November 28, 2014

Brandin Jefferson, issa samake, zach danjean, jaquincy nelson, brian okoye, yee wong

University of louisiana at lafayette

CMPS 453 – Dr. Ashok Kumar

project report

ul housing assignment

# Executive Summary

The following document is an amalgamation of all the reports created by the group that worked on the UL Housing Department’s project for CMPS 453 at the University of Louisiana at Lafayette. Within this document are descriptions of the way the project was planned, developed, and eventually deployed. These include aspects of development like the resources utilized by the team to complete the project, the risks we would encounter, detailed diagrams depicting the system’s inner workings, and how it was tested.

Table of Contents

[Executive Summary 1](#_Toc404962255)

[List of Figures 2](#_Toc404962256)

[List of Tables 3](#_Toc404962257)

[Introduction 4](#_Toc404962258)

[Project Management 5](#_Toc404962259)

[Life Cycle Model 5](#_Toc404962260)

[Risk Analysis 5](#_Toc404962261)

[Hardware and Software Resource Requirements 5](#_Toc404962262)

[Requirement Specifications 6](#_Toc404962263)

[Use Case Diagram 6](#_Toc404962264)

[Rationale 8](#_Toc404962265)

[Non-Functional Requirements 8](#_Toc404962266)

[Architecture 9](#_Toc404962267)

[ARCHITECTURE MODEL 9](#_Toc404962268)

[ARCHITECTURE STYLE 12](#_Toc404962269)

[TECHNOLOGY, SOFTWARE, AND HARDWARE 12](#_Toc404962270)

[RATIONALE FOR ARCHITECTURE STYLE AND MODEL 13](#_Toc404962271)

[Design 14](#_Toc404962272)

[Graphic User Interface Design 14](#_Toc404962273)

[STATIC MODEL SEQUENCE DIAGRAMS 17](#_Toc404962274)

[DYNAMIC MODEL SEQUENCE DIAGRAMS 18](#_Toc404962275)

[RATIONALE FOR DETAILED DESIGN MODEL 21](#_Toc404962276)

[TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL 21](#_Toc404962277)

[Test Plan 22](#_Toc404962278)

[SYSTEM LEVEL TEST CASES 22](#_Toc404962279)

[TRACEABILITY OF TEST CASES TO USE CASES 26](#_Toc404962280)

[TECHNIQUES FOR TEST GENERATION 27](#_Toc404962281)

[References 28](#_Toc404962282)

# List of Figures

[Figure 1: Room Storage Use Case 6](#_Toc404962237)

[Figure 2: Scenarios 9](file:///C:\Users\Brandin%20Jefferson\Documents\GitHub\CMPS453-Docs\FinalProjectReport.docx#_Toc404962238)

[Figure 3: Logic View 10](file:///C:\Users\Brandin%20Jefferson\Documents\GitHub\CMPS453-Docs\FinalProjectReport.docx#_Toc404962239)

[Figure 4: Physical View 10](file:///C:\Users\Brandin%20Jefferson\Documents\GitHub\CMPS453-Docs\FinalProjectReport.docx#_Toc404962240)

[Figure 5: Development View 11](file:///C:\Users\Brandin%20Jefferson\Documents\GitHub\CMPS453-Docs\FinalProjectReport.docx#_Toc404962241)

[Figure 6: Process View 11](file:///C:\Users\Brandin%20Jefferson\Documents\GitHub\CMPS453-Docs\FinalProjectReport.docx#_Toc404962242)

[Figure 7: Model View Controller Style 12](file:///C:\Users\Brandin%20Jefferson\Documents\GitHub\CMPS453-Docs\FinalProjectReport.docx#_Toc404962243)

[Figure 8 - GUI: Login Screen 14](#_Toc404962244)

[Figure 9 - Student Interface 15](#_Toc404962245)

[Figure 10 - Admin Interface 16](#_Toc404962246)

[Figure 11 - Static Diagram: Room Change (Student)/(Admin) 17](#_Toc404962247)

[Figure 12- Dynamic Diagram: Create New ID 18](#_Toc404962248)

[Figure 13- Dynamic Diagram: Request New Room 19](#_Toc404962249)

[Figure 14- Dynamic Diagram: Decide Room Requests 20](#_Toc404962250)

# List of Tables

[Table 1- Use Case: Request Room 7](#_Toc404962251)

[Table 2 – Administrator Login 7](#_Toc404962252)

[Table 3 - Accept/Deny Request 8](#_Toc404962253)

[Table 4 - Testing Example 26](#_Toc404962254)

# Introduction

# Project Management

## 

## Life Cycle Model

The software development lifecycle model chosen for this project is a mixture of both waterfall and agile software development methods. The beginning stages we have used the waterfall model for project planning as well as for writing the requirements. The waterfall method is very useful when starting to acquire the customer’s requirements and create documentation. Later on when the design portion is completed, we will be transitioning to the agile method. The agile method will later be used during the design, development and testing stage of the project. The method’s flexibility and the ability for processing stages to run concurrently would allow adaptations to the customer’s requirements.

## Risk Analysis

The risks for the project include the possibilities such as a group members losing motivation, group members dropping out of the class, and group members deciding to switch teams. This would result in staff turnover and management change risks within the project. The best way to handle these risks is for each member to try their best to complete the project. Motivation will be a combination of a finished project and a good grade. This should also keep members from wanting to switch teams.

## Hardware and Software Resource Requirements

The hardware that will be used to carry out the development will be any computer running the Windows 7 operating system. To test multiple threads, if it is required, then the computers within the CMPS lab will be in use heavily. Printers in the CMPS lab and other labs around the U.L. Lafayette campus will be used to print out documentation.

We will use programming software as well as database software to implement code for the development. The IDEs will be used for interfacing and will be used alongside HTML script and data basing software. Ruby on Rails will be the programming language of choice as well as MySQL to help implement a database. Git will be used as a repository for the applications development. Web applications such as google docs and google hangout will be used as a means of communication for members within the group. UL Lafayette email will be used to communicate with client when necessary. The Facebook messenger and application will also be used for communication.

# Requirement Specifications

## Use Case Diagram

**Use Cases (Scenarios)**

Student

Administrator

Database

Database

Database

Database

Figure 1: Room Storage Use Case

|  |  |
| --- | --- |
| Use Case Name | Requesting a new room |
| Participating Actors | Students |
| Flow of Events | 1. Student enters website. 2. Student fills out application. 3. Student submits application. 4. Confirmation email is sent to school email address. 5. Student confirms email address. 6. Request is sent to queue of requests. |
| Alternate Flows | 3a. Information is not completely filled.  3b. Display error message.  3c. Remain on page.  4a. CLID is invalid.  4b. Display error message.  4c. Remain on page. |
| Entry Condition(s) | Student has a valid CLID. |
| Exit Condition(s) | Form is correctly filled out and entered; alternately, user cancels request. |
| Special Requirements | N/A |

Table 1- Use Case: Request Room

|  |  |
| --- | --- |
| Use Case Name | Administrator Login |
| Participating Actors | Students |
| Flow of Events | 1. Admin enters website. 2. Administrator select the link for logging in. 3. Administrator enters credentials. 4. Administrator submits credentials. 5. Request Queue displayed. |
| Alternate Flows | 4a. Username or password is invalid.  4b. Display error message.  4c. Remain on page. |
| Entry Condition(s) | Administrator has a valid username and password. |
| Exit Condition(s) | User presses return button; |
| Special Requirements | Must be registered within the program as an admin for the correct privileges. |

Table 2 – Administrator Login

|  |  |
| --- | --- |
| Use Case Name | Accept/Deny Request |
| Participating Actors | Students |
| Flow of Events | 1. Administrator successfully logs in to website. 2. Website presents all requests in the form of a list with an accept and deny button for each item. 3. Administrator selects either accept or deny. 4. Request is removed from the queue. 5. Email containing the decision is sent to the student. 6. Page refreshes to show new queue. |
| Alternate Flows | N/A |
| Entry Condition(s) | Administrator has logged in. |
| Exit Condition(s) | User presses log out button. |
| Special Requirements | Must be registered within the program as an admin for the correct privileges. |

Table 3 - Accept/Deny Request

## Rationale

It was determined that, in order to efficiently perform any of the client’s tasks, a database would be necessary. Since all of the tasks needed one, it was decided that all of the projects would be completely based around the database. The only extra things required will be interfaces unique to the individual projects.

## Non-Functional Requirements

**Product requirement**

The requirements include execution speed, reliability and accessibility of the system .The room-change system should be available to all current residents on campus during weekdays and weekends. Downtime for site maintenance shall not exceed 24 hours. The system should be easy to use by residents and should be organized in such a way that user errors are minimized.

**Organizational requirement**

Users of the room-change system shall authenticate themselves using their CLID and user-defined password. Priority of the room-change service should follow the first-come, first-served practice.

**External requirement**

The system shall implement resident privacy provisions. Maintenance should be done on the system regularly.

# Architecture

## ARCHITECTURE MODEL

The chosen model is the 4+1 View Model. The following figures describe the individual views.

Occupancy map

Occupancy map

Occupancy map

Occupancy map

Figure 2: Scenarios

**Use Cases (Scenarios)**

Student

Administrator

Database

Database

Database

Database

**External Desktop PC**

Students <Web Interface>

**External Desktop PC**

Students <Web Interface>

**External Desktop PC**

Students <Web Interface>

**External Desktop PC**

Students <Web Interface>

**Desktop PC**

Administrators <Application>

**Desktop PC**

Administrators <Application>

**Desktop PC**

Administrators <Application>

**Desktop PC**

Administrators <Application>

**Laptop**

Students

<Web Interface>

**Laptop**

Students

<Web Interface>

**Laptop**

Students

<Web Interface>

**Laptop**

Students

<Web Interface>

**Database**

**Database**

**Database**

**Database**

Figure 3: Logic View

**Web Interface**

Methods that the student interacts with

**Web Interface**

Methods that the student interacts with

**Web Interface**

Methods that the student interacts with

**Web Interface**

Methods that the student interacts with

**Application**

Methods that the admin interacts with

**Application**

Methods that the admin interacts with

**Application**

Methods that the admin interacts with

**Application**

Methods that the admin interacts with

**Database Manager**

Holds all methods and functions that allow editing and retrieval of info from database.

**Database Manager**

Holds all methods and functions that allow editing and retrieval of info from database.

**Database Manager**

Holds all methods and functions that allow editing and retrieval of info from database.

**Database Manager**

Holds all methods and functions that allow editing and retrieval of info from database.

**Database Files**

Contain the actual files

**Database Files**

Contain the actual files

**Database Files**

Contain the actual files

**Database Files**

Contain the actual files

Figure 4: Physical View

For the physical view, administrators are the only ones capable of interacting with the database without an internet connection. They have direct access to it through the application they will be supplied with, hence the dotted line.

Figure 5: Development View

User Interfaces, Request Queue,

Database Management, Mail verification

SQL databases, Base Rails framework,

User Level

Public Level

Private Level

Figure 6: Process View

<process>

RequestApplication

<process>

HousingDatabaseAccess

<process>

ApplicationDecision

<process>

DisplayMap

MapController

* getFloors()
* getOccupancy()

HousingController

* findFloor()
* changeOccupancy()
* findStudent()
* setStudentRoom()

RequestQueue

* getFloors()
* getOccupancy()

StudentRequestForm

* displayOptions()
* sendOptions()

## ARCHITECTURE STYLE

The Model-View-Controller style was chosen. To represent this, refer to *Figure 6* below.

**Database**

1. Store student records
2. Store room info
3. Edit student records
   1. Optional-Edit room info
4. Query student or room from ID

**Browser Interface**

1. Create new applications
2. Retrieve old applications
3. Send room change requests

**Admin Browser Interface**

1. Approve room change requests
2. Query student records
   1. Display queried records

**Admin Program Interface**

1. Approve room change requests
2. Query student records
   1. Display queried records
3. Activate floor plan interface

## TECHNOLOGY, SOFTWARE, AND HARDWARE

Figure 7: Model View Controller Style

* Languages
  + Ruby
  + SQL
* Tools
  + Rails
  + MySQL
* IDE
  + Aptana Studio – A Ruby centric editor modeled after the Eclipse IDE. Comes with built in terminals and editing windows or browsers for test development. Rails is fully supported, thus making everyone’s job easier.

## 

## RATIONALE FOR ARCHITECTURE STYLE AND MODEL

The 4+1 model view was chosen as the architecture model simply because it is widely used. The various descriptions that are required of it had forced the team to look into the project in a more in-depth pattern than had previously been planned. Since the process of completing it would provide not only a clear description of the project and its outcomes but also help the team with future assignments, it was decided that this method would be best.

As for the style, we decided upon a combination of the repository style and the transaction-processing style. The reason for this combination is due to the underlying features of the application. The program has everything based upon a database that stores different records for several years each. Students needed to be capable of adding their information to the database automatically while requiring minimal input from administrators to increase time efficiency. Simultaneously, administrators needed to be capable of accessing records either to edit them directly or perform some other action. The repository style is excellent both for storing information for long periods of time and maintaining information that is frequently edited. The transaction-processing style describes exactly how both administrators and students will interact with the system.

# Design

## Graphic User Interface Design

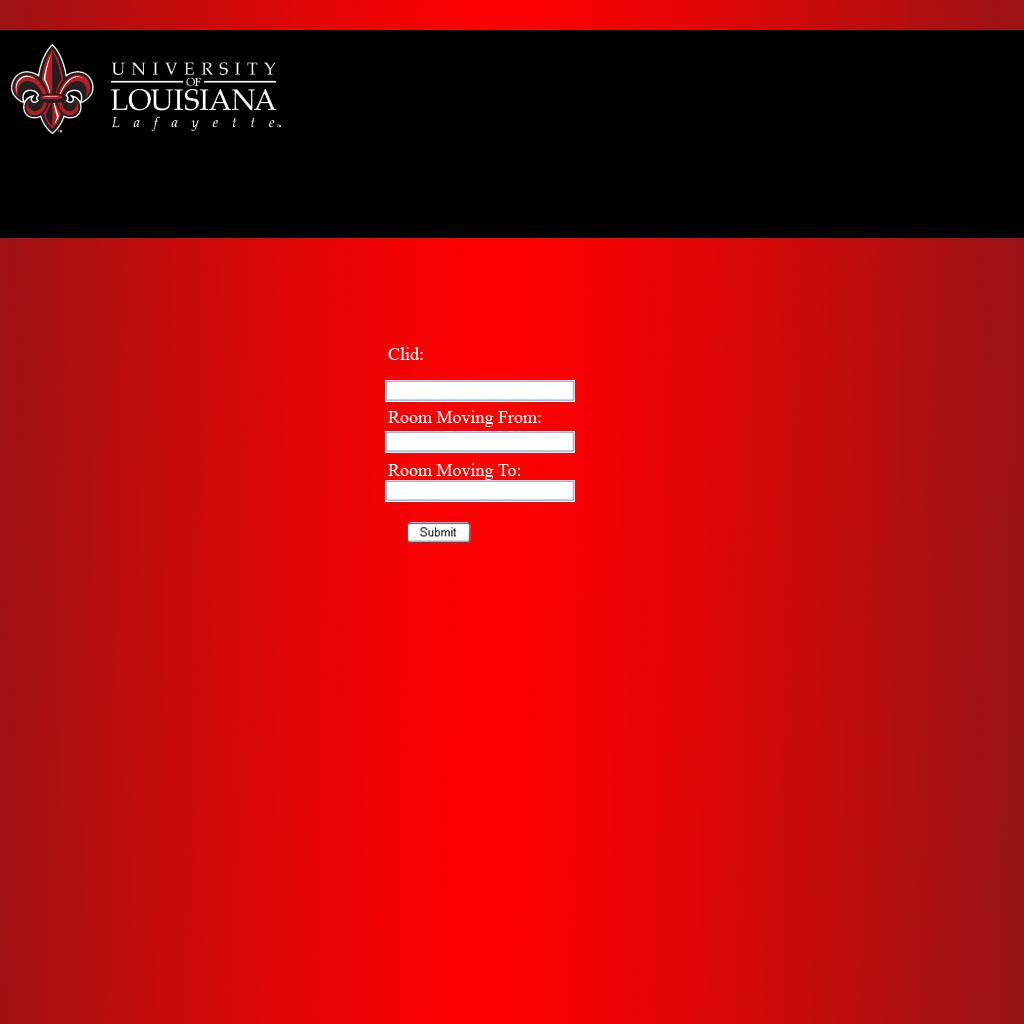


Figure 8 - GUI: Login Screen

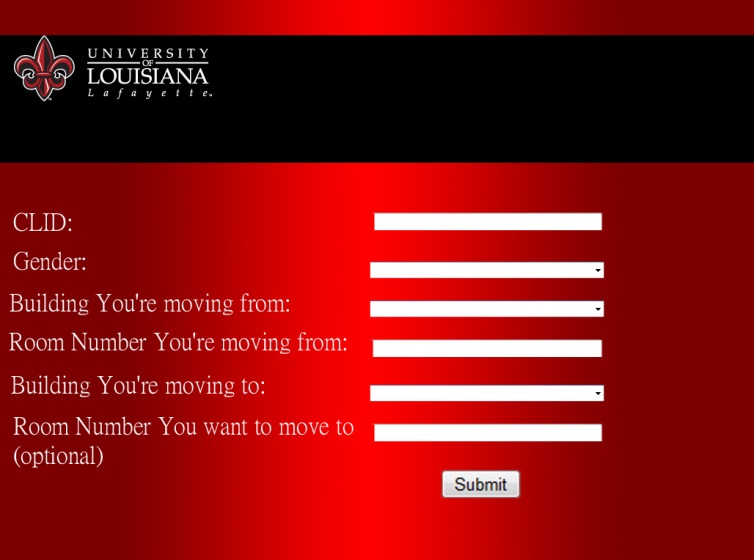


Figure 9 - Student Interface

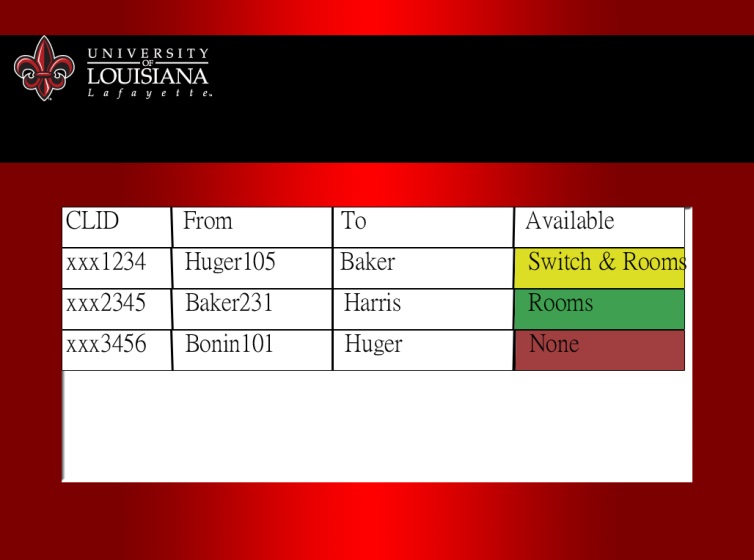


Figure 10 - Admin Interface

## STATIC MODEL SEQUENCE DIAGRAMS

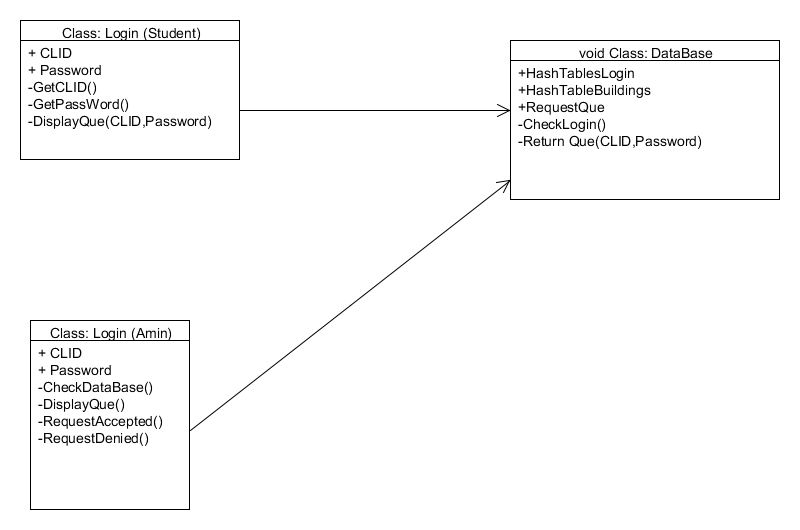


Figure 11 - Static Diagram: Room Change (Student)/(Admin)

## DYNAMIC MODEL SEQUENCE DIAGRAMS

sd Create New ID (void) : void

recordlist : StudentRecord

ids : StudentID

verifyID (id, password)

: boolean

*exists*

alt

[exists == false]

Login (id, password)

: void

mail : MailVerifier

createID(email, password)

: void

verifymail(email):

boolean

*valid*

alt

[valid == false]

notvalid() : void

errorMsg()

: void

else

AddID(): void

else

displayPage() :

void

Figure 12- Dynamic Diagram: Create New ID

Verifies a student’s identity if their CLID has not been used before by sending an email to their school email address. If it’s found to be valid, then the student is able to continue.

sd Request Room Change (void) : void

holder : Interface

db : HousingRecord

selectRoomChange

displayOptions

available(building, room)

*available*

alt

[available == true]

updateAvailability(id)

successMsg

exit

[else]

updateAvailability(id)

notFirstMsg

exit

rq : Request Queue

add

Figure 13- Dynamic Diagram: Request New Room

When a student registers for a room or building, the interface will send a query to the database to determine the availability of the room or any rooms respectively. A message displaying whether there is a chance at the moment or not is then printed to the screen.

sd Decide Room Requests

Interface Manager

DisplayRequests

Request Queue

*Request Queue*

DecideRequest

removeRequest

loop

sendMail

Figure 14- Dynamic Diagram: Decide Room Requests

After the admin logs in, they will have a view of every confirmed request. From there, they can accept or deny a request, the result of which will be an email being sent to the student with the correct response.

## RATIONALE FOR DETAILED DESIGN MODEL

We decided upon the given design by viewing other websites and examples. The final product that we’re creating is not unique by any means and, as such, already has plenty of use. The group chose the aspects that applied most appropriately, which placed in their simplest form. The ideology of “the simpler, the better” was heavily in play here due to the potential massive amount of traffic the client may receive in comparison to the number of employees available. So a minimum number of tasks are available at any one time for a user to choose, whether they be a student or administrator in order to streamline the process of whatever needs to be accomplished.

## TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL

The requirements document lists all of the features described by sequence diagrams and the user interface in this document. Each feature, save those that are no longer going to be used due to changes in the project parameters, has been implemented in greater detail than was described by the requirements. This was thanks to the group now having a clear set of tools and designs available for use, rather than just a vague idea of how the project should be handled.

# Test Plan

## SYSTEM LEVEL TEST CASES

|  |
| --- |
| Test Name: Data Retrieval  Subsystem: Data Base  Description: Test the retrieval of specific data  Pre-Condition: The user has a valid username and password |

|  |  |  |
| --- | --- | --- |
| Steps | Expected System Response | Pass/Fail |
| 1. The user clicks to access a certain data element | The system displays a message for the user to specify their search category | NA |
| 1. The user enters the specified category | The system searches for entered category from within the database | NA |
| 1. The user clicks to view search | The system displays a print out of the table form the database | NA |

|  |
| --- |
| Post -Condition: The user is shown a print screen of defined search item |

|  |
| --- |
| Test Name: Secure Login  Subsystems: Web Application/Database  Description: Test that the users login credentials are correct  Pre-Condition: The users defined credentials are correct |

|  |  |  |
| --- | --- | --- |
| Steps | Expected System Response | Pass/Fail |
| 1.The user selects to access the Housing database | The system displays a login GUI  With entries for a username and password and Account type | NA |
| 2. User enters in necessary credentials | The system collects the entered data and selects which database to compare from based on Account type | NA |
| 3.The user is a student | The system displays a GUI inquiring about Room Changes | NA |
| 4. The user is an administrator | The system displays a GUI of options of Admin Actions | NA |

|  |
| --- |
| Post Condition: The user is displayed with a GUI asking for their respected requirements |

|  |
| --- |
| Test Name: Room Request  System: Database  Description: Test the Database Room Request service  Pre-Condition: The user have valid login credentials |

|  |  |  |
| --- | --- | --- |
| Steps | Expected System Action | Pass/Fail |
| 1.The user selects to access the Housing database | The system displays a login GUI  With entries for a username and password and Account type | NA |
| 2. User enters in necessary credentials | The system collects the entered data and selects which database to compare from based on Account type | NA |
| 3.The user is a student | The system displays a GUI inquiring about Room Changes | NA |
| 4. The user selects a room request | The system collects information about the table element within database and writes to another table with user information. | NA |

|  |
| --- |
| Post -Condition: Room Request is accepted and information about the requested room and user is placed within another table |

|  |
| --- |
| Test Name: Edit Permission for Admin  System : Web Application/Database  Description: Test the Database for Administrator changes  Pre-Condition: The user login credentials are valid |

|  |  |  |
| --- | --- | --- |
| Steps | Expected System Action | Pass/Fail |
| 1.The user wants to access the Housing Database | The system displays a login GUI  With entries for a username and password and Account type | NA |
| 2. User enters in necessary credentials | The system collects the entered data and selects which database to compare from based on Account type | NA |
| 3. The user is an administrator | The system displays a GUI of options of Admin Actions | NA |
| 4.The user selects an option | The system compare option with compared with a predefined variable | NA |
| 5.The user selects to edit or exit | The system allows user to either edit selected elements or return to previous menu | NA |

|  |
| --- |
| Post-Condition: The system displays a list mapped to the given option and able to edit. |

|  |
| --- |
| Test Name: Remove Room Request for Admin  System : Web Application/Database  Description: Test the Database for Room Change Removal  Pre-Condition: The user login credentials are valid |

|  |  |  |
| --- | --- | --- |
| Steps | Expected System Action | Pass/Fail |
| 1.The user wants to access the Housing Database | The system displays a login GUI  With entries for a username and password and Account type | NA |
| 2. User enters in necessary credentials | The system collects the entered data and selects which database to compare from based on Account type | NA |
| 3. The user is an administrator | The system displays a GUI of options of Admin Actions | NA |
| 4.The user selects to view Room Change Request List | The system displays a list of requested room changes of students | NA |
| 5.The user searches for specific element | The system will display a drop down menu with specified element (Gender, Room Type, Building) | NA |
| 6.The user will any of the elements from menu | The system will filter through the list given the specified criteria and display result | NA |
| 7.The user selects a room request from the displayed list if any | The system will allow the user to edit selected item |  |

|  |
| --- |
| Post-Condition: The system display a list of Room Request and the user is able to remove items from list |

## TRACEABILITY OF TEST CASES TO USE CASES

In this test plan, we use only one main use case, which is the requesting a new room use case. We split this use case into subparts which represent the main steps of this use case and then, we elaborate a test case for each subpart.

|  |  |
| --- | --- |
| USE CASE | TEST CASES |
| **Requesting a new room** |  |
| 1. Student logs into Housing website. | Secure Login |
| 1. Student navigates to request form | Room Request |
| 1. Request form is filled out. | Room Request |
| 1. Form is sent to the database. | Room Request |
| 5. System displays list of unanswered room requests | Edit Permission for admin |
| 6. Admin selects a room request | Edit Permission for admin |
| 1. System displays information pertaining to that request | Edit Permission for admin |
| 1. System replaces student’s current file with new one | Remove Permission for Admin |
| 9- List of unanswered room requests displayed | Remove Room Request for Admin |

Table 4 - Testing Example

## TECHNIQUES FOR TEST GENERATION

We use both black box techniques as well as white box techniques in the test plan. We use white box testing for the data retrieval test because the tester needs to have an excellent knowledge of how the program components interact. Also, the white box testing is highly effective in detecting and resolving problems, because bugs can often be found before they cause trouble. The tester should get the right data request. The criteria used for measuring the quality of the data retrieval test are the correctness of the data, the reliability of the database and the efficiency of the process. We use black box testing for the remaining tests because the tester does not need to know any internal structure and coding in the program. The tester generates inputs and gets in return outputs that will compare to user requirements. The first test is the secure login the tester tests to see if he gets the correct response from the web applications when he enters the required information. The quality of this test will be measured based on the correctness of the response, the reliability and efficiency of the function. The second black box test is the Room request test .For this test, the tester logs in as a student and make a room change request then check that the request has been successfully added to the queue of room requests which is one of the user requirements. The quality of the test is measured based on the reliability of the function. The next black box test is to test if the administrator is able to make changes to the database such make room changes for the occupants, which is another user requirement. The quality of the test will be measure based on 3 criteria: the correctness of the changes, the reliability of the function and efficiency of the process. The last black box test is to test if the administrator can remove requests from the room change requests list, as they are processed to keep the list updated which is the last user requirement. The quality of the test will be measured based on the reliability of the function.

# References

Badgerati. “Software Engineering – Use Case Diagrams / Descriptions.” *Computer Science Source*. Computer Science Source, 22 Nov 2009. Web. 29 Sep 2014.

Bell, Donald. UML basics: The sequence diagram. IBM Corporation, 14 Feb. 2004. Web. 26 Oct 2014.

Johnson, S. “Example: Software Architecture Document.” California State University: Northridge, 21 March 1999. Web. 11 Oct. 2014.

Krutchen, Philippe. “Architectural Blueprints – The “4+1” view Model of Software Architecture.” IEEE Software 12.6 (1995): 42 – 50. Print.