Università degli studi di Salerno  
Corso di Ingegneria del Software

**RooManageR**

Presentazione



14/04/2016

Coordinatore del progetto:

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

Partecipanti:

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

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

Sommario

[Cos’è RooManageR? 4](#_Toc448398345)

[Perché RooManageR? 4](#_Toc448398346)

[Panoramica sulla documentazione 5](#_Toc448398347)

[**Documento di Analisi dei Requisiti** 5](#_Toc448398348)

[Requisiti Funzionali: 5](#_Toc448398349)

[Requisiti Non Funzionali: 6](#_Toc448398350)

[Modello a Oggetti 6](#_Toc448398351)

[Modello Dinamico 6](#_Toc448398352)

[Descrizione dell’interfaccia grafica 6](#_Toc448398353)

[**System Design Document** 7](#_Toc448398354)

[Panoramica del Sistema 7](#_Toc448398355)

[Vincoli di Design 7](#_Toc448398356)

[Architettura del sistema e del database 7](#_Toc448398357)

[Sicurezza e controllo degli accessi 8](#_Toc448398358)

[Flusso di Controllo Globale 8](#_Toc448398359)

[Configuration 8](#_Toc448398360)

[Start up and Shutdown 8](#_Toc448398361)

[Exception Use Cases 8](#_Toc448398362)

[**Object Design Document** 9](#_Toc448398363)

[Compromessi di Design 10](#_Toc448398364)

[Sicurezza ed efficienza 10](#_Toc448398365)

[Documentazione delle guideline dell’interfaccia 10](#_Toc448398366)

[Pacchetti 10](#_Toc448398367)

[Interfaccia delle classi 10](#_Toc448398368)

[**Testing** 11](#_Toc448398369)

[Test Plan 12](#_Toc448398370)

[Test Case Specification 15](#_Toc448398371)

[Test Summary Report 15](#_Toc448398372)

# Cos’è RooManageR?

RooManageR è un software che si propone di essere semplice e veloce e di permettere la gestione di tutte le funzioni indispensabili alla gestione di una o più strutture.  
Saranno presenti due diverse modalità, una per la tipologia d’utente proprietario e una per la tipologia d’utente portinaio.

L’utente portinaio avrà a disposizione un sottoinsieme delle funzioni del proprietario.

L’utente verrà portato alla schermata contenente le funzioni relative alla sua mansione in modo automatico in seguito al login.

In particolare sarà possibile:

* Aggiungere e rimuovere le strutture (Modalità proprietario)
* Aggiungere e rimuovere le stanze (Modalità proprietario)
* Assegnare e rimuovere un portinaio a una struttura (Modalità proprietario)
* Iniziare e terminare una visita o una permanenza in una determinata stanza
* Visualizzare l’elenco di tutte le operazioni svolte
* Visualizzare il registro clienti, contenente tutte le visite e le permanenze che sono state avviate e concluse dall’inizio dell’utilizzo di RooManageR, dettagliate di data e ora di inizio e conclusione.
* Inserire, leggere e modificare le anagrafiche relative alle visite e alle permanenze
* Inserire, leggere e modificare le anagrafiche relative ai portinai delle varie strutture (modalità proprietario)

Il software client sarà scritto in java, mentre i dati permanenti saranno archiviati in un database remoto.

# Perché RooManageR?

Abbiamo scelto di creare RooManageR dopo aver notato che nonostante la grande diffusione dell’informatica negli ultimi anni, molti gestori di strutture utilizzano ancora registri cartacei per segnalare ingressi ed uscite relativi a visite e permanenze, con conseguente accumulo di materiale fisico nel corso del tempo e danno a carico dell’ambiente.

Molto spesso, tuttavia, in queste strutture sono presenti anche dei computer, per cui una soluzione come RooManageR che non presenta requisiti tecnici particolarmente gravosi (basta un computer in grado di eseguire Java ed una connessione ad internet) è eseguibile dalla grande maggioranza delle persone, richiedendo costi di aggiornamento molto bassi o addirittura nulli.

# Panoramica sulla documentazione

## **Documento di Analisi dei Requisiti**

La fase di Analisi dei requisiti comprende i compiti che servono a determinare le condizioni da rispettare nella realizzazione di un progetto, tenendo conto dei requisiti possibilmente in conflitto da parte dei vari stakeholders e analizzando, documentando, convalidando e gestendo i requisiti software e di sistema.

La fase di analisi dei requisiti è fondamentale per determinare il successo o il fallimento di un progetto.   
I requisiti dovrebbero essere documentati, tracciabili e sufficientemente dettagliati per poter essere implementati in una fase successiva.

La fase di analisi dei requisiti include tre tipi di attività:

* **Acquisizione della conoscenza**: Si discute con i clienti e gli interessati dal progetto
* **Analisi dei requisiti**: si determina se i requisiti sono chiari, completi, consistenti e non ambigui, e si risolvono tutti i conflitti individuati.
* **Registrazione dei requisiti**: In questa fase i requisiti possono essere documentati in varie forme, che solitamente includono un sommario e spesso documenti in linguaggio naturale come casi d’uso e scenari.

Per RooManageR, dopo aver introdotto il sistema e averne individuato requisiti funzionali e non funzionali, sono stati inseriti i modelli del sistema.   
Sono stati descritti due scenari che descrivono un flusso di eventi tipico per le due tipologie di utente e sono stati descritti tutti i possibili casi d’uso del sistema per le due tipologie d’utente.

Requisiti Funzionali:

Il **Proprietario** deve poter:

* Creare e gestire le proprie strutture
* Creare e gestire le stanze locate all’interno delle proprie strutture
* Creare e gestire le anagrafiche
* Creare e gestire le visite all’interno delle proprie strutture
* Gestire gli inquilini all’interno delle proprie strutture
* Nominare un’anagrafica come portinaio per una o più delle sue strutture
* Visualizzare un registro operazioni
* Visualizzare il registro movimenti per ognuna delle sue strutture

Il **Portinaio** deve poter:

* Creare e gestire le visite all’interno della propria struttura
* Gestire gli inquilini all’interno della propria struttura
* Creare e gestire le anagrafiche
* Visualizzare un registro operazioni
* Visualizzare il registro movimento per la struttura presso cui è in servizio

### Requisiti Non Funzionali:

* Il tempo di risposta del server deve essere minore di 2 secondi
* Il tempo di avvio deve essere inferiore a 5 secondi
* Deve essere garantito un “up-time” del 98% annuo
* L’applicazione non deve avere “freeze” dell’interfaccia
* L’applicazione non deve avere alcun “**ANR**” (**A**pplication **N**ot **R**esponding)
* L’applicazione dev’essere multipiattaforma

### Modello a Oggetti

Per quanto concerne il modello ad oggetti:

* È stato inserito il class diagram che rappresenta gli entity object del sistema

### Modello Dinamico

Relativamente al modello dinamico:

* Sono stati inseriti i sequence diagram per ogni caso d’uso e per ogni eccezione connessa ai particolari casi d’uso
* Sono stati inseriti gli state chart diagram per le entità visita e permanenza, le uniche che possiedono un vero stato (Iniziata – in corso – terminata) all’interno del sistema.

### Descrizione dell’interfaccia grafica

Per la descrizione dell’interfaccia grafica:

* Sono stati inseriti screenshot dell’intero sistema ed è stato descritto il navigational path che le due tipologie di utenti potranno seguire per svolgere le varie operazioni.

## **System Design Document**

Il processo di System Design serve a definire l’architettura, i componenti, i moduli e i dati necessari a soddisfare i requisiti di un sistema.

La fase di System Design può essere vista come l’applicazione della teoria dei sistemi allo sviluppo di un prodotto.

C’è sovrapposizione con le discipline di analisi dei sistemi, architettura dei sistemi ed ingegneria dei sistemi.

Questo documento descrive il sistema a livello architetturale, includendo i sottosistemi e i loro servizi, il mapping hardware-software, la gestione dei dati, la matrice di controllo degli accessi, la struttura di controllo globale del software e le boundary condition.

Il documento di System Design dovrebbe definire come implementare tutti i requisiti del RAD, e dovrebbe fornire una guida di base per poi scendere ulteriormente in dettaglio nel processo che condurrà verso una soluzione realizzabile.

I destinatari del documento di System Design sono tutti i software architect e i liaison che fanno parte del team di sviluppo di ciascun sottosistema.

### Panoramica del Sistema

Nello scrivere il documento di System Design di RooManageR, abbiamo introdotto il nostro sistema spiegando a chi è rivolto e quali sono le possibilità offerte alle due tipologie di utenti che potranno effettuare il login.

### Vincoli di Design

Abbiamo poi analizzato i vincoli di design, relativi nel caso specifico alle macchine su cui dovrà girare il sistema software e sui requisiti di connessione richiesti, e abbiamo individuato l’architettura del sistema, che sarà formato da:

* Un’applicazione Java che girerà sulla macchina dell’utente
* Un’applicazione server in php che interagirà con il database
* Un database relazionale remoto su cui saranno archiviati i dati

### Architettura del sistema e del database

Sono stati descritti i sottosistemi in cui è scomposta l’applicazione Java, e l’architettura del database con le relative tabelle e campi.

### Sicurezza e controllo degli accessi

Nella sezione del documento dedicata alla sicurezza, è stato descritto come in fase di login l’utente dovrà inserire il proprio codice fiscale e password, e in base alla mansione assegnatagli internamente al database sarà indirizzato in modo automatico alla modalità del programma che contiene le funzionalità per lui pensate.

In oltre, è stata rappresentata la matrice di controllo degli accessi specificando per ogni coppia Tipologia di utente – Azione la possibilità di svolgimento.

### Flusso di Controllo Globale

Nella sezione dedicata al flusso di controllo globale è stato specificato che in RooManageR non esiste una sequenza di operazioni prestabilita, ma il flusso è guidato dagli eventi (event-driven), e viene avvito un thread quando è necessario interagire con il server per inviare e ricevere risposte dal database.

Questa scelta è stata presa per evitare “freeze” dell’interfaccia grafica in attesa di una risposta.

### Configuration

Nella sezione Configuration è stato descritta la fase di creazione e distruzione per gli oggetti persistenti.

### Start up and Shutdown

Nella sezione Start up and Shutdown viene descritto quando, come e da chi dovrebbero essere avviate le diverse componenti del sistema.

### Exception Use Cases

Nella sezione Exception Use Cases è stato descritto il principale errore che può affliggere RooManageR, cioè quello relativo alla comunicazione con il server, e la reazione che dovrebbe avere il sistema software nel verificarsi di tale situazione.

## **Object Design Document**

Il documento di Object Design descrive i compromessi a cui sono giunti gli sviluppatori, le linee guida che hanno seguito nello sviluppo dei sottosistemi, la decomposizione dei sottosistemi in pacchetti e classi e l’interfaccia delle classi.

L’object design document è usato per scambiare informazioni sull’interfaccia tra i team e come riferimento durante il testing.

Il target dell’Object Design Document include gli architetti di sistema, gli sviluppatori che implementano ciascun sottosistema ed i tester.

Esistono tre approcci principali nel documentare l’object design:

* Il primo approccio consiste nel documentare il modello di object design nello stesso modo in cui abbiamo documentato il modello di analisi o il modello di system design: scriviamo e manteniamo un modello UML e manteniamo il documento automaticamente.   
  Gli svantaggi di questa soluzione includono la ridondanza con il RAD e lo sforzo necessario per mantenere la consistenza con il RAD stesso. Inoltre, l’ODD duplica le informazioni contenute nel codice sorgente, e richiede un alto livello di sforzo ogni volta che il codice cambia.   
  Questo conduce spesso a inconsistenze tra RAD e ODD, che risultano inaccurati o non aggiornati.
* Il secondo approccio consiste nel trattare il modello di object design come estensione del modello di analisi.   
  Il vantaggio di questa soluzione è che mantanere la consistenza tra RAD e ODD diventa molto più semplice come risultato della riduzione della ridondanza.   
  Gli svantaggi di questa soluzione includono l’affollamento del RAD con informazioni che sono irrilevanti per il cliente e l’utente.
* Il terzo approccio è includere l’ODD nel codice sorgente.   
  Possiamo generare l’ODD usando un tool che analizza il codice sorgente e estrae le informazioni rilevanti, come ad esempio Javadoc.   
  Il vantaggio di questo approccio è che la consistenza tra il modello di object design e il codice sorgente è molto più semplice da mantenere: Quando vengono effettuati cambiamenti nel codice sorgente, i commenti javadoc vengono aggiornati e l’ODD viene rigenerato.   
  **Quest’ultimo è l’approccio che è stato scelto per RooManageR.**

### Compromessi di Design

Nella sezione *compromessi di design* è stato spiegato come sia stato necessario sacrificare alcuni dettagli di implementazione per rispettare i tempi previsti, ed è stato specificato che verrà utilizzato un database remoto per la gestione dei dati persistenti necessari al sistema.

### Sicurezza ed efficienza

Nella sezione *sicurezza ed efficienza* è stato specificato quali tipi di dati saranno controllati e quali invece saranno accettati senza verifiche per ottimizzare i tempi di risposta dell’applicazione.

### Documentazione delle guideline dell’interfaccia

Nella sezione *documentazione delle guideline dell’interfaccia* sono state specificate le linee guida da seguire durante lo sviluppo del codice, come le convenzioni da utilizzare per i nomi delle classi e dei metodi.

### Pacchetti

Nella sezione *pacchetti* sono stati individuati e descritti tutti i pacchetti in cui è stato diviso il software.

### Interfaccia delle classi

Nella sezione *interfaccia delle classi* è stata presa in esame ogni classe e ne è stata analizzata la struttura interna attraverso la javadoc.

## **Testing**

Il testing è il processo che permette di individuare le differenze tra il comportamento atteso specificato dal modello del sistema e quello osservato una volta che il sistema è stato implementato.

Il test di unità trova le differenze tra il modello di object design e la componente corrispondente.

Il test strutturale trova le differenze tra il modello di system design e un sottinsieme di sottosistemi integrati.

Il test funzionale trova le differenze tra il modello dei casi d’uso e il sistema.

In fine, il test delle performance trova le differenze tra i requisiti non funzionali e le performance reali del sistema.

Quando vengono trovate delle differenze, gli sviluppatori individuano i difetti che causano le failure in questione e modificano il sistema per correggerli.   
In altri casi, la causa della differenza è identificata nel modello del sistema, e il modello viene aggiornato per riflettere lo stato del sistema stesso.

Dal punto di vista della modellazione, il testing ha lo scopo di mostrare che l’implementazione del sistema non è consistente con i modelli del sistema.

Quest’attività è contraria a tutte le altre attività che abbiamo descritto nei capitoli precedenti: Analisi, design, implementazione, comunicazione e negoziazione sono attività costruttive, mentre il testing punta a demolire il sistema.

Di conseguenza, il testing viene normalmente portato avanti da sviluppatori che non sono stati coinvolti nel processo di costruzione nel sistema.

L’attività di testing è documentata attraverso quattro tipi di documenti.

* Il Test Plan si concentra sugli aspetti manageriali del testing.   
  Documenta l’ambito, l’approccio, le risorse ed il programma delle attività di testing.   
  I requisiti e i componenti da testare sono identificati in questo documento.
* Ogni test da effettuare è documentato dal documento di Test Case Specification, che contiene gli input, i driver, gli stubs e gli output attesi per ogni test, oltre ai task che ci si aspetta vengano eseguiti.
* Ogni esecuzione di ogni test è documentata nel Test Incident Report. Vengono registrati il risultato reale del test e la differenza rispetto al risultato atteso.
* Il Test Report Summary elenca tutte le failure scoperte durante i test che è necessario investigare.  
  Dal Test Report Summary, gli sviluppatori analizzano e prioritarizzano ogni failure e piano di cambiamento nel sistema e nei modelli.   
  Questi cambiamenti possono chiaramente portare a nuovi casi di test e a nuovi test.

### Test Plan

#### Relazioni con altri documenti

Nella sezione *Relazioni con altri documenti* vengono evidenziate le relazioni tra il Test Plan e il RAD, l’SDD e l’ODD.

#### Panoramica del sistema proposto

Nella sezione *Panoramica del sistema proposto* vengono esposti i criteri in base ai quali verrà testato RooManageR.   
I criteri in questione saranno quelli di:

* **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 stanza in cui è ancora in corso una permanenza o di una struttura da cui non sono state rimosse tutte le stanze
* **Disponibilità**: Una volta che il sistema sarà rilasciato, sarà disponibile ogni volta che l’utente ne richiederà l’utilizzo

#### Criteri di successo e insuccesso

Nella sezione *criteri di successo e insuccesso* viene spiegato che viene considerato che un test case ha avuto “Successo” quando il risultato reale non corrisponde a risultato atteso, mentre “fallisce” se il risultato osservato risulta essere uguale all’oracolo.

#### Approccio

Nella sezione *approccio* viene descritta la stretegiadi testing adottata per RooManageR.

Viene spiegato che verrà svolto l’unit testing di ogni unità, e verrà successivamente effettuato il functional testing per verificare la corrispondenza tra i casi d’uso individuati in fase di analisi dei requisiti e le effettive funzionalità del programma.

#### Linee Guida

Nella sezione *Linee guida* è descritto come l’obbiettivo principale nel testare RooManageR sia stato il determinare la corrispondenza tra il software e i requisiti funzionali.

In oltre, viene specificato che per la costruzione dei test ci si avvarrà esclusivamente della specifica dei requisiti, ignorando come è stato realizzato il sistema al suo interno (black-box).

#### Unit Testing

Nella sezione *Unit Testing* viene spiegato che sarà effettuato il testing per ogni entity object individuato nel sistema.

#### Integration Testing

Nella sezione *Integration Testing* viene descritto come sarà eseguito l’integration testing.

#### System Testing

Nella sezione *System Testing* viene descritto come verrà eseguito il system testing. In RooManageR, è stato eseguito esclusivamente il testing funzionale.

#### Sospensione e Ripristino

Nella sezione *sospensione e ripristino* viene descritto come ci si comporterà relativamente al testing in caso di individuazione di errori.   
Se verranno riscontrati errori o fallimenti, il processo di testing verrà sospeso per tutte le componenti, e durante la fase di sospensione si passerà alla correzione degli errori da parte degli sviluppatori.

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

In oltre, ad ogni ripristino del testing saranno effettuati nuovi test su ciò che è stato corretto, secondo la tecnica del *regression testing*, per verificare se le correzioni hanno introdotto nuovi errori e per accertare in generale la correttezza dell’intero gruppo di componenti.

#### Materiale per il testing

Nella sezione *materiale per il testing* vengono elencate le risorse hardware e software impiegate nella fase di testing di RooManageR.

#### Test Cases

Nella sezione *test cases* vengono descritti i casi da testare secondo una tabella che comprende:

* Id del caso di test
* Nome del caso di test
* Obiettivo del caso di test
* Condizioni di ingresso del caso di test
* Flusso degli eventi
* Condizioni d’uscita

#### Testing Schedule

Nella sezione *testing schedule* si descrive la gestione dei rischi e la durata delle varie fasi del testing.

### Test Case Specification

#### Relazione con altri documenti

Nella sezione *relazione con altri documenti* si specifica che i nomi dei test case fanno riferimento a quelli individuati nel Test Plan

#### Test Case

Nella sezione dedicata ad ogni test case vengono specificati:

* Nome del caso di test
* Descrizione del caso di test
* Elementi testati nel caso di test
* Specifiche dell’input
* Output atteso
* Requisiti dell’ambiente
* Eventuali procedure speciali richieste
* Dipendenze tra i casi

### Test Summary Report

#### Sommario dei test effettuati

La sezione *sommario dei test effettuati* descrive quali tipologie di testing sono state eseguite su RooManageR.

#### Riepilogo dei risultati

La sezione *riepilogo dei risultati* riassume i risultati dei test eseguiti, specificando che solo due dei casi di test selezionati hanno dato “successo”, cioè che solo due dei casi di test hanno rilevato inconsistenze tra l’output atteso e l’output reale.