendo sempre conto dei costi dell’attività di testing.

La **ripresa** del test avviene soltanto quando tutti i problemi relativi alla sospensione dello stesso sono stati risolti. L’attività di testing riprenderà a partire dal test case che ha causato la sospensione.

# Strumenti per il testing

La dotazione software richiesta per l’attività di testing è la seguente:

* ***PHP Unit,*** integrato nell’IDE *PHP STORM*, utilizzato per lo sviluppo in locale del software. Tale strumento è impiegato sia nel testing di unità che di integrazione.
* ***Selenium,*** utilizzatoper poter effettuare il testing di sistema.
* Un qualunque browser Web tra i vari *Firefox, Google Chrome, Internet Explorer, Safari, Opera, ecc*.
* **Intellij IDEA Coverage Function**: Questa funzionalità integrata all’interno dell’IDE permette di controllare la percentuale di metodi, classi, linee di codice coperte dai casi test creati.