MEDICALOCR

PRESCRIPTION DATA EXTRACTOR APP

Judges' Report

Team 10

Contents

1 Overall Introduction **2**
- 1.1 Glossary . . . 2
- 1.2 Product Perspective . . . 2

2 Usage and Feature Listing **3**
- 2.1 Starting MedicalOCR . . . 3
- 2.2 The Complete Feature Reference . . . 3
  - 2.2.1 Basic Set of Features . . . 3

3 System Architecture **4**
- 3.1 High Level View . . . 4
- 3.2 Functional View: Use Case Diagram . . . 5
- 3.3 Flow Diagram . . . 6
- 3.4 User Action View . . . 7
  - 3.4.1 Browse images . . . 8

4 Languages, Technologies and Data Sources Used **9**

5 Test Plan **9**
- 5.1 Features to be Tested . . . 9
- 5.2 Approach . . . 9
  - 5.2.1 Testing Levels . . . 9
  - 5.2.2 Meetings . . . 10
  - 5.2.3 Responsibilities . . . 10
  - 5.2.4 Test Case Pass / Fail Criteria . . . 10
  - 5.2.5 Test Deliverables . . . 10
- 5.3 Environmental Needs . . . 11
- 5.4 White Box Testing (Unit Testing) . . . 11
- 5.5 Black Box Testing . . . 12

6 Actual Testing Done **13**
- 6.1 Process Model . . . 13
- 6.2 Unit Testing . . . 13
- 6.3 System Testing . . . 13
- 6.4 Regression Testing . . . 14
- 6.5 Beta Testing . . . 14

1 Overall Introduction

MedicalOCR, is a software developed as an initiation of digital convergence in the medical prescriptions handwritten by doctors. Doctors write their prescriptions in the conventional way (i.e., using their pen and paper). From the scanned version of the prescription, a handwritten character recognition is followed to capture the data (name of the patient, symptoms, findings, prescription of medicine, tests, advice, etc.) written by the doctor. Since, the accuracy rate of the state-of-the-art hand written character reorganization is not still up to the acceptable level, we have applied apply an error correction mechanism to reduce the errors. The solution does not oppose the age-old convention and affordable as it is mostly a software solution with a minimum hardware requirement.

1.1 Glossary

Term	Definition
MedicalOCR	The name of the developed software.
User	A person who interact with the software system.

1.2 Product Perspective

MedicalOCR is designed to be a Machine Learning based handwriting recognizing solution, without any possibility of downtime, fraud, censorship or third-party interference. The application is safe, secure and faster with no maintenance cost. It has one active actor, the user, who will interact through keyboard and mouse. The software system will interpret the users action and act accordingly and show the output result on screen. The Graphical User Interface is pleasant to look at and intuitive to use.

2 Usage and Feature Listing

In this section, we describe all the features that are included in MedicalOCR . The input and output specifications of each command have also been described in detail. Additionally, we have also added the justification for additional features, that were not a part of the client's initial requirements list.

2.1 Starting MedicalOCR

The user starts the software by-

- Opening the terminal and running the makefile by entering the command 'make'.

2.2 The Complete Feature Reference

2.2.1 Basic Set of Features

The following Input - Response pairs are observed when correct operations are processed. These represent the basic set of features that the application has.

Basic features

1. **Load prescription.**
   Input: : Image
   Output: : Pre-Processed Image.

2. **Locate Hand Written Regions**
   Input: : Pre-Processed Image.
   Output: : Image with bounding boxes covering handwritten words (Segmented Block).

3. **Handwritten Text Recognition.**
   Input: : Segmented Block.
   Output: : Predicted Text using Trained OCR model.

4. **Replace Handwritten Regions With Printed Text.**
   Input: : Processed Image, Predicted Text.
   Output: : Final Image.

5. **Extract Basic Details.**
   Input: : Processed Image.
   Output: : Frame/Window with all extracted information.

3 System Architecture

In this section, we shall describe in detail the architecture of the software, which aids the user in understanding how the system will behave under various circumstances. We start with the high level architecture of the software, and then move into smaller units describing the relation between various classes and methods through well-known diagrams.

The software architecture is organized in to views which can be seen as different types of blueprints, similar to those in traditional building architecture. Views can be seen as instances of a viewpoint, where the viewpoint describes the software architecture from a specific perspective. We have described the following views in our architecture.

- High Level View
- Functional View
- Flow Diagram
- User action view

3.1 High Level View

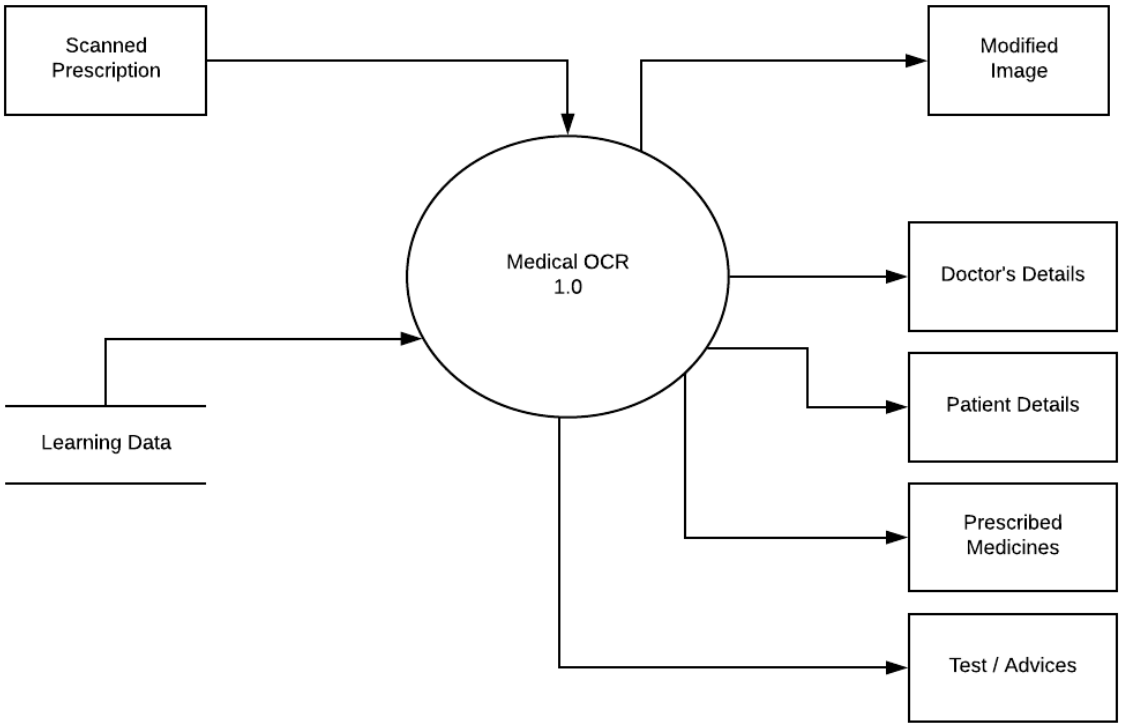

Figure 1: A high level overview of the control flow Medical OCR

The high level overview shows the interaction of the software with the external environment. It also shows the final output.

3.2 Functional View: Use Case Diagram

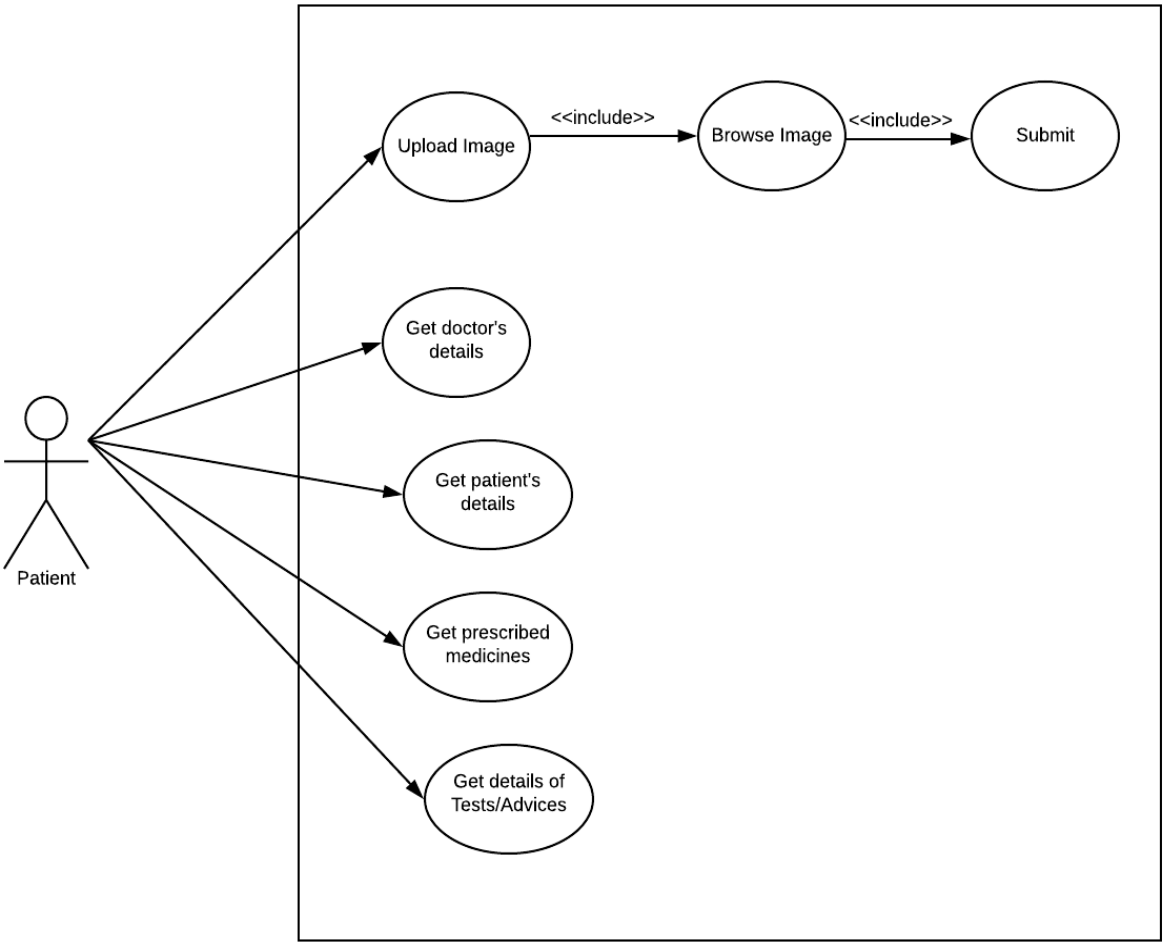

Figure 2: Use Case Diagram Medical OCR

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

In this application we have only one actor i.e. patient. The puser can get the doctor's details, patient details, medicines and test information from the prescription.

3.3 Flow Diagram

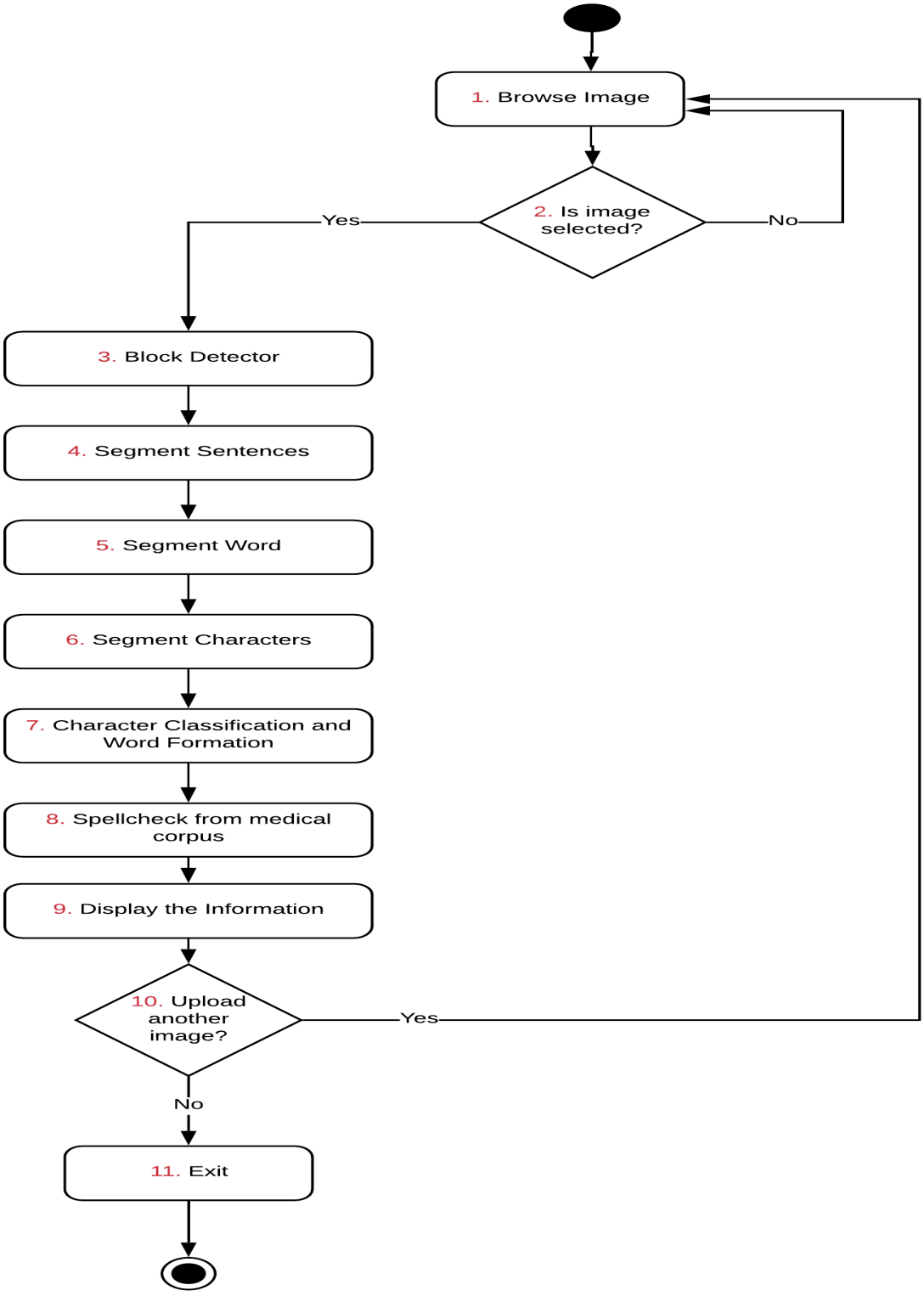

Figure 3: The Flow Diagram of Medical OCR.

A flowchart is a type of diagram that represents an algorithm, workflow or process, showing

the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. In Medical OCR first the image in selected. The written text blocks are detected. After that using deep learning models, characters are segmented and classified. The classified word is searched in the medical dictionary and the most appropriate written in place of handwritten text.

3.4 User Action View

So far, we have described the various features implemented in the application and the different classes used to organize it. However, to get a deep understanding of the system behaviour, it is important to know the happenings of the implementation within the classes.

These flow diagrams will be used in white box testing of this application, later in the document.

3.4.1 Browse images

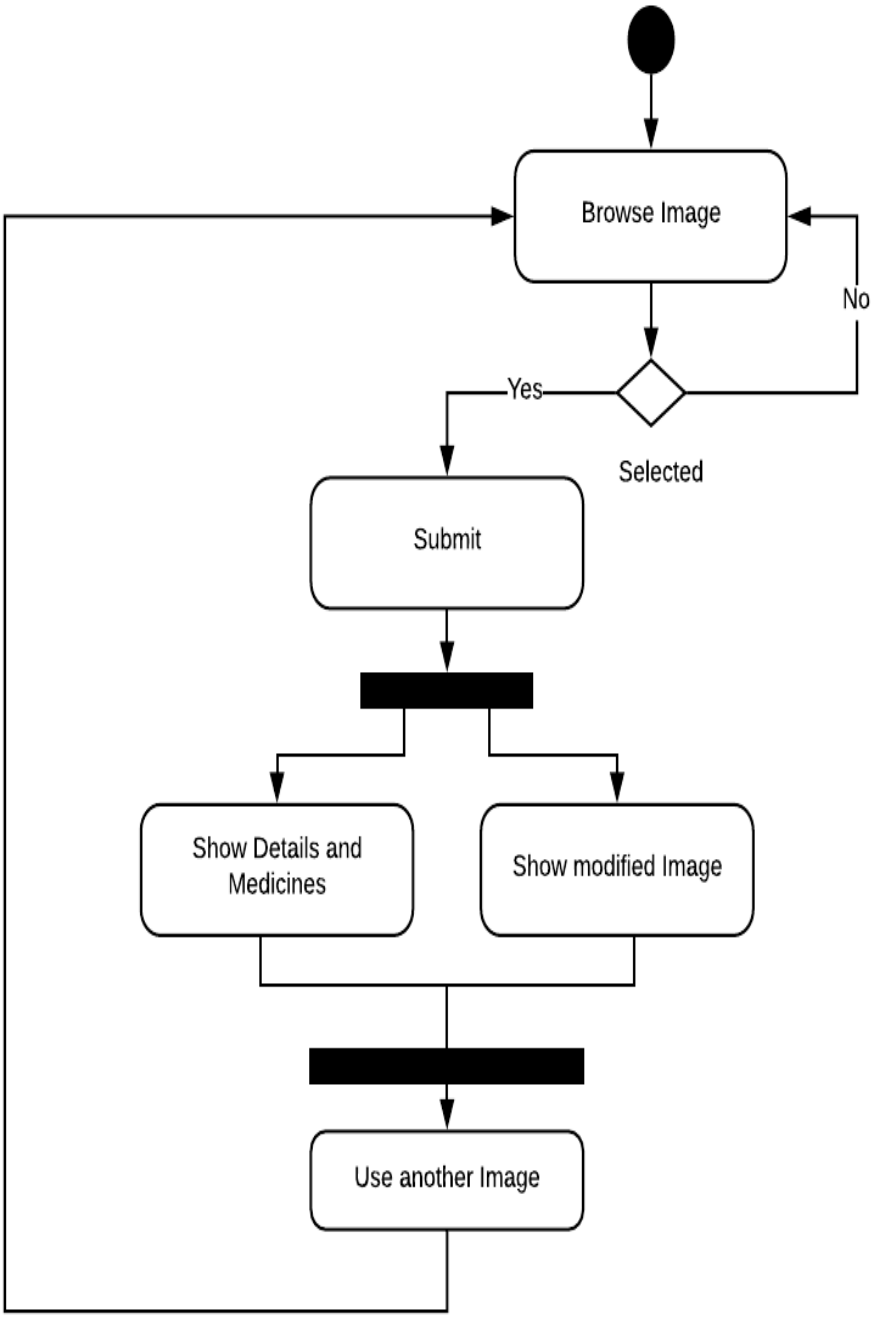

Figure 4: Flow Diagram for *Browse images*.

4 Languages, Technologies and Data Sources Used

The following table summarizes the Languages and Technologies used by various modules and parts of Medical OCR.

Languages, Libraries	Purpose
Python	Fundamental implementation of the project.
OpenCV	Used for word segmentation and for modifying prescription to display editted details.
TensorFlow	Used to implement the Recurrent Neural Network(RNN)
Pandas, NumPy	Used for data manipulation and analysis.
QtPy	Used for UI/UX design of front-end to get user input for uploading prescription image and displaying the doctor, patient and medical drugs, dosage and tests details.

5 Test Plan

In this section, we discuss the test plan for the MedicalOCR. The primary focus of this plan is to ensure that the application adheres to the customers' needs.
The project will have four levels of testing, Unit, Regression, System & Functional, and Beta Testing. The details for each level are addressed in the following sections.

5.1 Features to be Tested

The following is a list of the features to be focused on during testing:

Get the doctor and patient details from the prescription.
Get the medical drugs, tests, symptoms observed by the doctor from the prescription.

5.2 Approach

5.2.1 Testing Levels

The testing for the MedicalOCR project will consist of **Unit, Regression, System and Functional, and Beta** test levels. It is hoped that there will be at least one full time independent test person for system and functional testing.

Unit Testing will be performed by the developer using White Box testing techniques and will be approved by the development team. Proof of unit testing (test case list, sample output, defect information) must be provided by the programmer to the team before unit testing will be

accepted and passed on to the test person.

Regression Testing will be performed by the test team in parallel to the development of the software. After every modification, the software will be checked to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

System and Functional Testing will be performed by the test team and development team leader with assistance from the individual developers as required. Using black box testing techniques, testers will examine the high-level design and the customer requirements specification to plan the test cases to ensure the code does what it is intended to do. Functional testing will be done ensuring that the functionality specified in the requirement specification works. No specific test tools will be used for this project. The application will enter in this level only after all the bugs from Unit testing are removed.

Beta Testing will be performed when a partial or full version of the software is available. The software will be offered to one or more potential users (beta testers). These users will install the software and use it by referring to the user guide, and they will report any errors revealed during usage back to the developers.

5.2.2 Meetings

The test team will meet once every two days to evaluate progress to date and to identify error trends and problems as early as possible. The test team leader will meet with development team once every two days as well. Additional meetings can be called as required for emergency situations.

5.2.3 Responsibilities

1. The development team leader will be responsible for the verification and acceptance of all unit test cases.
2. The test team leader is responsible for all test cases and their documentation.
3. The entire team will participate in the review of the software as well as the review of required changes as a result of defects discovered during development and testing.

5.2.4 Test Case Pass / Fail Criteria

A test case passes the test if the actual output produced is same as the expected output. If not, the test case fails.

5.2.5 Test Deliverables

The following must be delivered before actual testing can be started:

1. Source code of each class

2. Test plan

The following must be delivered after the testing is completed:

1. Test Reports
2. Problems report (if any)

5.3 Environmental Needs

The following elements are required to support the overall testing at all:

1. Linux
2. Python
3. NumPy
4. TensorFlow
5. Pandas
6. OpenCV
7. PyQt 4

5.4 White Box Testing (Unit Testing)

White-box testing is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). We use test cases to exercise all the paths through the code. The test cases must be such that they test all the possible paths in the code.

We have highlighted flow diagram in Section 3.4. We discuss the test cases for the aforementioned below. With reference to Fig. 8

Parameters:
Boolean imageSelected, *Boolean* uploadAnother

Test Cases:
We have 3 independent paths through the code. Following are the paths and their corresponding test cases:

1. 1-2-1-2:
   $imageSelected = false$
2. 1-2-3-4-5-6-7-8-9-10-11:
   $imageSelected = true$, $uploadAnother = false$
3. 1-2-3-4-5-6-7-8-9-10-1-2:
   $imageSelected = true$, $uploadAnother = true$

5.5 Black Box Testing

Black-box testing is a method of software testing that tests the functionality of an application as opposed to its internal structures or workings (i.e. white-box testing). We have incorporated the following test cases in black-box testing. Even though the test cases are not exhaustive, it spans every command used in the application and tries to incorporate most of the exceptions possible.

ID	Description	Expected Results
1	Software is loaded	Main window to upload the prescription appears
2	**Precondition**: Main window has been launched. **Command**: Click the ”upload” button and browse the image desired.	The image gets uploaded.
3	**Precondition**: Image has been uploaded. **Command**: CLick on the ”submit” button.	All the fields get filled with the relevnt information and the processed images are shown.

6 Actual Testing Done

In this section, we describe the software development process model used and the testing that has been performed in different stages of the product development. It includes all the four types of testing mentioned in the earlier section.

6.1 Process Model

In the creation of MedicalOCR, we used the popular and simple ***Iterative Development*** model. We justify the selection of this model as follows:

- The problem was well understood.
- The project was known to be of short duration, and time was of essence.
- New requirements were realized on further developing the software, and incremental addition of features was of utmost importance.
- This model allowed the prioritizing of requirements, as well as quick deliveries of versions.
- This model also allowed the incorporation of user feedback, and appropriate changes.

6.2 Unit Testing

The unit testing has been implemented using the white box test cases described in Section 5.4. For unit testing, we wanted to write test cases such that each of these paths is tested at least once. We successfully modeled these test cases in Section 5.4.

6.3 System Testing

Ideally, we would like to test every possible thing that can be done with our program. But, as we said, writing and executing test cases is expensive. We wanted to make sure that we definitely write test cases for the kinds of things that the customer will do most often or even fairly often. Our objective was to find as many defects as possible in as few test cases as possible. We started by looking at each customer requirement to make sure that every single customer requirement has been tested at least once.

Hence, we performed system testing on our final product, on all the test cases mentioned in black box testing in Section 5.5.

Bugs were found in between which were rectified and retested using the Iterative Development model. However, on our final product, we found that the actual outputs generated by the system in both the modes were found to match the expected outputs for every test case except for two test cases mentioned below.

6.4 Regression Testing

We have used the above described Unit Tests and Black Box Tests to evaluate our system at various stages of development. Whenever a new iteration of our software was generated, we tested it on the applicable subsets of the above described tests, and proceeded to add more features only after the bugs discovered were eliminated. So, in effect we have implemented Regression Testing.

6.5 Beta Testing

Apart from the Black Box Testing done on the final product, a large number of other tests were also done before the product was moved for evaluation. They include the following.

1. The product was run on different operating systems and found to have worked on any system with Ubuntu 14.04 or higher.

2. In order to eliminate systematic errors arising out of use of a single machine for testing, we have used an array of different laptops for testing at different stages of development.

3. The user manual and the executable was handed over to around 10 third parties who had no prior knowledge of the application design. This helped us to understand the user-friendliness and ease of use of the interface design.