 Università degli studi di Salerno  
Corso di Ingegneria del Software

**RooManageR**

Test Plan Document

Versione 1.0.0



06/04/2016

Coordinatore del progetto:

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Andrea De Lucia |  |

Partecipanti:

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Gargiulo Emanuele | 0512102244 |
| Izzo Giandomenico | 0512102292 |

|  |
| --- |
| Scritto da: Gargiulo Emanuele, Izzo Giandomenico |

Sommario

[1. Introduzione 4](#_Toc447709040)

[2. Relazioni con altri documenti 4](#_Toc447709041)

[2.1 Relazioni con il documento di Analisi dei Requisiti 4](#_Toc447709042)

[2.2 Relazioni con il documento di System Design 5](#_Toc447709043)

[2.3 Relazioni con il documento di Object Design 5](#_Toc447709044)

[3. Panoramica del Sistema Proposto 5](#_Toc447709045)

[4. Criteri di successo e insuccesso 6](#_Toc447709046)

[5. Approccio 6](#_Toc447709047)

[5.1 Linee guida 7](#_Toc447709048)

[5.2 Unit Testing 7](#_Toc447709049)

[5.3 Integration Testing 8](#_Toc447709050)

[5.4 System Testing 8](#_Toc447709051)

[6. Sospensione e ripristino 8](#_Toc447709052)

[6.1 Sospensione 8](#_Toc447709053)

[6.2 Ripristino 8](#_Toc447709054)

[7. Materiale per il testing 9](#_Toc447709055)

# Introduzione

Il testing consiste nel trovare le differenze tra il comportamento atteso specificato attraverso il modello del sistema e il comportamento osservato dal sistema implementato.

Lo scopo di tale attività è quello di eseguire il sistema e rilevare eventuali problemi.

È particolarmente importante, dunque, massimizzare il numero degli errori riscontrati in modo tale che questi ultimi possano essere successivamente corretti. Inoltre, è di fondamentale importanza definire casi d’uso in modo da testare le funzionalità del sistema.

Come si può ben capire tale attività va in contrasto con quelle svolte in precedenza: analisi, design e implementazione sono attività costruttive mentre il testing risulta essere un’attività distruttiva in quanto tenta di “rompere il sistema”.

Questo documento descrive il piano di testing relativo al sistema RooManager.

Esso supporta i seguenti obiettivi:

* Identificare i componenti software che dovrebbero essere testati;
* Elencare i requisiti (raccomandati) da testare;
* Consigliare e descrivere le strategie di test da impiegare;
* Identificare le risorse richieste e fornire una stima dei costi;
* Definire deliverable e soggetti responsabili.

# Relazioni con altri documenti

## 2.1 Relazioni con il documento di Analisi dei Requisiti

I test dovranno tenere conto delle specifiche espresse nel RAD

Inoltre sarà data rilevanza anche ai requisiti non funzionali ed ai vari modelli prodotti in fase di analisi dei requisiti

1. Requisiti Funzionali, RAD
2. Requisiti non Funzionali, RAD
3. Inserisci/Carica Anagrafica
4. Inserisci/Carica Struttura
5. Modifica struttura
6. Rimuovi Struttura
7. Modifica Anagrafica
8. Inizio visita
9. Inizio Permanenza
10. Conclusione di una permanenza
11. Conclusione di una visita
12. Aggiungi/Rimuovi Portinaio
13. Modifica Stanza
14. Aggiungi Stanza
15. Rimuovi Stanza

## 2.2 Relazioni con il documento di System Design

Il testing dovrà garantire la coerenza tra il software e gli obiettivi di design definiti all’interno del System Design Document.

In questo documento si andranno ad analizzare i conflitti e le inconsistenze tra le componenti del sistema e i suoi obiettivi.

Si analizzerà, quindi, l’intera struttura del sistema al fine di comprendere quelle che sono le differenze esistenti tra l’effettivo comportamento del sistema e quello atteso dall’utente.

## 2.3 Relazioni con il documento di Object Design

La fase di testing dovrà considerare il contenuto del documento di Object Design, in quanto quest’ultimo rappresenta la base per la realizzazione dell’implementazione, fondamentale per il testing.

# Panoramica del Sistema Proposto

L’obiettivo principale di RooManager è quello di fornire un sistema unico che permetta di controllare tramite un’unica interfaccia tutti gli aspetti relativi alla gestione di una o più strutture

Dopo essere stato suddiviso in sottosistemi e ogni sottosistema in componenti, ognuno di questi sottosistemi e i relativi componenti saranno testati indipendentemente.

Nella fase di Integration testing saranno testate le funzionalità in modo indipendente.

Nella successiva fase di System Testing verrà testato l’intero sistema andando a mettere insieme tutte le sue componenti e verificando la corretta interazione tra esse.

Le caratteristiche da testare per il corretto controllo del funzionamento di ciascuna funzionalità saranno:

* Robustezza: Il sistema dovrà garantire affidabilità e quindi robustezza agli input invalidi forniti dagli utenti. Esso, quindi, elaborerà i dati forniti dall’utente e nel caso in cui gli stessi siano errati lancerà un messaggio di errore per avvisare l’utente che i dati inseriti non sono validi.
* Usabilità: analisi di ogni forma di interazione corrisposta da messaggi di aiuto (in caso di errore);
* Affidabilità: Il sistema dovrà garantire affidabilità sulle funzionalità implementate. Esso verrà sviluppato in modo tale da garantire un controllo rigido dei dati immessi dall’utente:
  + Mancato inserimento di dati fondamentali per l’operazione;
  + Rimozione di una struttura in cui è ancora in corso una permanenza o dalla quale non sono state rimosse tutte le strutture
* Disponibilità: Una volta che il sistema sarà rilasciato sarà disponibile ogni volta che l’utente ne richiederà l’utilizzo

# Criteri di successo e insuccesso

In generale, un test case ha “SUCCESSO” quando il risultato reale non corrisponde al risultato atteso, altrimenti il test case “FALLISCE”, e in questo caso il risultato osservato sarà uguale all’oracolo.

Il risultato atteso è definito separatamente per ogni test case.

# Approccio

Questa sezione descriverà il generale approccio al processso di testing e la strategia di Integration testing utilizzata.

Le tipologie di testing che verranno svolte per testare RooManager saranno:

* Unit Testing: un test che coinvolge una singola funzionalità del sistema e verrà svolto indipendentemente dalle altre unità;
* Integration Testing: un test che coinvolgerà più unità che interagiscono tra loro e che vengono integrate per essere testate insieme;
* System Testing: un test nel quale andremo a coinvolgere l’intero sistema e che quindi testa il software nella sua interezza, analizzando le interazioni tra tutti i gruppi di unità e le funzionalità. -  **DEL SYSTEM TESTING VERRA’ EFFETTUATO SOLO IL FUNCTIONAL TESTING, PER RAGIONI DI TEMPO**

Va segnalato che alcune restrizioni sui dati inseribili non sono replicabili in fase di testing in quanto sono parte integrante di come funziona il programma da un punto di vista grafico.

Volendo fare un esempio: Quando si testa la creazione di un anagrafica, è possibile inserire un qualunque valore come Nazionalità, in quanto l’entità anagrafica accetta come valore nazionalità una stringa.   
Dal programma, invece, l’elenco delle nazionalità (non modificabile) verrà caricato direttamente dal database e sarà selezionabile esclusivamente tramite una Jcombobox.

Stessa cosa, in fase di creazione di una stanza, dal programma è possibile creare una stanza che appartenza soltanto a una struttura già esistente, dato che l’utente seleziona graficamente la struttura da una lista, mentre in fase di testing è possibile scrivere una qualunque stringa nel campo “Nome struttura”.

In oltre, diverse restrizioni dunque sono volutamente gestite tramite interfacce grafiche bloccanti (JDialog) che non permettono di aggirare formati dati sbagliati, ma questo tipo di errore sarebbe riscontrabile in fase di testing ed è stato pertanto deciso di ignorarli.

Pertanto, il System Testing sarà ripetuto avviando il programma simulandone un uso reale, in quanto è stato ritenuto che sia l’unico modo per individuare errori che altrimenti potrebbero sfuggire.

## 5.1 Linee guida

L’obbiettivo sarà quello di determinare quanto il software sarà conforme alla specifica dei requisiti funzionali.

Per la costruzione dei test ci si avvarrà solo della specifica dei requisiti, ignorando completamente come sia stato realizzato il sistema al suo interno.

Se ben condotto, il testing, riuscirà a determinare il manifestarsi di molti malfunzionamenti relativi alle funzionalità e si potrà evidenziare il mancato rispetto di qualche requisito funzionale.

## 5.2 Unit Testing

Sarà effettuato un testing per ogni unità del sistema individuata.

Se il test produrrà un risultato diverso da quello atteso (oracolo), esso verrà segnalato al team di implementazione in modo tale da permettere loro di effettuare la manutenzione.

## 5.3 Integration Testing

Dopo aver testato le singole funzionalità del sistema, verrà eseguito il testing su gruppi di componenti sarà considerato valido solo quando essi soddisferanno i vincoli e le interfacce dettate dalle specifiche dell’ Object Design e dalle specifiche del System Design.

In generale, dato che l’esecuzione di ogni funzionalità parte sempre dall’interazione dell’utente con il sistema, è stata scelta la tipologia di sviluppo top-down.

Questa strategia è stata scelta perché si è notato che il numero di stub richiesti per seguire tale strategia è molto basso, a differenza del numero di stub e di driver che sarebbero necessari per una strattegia di un’altra tipologia.

Se il test produrrà un risultato diverso da quello atteso (oracolo), esso verrà segnalato al team di implementazione in modo tale da permettere loro di effettuare la manutenzione.

## 5.4 System Testing

Il testing dell’intero sistema utilizzerà i requisiti funzionali, non funzionali e gli use case per determinare il successo/insuccesso dello stesso.

Se il test produrrà un risultato diverso da quello atteso (oracolo), esso verrà segnalato al team di implementazione in modo tale da permettere loro di effettuare la manutenzione.

# Sospensione e ripristino

Nel testare RooManager, potranno esserci casi in cui bisogna sospendere il testing e casi in cui bisogna ripristinarlo.

## 6.1 Sospensione

Durante le fasi di Integration Testing e System Testing, se verranno riscontrati errori o fallimenti, il processo verrà interrotto per tutte le componenti.

Durante la sospensione del testing si passerà alla fase di correzione da parte degli sviluppatori.

## 6.2 Ripristino

Il ripristino del testing di un gruppo di componenti del software sarà svolto quando gli errori riscontrati saranno corretti.

Inoltre, ad ogni ripristino del testing, saranno effettuati nuovi test su ciò che è stato corretto (regression testing), per verificare la correttezza dell’intero gruppo di componenti.

Dopo questa verifica, saranno svolti diversi test sul gruppo di componenti corretti oppure si passerà al testing di un altro insieme di unità dell’applicazione.

# Materiale per il testing

Le risorse hardware utilizzate per effettuare il testing saranno:

* Computer portatili per lavorare sui vari documenti del testing ed analizzare e testare il sistema, con diversi sistemi operativi

I software che saranno utilizzati per effettuare il testing saranno:

* Microsoft Office 2016, come applicazione per la scrittura di test case e test plan
* Netbeans IDE per la scrittura dei test case

# Test Cases

## TC UC 1

|  |  |
| --- | --- |
| ID | TC UC 1 |
| Nome | Inserisci / Carica Anagrafica |
| Obiettivo | Testare le funzionalità di aggiunta e caricamento di un’anagrafica dal database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene creata un’anagrafica 2. Vengono assegnati tutti i dati relativi a un’anagrafica 3. L’anagrafica viene inserita nel database 4. L’anagrafica viene cercata nel database per verificare se è stata inserita correttamente |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.2 TC UC 2

|  |  |
| --- | --- |
| ID | TC UC 2 |
| Nome | Modifica Anagrafica |
| Obiettivo | Testare le funzionalità di modifica di un’anagrafica dal database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene cercata un’anagrafica nel database 2. Vengono modificati i dati dell’anagrafica 3. Viene cercata nuovamente l’anagrafica nel database per verificare se i cambiamenti sono stati apportati correttamente |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.3 TC UC 3

|  |  |
| --- | --- |
| ID | TC UC 3 |
| Nome | Inserisci / Carica Struttura |
| Obiettivo | Testare le funzionalità di aggiunta e caricamento di una struttura dal database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene creata una struttura 2. Vengono assegnati tutti i dati relativi a una struttura 3. La struttura viene inserita nel database 4. La struttura viene cercata nel database per verificare se è stata inserita correttamente |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.4 TC UC 4

|  |  |
| --- | --- |
| ID | TC UC 4 |
| Nome | Inserisci / Carica Stanza |
| Obiettivo | Testare le funzionalità di aggiunta e caricamento di una stanza dal database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene creata una stanza 2. Vengono assegnati tutti i dati relativi a una stanza 3. La stanza viene inserita nel database 4. La stanza viene cercata nel database per verificare se è stata inserita correttamente |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.5 TC UC 5

|  |  |
| --- | --- |
| ID | TC UC 5 |
| Nome | Modifica Stanza |
| Obiettivo | Testare la funzionalità di modifica di una stanza nel database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene cercata una stanza nel database 2. Vengono modificati i dati della stanza 3. Viene cercata nuovamente la stanza nel database per verificare se i cambiamenti sono stati apportati correttamente |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.6 TC UC 6

|  |  |
| --- | --- |
| ID | TC UC 6 |
| Nome | Modifica Struttura |
| Obiettivo | Testare la funzionalità di modifica di una struttura nel database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene cercata una struttura nel database 2. Vengono modificati i dati della struttura 3. Viene cercata nuovamente la struttura nel database per verificare se i cambiamenti sono stati apportati correttamente |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 

## 8.7 TC UC 7

|  |  |
| --- | --- |
| ID | TC UC 7 |
| Nome | Aggiungi / Rimuovi portinaio |
| Obiettivo | Testare la funzionalità di aggiunta e rimozione di un portinaio nel database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene creata un’anagrafica per la mansione del portinaio 2. Vengono assegnati tutti i dati relativi al portinaio 3. Il portinaio viene inserito nel database 4. Il portinaio viene cercato nel database per assicurarsi che l’inserimento sia andato a buon fine 5. Il portinaio viene cancellato dal database |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.8 TC UC 8

|  |  |
| --- | --- |
| ID | TC UC 8 |
| Nome | Inizio / Carica Permanenza |
| Obiettivo | Testare la funzionalità di inserimento e caricamento di una permanenza nel database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene creata un’entità anagraficaStanza di tipo permanenza 2. Vengono assegnati tutti i dati relativi alla permanenza 3. L’entità viene aggiunta al database 4. L’entità viene cercata nel database per assicurarsi che l’inserimento sia andato a buon fine |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.9 TC UC 9

|  |  |
| --- | --- |
| ID | TC UC 9 |
| Nome | Inizio / Carica Visita |
| Obiettivo | Testare la funzionalità di inserimento e caricamento di una visita nel database |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene creata un’entità anagrafica stanza di tipo visita 2. Vengono assegnati tutti i dati relativi alla visita 3. L’entità viene aggiunta al database 4. L’entità viene cercata nel database per assicurarsi che l’inserimento sia andato a buon fine |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.10 TC UC 10

|  |  |
| --- | --- |
| ID | TC UC 10 |
| Nome | Termina Permanenza |
| Obiettivo | Testare la funzionalità che permette di terminare una permanenza |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene selezionata la l’anagraficaStanza di tipo permanenza da contrassegnare come terminata all’interno del database 2. Vengono inseriti la data e l’ora d’uscita e il prezzo da pagare per la permanenza 3. La permanenza aggiornata con i dati di conclusione viene cercata nel database per accertarsi che l’operazione sia andata a buon fine |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.11 TC UC 11

|  |  |
| --- | --- |
| ID | TC UC 11 |
| Nome | Termina Visita |
| Obiettivo | Testare la funzionalità che permette di terminare una visita |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene selezionata la l’anagraficaStanza di tipo visita da contrassegnare come terminata all’interno del database 2. Vengono inseriti la data e l’ora d’uscita 3. La visita aggiornata con i dati di conclusione viene cercata nel database per accertarsi che l’operazione sia andata a buon fine |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.12 TC UC 12

|  |  |
| --- | --- |
| ID | TC UC 12 |
| Nome | Rimuovi stanza |
| Obiettivo | Testare la funzionalità che permette di rimuovere una stanza |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene selezionata la stanza da rimuovere dal database 2. La stanza viene rimossa |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

## 8.13 TC UC 13

|  |  |
| --- | --- |
| ID | TC UC 13 |
| Nome | Rimuovi struttura |
| Obiettivo | Testare la funzionalità che permette di rimuovere una struttura |
| Condizioni d’ingresso |  |
| Flusso d’eventi | 1. Viene selezionata la struttura da rimuovere dal database 2. La struttura viene rimossa |
| Condizioni d’uscita | Le operazioni vengono svolte correttamente o viene visualizzato un messaggio d’errore che specificherà quale passaggio non è andato a buon fine |

# Testing Schedule

Di seguito saranno elencate la gestione dei rischi e la durata della fase di testing, l’organizzazione delle attività di testing e la schedulazione delle attività.

## Programma

La fase di testing inizierà il giorno Lunedì 4 Aprile e terminerà il giorno Giovedì 7 Aprile 2016.   
Data la fiducia nelle unità su cui si basa il sistema, che sono state ampiamente testate durante lo sviluppo, nonché la scarsità del tempo a disposizione, si è deciso di procedere soltanto con il test delle unità e il test funzionale del sistema, cioè andremo a verificare che il programma sia in grado di svolgere senza problemi i compiti per i quali è stato progettato, senza considerare la struttura interna del sistema (testing black-box).

La fase di testing è stata così divisa:

* Unit testing ( 4 Aprile 2016)
* Functional Testing ( 5 Aprile 2016 – 7 Aprile 2016)
  + Consultazione dei casi d’uso
  + Definizione dei casi d’uso da testare
  + Test del sistema

# 