Test Plan
Mobeye Mobile Application

Prepared by:

CB05_05

22/09/2020

Table of contents

1. INTRODUCTION

This project is created for the Dutch company Mobeye which specialize in innovative alarm and telemetry technology. The company was founded in 2008 with the idea to help people and organizations to make their life safer and easier by providing them with a meaningful way to secure, control and monitor their property and devices remotely.

The goal of this project is to design and develop a mobile application that would contribute to the Mobeye existing notification system.

Due to increasing demand, Mobeye has decided to create a mobile application that their customers can use as another way to receive any important alarm system messages. The mobile application would also offer a more convenient way for the customers to access some basic control functions.

By logging into their personal account, the users would be able to control and monitor their devices. They would receive a push notification in case there is an emergency and they would also have the possibility to view relevant data concerning the alarm that has been forwarded to their device. The users would also have the possibility to access some basic control functions as for example to arm or disarm a device. Furthermore, there would be the option to go to the Mobeye's web portal from where their customers could acquire full control.

2. OBJECTIVES AND TASKS

Objectives

For creating a well rounded and feature rich application, while using an agile methodology of development, continuous changes will have to be made to the product. In 3 week stretches (called sprints) a new version of the project will be supplied to the client.

A team of six people will be working simultaneously on different aspects of the software. For a smooth improvement between the versions of the application and also a general control of functionality, needed parts of the project will be tested.

This will be done in multiple ways to ensure coverage of all possible problems that may arise.

Tasks

Task for implementing testing to the project will include:

1. Researching the best way to utilize testing methods
2. Deciding on exact methods that will be used
3. Dividing work between team members for the different stages of

3. SCOPE

General

While the project is in a development phase we will test the connectivity of the app with the Mobeye portal, but we will also test the user features like displaying an alarm notification and user input when the user responds to the alarm.

Tactics

The tactics we will use for the testing are pretty simple. We will first use unit testing for the methods and objects found in our classes and after the unit testing is completed we will use GitLab runners for the integration testing to take place when the code is pushed to the GitLab repository. If an integration test fails then the repository mentainer will notify the developer who pushed the code that his commit was flawed.

4. TESTING STRATEGY

4.1 Unit Testing

Definition:

For the unit testing phase we plan to test the code and logic of every class found in our project, so as a result every class will have its own test class. Because of this practice we will have a better understanding of why something went wrong and find a solution faster. For each class we will unit test every feature and the process will be declared complete when all the tests have passed.

Participants:

In the unit testing will participate all the members of the team and they would also be responsible for the Unit Testing process.

Methodology:

The unit test scripts will be written by the back end developers and they will be in the order of importance from least to the most important. The testing activity will take place inside the Visual Studio IDE.

4.2 System and Integration Testing

Definition:

System Integration Testing is defined as a type of software testing carried out in an integrated hardware and software environment to verify the behavior of the complete system. System and Integration Testing is performed to verify the interactions between the modules of a software system.
For Example, software and/or hardware components are combined and tested

progressively until the entire system has been integrated.

Participants:

System and Integration Testing will be conducted by all the members of the team and they would also be responsible for the System and Integration Testing process.

Methodology:

The System & Integration testing will take place when a developer pushes something in the project repository, the test scripts will be written by the developer that pushes his code to the repository and the tests will be run on a GitLab runner. We will have a branch for every feature of the project and separate GitLab runners to test those features. This way we have a more specific testing that will help the developers find and fix errors faster but also the code will be more maintainable and avoid merging errors.

4.4 User Acceptance Testing

Definition:

The purpose of the acceptance test is to confirm that the system is ready for operational use. During acceptance tests, end-users (customers) of the system compare the system to its initial requirements.

Participants:

A group of people (end-users) will be selected. This will include people:

1. Past users of the client’s other products
2. Of different social groups
3. With different background

Methodology:

Firstly, a state of the product, ready for user testing, needs to be achieved.

From there, a test script will have been prepared. This will guide users, unfamiliar with the application, through its features.The testing will be monitored and recorded. This will provide data needed to further understand the usability of the application.

Needed insights will be the following:
1. Flow of most frequent usage
2. Coherence
3. Ease of use
4. Bugs that may appear
5. Rating of the app itself

4.5 Automated Regression Testing

No automated regression tools like Selenium or Katalon Studio would be used in the project for the moment, as CI pipeline in GitLab would be created. This pipeline would automatically build the application and run the tests each time a new feature is added to the main branch, ensuring that the new functionality has not caused any unintended effects and that the system or component still works as specified in the requirements.

However, if agreed upon, the team could test the application as well by using automated regression testing.

4.6 Beta Testing

After the final product is developed and completely tested, the team would apply beta testing as a final form of validation before presenting the mobile application to the client. So if there are any components that are somehow ambiguous or complicated for the user, the developers could fix these functionalities and/or features.By using Beta testing as one of the testing strategies for the mobile application,the team would validate the final product for functionality, usability, reliability, and compatibility.

5. HARDWARE REQUIREMENTS

For the different methods of testing, a list of hardware components will be required.

Firstly, a way to test the sensor data, a sensor provided by the client will be ideal.

For the continuous integration, a computer capable of handling an uninterrupted deployment of the gitlab-runner will be needed

For the user acceptance testing, two mobile smartphones will be needed. One that has “Android” and one with “iOS” operating systems that can run the application.

6. TEST SCHEDULE

The following testing methods will be initialized with the first software components that are added:

1. Unit testing - all new software changes will be accompanied with corresponding Unit tests
2. Continuous integration testing - with each change in the project online repository, a pipeline will be initiated to ensure functionality remains in the application

User acceptance testing will begin with the development of a version that has a responsive GUI and a minimum number of features that need to be tested.

7. CONTROL PROCEDURES

Problem Reporting

Problems will undoubtedly arise. Most reported problems are expected to be received from either User testing. This testing method provides an “error message” containing information of where, what and why a test has failed. That way these issues will be almost immediately dealt with.

If or when an instance of Continuous integration testing fails, again an “error message” accompanies the raised problem that gets recorded and saved for when the problem needs to be fixed. In this case, a person responsible will have to troubleshoot the issue and resolve it as soon as possible.

Change Requests

Changes to the final product will be discussed between the team and only final decisions will be allowed to be added to the software.

8. FEATURES TO BE TESTED

- Communication between our API and the Mobeye portal;
- The app receiving information from the API;
- The app displaying information;
- The app giving the user push notifications;
- The notification sound
- The app being able to take user input;
- The app sends the user input to the API.

9. FEATURES NOT TO BE TESTED

The following functions will not be tested:
- How the Mobeye portal handles the information we send;
- How often the information is updated.

10. RESOURCES/ROLES & RESPONSIBILITIES

Each developer will be responsible for their code and the unit testing that comes with it. If a pushed code fails the integration test then the developer who pushed it will be notified so he can start fixing it. The unit testing environment will be the Visual Studio IDE and for the integration testing we will use GitLab Runner.

11. DEPENDENCIES

The main dependency would be the availability of the test-items. Software components need to be ready some time before the actual deadline, so that they could be unit tested. It might happen that a component is not ready in time and no tests are created for it.

Another constraint might be insufficient planning. Not setting correct deadlines and wrongly estimating necessary hours for the completion of a task with the test cases would result in slowing down the working process and missing deadlines.

12. RISKS/ASSUMPTIONS

One of the greatest risks the team might face is not delivering a component on time. This delay in delivery, might slow down the testing procedure afterwards, which could result in increased workload in order to meet the deadlines. It might also be needed for developers who have already finished their tasks to take up additional work,so that the team could finish the necessary deliverables on time.

Another issue might occur if a teammate does not unit test their code before merging it with the main software. Even if the newly added functionality does not raise any issues when combined with the rest of the application, it might cause some integration problems in the future.

As the agile methodology is new for the whole team, and the requirements are constantly changing, this might cause some delay in delivery of test components as well. New functionality could constantly be added or updated, so the team would have to quickly change focus and test the newly required functionality. It would also be vital in such a situation to prioritize the requirements once again and focus on the ones that are top priority.

13. TOOLS

The team would use a number of automation tools for testing. If needed, new tools could be added to the list at a later date.

- CI/CD GitLab pipeline - it would automate the integration and deployment and would ensure that each time a new change has been made and added to the main application, no unexpected behaviour would occur. By using GitLab's pipeline, issues would be caught early and the number of integration problems would be reduced, allowing the team to deliver software more rapidly.