**Doorbell Test Plan**

| **Document Information** | |
| --- | --- |
| **Project Name** | Doorbell Software Validation and Production Testing |
| **Project Number** |  |
| **Project Account Code** |  |
| **Author** |  |
| **Issue Date** |  |

| **Document History** | | | |
| --- | --- | --- | --- |
| **Version** | **Date** | **Summary of change** | **Author** |
| 1.0 |  |  |  |
| 2.0 |  |  |  |

| **Approvals** | | | |
| --- | --- | --- | --- |
| **Name** | **Title** | **Signature** | **Escalation** |
|  |  |  |  |

| **DOCUMENT CIRCULATION** | | | |
| --- | --- | --- | --- |
| **Version** | **Date** | **Person/Group** | **Author** |
|  |  |  |  |

**CONTENTS**

[1. INTRODUCTION 3](#_heading=h.1fob9te)

[2. OBJECTIVES 3](#_heading=h.3znysh7)

[3. SCOPE 4](#_heading=h.2et92p0)

[3.1 System Diagram 4](#_heading=)

[3.2 Test Plan Diagram 5](#_heading=h.xaplcswo7gz3)

[4. TEST PLAN 6](#_heading=h.tyjcwt)

[5. TEST AUTOMATION STRATEGY APPROACH 9](#_heading=h.3dy6vkm)

[5.1 Automation Framework Selection 9](#_heading=h.1t3h5sf)

[5.2 Automation Tool Selection 10](#_heading=h.ci6kpkvhcj3r)

[5.3 Reporting, Metrics and Monitoring 10](#_heading=h.i9fdfg6t3yr5)

[5.4 Roles and Responsibilities 10](#_heading=h.d9tp2db0n798)

[5.5 Resources and Environments 11](#_heading=h.x49oo6w2zc3k)

[6. FACE RECOGNITION VALIDATION EXAMPLE 12](#_heading=h.53y54alzhnv)

[6.1 Task Overview 12](#_heading=h.4d34og8)

[6.2 Test Scope 12](#_heading=h.gh9lzlvqwfyh)

[6.3 Test Plan 13](#_heading=h.17dp8vu)

[6.3.1 Test Case Design 14](#_heading=h.q0tf0a4k40f0)

[6.3.2 Test Automation Framework 21](#_heading=h.fn8lzpc59kwk)

[6.3.3 Test Tools 21](#_heading=h.tuklwqrk7o4o)

[6.4 Test Execution 22](#_heading=h.3rdcrjn)

[6.5 Test Metrics & Reports 25](#_heading=h.1ksv4uv)

[6.6 Test Schedule 26](#_heading=h.26in1rg)

[7. SIGN OFF / APPROVALS 27](#_heading=h.2xcytpi)

# INTRODUCTION

This document deals with the design of a software test plan for the doorbell product. The test plan is focused to the development validation production acceptance testing of the software. The design of the test process is focused on the structure of the test, the testing techniques, tearing down the objectives, the equipment, the tools for automation, the test environment, the metrics flow into several systems and processes(teams), and the reports.

# OBJECTIVES

Objective 1: System diagram of the system under test.

Objective 2: Top-level test plan for the doorbell product.

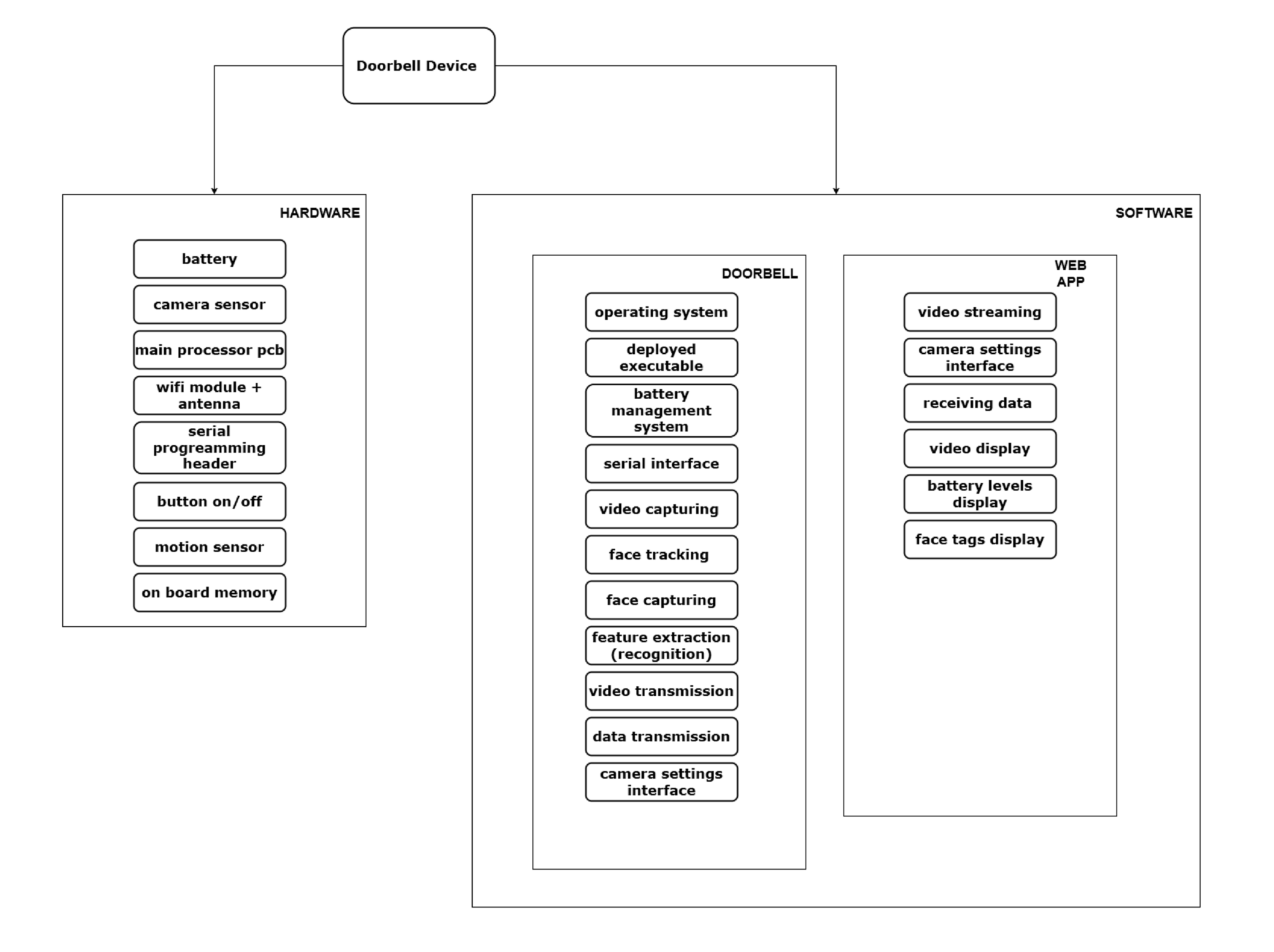
Objective 3: Proposal of test automation strategy including equipment and tools.

Objective 4: Example of an automated test case at the process of validation of the face recognition function. Tools and techniques.

# SCOPE

## System Diagram

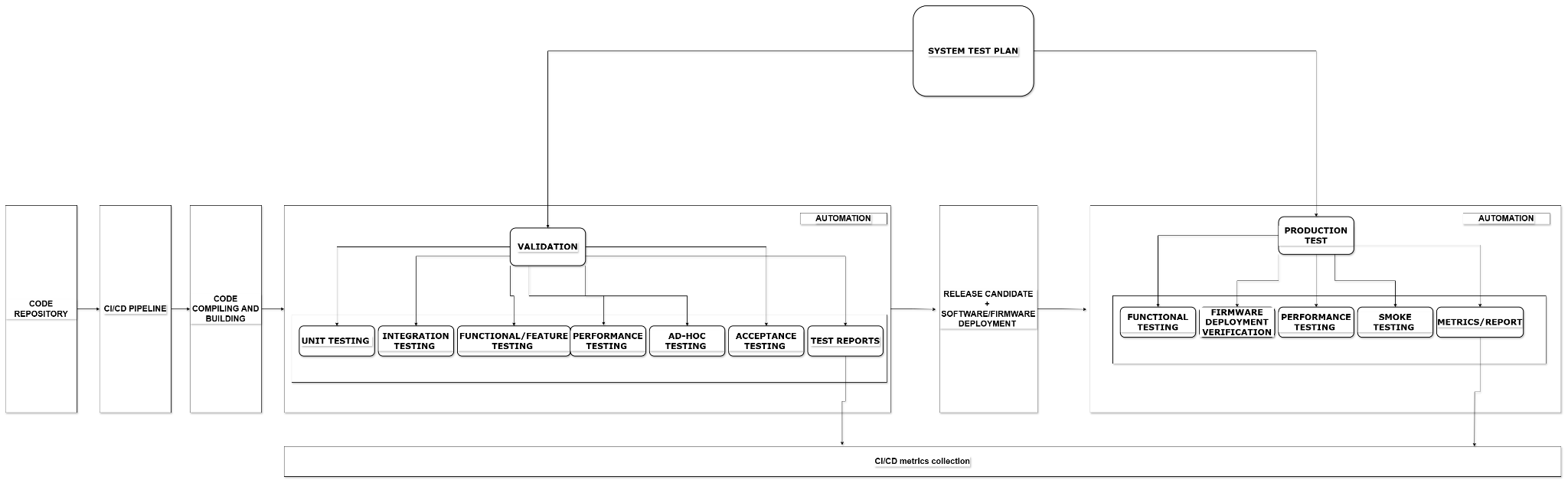
The following diagram tears down the Doorbell system into subsystems and groups them in hardware and software. The software section is splitted to Doorbell device software/firmware and the Web App software.



## Test Plan Diagram

The following flowchart shows a generic structure of a test plan for the Doorbell product. It includes validation testing and production acceptance testing.The validation testing is responsible for full software testing before deployment. The production testing is responsible for lighter software testing and focuses mainly on the validation of the desirable performance of the whole system - user-accepted performance.

As we have seen in the system diagram (3.1) , the Doorbell product consists of the software/firmware of the device and the software of the Web app. The following test plan is applied to both objects.



# 4. TEST PLAN

As we can see in the flowchart (3.2) our testing plan is split into 2 sections. The validation testing and the production testing. At the validation testing we are focused on many types of testing techniques and our main goal is to verify that a new piece of code/feature/code tweak works as expected. At the production testing, we are focused on the final tests before we release the firmware/software.

Once there is a new compiled built of software for the device or Web app, it will go automatically into the validation stage. At this stage, we will perform a series of tests that will validate the functionality and the stability of the software. The basic types of testing are Unit testing,Integration testing, Functional/Feature testing,Performance testing, AD-HOC Testing,Acceptance Testing and finally report creation.

Once our software is validated, a release candidate will be created and a final deployment will take place. At this stage, we need to go through production testing. the production testing is mostly focused to the validation of the main functionalities and stability of our system. We program a physical device and we perform Functional testing, Firmware Deployment validation, Performance Testing, Smoke Testing, and final report creation.

**Validation**

* Unit testing

The Unit testing includes small tests in the codes. We are focusing on main checks for example error handling, edge cases, input validation, logic testing, exceptions, and output testing.

* Integration Testing

Integration testing includes various types of testing into the code. For example Top-Down, Bottom-Up, Sandwich testing, interface testing, and Big Bang Testing. Particularly for the Doorbell software and Web App, we could perform checks on video transmission, video streaming, face tracking data transmission, and camera settings updating. Also, we can perform tests on the Web App API interaction.

* Functional/Feature Testing

At this stage we are mostly focused on checking the key functionalities of the code through sanity testing, positive/negative testing, usability testing, and of course regression testing. Particularly for the Doorbell product, at this stage, we should check the video streaming feature, the face tracking feature, the face recognition feature, the database that stores all the frame captures that will be used for the neural network of the face recognition algorithm, the face tracking data transmission from the device to the Web App (e.g face Ids), the camera settings updating mechanism, the battery management system that monitors the battery levels and prints them on the Web App, the camera connection status and stability of the Web App in connection abnormalities etc.

* Performance Testing

At this stage, we are pushing the system beyond its normal conditions and we observe its stability, identify braking points, we understand how it handles the high load if it recovers but also if it meets the formal product specifications ( eg. battery life, max. number of faces that can be saved) . The theoretical approach of these types of tests is load testing, stress testing, scalability testing, stability-endurance, volume testing, and spike testing.

For example, load testing of the video streaming (check of the stream performance with multiple users), stress testing of the camera (camera in use for long periods - endurance), load testing of the face tracking and recognition (measure score of successful tracking and face recognition), measurements of the latency between video capture, face detection, face display for long periods ( eg. 24h,1 week - endurance testing ), the latency on the database (when a new face is detected and the algorithm pulls saved frames from the database) multi-user tests, CPU/memory/battery consumption checks.

* AD-HOC Testing

At this stage, we start with some manual exploratory testing and try to find weak areas in the system. Then we create some automated tests that simulate specific user journeys or abnormal conditions.

* Acceptance Testing

At this stage, we will perform a full test of all the features and functionalities of the system. The main types of tests at this stage are functional and non-functional acceptance testing. Functional testing example: Ensuring the video capturing,video streaming, face tracking and ,face recognition work as expected. Non-functional testing example: We perform a real condition test using 50 - 100 simulated users and then ensure that the video streaming is working as expected and the app is working as expected too. Also , check if the app works fine with many types of devices.

* Test Reports

The most important part of the validation is the test reports. the test reports should meet the QA standards. The automated production of test reports can be done through various tools and plugins. Our main goal is to feed our CI/CD pipeline with these reports but also store them in a central database for future analysis.

**Production Testing**

* Functional Testing

At this stage, we perform tests that will validate the main features according to the product specifications. Checking if the users can access the camera settings, check credential inputting for login, and error handling on the Web app.

* Firmware Deployment Testing

At this stage, we ensure that the final firmware is installed to our hardware correctly and operates as expected. The important tests that will be performed at this stage are the deployment process, the validation of post-deployment, compatibility checks, security checks, and some performance checks. Particularly for the Doorbell product we need to check the firmware upgrade, to check if the video streaming is working as expected after firmware updates, if the camera settings are fully working after a firmware update, backward compatibility checks, and a stress test of multiple cameras firmware update.

* Performance Testing

At this stage, we can add similar automated tests as on Validation performance testing.

* Smoke Testing

Full feature testing at the deployed firmware/software of the device and the Web app. Here we test only the critical functionalities for example API testing, ensuring that the apps start with no errors, the UI elements are fully working, the device database is working as expected and the face recognition is pulling data with no issues.

* Metrics/Report

A final test report is very important at this stage as it can be very useful for analysis of defects, bugs etc.

## 

# 5. TEST AUTOMATION STRATEGY APPROACH

Both validation and production testing can be automated through a customized testing framework setup. Planning is the first step as we need to decide which type of automation we need (data-driven, keyword-driven, hybrid, modular). The next step is the tool selection. We set the in-scope items that need to be automated. The main framework design and the test bed preparation are next. We create all the test cases and scripts that need to be used and we finally need to check if the deliverables are good enough and cover the whole QA process.

## 5.1 Automation Framework Selection

Based on our Scope, the most suitable Automation Framework is the **Hybrid**. We can combine **Keyword-driven** and **Data-driven** testing.

A high-level test scenario can be created through keyword testing. A good tool for this purpose is the **Cucumber** plugin. We create scripts that are based on user stories/journeys. The keywords will be mapped to functions of the automation scripts (Pytest). In this way, we have a simple and readable structure for all our test cases. The test cases can be easily tweaked with no need of heavy script tweaking.

The low-level scripting can be done in **Pytest**. We can build comprehensive tests with multiple inputs, store the outputs in many types of files (eg.CSV,JSON) or store them in a database. The main Pytest parametrization can help us feed data into the individual scripts dynamically. This is the ultimate option for negative testing, edge cases, and many other types of tests.

The Web app validation will require some back-end and some front-end tests. For the back-end testing for instance Unit testing, and integration testing we can use Pytest. On the front-end testing of the Web app, we need a tool that has the ability to simulate user interactions. A good option is **Selenium** which can be used specifically for UI behavior checks (eg. how the frontend interacts with the backend through API calls).

## 5.2 Automation Tool Selection

* Cucumber, Pytest: Software/firmware testing of the Doorbell device
* Selenium, Pytest: Web app back-end and front-end testing
* GitHub Actions: CI/CD Integration. Selenium, Cucumber, and Pytest can be integrated into a CI/CD pipeline. In this way, we can create a continuous validation process. Whenever there is a pushed code change, the automated script execution will take place.

## 5.3 Reporting, Metrics and Monitoring

One more important part of the testing process is the creation of accurate and correctly structured test reports. As soon as we have 2 stages of testing ( validation and production testing) we need to address the test report structure accordingly. Also, it is very useful to feed the CI/CD with the reports but also saving them in a central database. We can produce reports in HTML or XML formats and feed them into the CI/CD so they will be part of the build process.

Another useful action at this stage is Monitoring and alerting. We can set up CI/CD alerts or notifications based on the test results. Of course, we can also use third-party test management tools like TestRail. It can be integrated well with our suggested automation tools.

## 5.4 Roles and Responsibilities

A quite challenging part of the automation framework is maintenance. Every time we have a new release cycle which means that new functionalities will be added to our firmware, all the scripts will need to be tweaked, reviewed accordingly but also to be validated. It is important at this stage, to have a reliable monitoring system of the automation process. Any potential alerts, issues,bugs will need to be investigated by the test team and the developers. At this stage, it is important for the to teams to interact efficiently through a management system (eg.Jira, TestRail).

The potential developer resources engagement will take place in cases of bugs in the new features, regression testing failures, and unclear expectations on specific test cases. However, there are many other failure events during automation, such as testing environment failures, network issues, test configuration mistakes, and dependency issues. Again good cooperation between the developers and testers will resolve most of the issues.

## 5.5 Resources and Environments

As soon as we have to deal with a device, this means that we will need some hardware development for testing. The hardware team should address a number of requirements from the QA department.

**Testing Rig 1:** The first stage after manufacturing of a device is inspection and electronics testing. for this purpose, the hardware designers should create a testing rig that is capable of checking the main electronics. Power supply, correctly fitter SMD and conventional components, correct wiring, correct assembly.

**Testing Rig 2:** The main equipment for the QA is a testing rig. The testing rig will have a special slot where the Doorbell device fits into and connects to a serial port header. The header connector should support RS232 or RS422. Once the device is connected, we should develop and integrate a small serial interface diagnostic tool that checks the health status of the serial interface. This Testing Rig 2 can be used by the Validation team but also the production testing team.

**Camera Test Rig:** A special testing rig for lens calibration and optical performance testing.

**WIFI Test Rig:** A special tool that checks the performance of the wifi module in sending and receiving data. (eg.Wifi Analyzer). Also, these tests can be combined with third-party software tools that check the video stream quality and the bandwidth and latency of the transmissions. Of course, most of these tests will be automated, but it is critical for the QA team to be able to perform manual tests in order to build more accurate test cases in the automation scripts.

**Software tools:** Pycharm, Github, various libraries and plugins installation, Wireshark, RS232 analyzer, PUTTY, Onviff, VLC.

# 6. FACE RECOGNITION VALIDATION EXAMPLE

## 6.1 Task Overview

This is the last task and focuses on the face recognition function validation. We will create a full plan of the test strategy, objectives, deliverables, tools, and testing environments.

## 6.2 Test Scope

The face recognition consists of some modules that work in the background. The automated validation process will perform a series of tests on the individual modules but also combined module testing. The main modules that need to be tested are :

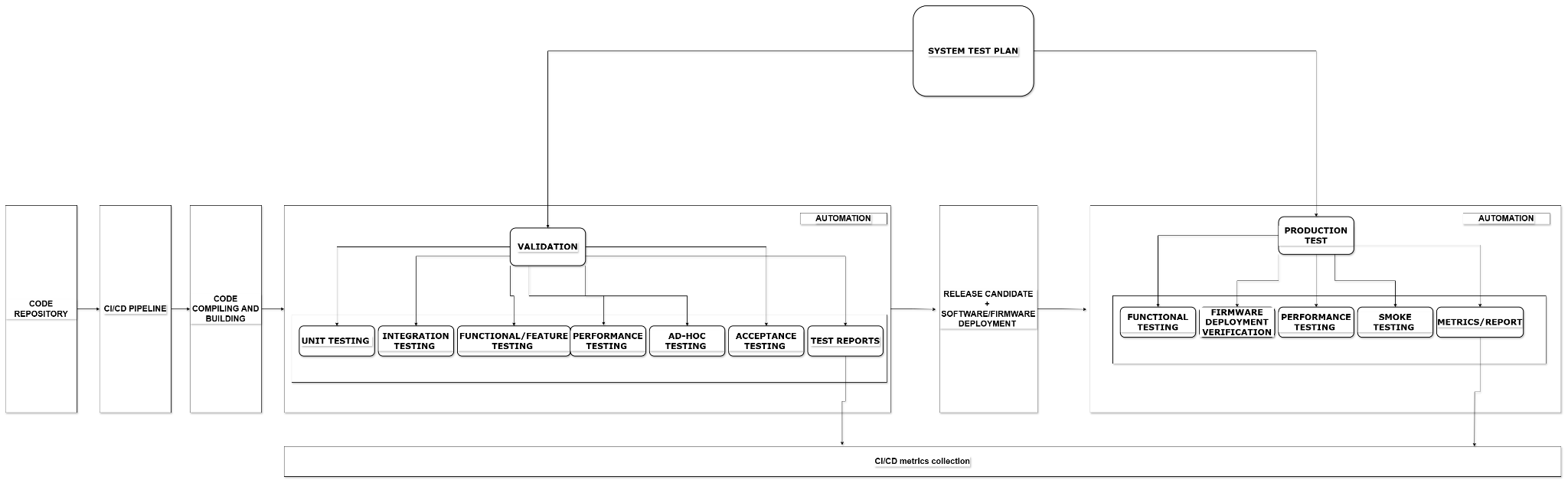
* video capture module
* the face detection module
* the face feature module
* the tag data storage mechanism
* the database module
* the face recognition module
* the face tag printer module

## 6.3 Test Plan

As we can see in the flowchart (3.2) our testing plan is split into 2 sections. The validation testing and the production testing. At the validation testing we are focused on many types of testing techniques and our main goal is to verify that a new piece of code/feature/code tweak works as expected. At the production testing, we are focused on the final tests before we release the firmware/software.

Once there is a new compiled built of software for the device or Web app, it will go automatically into the validation stage. At this stage, we will perform a series of tests that will validate the functionality and the stability of the software. The basic types of testing are Unit testing,Integration testing, Functional/Feature testing,Performance testing, AD-HOC Testing,Acceptance Testing and finally report creation.

Once our software is validated, a release candidate will be created and a final deployment will take place. At this stage, we need to go through production testing. the production testing is mostly focused to the validation of the main functionalities and stability of our system. We program a physical device and we perform Functional testing, Firmware Deployment validation, Performance Testing, Smoke Testing, and final report creation.



### 6.3.1 Test Case Design

Here is a test case list of the modules/subsystems/mechanisms that need to be validated.

**Validation**

* Unit testing

| Face detection mechanism | Test of the accuracy of the algorithm to detect multiple faces |
| --- | --- |
| Frame grabbing mechanism | Test of the frame-grabbing mechanism. This part of the code should grab many face pictures and store them. They will be used for face comparison from the recognition code. |
| Frames Storage, retrieving from the local database | Test of the mechanism that stores frames (images) in the local database. |
| Facial feature detection accuracy | Test of the mechanism that detects eyes, nose, and mouth. |
| Face matching | Test of the mechanism that compares the saved images with the current detected face. |
| IDs/tags creation per face verification | Check the mechanism that creates tag items on top of the video feed. |
| IDs/tags storage, retrieving, and updating mechanism | Check the mechanism that stores the face data in the local database. |
| Error Handling mechanisms | Missing video feeds, corrupted frames, randomly dropped frames on the video feed |
| Edge Cases with invalid inputs | Corrupted saved image files, ultra high size or low size of images to be retrieved from the database |
| Video capturing mechanism | Checking if the algorithm capture video in a correct and robust way |

* Integration Testing

| Face detection & feature face extraction | Check the cooperation of these two modules and if there are unhandled cases/errors |
| --- | --- |
| Database | Check if the data are correctly stored, not corrupted, and correctly retrieved from other modules. Check if the face ID/tags match the database-stored records. |
| Validation of camera video feed module and face detection module | Check if the video feed will be captured correctly and if no dropped frames or lagging will appear once it goes through the face tracking module |
|  |  |

* Functional/Feature Testing

| Face Recognition of one face | Validation of the face recognition mechanism with one face by providing scores about accuracy and stability |
| --- | --- |
| Face Recognition of many faces | Validation of the correct performance with multiple faces recognition |
| Face matching in normal conditions | Validation of the detected face ID/TAG and it’s storage into the database correctly |
| Face-matching speed checks | Verification of acceptable speed of face matching |

* Performance Testing

| **Main Test Cases** | **Description** |
| --- | --- |
| Face recognition latency | We measure the time for the module to recognise a face |
| Video feed frame rates | We observe the performance of the face recognition with very low and very high video resolution video feeds |
| Scalability test | Many faces to be tracked and recognised. WE observe the performance of recognition and the tag creation and storage |
| Stress tests | We populate the system with maximum faces, maximum users and we measure CPU/RAM of the processor |
| Load Testing | We populate the system with maximum faces, maximum users and we measure CPU/RAM of the processor |

* AD-HOC Testing

| **Main Test Cases** | **Description** |
| --- | --- |
| Abnormal inputs | We use abnormal conditions for example a face with a very bright background. Or a very low-brightness video feed. Semi-covered faces. We observe the performance of the face recognition set of modules |
| Trained and untrained faces | We feed the camera with a video that contains trained and untrained faces. We check the behavior of the system, the latency, the CPU/RAM usage, the accuracy of tag printing |
|  |  |

* Acceptance Testing

| **Main Test Cases** | **Description** |
| --- | --- |
| Accuracy of registered faces | Validation of the accuracy of the module to recognize pre-registered faces. Repeatability testing on this. |
| Accuracy of unregistered faces | Validation of the accuracy and robust performance of the recognition module when there are unregistered faces. It shouldn’t consider them as registered giving false tags. Repeatability testing on this. |
| End-to-end testing | Validation of the entire process of video capturing, video feeding into the face detection algorithm, face detection, face feature grabbing, frame grabbing, frame storing in the database, face recognition and comparison with stored images, face tag printing over the video feed. |
| System usability testing | Validation of any potential new feature in face recognition from a user’s perspective. Validation of the user interface clarity. |

**Production Testing**

* Functional Testing
* Firmware Deployment Testing
* Performance Testing
* Smoke Testing

### 6.3.2 Test Automation Framework

Based on our Scope, the most suitable Automation Framework is the **Hybrid**. We can combine **Keyword-driven** and **Data-driven** testing.

### 6.3.3 Test Tools

| **Category** | **Tools** |
| --- | --- |
| Test Automation Tools | Pytest, Selenium, Cucumber,OpenCv |
| CI/CD Pipeline | GitHub Actions |
| Test Management Tools | Jira, TestRail |

## 6.4 Test Execution

I will use as an example the Test Case in the **Unit Testing section** -> **Video capturing mechanism** -> **Checking if the algorithm captures video in a correct and robust way**

**Step 1:** We analyze the piece of code that captures the video and create specific test cases.

**Step 2:** We write some Cucumber scenarios :

Scenario: I attempt to open the camera feed

Given I have connected a camera to my system

When I attempt to open the camera feed

Then The camera feed should be opened successfully

**Produced Python Script**

@given("I have connected a camera to my system")

def step\_given\_camera\_connected(context):

# Assuming camera index 0 is used for local camera

context.camera\_index = 0

context.cap = cv2.VideoCapture(context.camera\_index)

if not context.cap.isOpened():

raise Exception("Camera not connected properly!")

@when("I attempt to open the camera feed")

def step\_when\_open\_camera(context):

context.camera\_opened = context.cap.isOpened()

@then("The camera feed should be opened successfully")

def step\_then\_camera\_opened(context):

assert context.camera\_opened, "Failed to open the camera feed"

**Pytest Test file :**

@pytest.fixture(scope="module")

def camera():

"""Fixture to initialize and release the camera."""

cap = cv2.VideoCapture(0) # Default camera index

yield cap

cap.release()

def test\_open\_camera(camera):

"""Test if the camera feed can be opened successfully."""

camera\_opened = camera.isOpened()

assert camera\_opened, "Failed to open the camera feed"

**Test Outputs :**

============================= test session starts =============================

collected 1 item

test\_camera.py . [100%]

============================== 1 passed in 0.45s =============================

### 

## 6.5 Test Metrics & Reports

This is a default report that can be used at the stages of Production testing.

| **TEST TITLE** | | **PRIORITY** | **TEST CASE ID** | | | **TEST NUMBER** | | | | **TEST DATE** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Face Recognition Validation | | 1 | 1 | | | 1 | | | |  |
| **TEST DESCRIPTION** | | | **TEST DESIGNED BY** | | | **TEST EXECUTED BY** | | | | **EXECUTION DATE** |
|  | | | PPantelidis | | | PPantelidis | | | |  |
|  | | | | | | | | | | |
| **TEST DESCRIPTION** | | **TEST DEPENDENCIES** | | **TEST CONDITIONS** | | | | **TEST CONTROL** | | |
| Unit Testing - data\_loader.py | | main.py | | pycharm | | | | test1.py | | |
|  | | | | | | | | | | |
| **STEP ID** | **STEP DESCRIPTION** | **EXPECTED RESULTS** | | | **ACTUAL RESULTS** | | **PASS / FAIL** | | **ADDITIONAL NOTES** | |
| 1 |  |  | | |  | |  | |  | |
| 2 |  |  | | |  | |  | |  | |
| 3 |  |  | | |  | |  | |  | |
| 4 |  |  | | |  | |  | |  | |

## 6.6 Test Schedule

| **Phase** | **Timeframe** | **Deliverables** |
| --- | --- | --- |
| Test Planning | 1 week | Test Plan, Documentation for Test Cases |
| Test Environment Setup | 1 week | Automation Framework setup, Tools installation |
| Test Case Development | 1 week | Writing of test cases, scripts, mechanisms that output metrics and data, performance checks of the scripts |
| Test Execution |  | Test Results, Performance Metrics, Auto-generated reports |
| Bugs / Exploratory test / Retesting |  | Retesting Results, Debugging, Faultfinding |
| Final Report | End of Test Cycle | Test summary Report, Metrics, Bugs and improvements |

## 

## 

## 

## 

## 

## 

## 

# 7. SIGN OFF / APPROVALS

All test plans are subject to sign-off – people within the program should know how their products are going to be tested and agree with the planned action. If not they should raise their issues and have the plan changed, if not their objections must be recorded.

For Example:

The following approve the contents and intentions of the XXXXXXXXXXXXX EISD Management Services Test Plan.

| **Name:** | **Appointment:** | **Approval Signature:** | **Signatory Remarks:** |
| --- | --- | --- | --- |
|  | Project Executive |  |  |
|  | Product Manager |  |  |
|  | Release Manager |  |  |
|  | Project Manager |  |  |
|  | Test Manager |  |  |
|  | Business Analyst |  |  |