Augmented Reality Room Designer: Agile Project Documentation

Software Engineering A Practitioner’s Approach 9th Edition Supplement

Jeffrey J. Yackley with Roger Pressman and Bruce Maxim

Table of Contents

[1.0 Introduction 2](#_Toc27933133)

[1.1 Purpose 2](#_Toc27933134)

[1.2 The AR Room Designer 2](#_Toc27933135)

[2.0 Process Model 3](#_Toc27933136)

[3.0 Use Cases 4](#_Toc27933137)

[3.1 Use Case 1: Create a Room 4](#_Toc27933138)

[3.2 Use Case 2: Place a Window 4](#_Toc27933139)

[3.3 Use Case 3: Place a Doorway 4](#_Toc27933140)

[3.4 Use Case 4: Place a Sensor 5](#_Toc27933141)

[3.5 Use Case 5: Place a Piece of Furniture 5](#_Toc27933142)

[3.6 Use case 6: Edit Object 5](#_Toc27933143)

[4.0 UML Model 6](#_Toc27933144)

[4.1 Use Case Diagram 6](#_Toc27933145)

[4.2 Deployment Diagram 6](#_Toc27933146)

[4.3 Class Diagram 7](#_Toc27933147)

[4.4 State Diagram 10](#_Toc27933148)

[4.5 Activity Diagram 11](#_Toc27933149)

[5.0 Customer Journey Map 13](#_Toc27933150)

[6.0 Personas 13](#_Toc27933151)

[7.0 UI Mock-up 14](#_Toc27933152)

[8.0 Testing Strategy 15](#_Toc27933153)

[8.1 Unit Testing 15](#_Toc27933154)

[8.2 Integration Testing 16](#_Toc27933155)

[8.3 Usability Testing 17](#_Toc27933156)

[8.4 Validation Testing 17](#_Toc27933157)

# Introduction

## Purpose

This document describes the software design of the Augmented Reality (AR) Room Designer. This case study is provided to supplement and clarify the processes, techniques, and theoretical knowledge provided in Roger Pressman’s and Bruce R. Maxim’s text, *Software Engineering: A Practitioner’s Approach,* 9th edition” McGraw-Hill, 2020. The text will be abbreviated and referenced in this document as SEPA.

We developed this case study following an agile philosophy of documentation. SEPA described agile documentation as the minimal descriptive information required to make sure the software development team, management, other stakeholders have a thorough understanding of the requirements and software to be built. Further following Agile principals, this document contains only information that the development team felt needed to be preserved and maintained. Additionally, we view this document as an iteration of the final design and therefore a work in progress. We invite critiques and suggestions for improvement of the document and model.

We view this document as being useful for instructors whom wish to explore the ideas of the AR Room Designer presented in SEPA in greater depth with their students. Additionally, students who would like to explore a more detailed example of the AR Room Designer architectural model will find this document useful.

## The AR Room Designer

The AR Room Designer is an augmented reality system. It uses a physical virtual reality headset, hand controllers, and room sensors. These interconnected devices will allow the user to see and manipulate a modifiable version of the living space in which the system is integrated through the deployment of the sensors. Once the system is established, a user is able to see an augmented 3D virtual version of the room they are standing inside and modify the dimensions of the room; the location, number, and dimensions of the door and windows; the number and location of security sensors; the number and location of placeholder furniture, wall-hangings, plants, and miscellaneous knick-knacks in the room; and can save or load their project. Future implementations will include upgrading the default placeholder objects to a diverse library of real products in each of the placeable object categories. In addition, future projects will allow the painting or wall-papering of surfaces and the ability to select various flooring options such as wood, tiles, and carpet. The six use cases presented as part of the case study were selected after consultation with the customer. Additional use cases will be added after the first incremental prototype is completed. The limitation to six use cases is based on minimizing the risk of the development team dealing with new technology and overcommitting to a wrong direction.

# Process Model

The software development team will use the evolutionary process model as show in Figure 1. The model the team selected is based on the desire to allow for rapid prototyping and iteration of design. The model is based on agile and spiral model principals in order to encapsulate feedback and risk management in to the decisions making for continuous evolution of the AR Room Designer project.

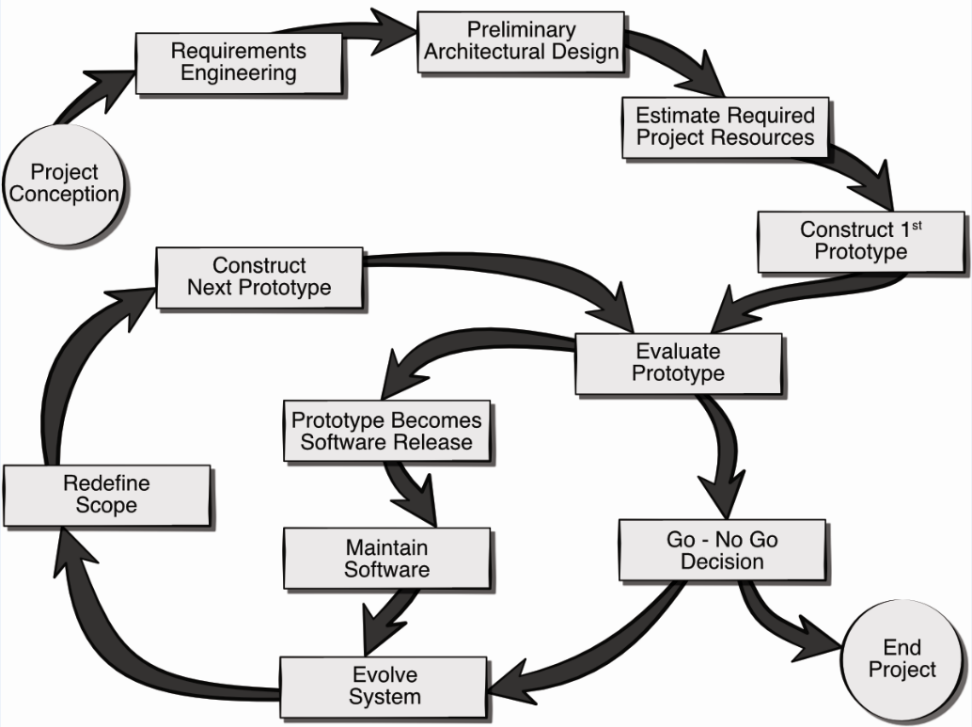


Figure 1. The AR Room Designer evolutionary process model.

# Use Cases

The following six uses cases were defined by the team and the client as the core system requirements for the delivery of working prototype. Note, this baseline functionality can be easily extended through the inclusion of more use cases as the project progresses. Each use case lists the name of the use case, primary actors, preconditions, description, and acceptance criteria.

## Use Case 1: Create a Room

**Primary Actor:** Homeowner / User

**Preconditions:** A project has been created to hold the room data.

**Description:** As a homeowner I want to be able to create a box-shaped room in my house project. I want the room to include a floor and four walls. I want to be able to set the width, length, thickness, and height of the floor and each of the four walls on creation.

**Acceptance Criteria:** I can create a room with my specified dimensions.

## Use Case 2: Place a Window

**Primary Actor:** Homeowner / User

**Preconditions:** A project has been created to hold the room data and a room has been created with at least one wall.

**Description:** As a homeowner I want to be able to be able to place a window in a wall. I should be able to click and drag a window object into a wall at any position in the wall. I should be able to set the width, length, and thickness of the window.

**Acceptance Criteria:** I can add a window to a wall in my room at my specified position and with my specified dimensions.

**Open Issue:** Need to address the situation where user places a door or window on top a different door or window.

## Use Case 3: Place a Doorway

**Primary Actor:** Homeowner / User

**Preconditions:** A project has been created to hold the room data and a room has been created with at least one wall.

**Description:** As a homeowner I want to be able to be able to place a door in a wall. I should be able to click and drag a door object into a wall at any position in the wall. I should be able to set the width, length, and thickness of the door.

**Acceptance Criteria:** I can add a door to a wall in my room at my specified position and with my specified dimensions.

**Open Issue:** Need to address the situation where user places a door or window on top a different door or window.

## Use Case 4: Place a Sensor

**Primary Actor:** Homeowner / User

**Preconditions:** A project has been created to hold the room data and a room has been created with at least a floor.

**Description:** As a homeowner I want to be able to be able to place a sensor in a room. I should be able to click and drag a sensor object into a room at any position on the floor, walls, door, or window.

**Acceptance Criteria:** I can add a sensor to my room at my specified position.

## Use Case 5: Place a Piece of Furniture

**Primary Actor:** Homeowner / User

**Preconditions:** A project has been created to hold the room data and a room has been created with at least a floor.

**Description:** As a homeowner I want to be able to be able to place a piece of furniture in a room. I should be able to click and drag a furniture object into a room at any position in the room. I should be able to set the width, length, height, and thickness of the piece of furniture.

**Acceptance Criteria:** I can add a piece of furniture in my room at my specified position and with my specified dimensions.

## Use case 6: Edit Object

**Primary Actor:** Homeowner / User

**Preconditions:** A project has been created to hold the room data, a room has been created, and some object exists to be modified.

**Description:** As a homeowner I want to be able to be able to re-position, re-size, and remove the floor, walls, windows, doors, sensors, and furniture objects in the room.

**Acceptance Criteria:** I can make changes to the objects in my room.

# UML Model

## Use Case Diagram

The six use cases are included in the use case diagram as seen in Figure 2. The only actor is the user who interfaces with the AR Room Designer System in order to manipulate the virtual environment. The user is able to create or place any of the five specified object classes through their respective use case. Each of these five use cases is then extended by the Edit Object use case which allows the user to edit the object properties such as transform, rotation, and scale in a 3D environment. The user can also select already created objects directly through the Edit Object use case in order to manipulate the objects properties as previously stated.

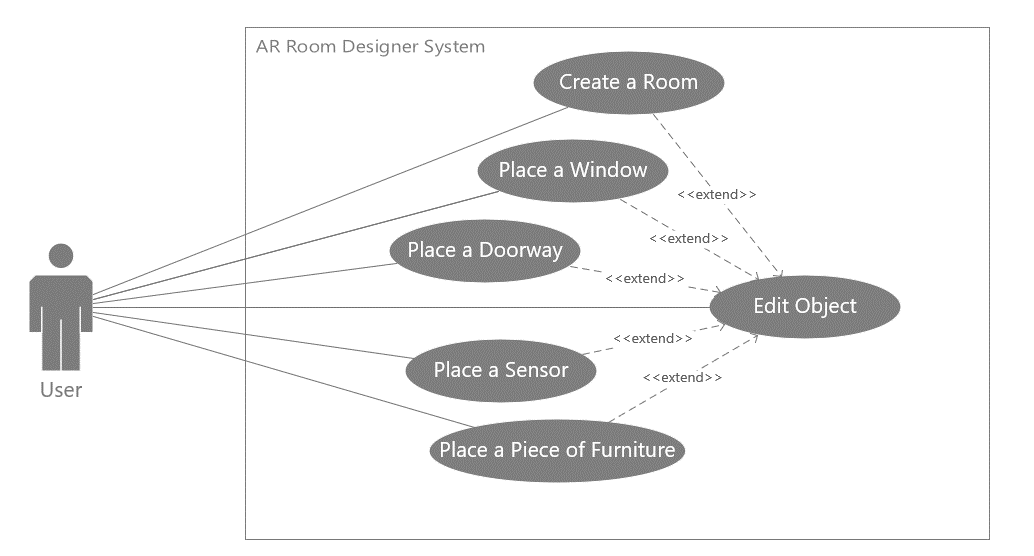


Figure 2. Use Case Diagram for the AR Room Designer System showing the six use cases for the project.

## Deployment Diagram

## 

A stylized deployment of the AR Room Designer System is illustrated in Figure 3. The AR Room Designer System software runs on a central processor for the system which we view as running on a user’s personal home computer or laptop. A user wears a VR headset which displays the augmented 3D reality of the room under design to the user. The user hold AR system controllers in order to manipulate the objects in the AR world. AR system room sensors take location, movement, and room data and communicate this back to the central processor. All devices are linked wirelessly.

A close up of a logo

Description automatically generated

Figure 3. Deployment diagram for the AR Room Designer System.

## Class Diagram

The class diagram for the core system of the AR Room Designer is depicted in Figure 4. The classes in the diagram are described below the figure.

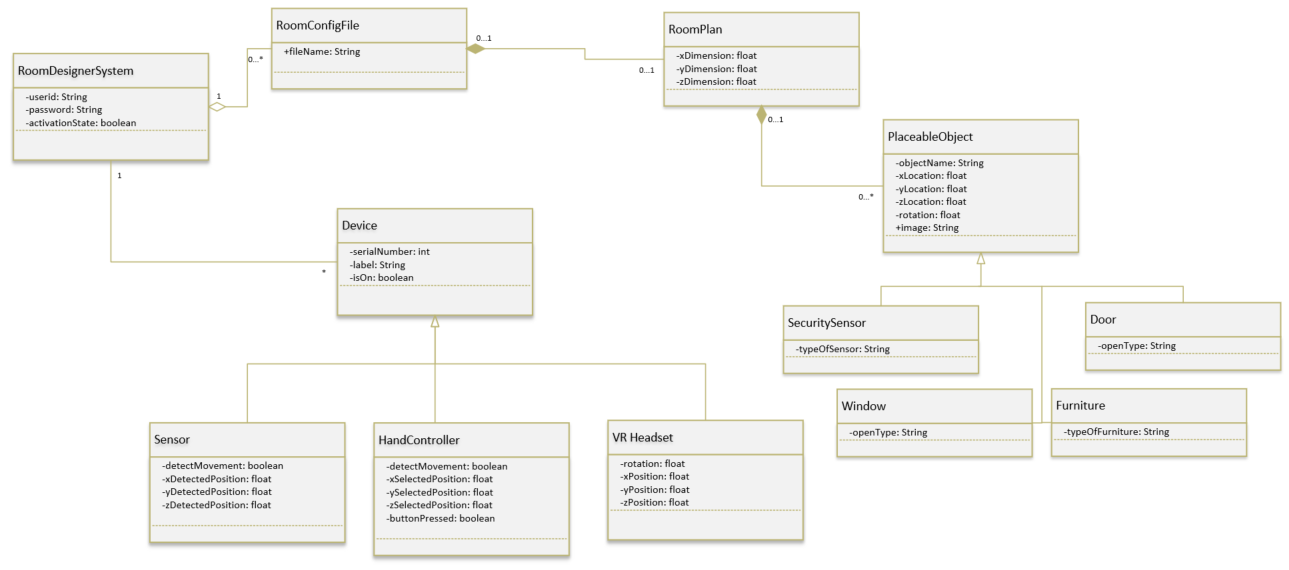


Figure 4. Class diagram for the AR Room Designer System

**RoomDesignerSystem**: the core singleton class (only one instance exists).

* **userid**: entered when logging into the system to differentiate users.
* **password**: a password to protect the user’s information.
* **activationState**: the state of the system as a whole.

**Device**: A generic parent class for different types of physical hardware systems that will send and receive input from the RoomDesignerSystem wirelessly.

* **serialNumber**: stores the unique identifier for the device.
* **label**: a name assigned to the device to make it clearer to the user what devices are currently connected to the system.
* **isOn**: true if the device is currently powered on and detected wirelessly by the system.

**Sensor**: a type of Device that is placed in a room by the user in order to detect their movements for the VR system to interpret and translate in to the virtual world.

* **detectMovement**: detects if user has moved position in the real world by comparing to stored positional values and returns true if different.
* **xDetectedPosition**: user real world last sensed x coordinate.
* **yDetectedPoistion**: user real world last sensed y coordinate.
* **zDetectedPosition**: user real world last sensed z coordinate.

**HandController**: a type of Device that in held by the user to help control and manipulate objects in the VR world.

* **detectMovement**: detects if the controllers have changed position in the real world by comparing to stored positional values and returns true if different.
* **xSelectedPosition**: controller real world last sensed x coordinate.
* **ySelectedPosition**: controller real world last sensed y coordinate.
* **zSelectedPosition**: controller real world last sensed z coordinate.
* **buttonPressed**: returns true if user has pressed a button on the controller.

**VRHeadset**: a type of Device that sits on the user’s head and over the user’s eyes in order to display the 3D virtual world.

* **rotation**: detects the current rotational position of the user’s head to determine the direction the user is facing.
* **xPosition**: detects the user’s current x coordinate of their head position.
* **yPosition**: detects the user’s current y coordinate of their head position.
* **zPosition**: detects the user’s current z coordinate of their head position.

**RoomConfigFile**: a file type that stores user’s room plans.

* **fileName**: user specified name of the save file.

**RoomPlan**: an object which holds the current dimensions of the room and aggregates placeable objects inside of its defined dimensions.

* **xDimension**: x coordinate boundary of room.
* **yDimension:** y coordinate boundary of room.
* **zDimension**: z coordinate boundary of room.

**PlaceableObject**: a generic parent class for the different types of objects that can be placed in a room by the user during the design process.

* **objectName**: a user specified name for the placed object.
* **xLocation**: the x coordinate of the placed object.
* **yLocation**: the y coordinate of the placed object.
* **zLocation**: the z coordinate of the placed object.
* **rotation**: the rotational position of the placed object.
* **image**: a predefined material to display in the world for the placed object.

**SecuritySensor**: a type of PlaceableObject that detects the presence of unauthorized presence in a room.

* **typeOfSensor**: a string which identifies the sub-type of sensor that is placed such as laser, infrared, camera, etc.

**Window**: a type of PlaceableObject that places a window in an exterior facing wall of the room.

* **openType**: a string which identifies the type of window that is to be placed such as a sliding, casement, bay, etc.

**Door**: a type of PlaceableObject that places a door in a wall of the room.

* **openType**: a string which identifies the type of door to be placed such as exterior, interior, sliding, etc.

**Furniture**: a type of PlaceableObject that places a piece of furniture or room décor in the room.

* **typeOfFurniture**: a string which identifies the sub-type of furniture to be placed in to the room such as house plant, bed, dresser, table, chair, wall hanging, etc.

## State Diagram

The state diagram in Figure 5 shows the high-level behavior of the system. It shows the four major states of operation that the system will enter during normal operation.

The system starts in the initial **Startup State** where a user selects a level to load. This selection transitions from the **Startup State** to the **Render Room State.** In the **Render Room State**, the system will display the room selected by the user and render the room as appropriate for the user based on the sensor signals coming from the hardware devices tracking the user in the real world. The user can then transition to the **Place/Edit Object State** by selecting an object to place from the options menu on their hand controls or by selecting an object to edit by selecting the object in the environment with their hand controls. In the **Place/Edit Object State** the user can place new objects in to the environment or select objects in order to edit their size, rotation, position or remove them from the environment completely. The user transitions from the **Place/Edit Object State** by deselecting their currently selected object they are placing or editing returning to the **Render Room State**. The user can then enter the final **Save and Quit Room State** by selecting to quit the program. The system will save their changes automatically.

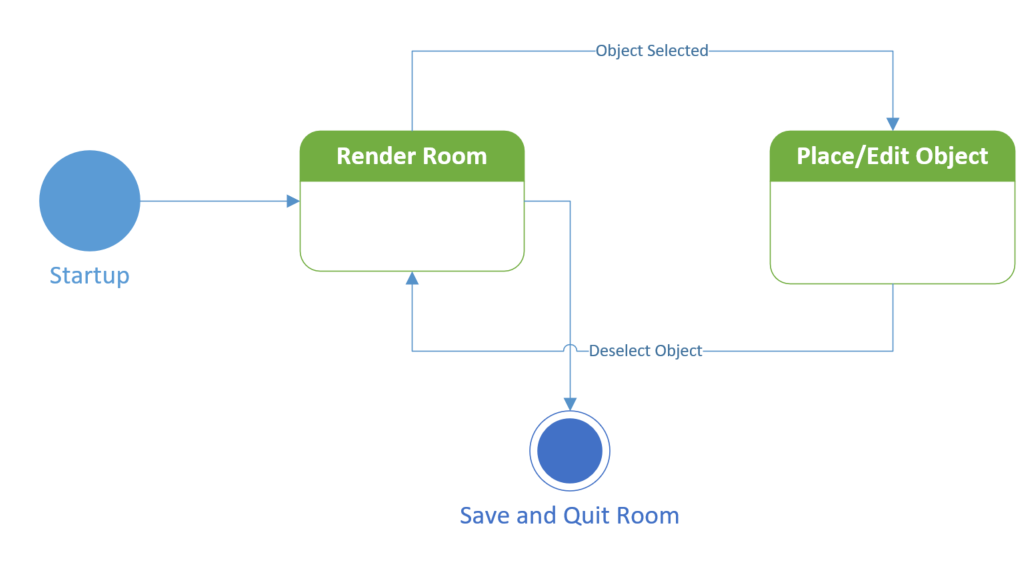


Figure 5. State diagram for the AR Room Designer System.

## Activity Diagram

The activity diagram shown in Figure 6 presents a more detailed description of the high-level behavior of the AR Room Designer Central Control System. Following the same process flow as the state diagram in the previous section, the user loads the system and **Opens a Room File**. This room file can either be new or one that the user has previously created and is now going back in to make changes. The system then forks in order to accomplish two separate processes. One process is to **Load the Room Data**, where the system loads either a default empty room based upon the initial sensor data from the real-world sensors or previous data stored in the room file. The second process is that the system must **Sense the user Position** through the three different types of real world sensors. If the user has moved position, then this will change how the user views the augmented virtual world and the way the room is rendered will need to be changed. These processes merge in order to **Render the Room**, which means displaying the current room as appropriate for the user. The user than has a choice of whether to **Place or Edit Objects** where the user can modify, create, or delete objects in the room. The user can also move which is picked up as movement by the real-world sensors which are constantly monitoring the user **Sensing Position**. Both of these choices require **Rendering the Room** again to the user. The user can also elect to save and quit the program.

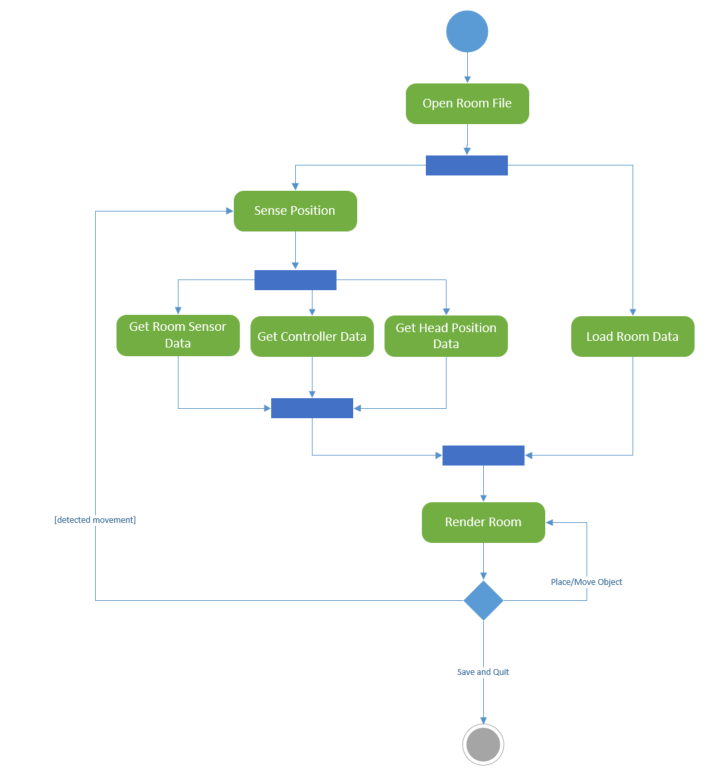


Figure 6. AR Room Designer high-level activity diagram.

# Customer Journey Map

A detailed customer journey map for the AR Room Designer system was created by the team in order to better understand the user experience of the customers (Figure 7). The goal of the customer is to find the optimal design layout for their taste and the minimal sensor placement for their home security system. The waypoints clarify the likely steps of the user as they attempt to accomplish their goal. Often, users may experience roadblocks or setbacks on the path to their goal and they should be modeled as part of the customer journey map. Here, the design team felt that the user process was relatively straightforward and so did not model any setbacks at the current time. It is important to keep in mind that further testing and evolution of the prototype may uncover hidden perils for the customer which would be important to update here in the living document.

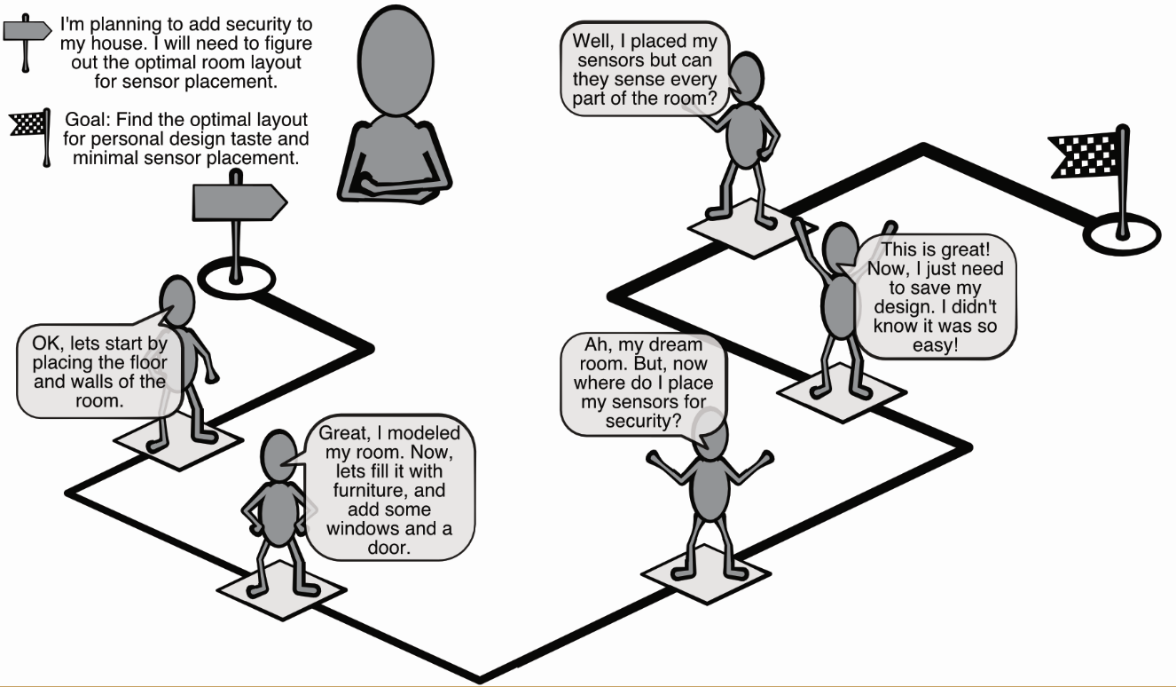


Figure 7. Customer journey map for the AR Room Designer system.

# Personas

A representative sample of a persona for the AR Room Designer system can be found in Figure 8. It is important to have a diverse collection of personas with which to use to evaluate your project. Here the team has provided one persona with the intention of adding more as the project progresses. Although the name and image of the persona is not particularly important, the rest of the information builds a potential customer profile to take in consideration when analyzing the project. For example, Elizabeth as a new VR user that suffers motion sickness might have difficulty adapting to the system. The developers would want to consider offering an in-depth tutorial and tool tips in HUD. The team might also want to add user options for enhanced motion smoothing, slower field of vision changes, and time limit warnings to help users adjust to VR motion. Further as Elizabeth needs help visualizing layouts and lines of sights it might be best to include options for a grid overlay in the AR environment in addition to showing highlighted sensor line of sight when selecting and moving security sensors.

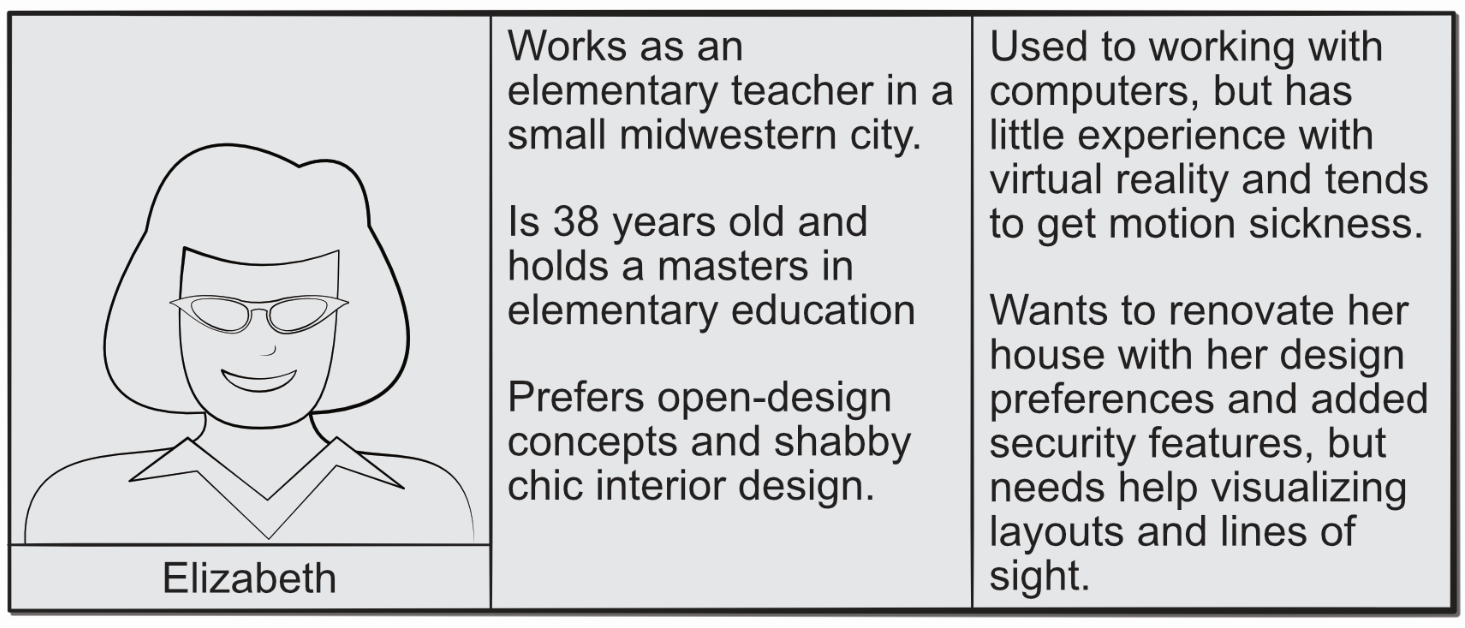


Figure 8. Sample persona to evaluate the AR Room Designer system.

# UI Mock-up

The initial user interface mock-up for the AR Room Designer system is presented in Figure 9. The view presented is as seen through the VR googles of the system. The screen shows the AR of a room in the user’s house. The various editable features have small buttons near their top-left corner which allow the user to reposition or remove the placed feature. The user also has the ability to use their hands to select actions and objects from a control menu that appears as a wrist-watch on the user’s right hand in the AR field of view.

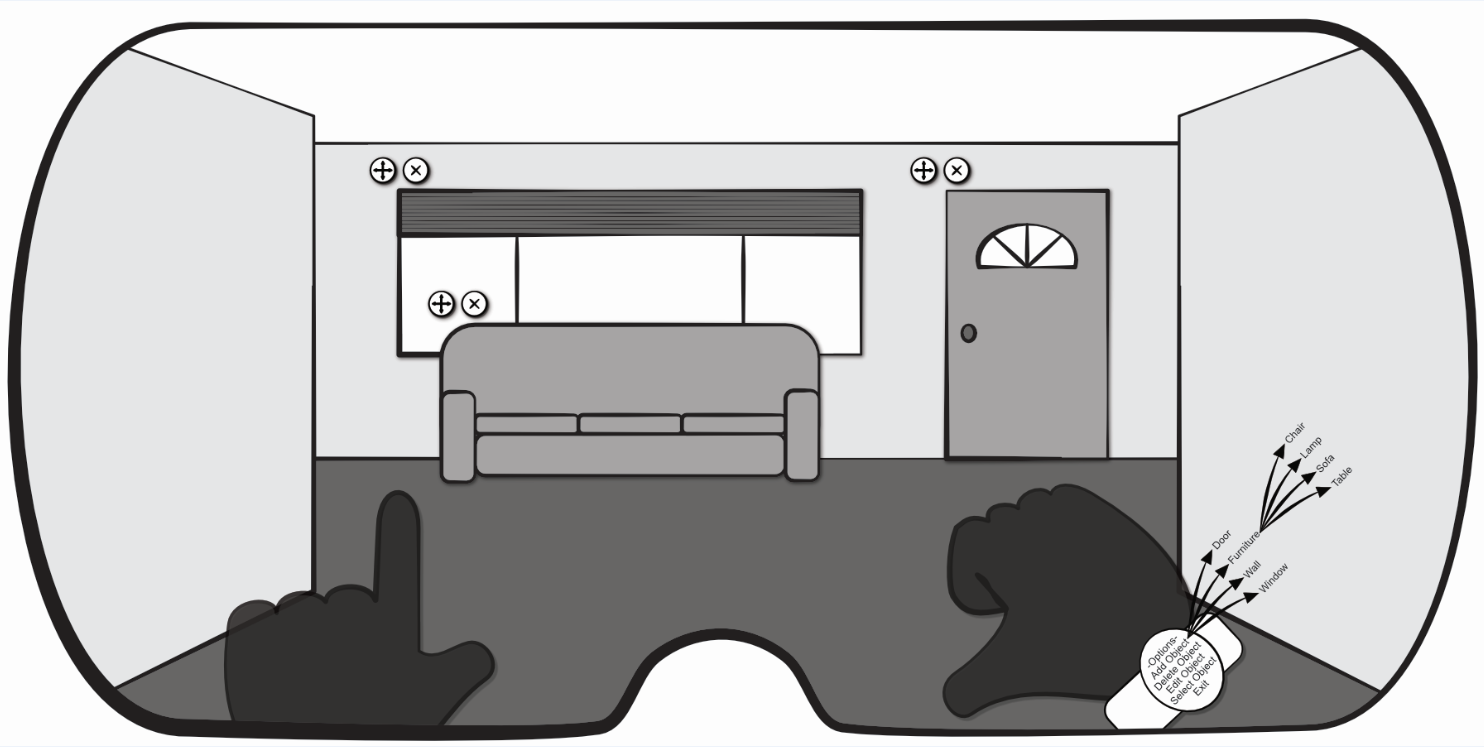


Figure 9. User interface mock-up for AR Room Designer system.

# Testing Strategy

It is important to define our test strategy in order to be able to accurately create our test cases as well as to manage the project timeline and make sure testing is performed by critical deadlines. A well-crafted test strategy will complement the various phases of the software development lifecycle model the team has selected.

The test strategy describes the approach to testing the software team will use in order to ensure the software delivered meets the requirements and quality standards set by the client. The development team have selected four categories of test cases as part of their test strategy: unit testing, integration testing, usability testing, and validation testing.

## Unit Testing

With unit testing we will test the individual methods which are the smallest testable parts of the program. The methods will be tested as they are created to validate the system as it is developed. In object-oriented testing, the class is seen as the smallest unit, yet the smallest testable part remains the methods of the classes. Therefore, regardless of your testing perspective (OOP vs. non-OOP) methods remain the target for unit tests. However, the tactics of testing must change with OOP since it can be difficult to test a single operation in isolation resulting in the need to test sequences of methods involved in a class. Further, if the class is defined as the smallest testable unit it implies that the team will need to use scenario or thread-based testing.

A good test case for a unit test will have a number and name to identify it, as well as a brief description of the test. The expected and actual output are noted. If the actual output of the test matches the expected output of the test then the test status is marked passed. Alternatively, if the actual and expected output do not match then the test is marked failed and indicates a potential bug in the system. Lastly, the test table should have a column that indicates which requirement or user story is associated with the test case.

For this type of unit testing, the developer would take each method defined for the classes listed in Figure 4 and create white box test cases to verify the correct execution of the logic paths in each method. Black box test cases for this type of unit testing, will focus on verifying that the actual method outputs match the expected method outputs for the set of method inputs based on the pre and post conditions for each method.

## Integration Testing

Integration testing ensures that the methods work correctly with each other in addition to ensuring all APIs call work as expected. Integration testing is often performed as automated testing, where test suites of integration tests are executed at some routine time. It is often functionally improbable for a developer to be able to execute an integration test suite on their own due to the number of tests and interconnections between the various software components.

It is important to note that integration testing needs to be performed incrementally so that small increments of a project are added together slowly in order to easier isolate issues and more thoroughly test the components. There are many approaches to incremental integration testing such as top-down, bottom-up, and continuous. Here the team decides to adopt a continuous integration approach where component will be merge in the evolving software once a day with the new daily build of the project which aligns with the team’s evolutionary process model.

Integration testing would be performed by considering each use case one at a time and ensuring the classes needed to accomplish a use case interact correctly. This would involve creating sequences of method calls that are needed to complete each use case correctly and varying the sequences of method calls to show the evolving prototype does not fail in unexpected ways. As each use case is integrated into the prototype regression testing will be performed to ensure new errors have not been introduced. This is the place the automated testing will be invaluable to the development team.

## Usability Testing

Usability is a type of high-order testing. For usability testing the development team is looking to evaluate the degree to which users can effectively interact with the program through its interfaces. As part of usability testing the team would make use of the personas developed in order to view determine scenario-based tests for how representative users may interact with the software.

Typically, the team would conduct a cognitive walk through of the 6 use cases from the perspective of the persona created for Elizabeth. The one of the developers would pretend to be Elizabeth and use the prototype mechanisms to step through each user story. The developer playing Elizabeth would talk aloud and they complete each use case. The other developers would be watching for places where Elizabeth gets stuck and cannot figure out how to accomplish her goal. These pinch points are places where the something in the interface needs to be changed.

## Validation Testing

Validation testing will be performed after integration testing has been passed and a prototype is ready for release during the review of user stories with the customer and when the test cases are reviewed by all the stakeholder on the project. Test cases here should be selected to demonstrate from a user perspective that all functional, non-functional, and other behavioral requirements have been satisfied.

This demonstration of function by the team for the product owner should be guided by the customer journey shown in Figure 7. Ideally, the product owner will have previously confirmed that the use cases are in fact what the system needs to accomplish to meet the customers’ needs. In usability testing, the developers are looking for places that the users fail to accomplish their goals. In validation testing, the developers are seeking to demonstrate that the system use cases as implemented deliver the product owner’s desired functionality and meet the required level of quality.