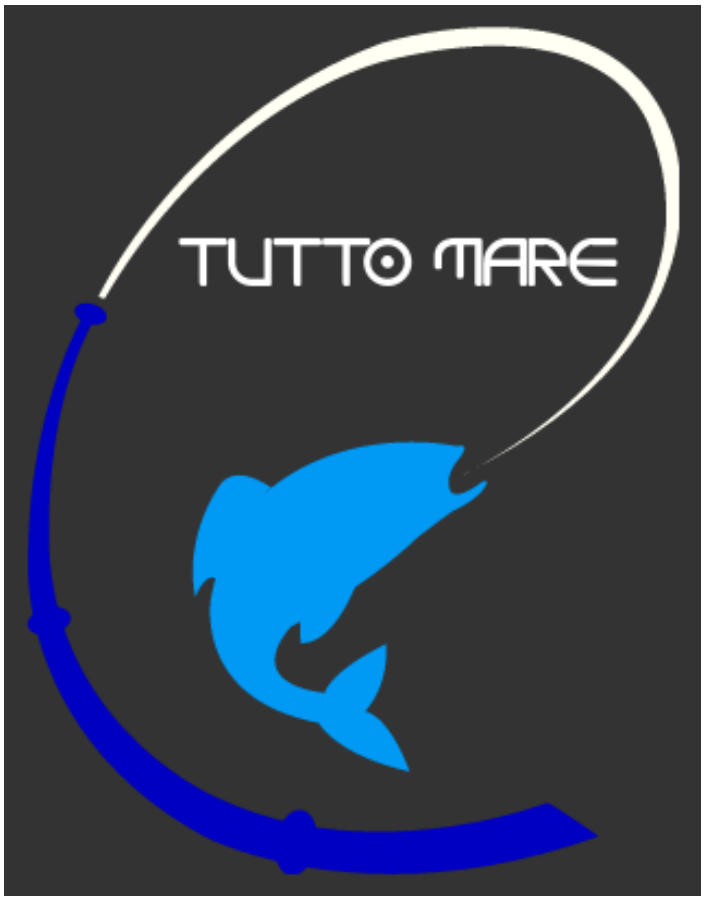
Test Plan



Di:

Sara Guerriero 0512103956
Torre Francesca 0512102366

Indice degli argomenti:

1. 0 Introduzione

Il testing è una tecnica che ha lo scopo di rompere il sistema, cioè quello di identificare errori con l'obiettivo di evitare che si presentino durante l'utilizzo del software da parte dell'utente, e non quello di garantire che il codice sia corretto. In altre parole si cerca di creare quanti più fallimenti o stati erronei in modo pianificato in modo da consentire agli sviluppati di correggerli prima che il prodotto sia consegnato al cliente. Lo scopo di questo documento è quello quindi di descrivere e pianificare l'attività di testing su determinate funzionalità con l'obiettivo di individuare errori mediante i casi di test.

2. 0 Riferimenti

Per verificare la corretta integrazione dei sottosistemi del sistema sono stati predisposti dei test case basati sui vari sottosistemi proposti in fase di System Design.

3. 0 Test di integrazione

Il testing di integrazione rappresenta una delle fasi di testing più importanti, in quanto consiste
nella verifica delle interazioni tra due o più componenti.
L'obiettivo del testing consiste nella verifica della corretta interazione tra le componenti e il
rispetto delle interfacce.
Questo documento ha il compito di identificare la strategia di testing di integrazione per il sistema.

3.1 Componenti da testare

La scelta delle componenti da testare segue la decisione di eseguire la strategia di testing Bottom-up.
Per il testing di integrazione, per la natura del linguaggio scelto, si andrà ad effettuare il testing all'interno di ogni sottosistema. Si testerà la "servlet" che utilizza una determinata classe all'interno di un sottosistema.
Il testing di integrazione si andrà a fare tramite il tool JUnit.

4. 0 Unit testing

Lo unit testing, come lo Integration testing, si focalizza su una singola componente usando un test stub e test driver, e esercitando tale componente tramite un test case.
L'obiettivo dello unit testing è che ogni sottosistema sia codificato e svolge la funzionalità prevista. Le unità candidate per il test sono prese dal modello a oggetti e dalla decomposizione dei sottosistemi.
I sottosistemi devono essere testati dopo che ogni classe e oggetto del sottosistema è stato testato individualmente.
Si andrà ad effettuare lo unit testing su una determinata componente solo dopo che le classi sono "stabili", cioè sono state realizzate tutte le funzionalità per quella determinata classe.
Se è una classe di servizio il testing di unità non si va ad effettuare, magari ogni sotto-team si può scegliere le proprie classi da testare.
Utilizzeremo un approccio white-box dove ci 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.
I test case si baseranno sulla conoscenza dell'implementazione delle componenti.

5. 0 Test Case

Le varie fasi di testing necessiteranno ognuna di test case utili ad individuare errori ed anomalie sia analizzando il codice che provando la sua esecuzione. Si sono individuate varie classi di equivalenza per ogni input che possa essere immesso per l'utilizzo di una o più componenti. In tal modo, è possibile sviluppare test case con input delle tipologie identificate per testare una o più unità.

6.0 Pass/Fail Criteri

Il testing ha successo se l'output osservato è diverso dall'output atteso: ciò significa che parliamo di SUCCESSO se il test rileva un failure.

In tal caso questa verrà analizzata e, se legata ad un fault, si procederà alla sua correzione.
Sarà infine iterata la fase di testing per verificare che la modifica non abbia impattato su altri componenti del sistema.

Viceversa parliamo di FALLIMENTO se il test non riesce ad individuare un errore.