SE 3XA3: Software Requirements Specification
ohm

Team 4, Ohm
Jonathan Brown, brownjs2
Graeme Crawley, crawleg
Ryan Marks, marksr2

November 18, 2016

Table 1: **Revision History**

Date	**Version**	**Notes**
11/13/2016	1.0	Initial Revision

Contents

List of Tables

List of Figures

1 Introduction

1.1 Overview

The resistance scanner project is the re-implementation of an open-source software application that allows a user to scan a resistor using their camera and read an on screen value representing the resistance of that resistor.

1.2 Context

This document is the Module Guide (MG). The Module Guide, written after the SRS, is used as an outline for the implementaion of all functional and nonfunctional specified in the SRS. This is done by the decomposition of the system into modules, showing it's structure and design. A brief explination of each module is given and the requirements outlined in the SRS are directly mapped to their modular implementations. An explination of the relationships between each module are also given. This is useful both for devlopers and maintainers as it allows them to more easily identify parts of the software.

For a more detailed description of the individual components of the system, the MIS is used to explain the semantics and syntax of the member functions of the classes used in the implementation of the modules.

1.3 Design Principles

This project implements the design principles of Single Responsibility, Separation of Concerns and Least Knowledge.

- **Single Responsibility:**
- **Separation of Concerns:**
- **Principle of Least Knowledge:**

2 Anticipated and Unlikely Changes

2.1 Anticipated Changes

AC1: The method of retrieving input images (Camera/Static Image).

AC2: The method of determining the axis of a resistor.

AC3: The platform of the application (Desktop to Mobile).

2.2 Unlikely Changes

UC1: The format of the output data.

UC2: The method used to scan the bands resistor.

UC3: The method used to get colours from the bands of the resistor.

UC4: The method used to calculate values from the resistor colours.

UC5: The content of the output.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

Module Number	Module Name
M1	Camera Input Module
M2	Image Input Module
M3	User Interface Module
M4	Resistor ID Module
M5	Band Location Module
M6	Colour Mapping Module
M7	Value Calculator Module

Level 1	Level 2
Hardware-Hiding Module	Camera Input Module Image Input Module
Behaviour-Hiding Module	User Interface Module
Software Decision Module	Resistor ID Module Band Location Module Colour Mapping Module Value Calculator Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

5 Module Decomposition

The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules

Secrets: System Camera or File System

Services: Provides input frames to the modules involved in image processing (M4, M5).

Implemented By: Operating System & OpenCV

5.1.1 Camera Input Module (M1)

Secrets: Camera Hardware

Services: Provides images produced by the camera in displayable formats and as a matrix of RGB vectors.

Implemented By: Operating System & OpenCV

5.1.2 Image Input Module (M2)

Secrets: File System

Services: Provides images stored locally in displayable formats and as a matrix of RGB vectors.

Implemented By: Operating System

5.2 Behaviour-Hiding Module

Secrets: Combination and use of Software Decision modules.

Services: Describes the behaviour of the application. Acts as a bridge between the Hardware Hiding and Software Decision modules.

Implemented By: –

5.2.1 User Interface Module (M3)

Secrets: View and View Controller

Services: Displays the results of any calculations performed by the software decision module and allows for user input.

Implemented By: ohm

5.3 Software Decision Module

Secrets: The algroithms and data structures used to perform the analysis of input frames.

Services: Computes the resulting resistance based on some input image produced by the Hardware Hiding modules.

Implemented By: ohm & OpenCV

5.3.1 Resistor ID Module (M4)

Secrets: Method of Body and Axis Identification

Services: Provides the location of the main axis of the resistor for further processing.

Implemented By: ohm & OpenCV

5.3.2 Band Location Module (M5)

Secrets: Band Detection and Sampling Method

Services: Provides the colours and relative locations of each band of the resistor.

Implemented By: ohm

5.3.3 Colour Mapping Module (M6)

Secrets: Algorithm to map sampled colours to known possible colours a resistor can take on.

Services: Produces the most likely real life colour of a band and it's value used in the computation of a resistor value.

Implemented By: ohm

5.3.4 Value Calculator Module (M7)

Secrets: Method of calculating resistance.

Services: Calculates the resistance of the resistor based on the

Implemented By: ohm

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR1	M3
FR2	M4, M5, M6, M7
NF1	M3
NF2	M3
NF3	M1, M2, M3, M4, M5, M6, M7
NF4	M3
NF5	M1, M2, M3, M4, M5, M6, M7
NF6	M1, M2, M3, M4, M5, M6, M7
NF7	M2, M4, M5, M6, M7
NF8	M5, M6, M7
NF9	M4, M5, M6, M7
NF10	M3
NF11	M1, M2, M3, M4, M5, M6, M7
NF12	M1, M2, M3, M4, M5, M6, M7
NF13	M1, M2, M3, M4, M5, M6, M7
NF14	M1, M2, M3, M4, M5, M6, M7
NF15	M1, M2, M3, M4, M5, M6, M7
NF16	M1, M2, M3, M4, M5, M6, M7
NF17	M3

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M2
AC2	M4
AC3	M1, M2, M3

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

Figure 1: Use hierarchy among modules