**A Software Design Specification**

by Anshuman Saboo

**Table of Contents**

1. Introduction 2
   1. Document Outline 2
   2. Document Description 3
      1. Introduction 3
      2. System Overview 4
2. Design Considerations 4
   1. Assumptions and Dependencies 4
   2. General Constraints 4
   3. Goals and Guidelines 5
3. Architectural Strategies 5
4. System Architecture 6
   1. Subsystem Architecture 7
5. Policies and Tactics 7
6. Detailed System Design 8
   1. Classification 8
   2. Definition 8
   3. Responsibilities 8
   4. Constraints 8
   5. Composition 8
   6. Uses/Interactions 8
   7. Resources 9
   8. Processing 9
   9. Interface/Exports 9
   10. Detailed Subsystem Design 9
7. Glossary 10
8. Bibliography 10

# Introduction

Vehicle management system is to manage the parking of different type of vehicles at a predefined field. All vehicles should be allowed to enter in the parking field if there is a vacant parking spot. The product will help in the services where labour wont be required. With the help of ultrasonic sensors we will detect the entry of a vehicle. These sensors feed the information to the Arduino IDE software which will enable the LED light colour depending on the situations.

## Document Outline

* Introduction
* System Overview
* Design Considerations
  + Assumptions and Dependencies
  + General Constraints
  + Goals and Guidelines
  + Development Methods
* Architectural Strategies
  + strategy-1 name or description
  + strategy-2 name or description
  + ...
* System Architecture
  + component-1 name or description
  + component-2 name or description
  + ...
* Policies and Tactics
  + policy/tactic-1 name or description
  + policy/tactic-2 name or description
  + ...
* Detailed System Design
  + module-1 name or description
  + module-2 name or description
  + ...
* Glossary
* Bibliography

## Document Description

### Introduction

* The document is intended for both the system developers and vehicle owners.It will give new project members a brief idea about the project and they can assist with any new modification. Users will be able to understand how the management system works like what message does the different LED light portrayed. Also intended for the Advertisers, investors, plot owner etc
* The software can also help in generating profits for the investors through parking waiting time.Another source of profit can be from advertisements by providing bill boards on the plot. Expansion of the system will be only by increasing the plot size where again software programmer have to define the plot size and capabilities.

**Arduino Sensors Information**—

https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm

**Software**—

https://www.arduino.cc/en/main/softwar

Note:

For the remaining sections of this document, it is conceivable (and perhaps even desirable) that one or more of the section topics are discussed in a separate design document within the project. For each section where such a document exists, a reference to the appropriate design document is all that is necessary. All such external (or fragmented) design documents should probably be provided with this document at any design reviews.

### System Overview

Product Functions

* The primary function of the product is to assist the user with parking safely and accurately.
* Generating a Parking fee based on parking time
* Feeding instruction to software through sensors for generating different LED lights to interact with user

The Arduino and its sensors will be placed at the entry and exit gates of the parking plot. LED lights will be placed at positions where the user can see it properly. The parking plots have to be well defined as the software will need the correct amount to operate and decide the situation.

# Design Considerations

## Assumptions and Dependencies

* The assumptions are of vehicle size and differences, like we have assumed that a truck will take the parking size of two cars.

## General Constraints

* The system constitutes of an arduino board powered by a 12V Battery. We code the arduino in C sharp and feed it to the arduino where it gets converted into assembly language and gets stored in its register. After this whenever the arduino is connected to a power source the assembly code starts running in a loop.
* The C sharp code uses two specific functions and global variables. Each time the arduino is powered up using a battery or an adapter power supply the script first runs the setup function once which initializes the variables etc. The the loop function runs unless the power runs out We used the ARDUINO 1.8.8 software to code the arduino The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software

## Goals and Guidelines

The goal is to achieve an effect vehicle management system where the users will directly interact with the system without a third party which can be a helper for example.

# Architectural Strategies

Describe any design decisions and/or strategies that affect the overall organization of the system and its higher-level structures. These strategies should provide insight into the key abstractions and mechanisms used in the system architecture. Describe the reasoning employed for each decision and/or strategy (possibly referring to previously stated design goals and principles) and how any design goals or priorities were balanced or traded-off. Such decisions might concern (but are not limited to) things like the following:

* Use of a particular type of product (programming language, database, library, etc. ...)
* Reuse of existing software components to implement various parts/features of the system
* Future plans for extending or enhancing the software
* User interface paradigms (or system input and output models)
* Hardware and/or software interface paradigms
* Error detection and recovery
* Memory management policies
* External databases and/or data storage management and persistence
* Distributed data or control over a network
* Generalized approaches to control
* Concurrency and synchronization
* Communication mechanisms
* Management of other resources

Each significant strategy employed should probably be discussed in its own subsection, or (if it is complex enough) in a separate design document (with an appropriate reference here of course). Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose). Sometimes it may be most effective to employ the "pattern format" for describing a strategy.

# System Architecture

This section should provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components. Don't go into too much detail about the individual components themselves (there is a subsequent section for detailed component descriptions). The main purpose here is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together to provide the desired functionality.

At the top-most level, describe the major responsibilities that the software must undertake and the various roles that the system (or portions of the system) must play. Describe how the system was broken down into its components/subsystems (identifying each top-level component/subsystem and the roles/responsibilities assigned to it). Describe how the higher-level components collaborate with each other in order to achieve the required results. Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected). Feel free to make use of design patterns, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them.

If there are any diagrams, models, flowcharts, documented scenarios or use-cases of the system behavior and/or structure, they may be included here (unless you feel they are complex enough to merit being placed in the Detailed System Design section). Diagrams that describe a particular component or subsystem should be included within the particular subsection that describes that component or subsystem.

Note:

This section (and its subsections) really applies only to newly developed (or yet-to-be developed) portions of the system. If there are parts of the system that already existed before this development effort began, then you only need to describe the pre-existing parts that the new parts of the system depend upon, and only in enough detail sufficient to describe the relationships and interactions between the old parts and the new parts. Pre-existing parts that are modified or enhanced need to be described only to the extent that is necessary for the reader to gain a sufficient understanding of the nature of the changes that were made.

## Subsystem Architecture

If a particular component is one which merits a more detailed discussion than what was presented in the System Architecture section, provide that more detailed discussion in a subsection of the System Architecture section (or it may even be more appropriate to describe the component in its own design document). If necessary, describe how the component was further divided into subcomponents, and the relationships and interactions between the subcomponents (similar to what was done for top-level components in the System Architecture section).

If any subcomponents are also deemed to merit further discussion, then describe them in a separate subsection of this section (and in a similar fashion). Proceed to go into as many levels/subsections of discussion as needed in order for the reader to gain a high-level understanding of the entire system or subsystem (but remember to leave the gory details for the Detailed System Design section).

If this component is very large and/or complex, you may want to consider documenting its design in a separate document and simply including a reference to it in this section. If this is the option you choose, the design document for this component should have an organizational format that is very similar (if not identical to) this document.

# Policies and Tactics

* The use of ultrasonic sensors is of paramount importance as it is important to detect any kind of vehicle in the entrance bay Multiple ultrasonic sensors would be needed to analyse their collective input to recognise the vehicle
* The Entrance LED to notify the vehicle operator that his vehicle is registered by the system . Use of the counter LED to notify oncoming vehicle if the Parking lot has reached its max capacity or not . Arduino to control the flow of current to make sure every system is working properly
* Counter variables in the code to make sure the number of vehicles and the capacity of the parking lot is calculated at every moment to prevent extra vehicles from entering the parking lot. Use of a buzzer if an object is staying in the entrance bay for more than required time thus blocking the way
* The loop halter in the code is responsible for making sure the code pauses its execution when the capacity of the lot is full Another condition in the code would be needed when the lot is empty then the exit counter should stop working to save power
* When the sensors confirms the detection of the vehicle. The software is fed the information via the Arduino board. Here system registers the number of vehicles and sends a command to the LED light to glow Green.

# Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

## Classification

The kind of component, such as a subsystem, module, class, package, function, file, etc. ....

## Definition

The specific purpose and semantic meaning of the component. This may need to refer back to the requirements specification.

## Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

## Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, postconditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

## Composition

A description of the use and meaning of the subcomponents that are a part of this component.

## Uses/Interactions

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclasses, and metaclasses.

## Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

## Processing

A description of precisely how this components goes about performing the duties necessary to fulfill its responsibilities. This should encompass a description of any algorithms used; changes of state; relevant time or space complexity; concurrency; methods of creation, initialization, and cleanup; and handling of exceptional conditions.

## Interface/Exports

The set of services (resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. .... For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

Much of the information that appears in this section is not necessarily expected to be kept separate from the source code. In fact, much of the information can be gleaned from the source itself (especially if it is adequately commented). This section should not copy or reproduce information that can be easily obtained from reading the source code (this would be an unwanted and unnecessary duplication of effort and would be very difficult to keep up-to-date). It is recommended that most of this information be contained in the source (with appropriate comments for each component, subsystem, module, and subroutine). Hence, it is expected that this section will largely consist of references to or excerpts of annotated diagrams and source code. Any referenced diagrams or source code excerpts should be provided at any design reviews.

## Detailed Subsystem Design

Provide a detailed description of this software component (or a reference to such a description). Complex diagrams showing the details of component structure, behavior, or information/control flow may be included in the subsection devoted to that particular component (although, unless they are very large or complex, some of these diagrams might be more appropriately included in the System Architecture section. The description should cover any applicable software component attributes (some of which may be adequately described solely by a source code declaration or excerpt).

# Glossary

An ordered list of defined terms and concepts used throughout the document.

# Bibliography

A list of referenced and/or related publications.

Brad Appleton <brad@bradapp.net>

http://www.bradapp.net