Contents

[1. Abstract 3](#_Toc467077821)

[2. Report Revision History 4](#_Toc467077822)

[2.1 Changes in Version 1.5 4](#_Toc467077823)

[2.2 Changes in Version 2.0 4](#_Toc467077824)

[3. Problem Statement 5](#_Toc467077825)

[3.1 Business Background 5](#_Toc467077826)

[3.2 Needs 5](#_Toc467077827)

[3.3 Objectives 5](#_Toc467077828)

[4. Requirements 6](#_Toc467077829)

[4.1 User Requirements 6](#_Toc467077830)

[4.1.1 Glossary of Relevant Domain Terminology 6](#_Toc467077831)

[4.1.2 User Groups 6](#_Toc467077832)

[4.1.3 Functional Requirements 6](#_Toc467077833)

[4.1.4 Non-functional Requirements 12](#_Toc467077834)

[4.2 System Requirements 14](#_Toc467077835)

[4.2.1 Functional Requirements 15](#_Toc467077836)

[4.2.2 Non-functional Requirements 22](#_Toc467077837)

[4.3 Requirements Trace Table 23](#_Toc467077838)

[5. Exploratory Studies 25](#_Toc467077839)

[5.1 Relevant Techniques 25](#_Toc467077840)

[5.2 Relevant Packages/Products 25](#_Toc467077841)

[5.3 Broader Impacts 25](#_Toc467077842)

[6. System Design 26](#_Toc467077843)

[6.1 Architectural Design 26](#_Toc467077844)

[6.2 Structural Design 26](#_Toc467077845)

[6.3 User Interface Design 31](#_Toc467077846)

[6.4 Behavioral Design 34](#_Toc467077847)

[6.5 Design Alternatives & Design Rationale 35](#_Toc467077848)

[7. System Implementation 37](#_Toc467077849)

[7.1 Programming Languages & Tools 37](#_Toc467077850)

[7.2 Coding Conventions 37](#_Toc467077851)

[7.3 Code Version Control 37](#_Toc467077852)

[7.4 Implementation Alternatives & Decision Rationale 37](#_Toc467077853)

[7.5 Analysis of Key Algorithms 37](#_Toc467077854)

[8. System Testing 38](#_Toc467077855)

[8.1 Test Automation Framework 38](#_Toc467077856)

[8.1.1 Steps for Installing Test Framework 38](#_Toc467077857)

[8.1.2 Steps for Running Test Cases 38](#_Toc467077858)

[8.2 Test Case Design 38](#_Toc467077859)

[8.2.1 Acceptance Test Cases 38](#_Toc467077860)

[8.2.2 System Test Cases 40](#_Toc467077861)

[8.2.3 Integration Test Cases 41](#_Toc467077862)

[8.2.4 Unit Test Cases 42](#_Toc467077863)

[8.3 Test Case Execution Report 42](#_Toc467077864)

[8.3.1 Unit Testing Report 43](#_Toc467077865)

[8.3.2 Integration Testing Report 44](#_Toc467077866)

[8.3.3 System Testing Report 45](#_Toc467077867)

[8.3.4 Acceptance Testing Report 46](#_Toc467077868)

[9. Challenges & Open Issues 49](#_Toc467077869)

[9.1 Challenges Faced in Requirements Engineering 49](#_Toc467077870)

[9.2 Challenges Faced in System Development 49](#_Toc467077871)

[9.3 Open Issues & Ideas for Solutions 49](#_Toc467077872)

[10. System Manuals 50](#_Toc467077873)

[10.1 Instructions for System Development 50](#_Toc467077874)

[10.1.1 How to Set Up Development Environment 50](#_Toc467077875)

[10.1.2 Notes on System Further Extensions 50](#_Toc467077876)

[10.2 Instructions for System Deployment 50](#_Toc467077877)

[10.2.1 Platform Requirements 50](#_Toc467077878)

[10.2.2 System Installation 50](#_Toc467077879)

[10.3 Instructions for System End Users 50](#_Toc467077880)

[11. Conclusion 51](#_Toc467077881)

[11.1 Achievement 51](#_Toc467077882)

[11.2 Lessons Learned 51](#_Toc467077883)

[11.3 Acknowledgment 51](#_Toc467077884)

[12. References 52](#_Toc467077885)

# Abstract

Erie Insurance currently works with its agents to help them display the dangers of distracted driving to their policyholders. This can often be very difficult for agents to do since the user is not able to experience the consequences of distracted driving for themselves in a safe way. In order to help solve this problem for the agents, we are creating a virtual reality experience to demonstrate how distracted driving can affect the policyholder. This virtual reality experience will utilize the Unity 3D engine and the Google Cardboard SDK to give the policyholder different scenarios in which they will have to make decisions influencing their outcome. This virtual reality experience will help the policyholder to understand how they can influence dangerous driving activities as well as to help stop them.

# Report Revision History

## Changes in Version 1.5

In this version, we have made the changes recommended to us by our advisor. We have added a new user requirement and functional requirement detailing more information regarding the specific tasks that the AI driver should perform. The use case mapping diagram has updated as well. Along with that, we have changed the name of our use case “Begin Experience” to “Experience Loop” to make more sense. References are now available and are used in section 5 to further explain our exploratory studies.

## Changes in Version 2.0

In this version, we have added our initial designs for the architecture, structure, interface, and behavior of the system. We have changed our architecture to the component-based architecture, which more accurately captures the way Unity objects build off each other to create the overall system. We have added and updated our requirements based off feedback from advisors and industry mentor. We have created test cases for our system, as well as the execution history. We have added the steps to set up the development environment and testing environment, build for the target platform, and install to the end user’s device.

2.3 Changes in Version 2.5

In this version, we have made further changes recommended to us by our advisor. We have modified the layout of the report in section 6.2 to better organize the descriptions that go with each individual image. We have also updated section 6.3 to be contained within one page for further formatting improvements.

# Problem Statement

## Business Background

Erie Insurance is a Fortune 500 insurance company employing thousands of people. Erie Insurance has been a figure in the insurance world for 90 years, and currently serves over 4 million customers in 13 states. They utilize and manage smaller agencies to deal directly with customers, selling them auto, home, life, and business insurance.

With the rise of technology, distracted driving has become more of a risk than ever before. As Erie Insurance invests in protecting people, they are taking the initiative in informing families about the dangers of driving while distracted.

## Needs

Currently, it is very difficult to display the dangers of distracted driving to a younger generation in a way that engages them. Erie Insurance is seeking an innovative solution in order to solve this problem.

## Objectives

This project aims to utilize virtual reality technology to create an immersive experience that engages users of all ages. The application will be distributed to agents around Erie's footprint and will effectively capture the younger audience.

# Requirements

## User Requirements

### Glossary of Relevant Domain Terminology

Virtual Reality (VR) – A simulation of a three dimensional environment

Cardboard – Google’s SDK created for smartphone devices

Headset – A head mounted device that displays virtual reality devices

Scene – A Unity scene is an aggregation of components that can be executed on its own

### User Groups

User – Any person engaging in our experience

### Functional Requirements

#### Project Scope (Use Case Diagram)

Figure 4.1 displays the system’s use case diagram. This gives a layout of the main user interactions that can occur as they use the system.



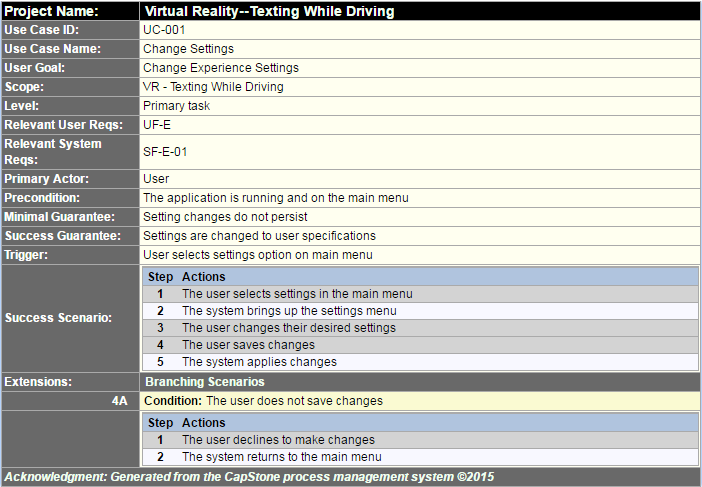
**Figure 4.1 - Use Case Diagram**

#### User Scenarios

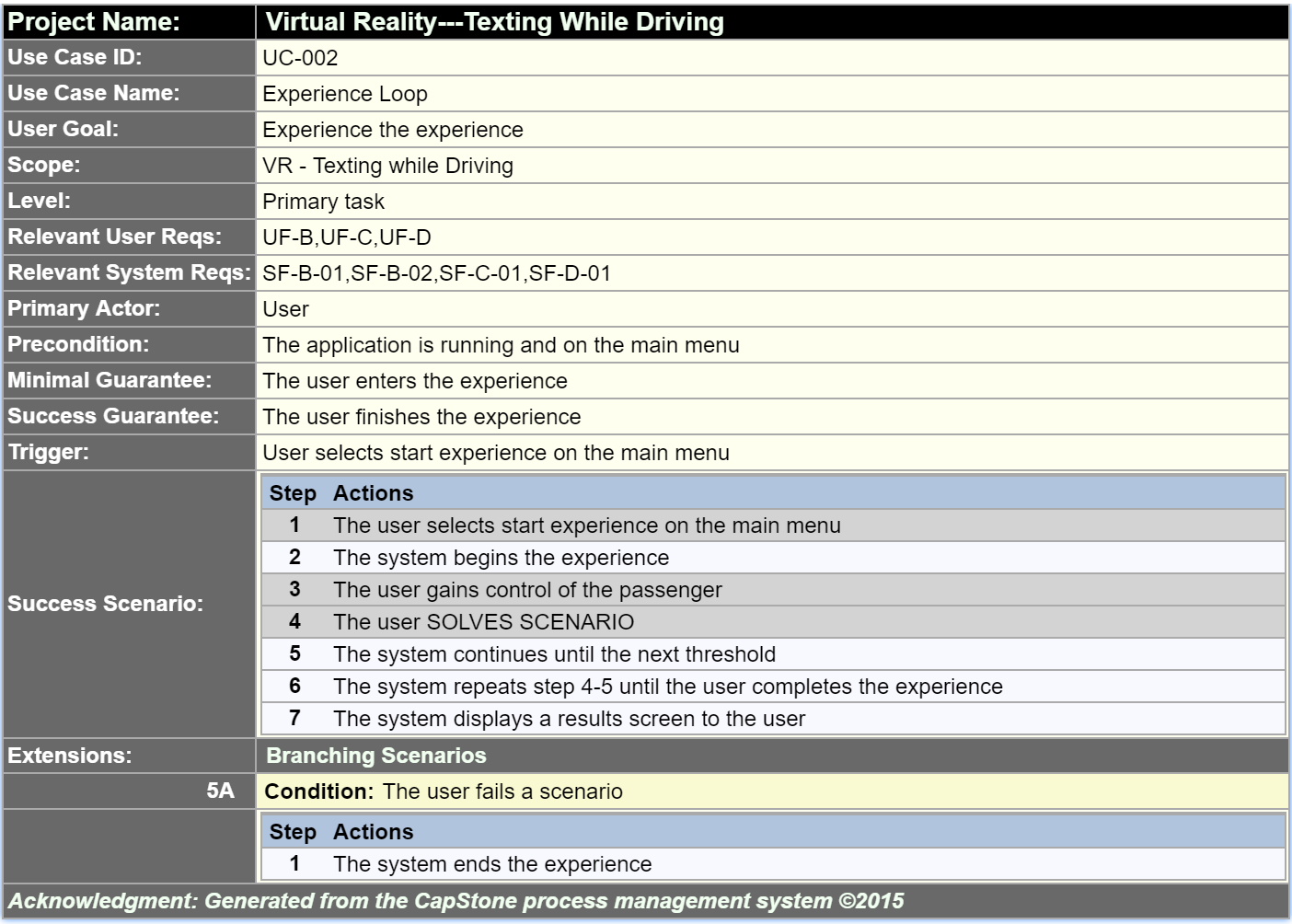
Figure 4.2 lists the details of the use cases that occur within the system. The use cases give an overview of the sequence of the interactions that occur with the user and the system.



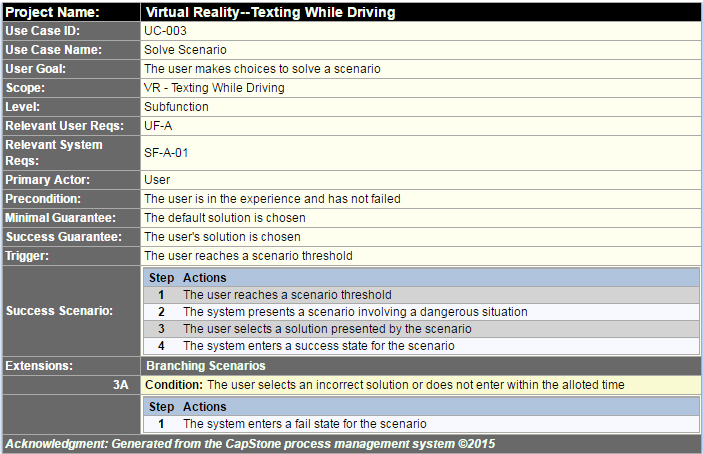
**Figure 4.2 - Use Case List**



**Figure 4.3 - Change Settings**



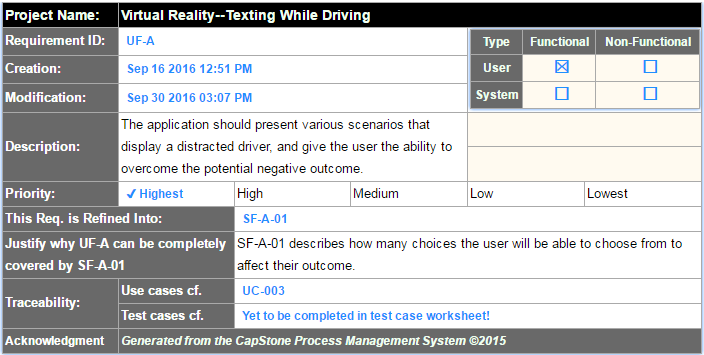
**Figure 4.4 – Experience Loop**



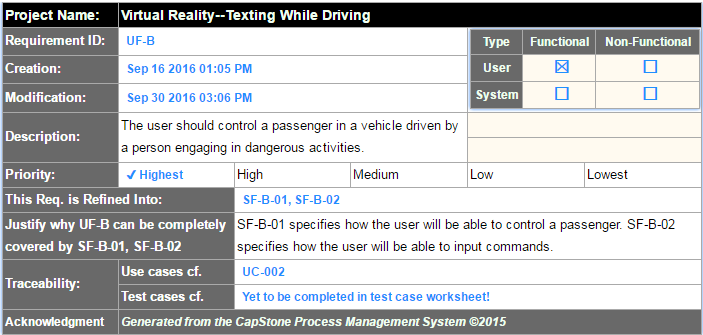
**Figure 4.5 - Solve Scenario**

#### List of User Functional Requirements

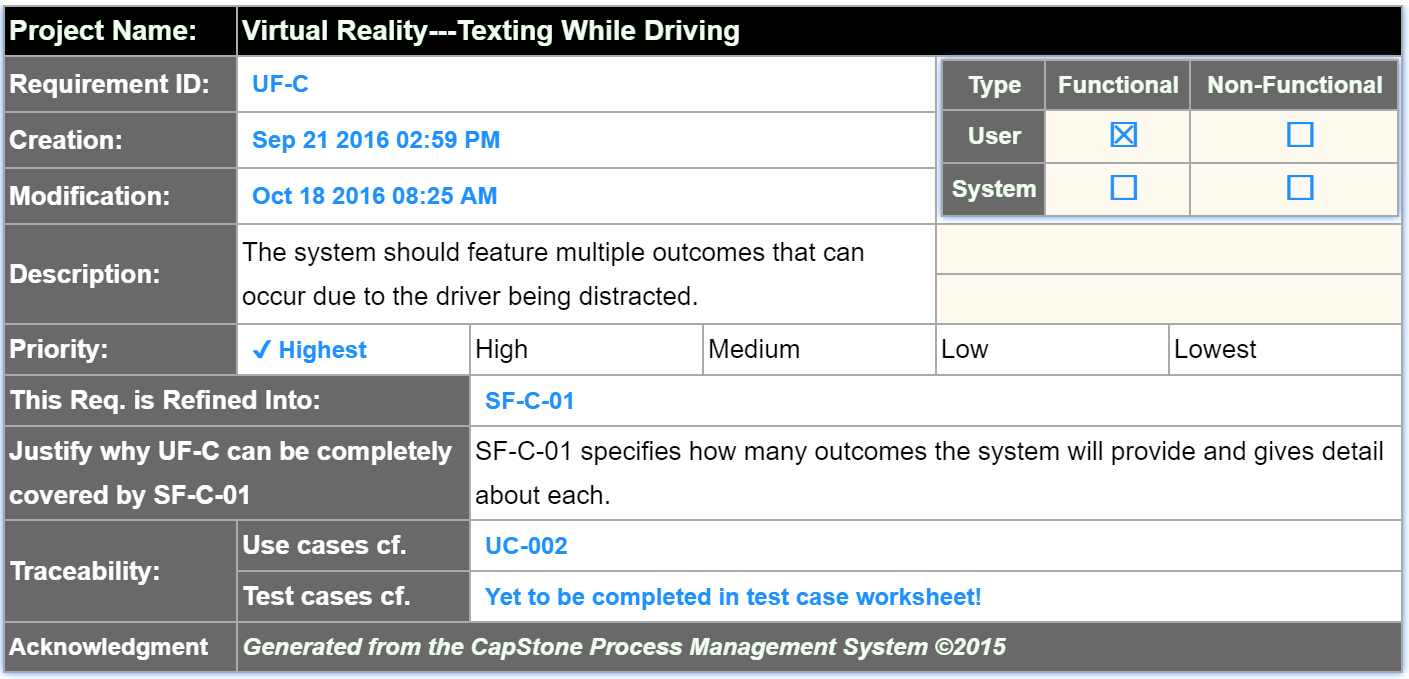
User functional requirements describe functionality that the system should provide.



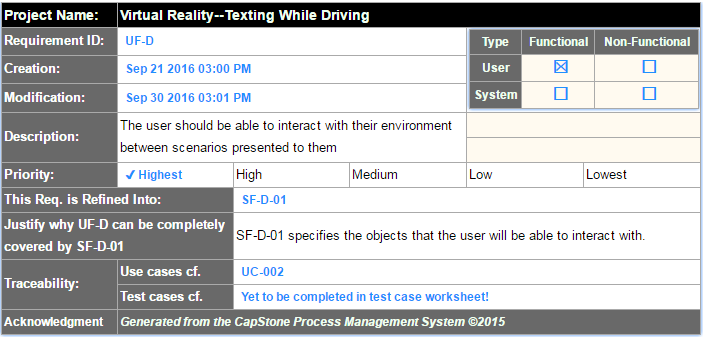
**Figure 4.6 - Requirement UF-A**



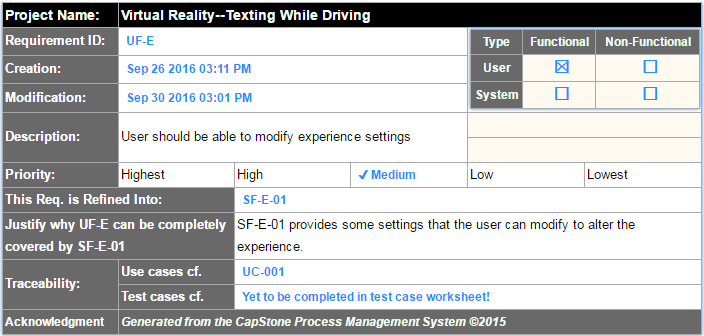
**Figure 4.7 - Requirement UF-B**



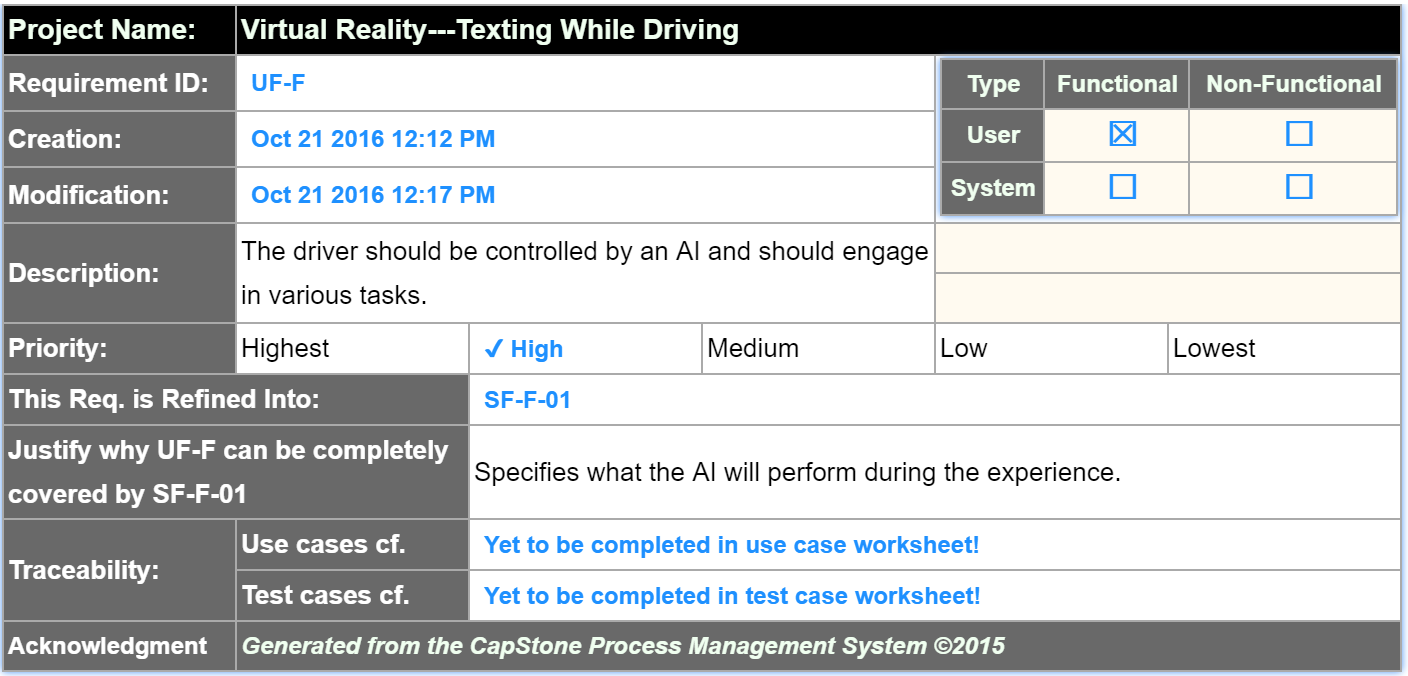
**Figure 4.8 - Requirement UF-C**



**Figure 4.9 - Requirement UF-D**



**Figure 4.10 - Requirement UF-E**



**Figure 4.11 - Requirement UF-F**

### Non-functional Requirements

Non-functional requirements describe the constraints and quality of the functionalities, providing testable features and specifying restrictions.

#### Product: Usability Requirements

Usability requirements describe how easily a user interacts with the system.

#### Product: Performance Requirements

Performance requirements describe how well a system performs in terms of time and resource usage.



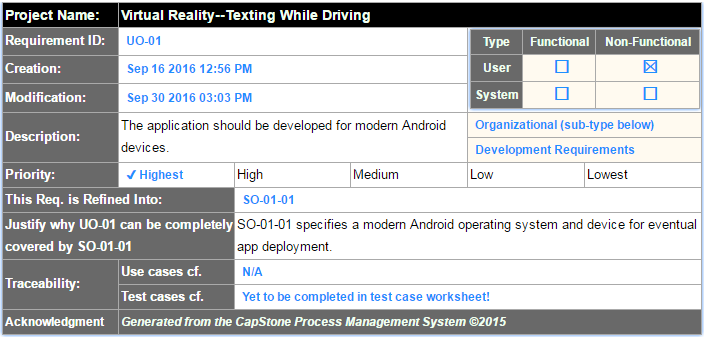
**Figure 4.12 - Requirement UP-01**

#### Product: Dependability/Security Requirements

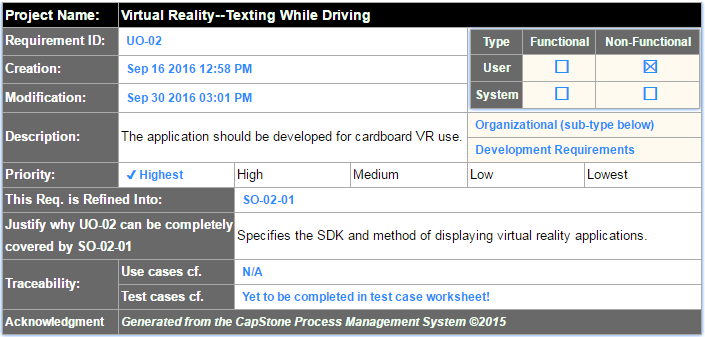
Dependability/Security requirements describe the reliability and security concerns of the project.

#### Organizational: Development Requirements

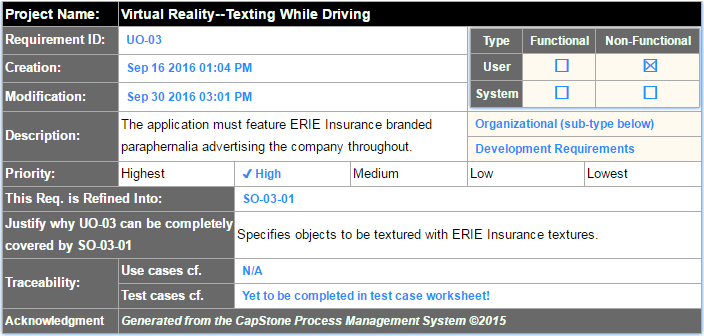
Development requirements specify development practices and constraints.



**Figure 4.13 - Requirement UO-01**



**Figure 4.14 - Requirement UO-02**



**Figure 4.15 - Requirement UO-03**

#### Organizational: Operational Requirements

Operational requirements describe conditions that a system must support.

#### Organizational: Environmental Requirements

Environmental requirements describe the look and feel of the system’s interface.

#### External: Safety/Security Requirements

Safety/Security requirements detail how the system will interact with other systems, and the security concerns of these interactions.

#### External: Cultural and Social Requirements

Cultural and social requirements describe how the system conforms to cultural and social expectations.

#### External: Political Requirements

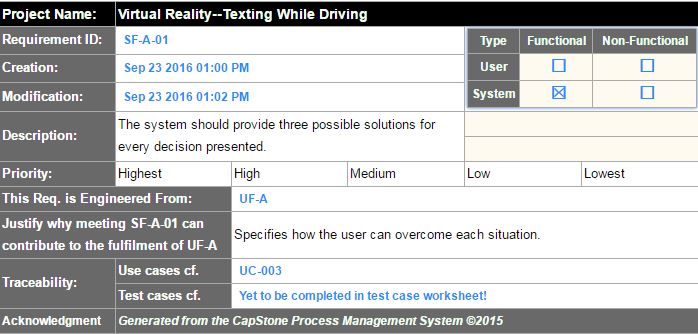
Political requirements detail how the system will influence different sections of the company.

## System Requirements

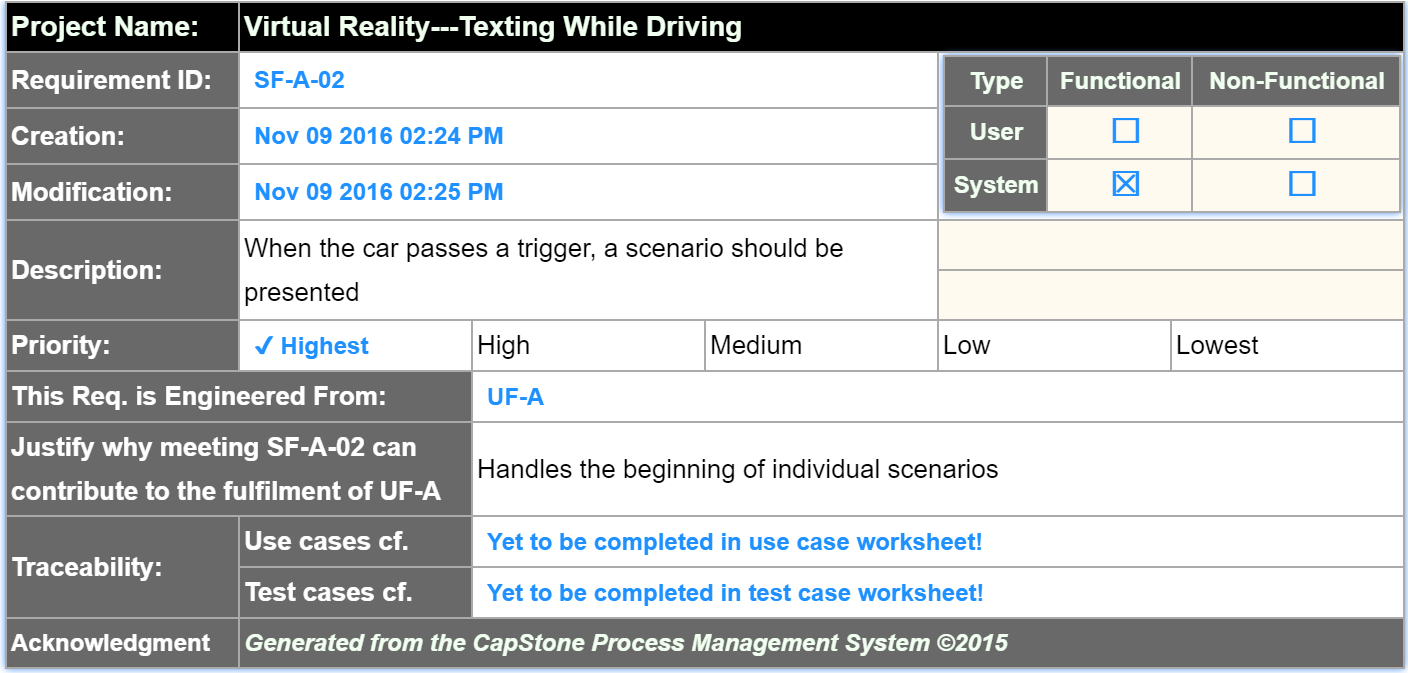
User requirements tend to be vague, so they are refined into system requirements. System requirements engineer and refine the user requirements into many detailed requirements that are much more descriptive and implementable.

### Functional Requirements

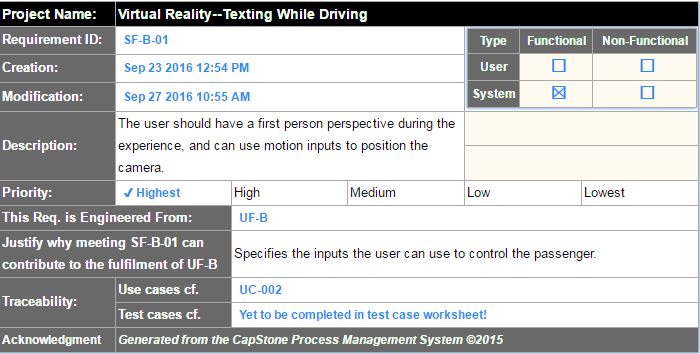
#### List of System Functional Requirements



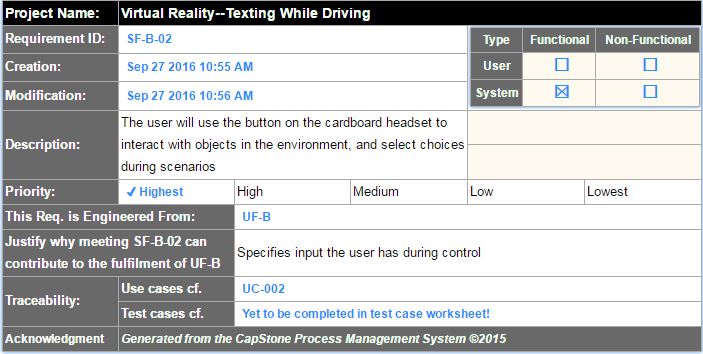
**Figure 4.16 - Requirement SF-A-01**



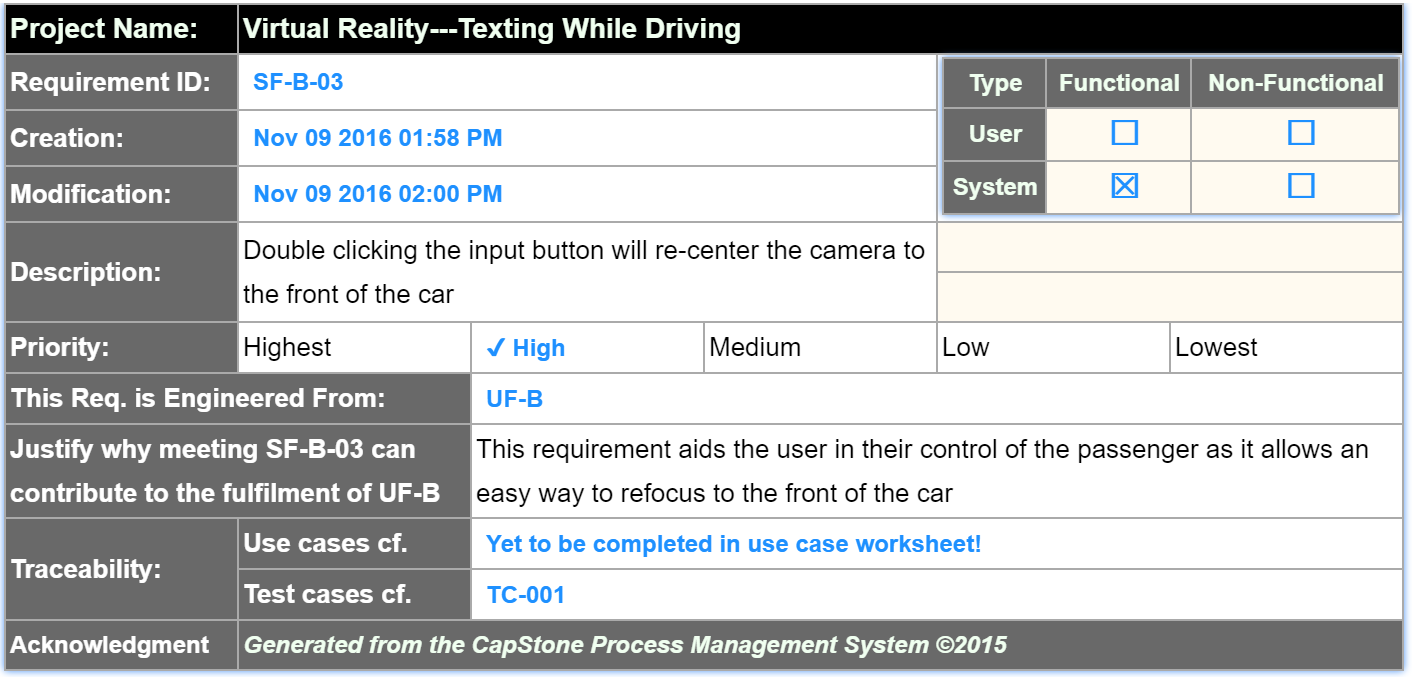
**Figure 4.17 - Requirement SF-A-02**



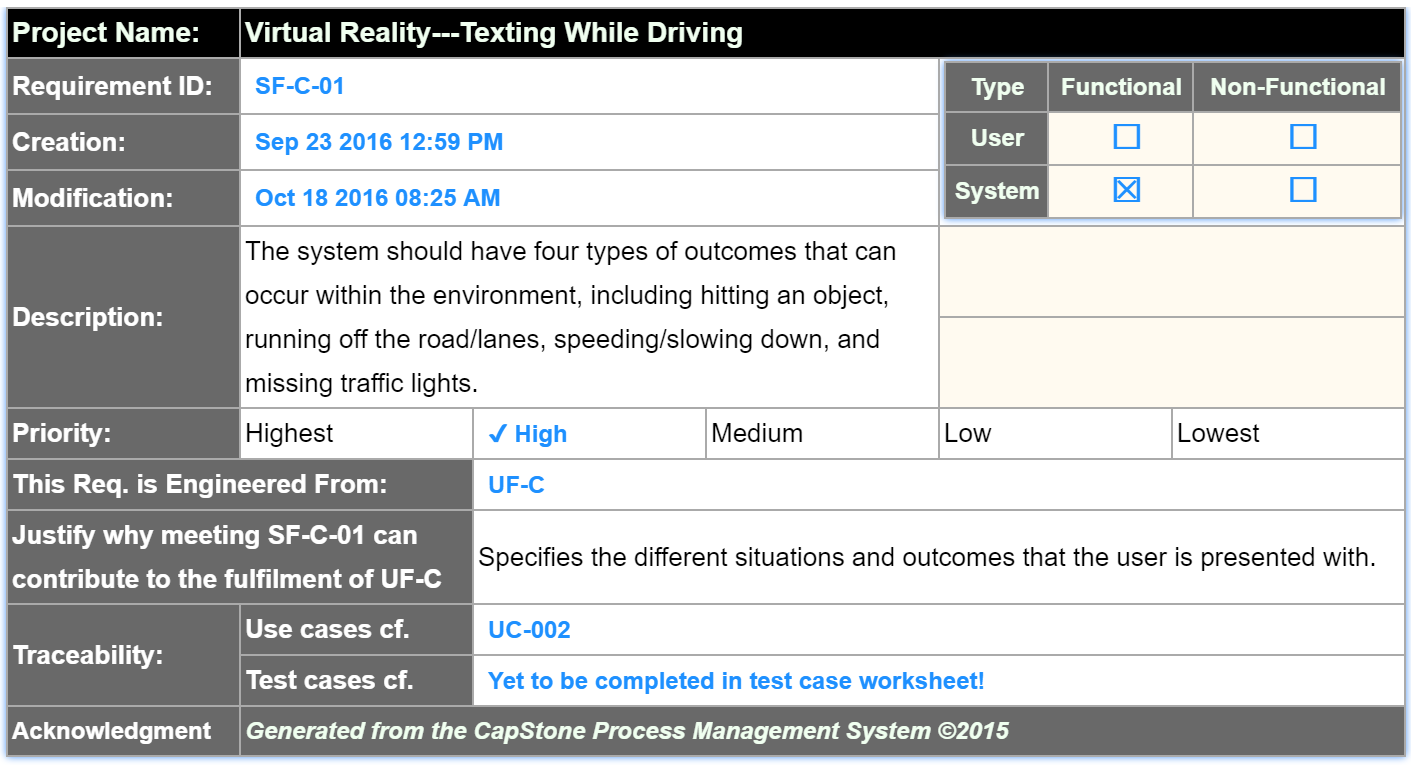
**Figure 4.18 - Requirement SF-B-01**



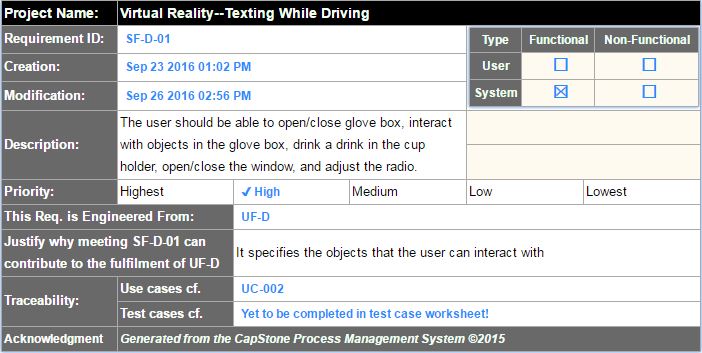
**Figure 4.19 - Requirement SF-B-02**



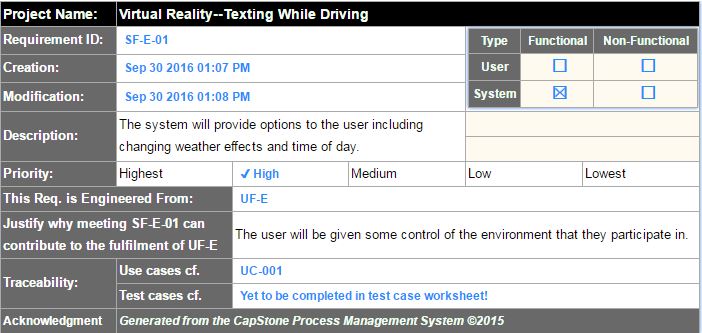
**Figure 4.20 - Requirement SF-B-02**



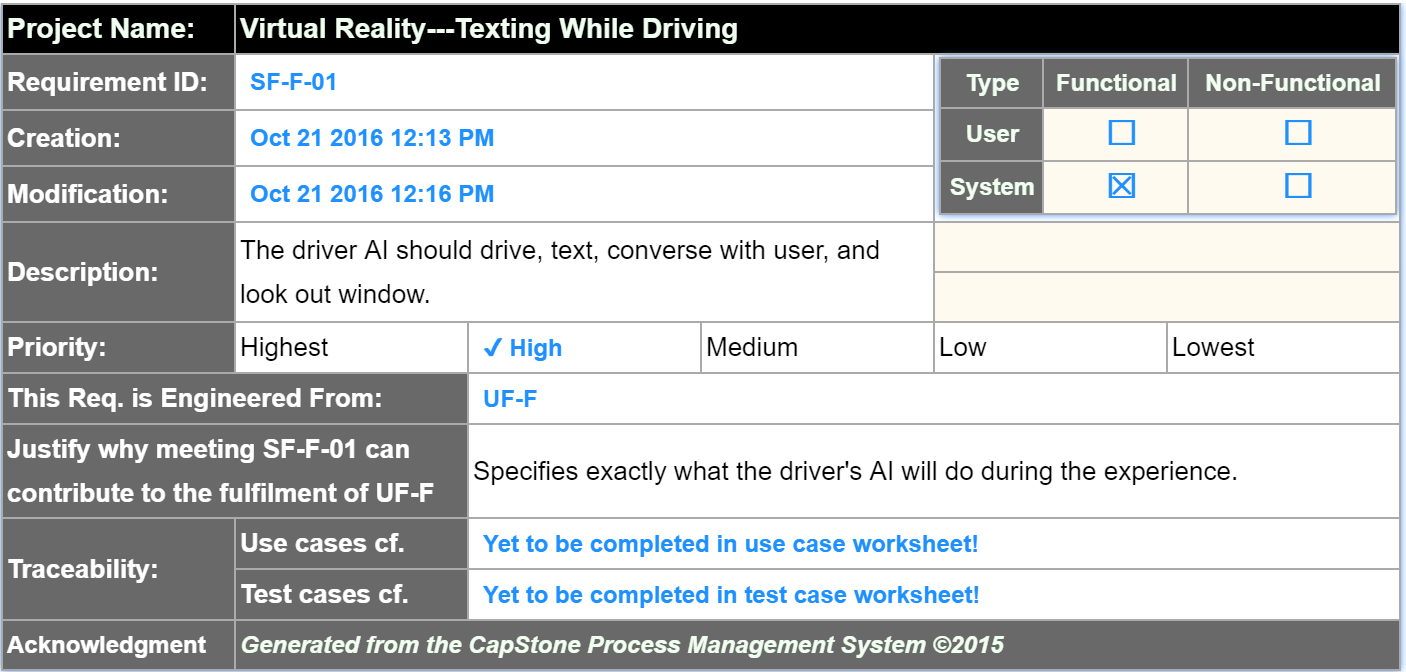
**Figure 4.21 - Requirement SF-C-01**



**Figure 4.22 - Requirement SF-D-01**



**Figure 4.23 - Requirement SF-E-01**



**Figure 4.24 - Requirement SF-F-01**

#### System Behavior

Figures 4.23 and 4.24 detail the sequence of flow between user and system, much like use cases. However, they give a more detailed look into the system, providing interaction between components in the system as well.



**Figure 4.25 - Experience Loop Sequence**



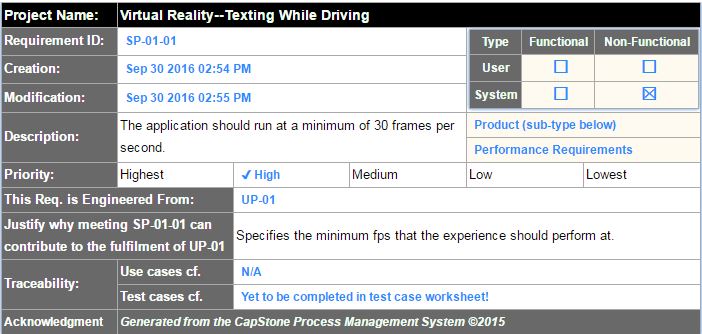
**Figure 4.26 - Change Settings Sequence**

#### Data Requirements

### Non-functional Requirements

#### Product: Usability Requirements

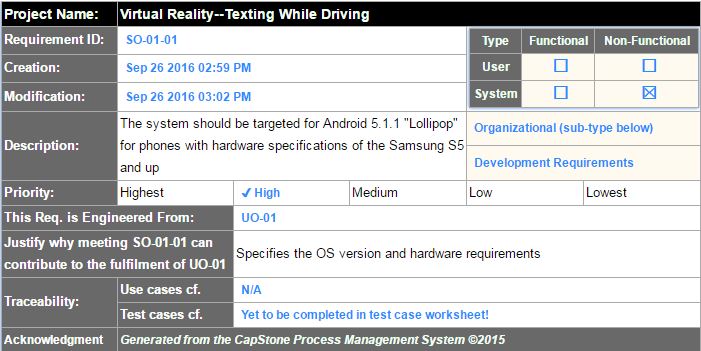
#### Product: Performance Requirements



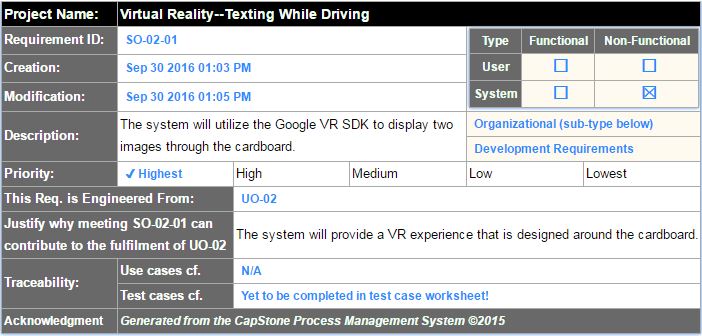
**Figure 4.27 - Requirement SP-01-01**

#### Product: Dependability/Security Requirements

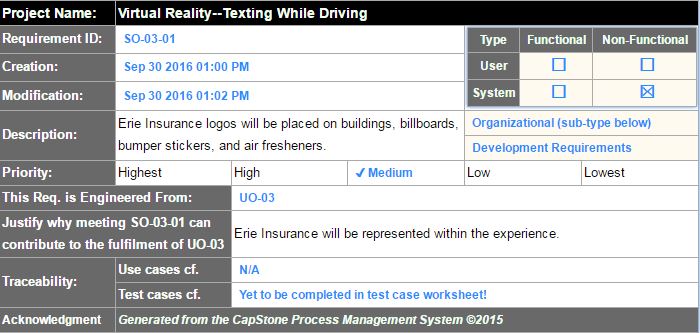
#### Organizational: Development Requirements



**Figure 4.28 - Requirement SO-01-01**



**Figure 4.29 - Requirement SO-02-01**



**Figure 4.30 - Requirement SO-03-01**

#### Organizational: Operational Requirements

#### Organizational: Environmental Requirements

#### External: Safety/Security Requirements

#### External: Cultural and Social Requirements

#### External: Political Requirements

## Requirements Trace Table

Figure 4.29 gives a breakdown of the system requirements that have been engineered from the user requirements.

**Figure 4.31 - Requirement Trace Table**

# Exploratory Studies

## Relevant Techniques

We will be using the Unity 3D game engine to create our application. We have chosen this engine because of its C# scripting, large community, and because it allows us to create an immersive VR experience very quickly. Along with Unity 3D, we will be using the Google VR SDK for Unity to adapt our project for VR use [6]. We also plan to take advantage of the Unity Asset Store to collect models, animations, and scripts to allow us to focus on implementing the requested features and not worry about having to create all of our assets from scratch. Within the Asset Store exists an important package called Unity Test Tools [4]. Unity Test Tools allows us various ways of testing including unit tests, integration tests, and assertion component to make sure our work is as bug free as possible. All of these technologies working together will allow us to create an experience that puts the user into the middle of a seemingly dangerous situation.

## Relevant Packages/Products

The main products and packages we will be using include Unity 3D, Google VR SDK, a variety of assets from the Unity Asset Store, the Android SDK to build from within the Unity engine, Unity Test Tools to complete our application testing, Visual Studio for writing C# scripts, and potentially more as we move forward.

## Broader Impacts

This virtual reality experience has the potential to help minimize distracted driving. Minimizing distracted driving means that there will be less accidents, less injuries, and less deaths because of distracted driving. Since the application runs on the Android operating system, which is used by millions of people every day, this application has the potential to reach a large number of drivers and passengers.

# System Design

## Architectural Design

The system will be using a component-based architectural design, which emphasizes the creation of components, which other components reuse to create a scene. Multiple scenes are sequenced together to create the overall system. Unity objects are a component that is self-contained, meaning that it can run on its own inside a scene. As objects are defined, they can be used in other objects to create large components that are combined to create complex scenes. Figure 6.1 shows our high-level architecture, which is consisting of a starting interface *GameObject* that has a composition with itself to allow the components to have other components that make it up.

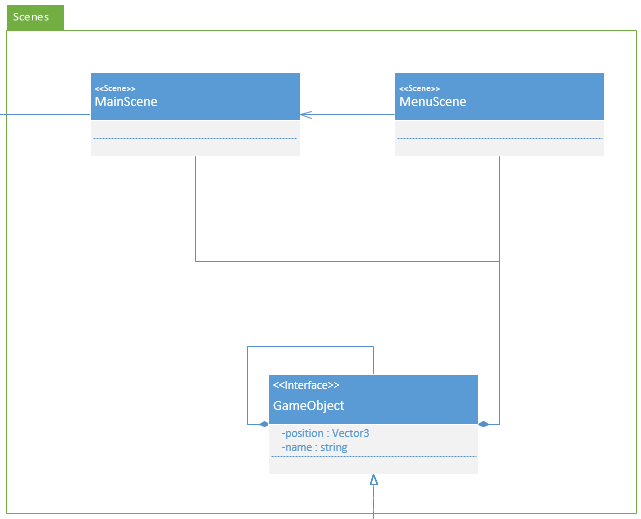


**Figure 6.1 - Architectural Design**

## Structural Design

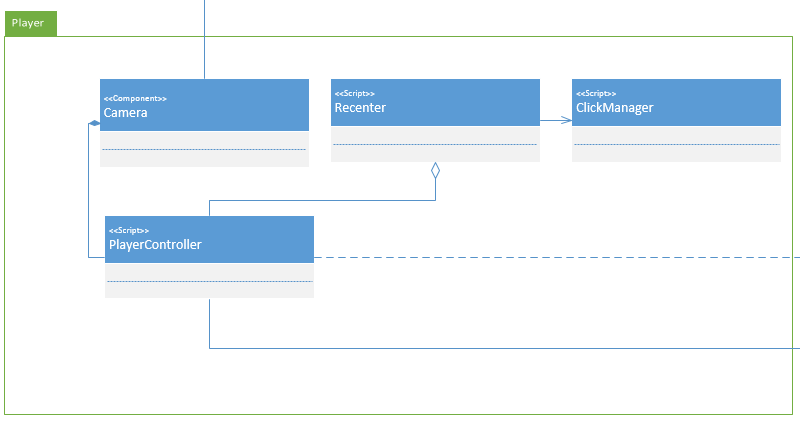
The structural diagram provides the detailed components that are defined in the architectural design. The basic components are refined into each individual component that can be reused to create the overall layout of the Unity scene.

Figure 6.2 represents the Scenes package within the structural diagram. This package will contain each scene within the experience and show how they connect to each other. This package also contains the main GameObject interface that all other components will be inheriting from throughout the system.



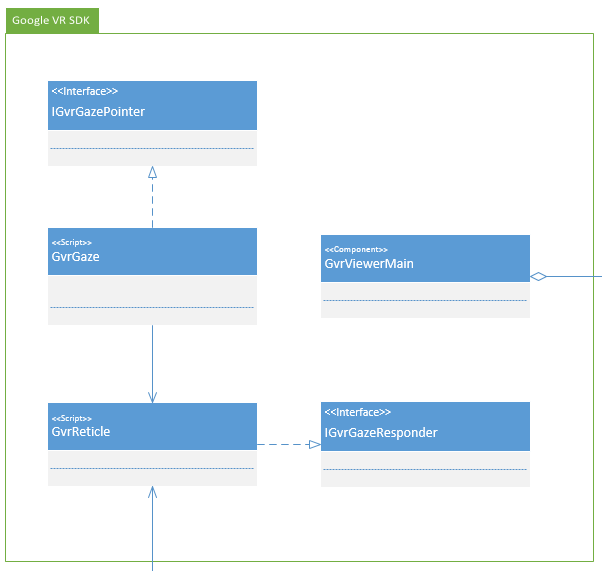
**Figure 6.2 - Structural Diagram (Scenes)**

Figure 6.3 is the package representing the player. The player is essentially a camera component using a player controller script to allow the user to move their head around to view and interact with what is happening in the scene. The player controller is able to perform actions with the environment such as reentering the user, which is shown below.



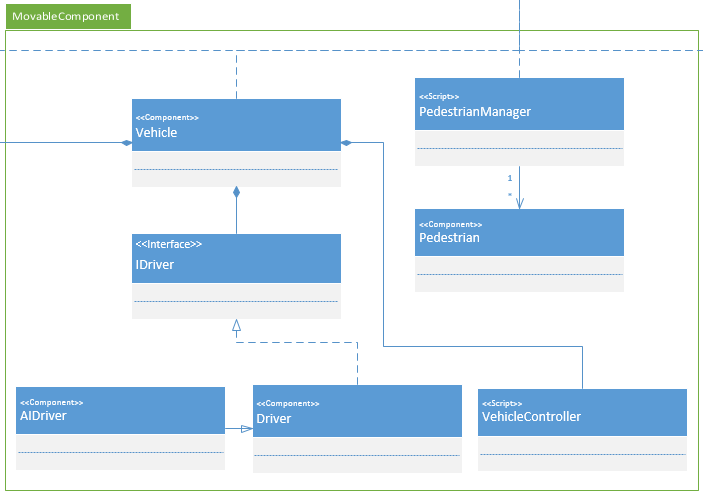
**Figure 6.3 - Structural Diagram (Player)**

Figure 6.4 shows many of the components that come from Googles VR SDK. As mentioned above, the player is a camera that is able to interact with the environment. To do this, the camera utilizes components, interfaces, and scripts in this package. This package allows components to be set as either objects causing interactions to happen or allows components to be the object that is interacted with.



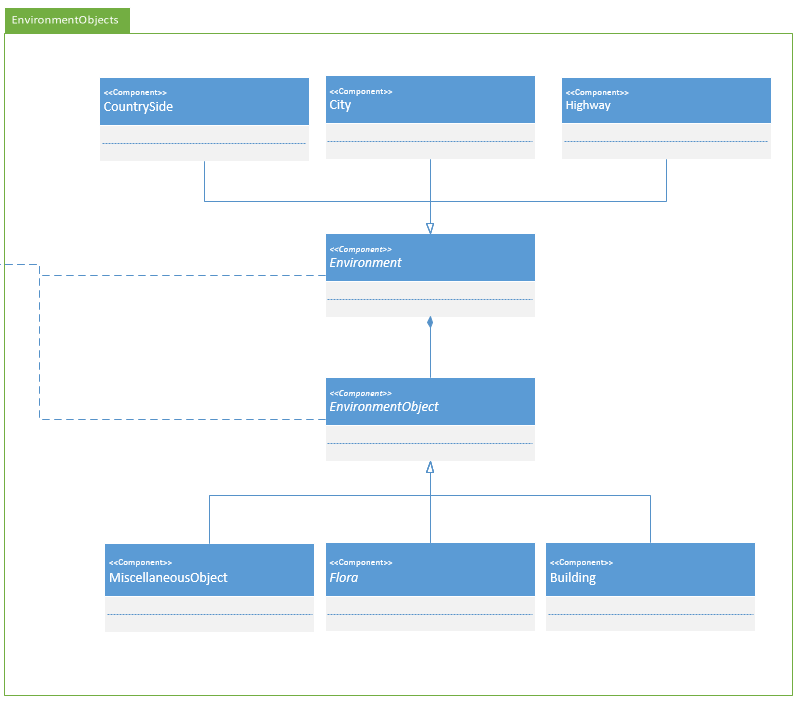
**Figure 6.4 - Structural Diagram (Google VR SDK)**

Figure 6.5 shows the MovableComponent package which consists of all components that will be moving in some way during the execution of the program. This package includes pedestrians (people, animals, etc.) and vehicles. The package also contains the scripts that these components will rely on to perform their movement and coordination.



**Figure 6.5 - Structural Diagram (MovableComponent)**

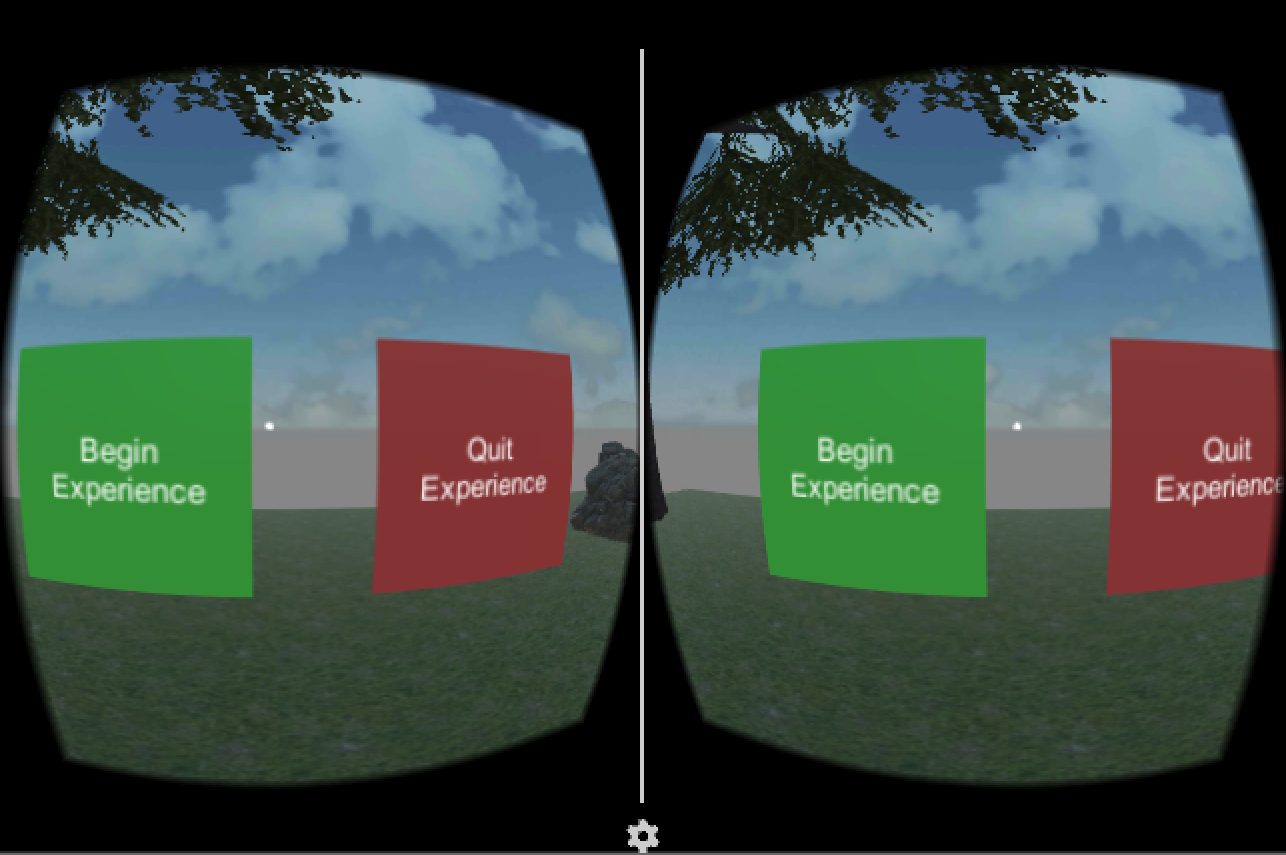
Figure 6.6 shows the EnvironmentalObjects package which contains objects that are non-moving and exist in the environment such as plants, buildings, and roadways. The hierarchy below demonstrates how full environments will be made up of smaller components such as what was listed previously.



**Figure 6.6 - Structural Diagram (EnvironmentObjects)**

## User Interface Design

Our user interface design is built around the technologies we are implementing. Virtual reality has a defined structure of displaying an image on two separate screens with logical angles that simulate what eyes see. With that, we are trying to create a very realistic depiction of riding in the car with a friend while the friend engages in dangerous activities. Google’s SDK has provided many useful assets that have helped create the menu screens and input management to allow the player to control the experience. Figures 6.7 through 6.9 show the view the player has throughout the experience.



**Figure 6.7 - Start Menu View**



**Figure 6.8 - Outside View**



**Figure 6.9 - Driver View**

## Behavioral Design

In Figure 6.10, the behavior of the system is displayed. The activity diagram shows the flow of the experience and gives the steps required to succeed in the system, as well as the fail state requirements.



**Figure 6.10 - Activity Diagram**

## Design Alternatives & Design Rationale

With our project, we are using Unity to create an experience that can run on mobile devices. Unity is designed with the component-based architecture in mind, and the way objects are implemented is based around that concept. Initially we looked into MVC, which is similar to our current design. However, each component in Unity essentially has its own model, view, and controller. The design would be complicated, and would not be as accurate as the component-based architecture.

# System Implementation

## Programming Languages & Tools

We are implementing our project using Unity, which takes advantage of C# for creating scripts. Unity provides an IDE called MonoDevelop, however we are using Microsoft Visual Studio, which can be used instead of MonoDevelop.

## Coding Conventions

We will adhere to the coding conventions designed around Unity development as well as Microsoft’s C# conventions. We will also be following Unity best practice for component design, which will help improve maintainability and performance.

## Code Version Control

As with all projects being worked on by multiple personnel, version control is very important for the efficiency of our workflow. We will be using a combination of Git and Unity SmartMerge for our version control which will handle branching and merge conflicts. We will be hosting our repository in GitHub.

## Implementation Alternatives & Decision Rationale

One alternative development tool we could have used instead of Unity is Unreal Engine. Unreal is another game engine that is widely available and features mobile development and also has Google Cardboard SDK support. With Unreal we would also be developing using C++ instead of C#. Our team decided to use Unity over Unreal because we are all more familiar with C# and virtual reality development is more popular with Unity, so the documentation and resources available will be better defined. Erie Insurance has stated that they are aware of the terms of service with Unity are we are still fine to proceed with development.

## Analysis of Key Algorithms

N/A

# System Testing

## Test Automation Framework

Our project is developed following the test-driven development methodology. In section 8, we will be covering the tests designed for our application as we continue developing it. In order for our system requirements to be verified, there will be tests created for each one to ensure correct implementation.

### Steps for Installing Test Framework

Our tests are designed using Unity Test Tools, which is an asset that allows assertions on Unity objects and scripts to verify that everything is working correctly. In order to install the testing framework, all that needs to be done is download Unity Test Tools from the Unity Asset Store and add it to an existing project.

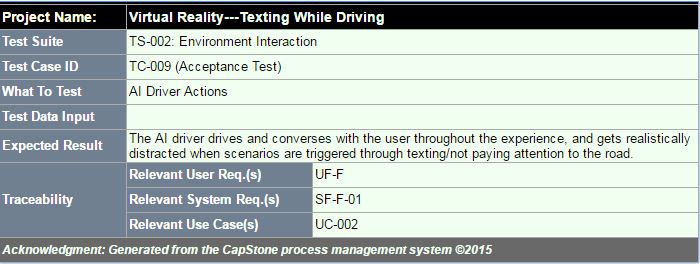
### Steps for Running Test Cases

In order to run a test case, the test case must be opened in Unity Test Tools. From there, the tests can be ran or modified to specified settings.

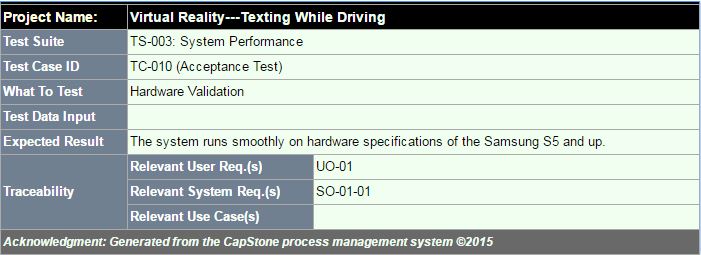
## Test Case Design

### Acceptance Test Cases

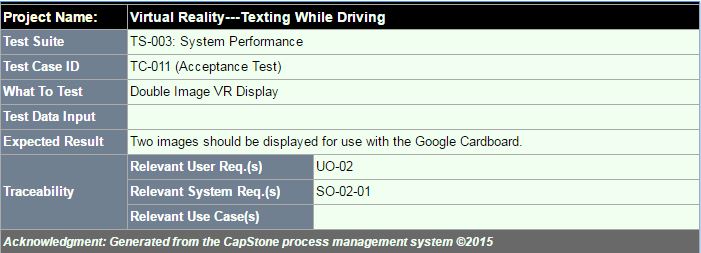
These test cases are specifically tailored to test user requirements. The tests verify that specific requirements are working as planned for the user. They ensure that the most important requirements provided by the end user are covered within the system.



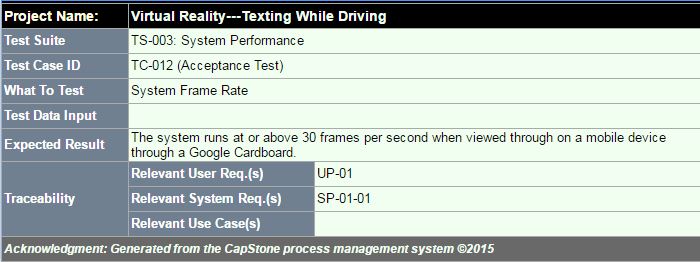
**Figure 8.1 - AI Driver Actions Test**

****

**Figure 8.2 - Hardware Validation Test**



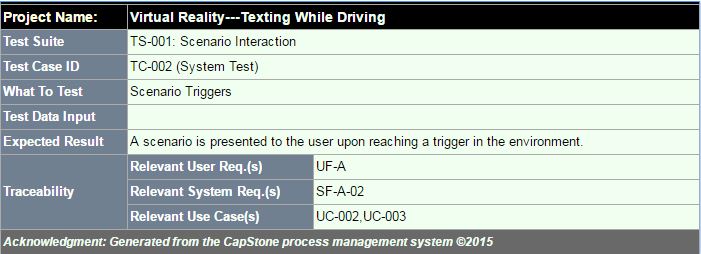
**Figure 8.3 - VR Display Test**



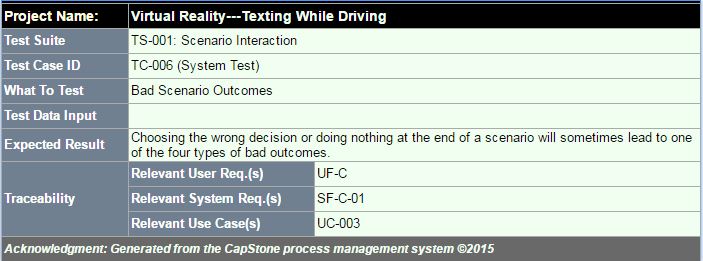
**Figure 8.4 – System Frame Rate Test**

### System Test Cases

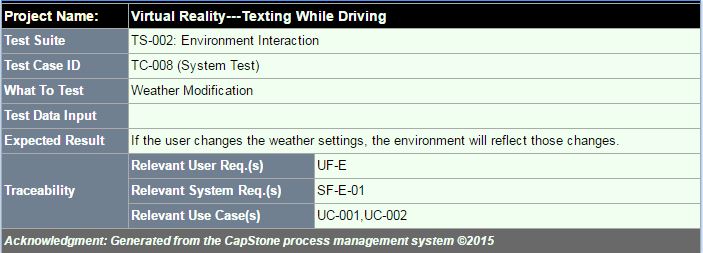
System tests covers major system functionalities, and tests specific system requirements.



**Figure 8.5 - Scenario Trigger Test**



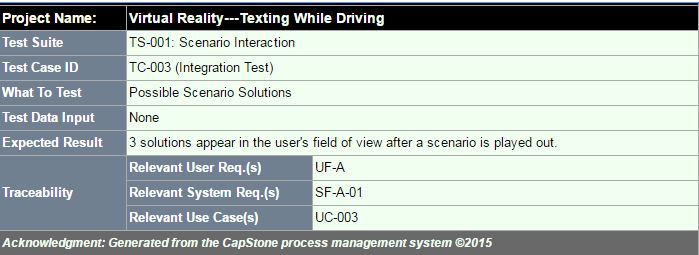
**Figure 8.6 - Bad Scenario Test**



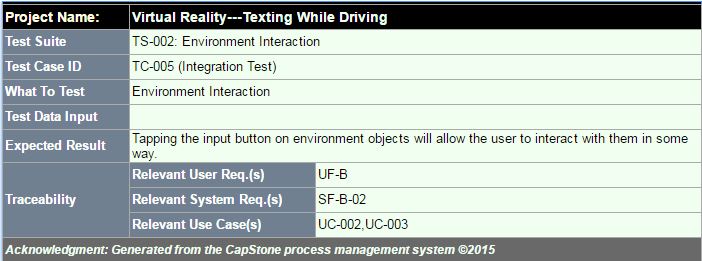
**Figure 8.7 - Weather Modification Test**

### Integration Test Cases

Integration test cases test the connection between the units of a system or subsystem.



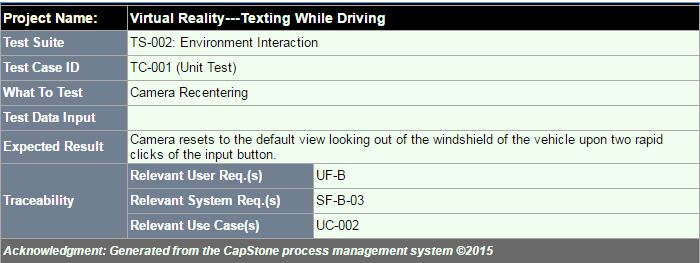
**Figure 8.8 - Scenario Solutions Test**



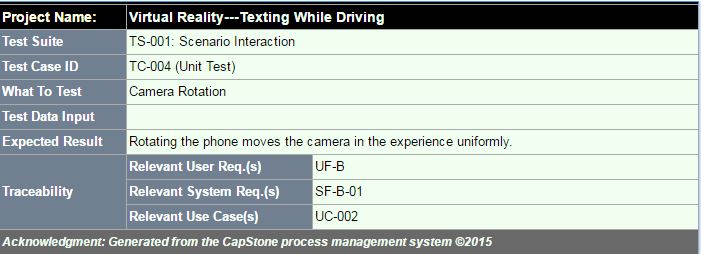
**Figure 8.9 – Environment Interaction Test**

### Unit Test Cases

Unit test cases test all parts of an individual unit within a system or subsystem.



**Figure 8.10 – Camera Recentering Test**

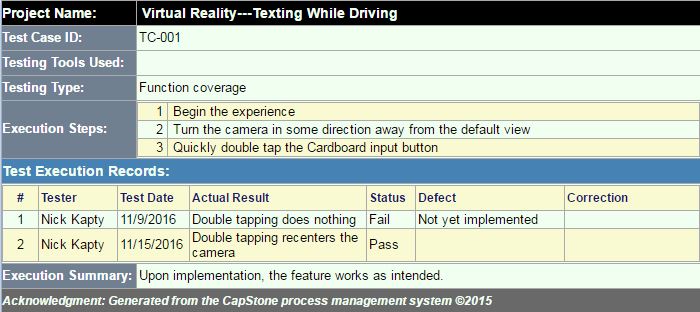


**Figure 8.11 – Camera Rotation Test**

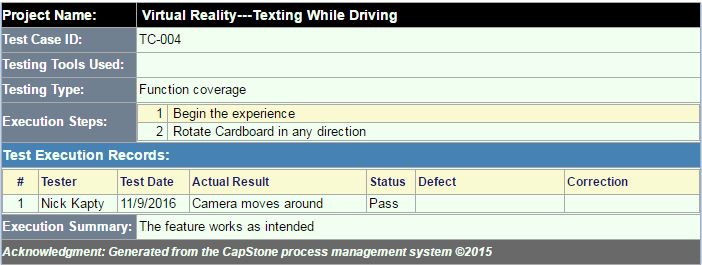
## Test Case Execution Report

The test case execution reports outline the steps taken to execute a given test case. They also provide the status of the test and any defects that will prevent the test from passing.

### Unit Testing Report

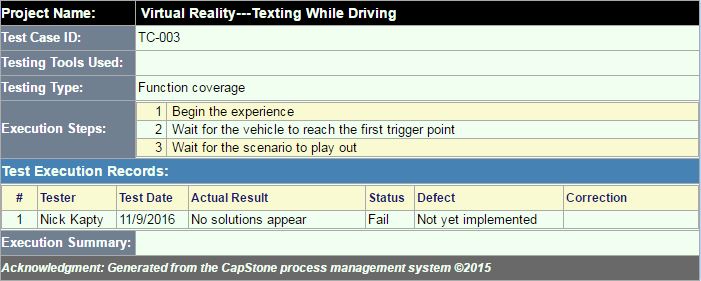


**Figure 8.12 – Camera Recentering Execution**

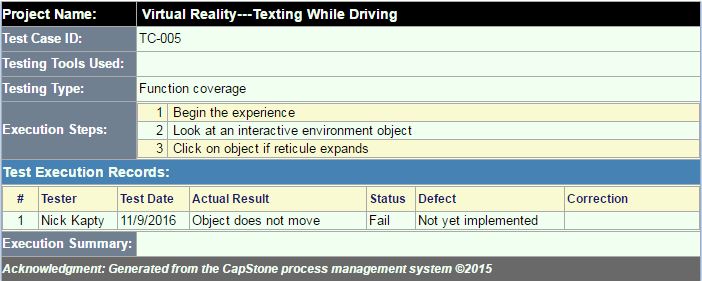


**Figure 8.13 – Camera Rotating Execution**

### Integration Testing Report

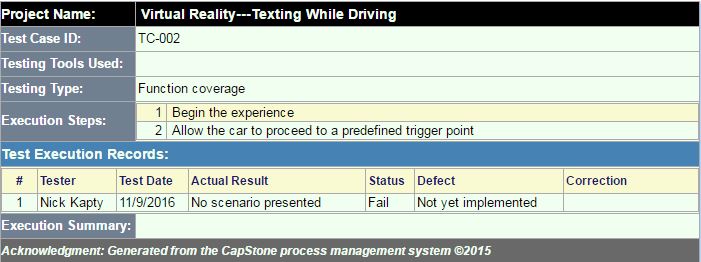


**Figure 8.14 – Scenario Solutions Execution**

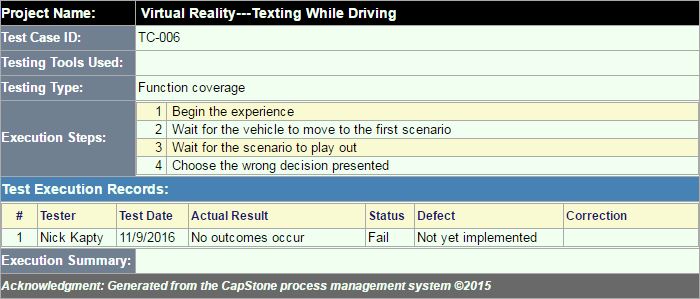


**Figure 8.15 – Environment Interaction Execution**

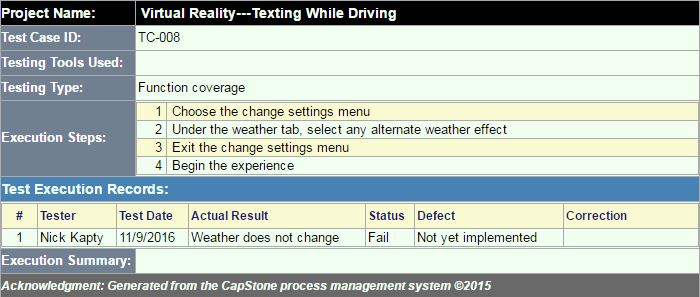
### System Testing Report



**Figure 8.16 – Scenario Interaction Execution**

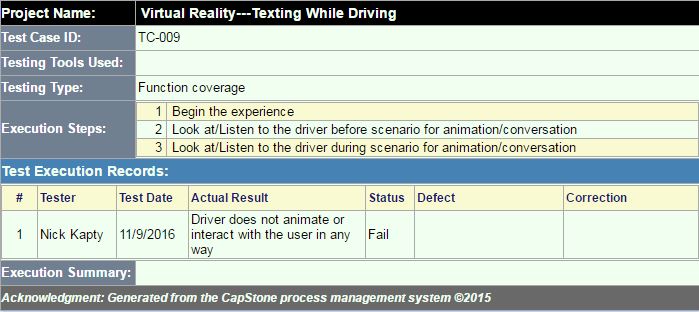


**Figure 8.17 – Bad Outcomes Execution**

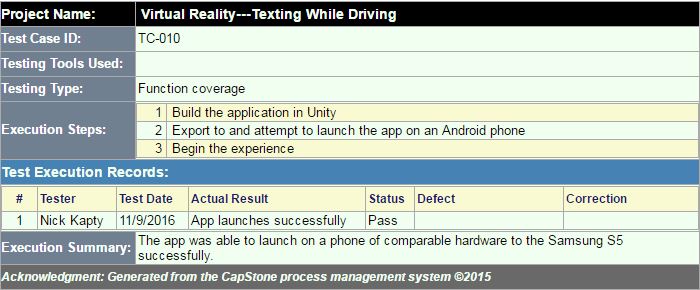


**Figure 8.18 – Weather Modification Execution**

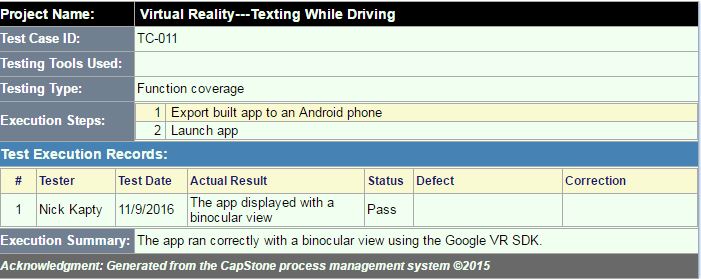
### Acceptance Testing Report



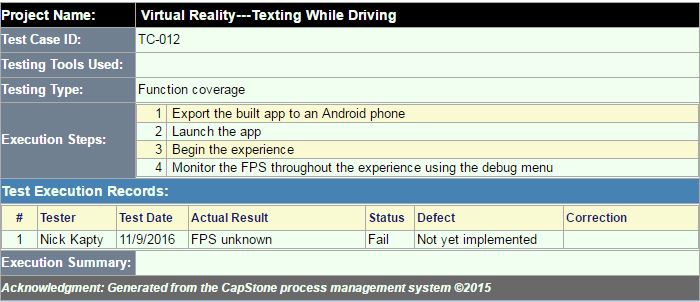
**Figure 8.19 – AI Driver Interaction Execution**



**Figure 8.20 – Hardware Validation Execution**



**Figure 8.21 – VR Display Execution**



**Figure 8.22 – System Frame Rate Execution**

# Challenges & Open Issues

## Challenges Faced in Requirements Engineering

We had trouble dealing with somewhat vague requirements provided by the industry sponsor, and were faced with the task of continuous meetings in order to get a clear understanding of the sponsor’s needs in regard to the system.

## Challenges Faced in System Development

Our first issue we faced was configuring version control to work with our system. Git alone does not work for Unity projects, and scenes are stored in binary files, so if a scene was worked on concurrently, it would not be able to merge. The documentation was confusing, and we failed to set it up properly a few times. We also had trouble with incompatible versions between Unity and the Google VR SDK. The SDK we originally had was out of date.

## Open Issues & Ideas for Solutions

N/A

# System Manuals

## Instructions for System Development

In order to develop the application, the environment must be set up. After the required steps are completed, the project must be opened in Unity. From there, any part of the system can modified.

### How to Set Up Development Environment

In order to develop the application, the developer must have Unity 5.4.1f1 installed as well as Git in order to pull from the repository. Once pulled, opening the project in Unity will allow for additional development.

### Notes on System Further Extensions

## Instructions for System Deployment

Steps to build and export to Android:

1. Select File > Build Settings
2. Select the platform as Android, then switch platform
3. Select Player Settings, in the resolution and presentation tab, select landscape left and use the 32-bit display buffer
4. Select Other Settings, change minimum API level to be Android 4.4 KitKat (API level 19)
5. Select Build to create APK

### Platform Requirements

In order to build and deploy the application, Unity is required. Along with that, the Android SDK and Java SDK must be installed as well.

### System Installation

To install on Android, the APK must be downloaded. After downloading, it can be installed and then started.

## Instructions for System End Users

N/A

# Conclusion

## Achievement

## Lessons Learned

## Acknowledgment

# References

[1] MSDN, C# Programmer’s Reference, Accessed on 10/21/2016

<https://msdn.microsoft.com/en-us/library/618ayhy6(v=vs.71).aspx>

[2] Unity, Unity Scripting Reference, Accessed on 10/21/2016

<https://docs.unity3d.com/ScriptReference/>

[3] Unity, Unity Manual, Accessed on 10/21/2016

<https://docs.unity3d.com/Manual/index.html>

[4] Unity, Unity Test Tools, Accessed on 10/21/2016

<https://unity3d.com/learn/tutorials/topics/production/unity-test-tools>

[5] Unity, Unity Community, Accessed on 10/21/2016

<https://forum.unity3d.com/>

[6] Google, Google VR SDK for Unity, Accessed on 10/21/2016

<https://developers.google.com/vr/unity/>