 INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

(ESCOM)

Integrantes: Galindo Reyes Agustin

Grupo: 3cM1

Profesor: Marco Antonio Dorantes González

Asignatura: Ingeniería de Software

Proyecto: App-agador

Fecha de entrega: 31 de mayo del 2019

Semestre: 19-2



Contenido

[Capítulo 1: Introducción 5](#_Toc10166635)

[1.1 Ámbito 6](#_Toc10166636)

[1.2 Marco teórico 6](#_Toc10166637)

[1.2.1 Batería 6](#_Toc10166638)

[1.2.2 Faros del automóvil 7](#_Toc10166639)

[1.2.3 Arduino Mega 10](#_Toc10166640)

[1.2.4 Bluetooth HC-05 11](#_Toc10166641)

[1.2.5 Sensor de ultrasonidos HC-SR04 12](#_Toc10166642)

[Capítulo 2: Estado del arte 15](#_Toc10166643)

[2.1 Propuesta 15](#_Toc10166644)

[2.2 Objetivo general 15](#_Toc10166645)

[2.3 Cronograma 15](#_Toc10166646)

[2.4 Metodología 16](#_Toc10166647)

[2.4.1 Trazar el diagrama de estructura 16](#_Toc10166648)

[2.4.2 Análisis de transformación 16](#_Toc10166649)

[2.4.3 Análisis de transacción 16](#_Toc10166650)

[2.4.4 Evaluación del diseño 17](#_Toc10166651)

[2.4.5 Preparación del diseño para la implantación 17](#_Toc10166652)

[2.5 Métrica de estimación 17](#_Toc10166653)

[2.5.1 Puntos Función sin Ajustar (PFsA). 18](#_Toc10166654)

[2.5.2 Cálculo del Factor de Ajuste (VAF). 18](#_Toc10166655)

[2.5.3 Puntos de función ajustados. 20](#_Toc10166656)

[2.5.4 Cálculo de líneas de código del sistema. 21](#_Toc10166657)

[2.5.5 Modelo Constructivo de Costes (Constructive Cost Model) 21](#_Toc10166658)

[2.5.6 Modelo básico de estimación 21](#_Toc10166659)

[Capítulo 3: Análisis 22](#_Toc10166660)

[3.1 Reglas de negocio 22](#_Toc10166661)

[3.2 Requerimientos funcionales 23](#_Toc10166662)

[3.3 Requerimientos no funcionales 23](#_Toc10166663)

[3.4 Gestión de riesgos 23](#_Toc10166664)

[Capítulo 4: Diseño 24](#_Toc10166665)

[4.1 Diagrama de casos de uso 24](#_Toc10166666)

[4.2 Diagrama de secuencia 26](#_Toc10166667)

[4.3 Diagrama de clases 27](#_Toc10166668)

[4.4 Diagrama de actividades 28](#_Toc10166669)

[Capítulo 5: Codificación 29](#_Toc10166670)

[5.1 Dispositivos Bluetooth disponibles 30](#_Toc10166671)

[5.2 Interfaz principal 31](#_Toc10166672)

[5.3 Programación del Arduino 35](#_Toc10166673)

[Conclusión 37](#_Toc10166674)

[Bibliografía 38](#_Toc10166675)

[Anexos 39](#_Toc10166676)

[Análisis Estructurado 39](#_Toc10166677)

[Diagrama de contexto 39](#_Toc10166678)

[Diagrama de flujo de datos lógico 40](#_Toc10166679)

[Diagrama de flujo de datos físicos 41](#_Toc10166680)

[Análisis Orientado a objetos 41](#_Toc10166681)

[Narrativa 41](#_Toc10166682)

[Identificación de objetos 41](#_Toc10166683)

[Análisis de métodos y atributos 42](#_Toc10166684)

[Diagrama de clases 43](#_Toc10166685)

Índice de imágenes

Figura 1.1 Diagrama de la batería…………………………….…………………………………………………………………….7

Figura 1.2 Diagrama de los faros del automóvil……………………………….…………………………………………….8

Figura 1.3 Diagrama de la conexión de los faros…………………………………………………………………………….9

Figura 1.4 Imagen física del Arduino…………………… ………………………………………………………………………10

Figura 1.5 Diagrama del bluetooth HC-05………..…………………………………………….…………………………….12

Figura 1.6 Sensor ultrasónico HC-SR04…………………………………………………………………………………………13

Figura 1.7 Funcionamiento del sensor HC-SR04 ……………………………………………………….………………….15

Figura 2.1 Metodología estructurada…………….……………………………………………………………………………..17

Figura 4.1 Diagrama Caso de usos…………………………………………………………………………………………………25

Figura 4.2 Diagrama de secuencias.………………………………………………………………………………………………27

Figura 4.3 Diagrama de clases………….…………………………………………………………………………………………..28

Figura 4.4 Diagrama de actividades controlador……….………………………………………………………………….28

Figura 4.5 Diagrama de actividades del usuario….………….…………………………………………………………….29

Figura 4.6 Diagrama físico del modulo externo..……….………………………………………………………………….29

Figura A.1 Diagrama de contexto…………………………………………………………………………………………………39

Figura A.2 Diagrama de flujo de datos lógico………….……………………………………………………………………40

Figura A.3 Diagrama de flujo de datos físico.. ………………………………………………………………………………41

Figura A.4 Diagrama de clases………………………………………………………………………………………………………43

Índice de tablas

Tabla 1.1 Características del sensor HC-SR04………………………………………….…………………………………..13

Tabla 2.1 Complejidad del componente.………………………………………………………………………………….….18

Tabla 2.2 VAF…………….………………………………………………………………………………………………………………..18

Tabla 2.3 Cuestiones del sistema…….………………………………………………………………………………………….,20

Tabla 2.4 Modelo de estimación…..…………………………………………………………………………………………….,22

Tabla 3.1 Reglas de negocio..…………..…………………………………………………………………………………………,23

Tabla 3.2 Requerimientos funcionales…………………………………………………………………………………………23

Tabla 3.3 Requerimientos no funcionales.…………………………………………………………………………………..23

Tabla 3.4 Gestión de riesgos…….………………………………………………………………………………………………….25

Tabla 4.1 Actor Controlador..…….………………………………………………………………………………………………..25

Tabla 4.2 Actor Usuario……………….………………………………………………………………………………………………25

Tabla 4.3 Actor Aplicación…………….……………………………………………………………………………………………..25

Tabla 4.3 Descripción del caso de usos………………………………………………………………………………………..26

# Capítulo 1: Introducción

A lo largo de la historia del hombre, se han estado buscando maneras mas cómodas y en la actualidad mas amigables con el ambiente de trasladarse de un lugar a otro. La opción mas frecuentada por el hombre es el uso del automóvil. Sin embargo, la mayoría posee un automóvil que hace uso de una batería, la cual tiene cierto tiempo de uso. La duración media de una batería es alrededor de 4 años, pero hay veces en las que la batería se descarga de manera rápida cuando se dejan componentes prendidos del automóvil como las luces o intermitentes, es por ello que a lo largo de este documento se expondrá la importancia del desarrollo de un prototipo de aplicación que ayude al usuario a saber si dejo las luces de su automóvil prendidas para evitar las descargas de su batería y ampliar un poco la vida de la batería.

## 1.1 Ámbito

Se plantea que este prototipo de aplicación este dirigida a personas que cuenten con un celular con Android 4.4 o mayor, también este prototipo de aplicación va dirigida a personas que cuenten con un automóvil que no esté equipado con tecnología que le permita conectar su celular con el carro, ya que ese tipo de automóviles cuenta con la capacidad de realizar lo que se propone con este prototipo.

## 1.2 Marco teórico

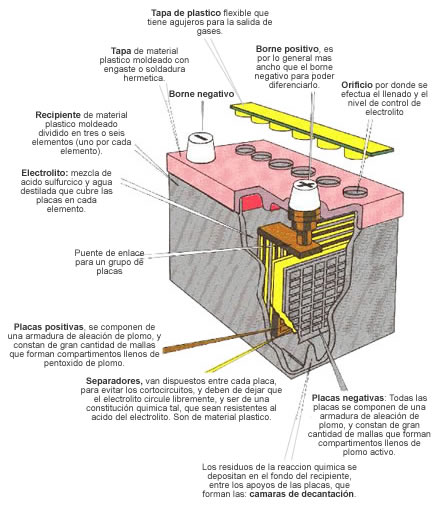
### 1.2.1 Batería

La batería es un elemento que generalmente encontramos en el bloque motor de nuestro vehículo. Su finalidad reside en el almacenaje de la energía eléctrica por medio de un proceso químico. Por ello, decimos que la batería es una pila electroquímica.

Está constituida por un acumulador que por lo general tiene nueve placas: cinco negativas y cuatro positivas, unidas de manera alterna por medio de un puente. Cada una de las partes de la batería está en un compartimento con una solución electrolítica que se compone de agua destilada y ácido sulfúrico, por lo que, al combinar esta disolución con las distintas placas de plomo, se produce una reacción química que genera corriente eléctrica. Cuando administramos electricidad a la batería, el proceso se invierte haciendo volver el sulfato desde las placas hasta el electrolito.

La máxima tensión que puede administrar este sistema es de 2,2v por lo que se unen varios en serie aumentando así el tamaño de las placas, lo que permite obtener baterías de 6, 12, 18 y hasta 24v.

Cuando hablamos de la capacidad de una batería, la expresamos en amperios por hora (Ah), es decir, la intensidad de corriente capaz de soportar sin ser recargada en una hora de uso.



**Figura 1.1 Diagrama de la batería**

La labor principal de la batería consiste en aportar la energía necesaria para la puesta en marcha del motor de arranque del vehículo que, después continúa recargándose por medio del alternador. Además, también sirve de apoyo al alternador cuando éste no puede suministrar toda la corriente que requieren otros consumidores eléctricos del automóvil, como los accesorios de confort o seguridad.

Por tanto, podemos decir que la batería, además de encargarse del sistema de encendido del motor, aporta alimentación a los equipos eléctricos cuando el automóvil está parado, como el cierre centralizado, la radio, el sistema de iluminación o el GPS. Pese a consumir una cantidad baja de energía, pueden llegar a descargar la batería si se somete a un uso prolongado de esta función.

Otra situación en la que la batería sirve de soporte al alternador es cuando el vehículo requiere una fuerte demanda de energía, como en el caso de tener el vehículo al ralentí. Si estamos en un atasco bajo la lluvia y de noche, por ejemplo, la batería cubrirá la electricidad requerida por los limpiaparabrisas, la calefacción, o los faros, ya que el motor no está aportando al alternador toda la potencia que éste necesita.

### 1.2.2 Faros del automóvil

El alumbrado del vehículo es un sistema clave en la seguridad activa dado que gracias a la iluminación podemos circular en situaciones de baja visibilidad, permitiéndonos ver con claridad, así como informando al resto de usuarios de la vía sobre nuestra presencia en la carretera, la dirección que vamos a tomar o la velocidad a la que estamos circulando.

El color de las distintas luces emitidas por nuestro vehículo está catalogado por normativa internacional, así los faros traseros serán de color rojo, los laterales o direccionales ámbar y los delanteros amarillos o de luz blanca, salvo en vehículos de emergencias.

Atendiendo a la ubicación que ocupan en las distintas partes de nuestro automóvil podemos dividir estos elementos lumínicos en tres grupos:

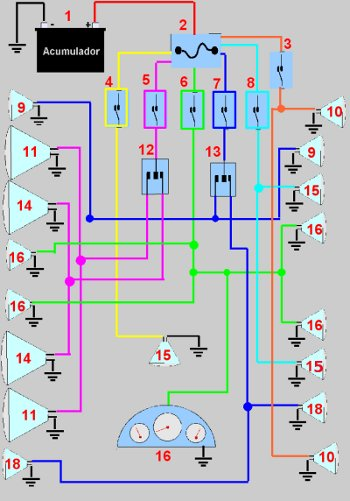
* Faros y luces auxiliares de iluminación delantera
* Faros frontales, laterales y traseros de señalización
* Luz interior de cortesía y otros dispositivos lumínicos



**Figura 1.2 Diagrama de los faros del automóvil**

Cada vez es más frecuente la utilización de circuitos electrónicos de control en el sistema de iluminación del [automóvil](http://www.sabelotodo.org/automovil/enlaceatemasautomovil.html), de esta forma en un auto actual es frecuente que las luces de carretera se apaguen solas si el conductor se descuida y las deja encendidas cuando abandona el vehículo, o, las luces de cabina utilicen temporizadores para mantenerlas encendidas un tiempo después de cerradas las puertas, de la misma forma existen diversas otras cuestiones vinculadas a la iluminación gobernadas por sistemas electrónicos. Dada esta situación se hace muy difícil generalizar, no obstante, se tratará de describir el sistema mínimo necesario.

En la Figura 1.3 se muestra un esquema de un sistema de iluminación típico de automóvil. Todos estos circuitos se alimentan a través de fusibles para evitar sobrecalentamiento de los cables en caso de posibles cortocircuitos.



**Figura 1.3 Diagrama de conexión de los faros**

En el esquema los elementos numerados son:

1.- Acumulador

2.- Caja de fusible

3.- Interruptor de luces traseras de reversa

4.- Interruptor de luz de cabina

5.- Interruptor de luz de carretera

6.- Interruptor de luces de ciudad

7.- Interruptor de luces de vía a la derecha

8.- Interruptor de luz de frenos

9.- Luces de vía

10.- Luces de reversa

11.- Luces altas de carretera

12.- Permutador de luces de carretera

13.- Interruptor de luces de vía

14.- Luces bajas de carretera

15.- Luces de frenos

16.- Luces de ciudad y tablero de instrumentos

18.- Luces de vía a la izquierda

En general cualquier automóvil tiene como mínimo Seis interruptores marcados con los números del 3 al 8 en la Figura 1.3 y cuya función es la siguiente:

|  |
| --- |
|  |

#3.- Encender las luces de reversa

#4.- Iluminar la cabina

#5.- Encender las luces de carretera

#6.- Encender las luces de ciudad

#7.- Poner a funcionar las luces de vía

#8.- Encender las luces de cola al frenar

Aunque los interruptores se han representado como uno solo por circuito, en algunos casos pueden ser varios conectados en paralelo para hacer la misma función; ejemplo: puede haber un interruptor de la luz de cabina en cada puerta y uno adicional en el tablero, o en la propia lámpara. Es muy frecuente un interruptor adicional para encender las luces intermitentes de avería.

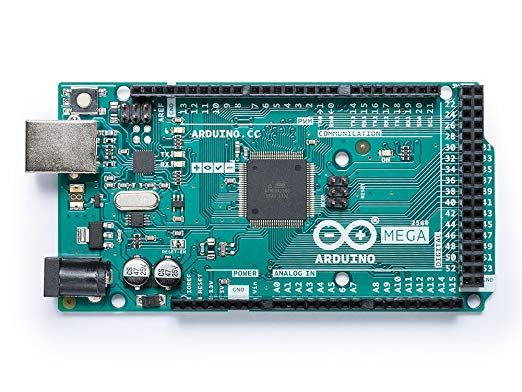
Dos permutadores de luces, uno para permutar las luces de carretera de altas a bajas y otro para seleccionar las luces intermitentes de vía de acuerdo con el giro a efectuar. Como indicadores de vía en algunos vehículos se usan las propias lámparas de frenos, en otros, lámparas aparte, comúnmente de color amarillo o ámbar.

### 1.2.3 Arduino Mega

Arduino es una marca de microcontroladores mundialmente conocida por los amantes de la electrónica, la programación y la robótica. Es un proyecto Open Source que pone a disposición de sus usuarios una amplia gama de dispositivos basados en el microcontrolador AtMega. Es posible comprar una placa Arduino armada o conseguir las piezas para uno mismo desarrollar sus propios dispositivos.

El Arduino Mega es probablemente el microcontrolador más capaz de la familia Arduino. Posee 54 pines digitales que funcionan como entrada/salida; 16 entradas análogas, un cristal oscilador de 16 MHz, una conexión USB, un boton de reset y una entrada para la alimentación de la placa.

La comunicación entre la computadora y Arduino se produce a través del Puerto Serie. Posee un convertidor usb-serie, por lo que sólo se necesita conectar el dispositivo a la computadora utilizando un cable USB como el que utilizan las impresoras.



**Figura 1.4 Imagen física del Arduino**

Arduino Mega posee las siguientes especificaciones:

* Microcontrolador: ATmega2560
* Voltaje Operativo: 5V
* Voltaje de Entrada: 7-12V
* Voltaje de Entrada(límites): 6-20V
* Pines digitales de Entrada/Salida: 54 (de los cuales 15 proveen salida PWM)
* Pines análogos de entrada: 16
* Corriente DC por cada Pin Entrada/Salida: 40 mA
* Corriente DC entregada en el Pin 3.3V: 50 mA
* Memoria Flash: 256 KB (8KB usados por el bootloader)
* SRAM: 8KB
* EEPROM: 4KB
* Clock Speed: 16 MHz

Arduino Mega puede ser alimentado mediante el puerto USB o con una fuente externa de poder. La alimentación es seleccionada de manera automática.

Cuando se trabaja con una fuente externa de poder se debe utilizar un convertidor AC/DC y regular dicho voltaje en el rango operativo de la placa. De igual manera se puede alimentar el micro mediante el uso de baterías. Preferiblemente el voltaje debe estar en el rango de los 7V hasta los 12V.

Arduino Mega posee algunos pines para la alimentación del circuito aparte del adaptador para la alimentación:

* VIN: A través de este pin es posible proporcionar alimentación a la placa.
* 5V: Podemos obtener un voltaje de 5V y una corriente de 40mA desde este pin.
* 3.3V: Podemos ubtener un voltaje de 3.3V y una corriente de 50mA desde este pin.
* GND: El ground (0V) de la placa.

Arduino puede ser programado de una manera muy fácil utilizando el lenguaje propio de Arduino junto con la interfaz Arduino IDE.

### 1.2.4 Bluetooth HC-05

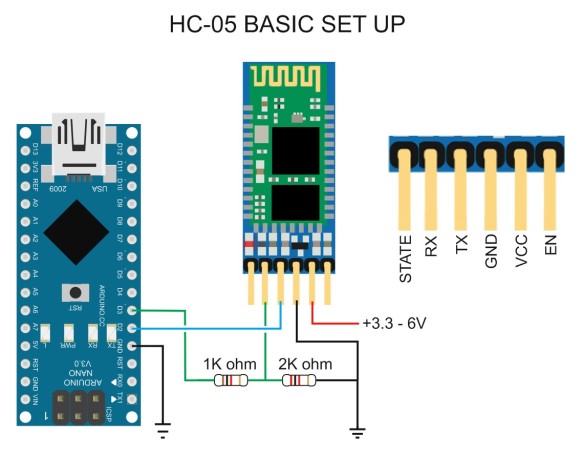
El módulo Bluetooth HC-05 es idea para utilizar en todo tipo de proyectos donde necesites una conexión inalámbrica fiable y sencilla de utilizar. Se configura mediante comandos AT y tiene la posibilidad de hacerlo funcionar tanto en modo maestro como esclavo. Eso quiere decir que puedes conectar dos módulos juntos, conectar tu robot al móvil o incluso hacer una pequeña red de sensores comunicados entre ellos con un maestro y varios esclavos.

El módulo Bluetooth HC-05 puede alimentarse con una tensión de entre 3.3 y 6V (normalmente 5V), pero los pines TX y RX utilizan niveles de 3,3V por lo que no se puede conectar directamente a placas de 5V. Debes utilizar dos pequeñas resistencias como divisor de tensión para que el módulo no se estropee. En las imágenes del producto podrás ver el esquema de conexionado. También dispone de un pulsador para entrar en modo comandos, aunque también lo puedes hacer por software utilizando el pin EN.

Tiene un LED incorporado que indica el estado de la conexión y si está emparejado o no en función de la velocidad del parpadeo.

**Características:**

* Protocolo Bluetooth: v1.1 / 2.0.
* Frecuencia: banda ISM de 2,4 GHz.
* Modulación: GFSK
* Potencia de transmisión: menos de 4dBm, Clase 2.
* Sensibilidad: Menos de -84dBm en el 0,1% BER.
* Ratio asíncronos: 2.1Mbps (Max) / 160 kbps.
* Síncrono: 1Mbps / 1Mbps.
* Perfiles de la ayuda: puerto serie Bluetooth (maestro y esclavo).
* Fuente de alimentación: + 3.3VDC 50mA. (soporta de 3.3 a 6V)
* Temperatura de trabajo: -5 ° C a 45 ° C.



**Figura 1.5 Diagrama del bluetooth HC-05**

### 1.2.5 Sensor de ultrasonidos HC-SR04

Dentro del mundo Arduino, el emisor/receptor de ultrasonidos HC-SR04 es quizás uno de los complementos más reconocibles. Este sensor se emplea en todo tipo de proyectos, siendo su principal uso el de reconocer obstáculos, aunque debido su buena resolución, también se emplea para medir las distancias a objetos.

Lo más singular del sensor de ultrasonidos HC-SR04 es quizás su "par de ojos", estos no son más que un emisor y un receptor de ultrasonidos que trabajan a una frecuencia de 40KHz (una frecuencia inaudible para las personas).



**Figura 1.6 Sensor ultrasónico HC-SR04**

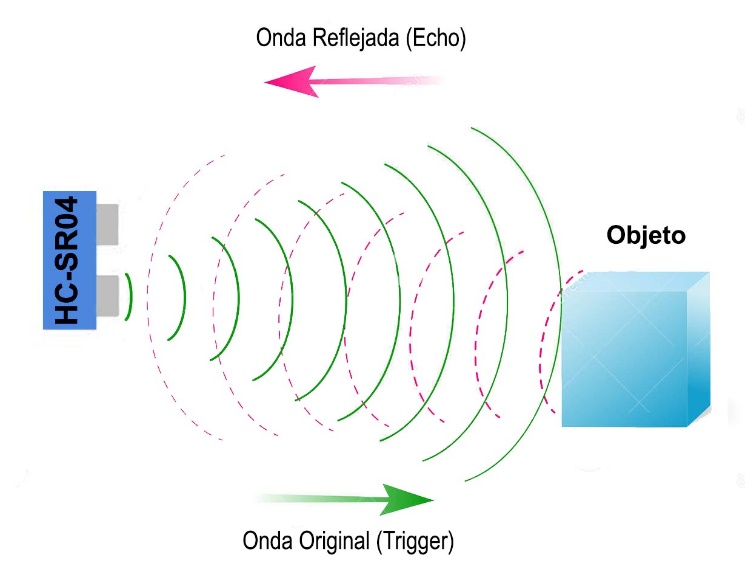
Este sensor consta de 4 pines, alimentación (Vcc), un disparador (Trig), el receptor (Echo) y masa (GND). Para el correcto funcionamiento del sensor es necesario el conectar estos 4 pines.

Las características básicas de este sensor y que se deben de tener en cuenta al trabajar con él son:



**Tabla 1.1 Características del sensor HC-SR04**

El principio en el que se basa su funcionamiento es muy sencillo, tan solo hay que generar una onda sónica en el emisor mediante un pulso en la patilla que pone "trig" (trigger o disparador), esta onda al encontrarse con algún obstáculo rebotará, volviendo al sensor y siendo registrada por el receptor, traduciéndose esta en un pulso en la patilla "Echo".



**Figura 1.7 Funcionamiento del sensor HC-SR04**

Con esto podemos hacer dos cosas, detectar un obstáculo esperando simplemente que Arduino reciba un "Echo" o contar el tiempo que transcurre desde que se manda el pulso por el trigger hasta que se recibe, de esta forma, y conociendo cual es la velocidad del sonido, podemos determinar de forma muy sencilla la distancia exacta a la que se encuentra el objeto en el que esta rebotando la señal.

Para aclarar un poco el factor de multiplicación que vamos a introducir en Arduino, basta con decir que la velocidad es igual al espacio dividido por el tiempo que se tarda en recorrer dicho espacio. La velocidad del sonido es conocida (343m/s) y el tiempo lo vamos a determinar, como el tiempo que transcurre desde que efectuamos el disparo hasta que recibimos el eco.

Aquí tenéis un pequeño resumen del cálculo que hay que hacer:

Pero como la onda ha recorrido el camino dos veces (ida y vuelta) hay que dividir entre dos para conocer la distancia a la que se encuentra el objeto.

# Capítulo 2: Estado del arte

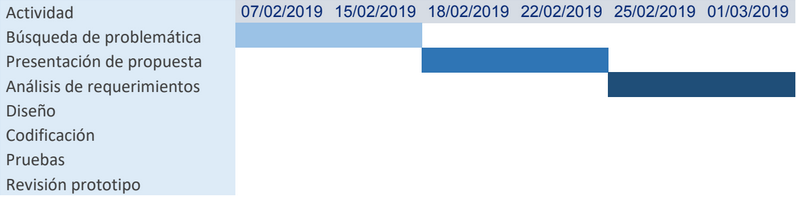
## 2.1 Propuesta

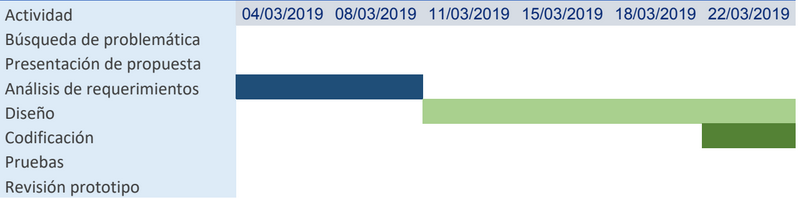
Se propone desarrollar una aplicación que nos permita recibir notificaciones en caso de que el usuario deje los faros de su automóvil encendidos. El usuario será capaz de apagar los faros desde la aplicación. Esta aplicación tiene como propósito ayudar al usuario a evitar que su batería se descargue por ese tipo de descuidos.

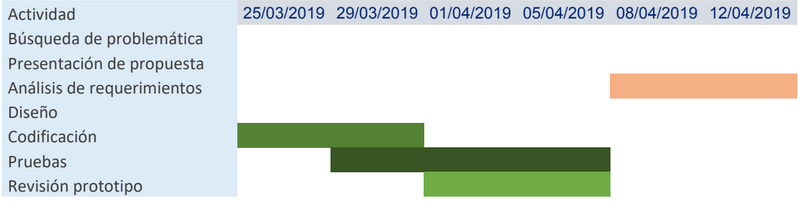
## 2.2 Objetivo general

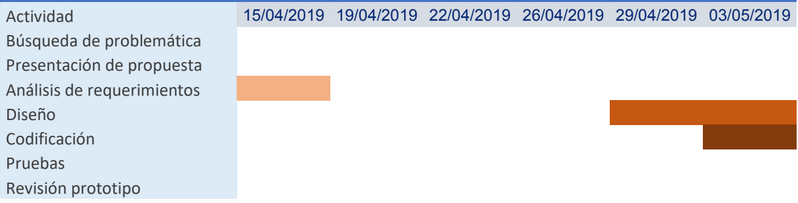
El objetivo general de este proyecto es ayudar al usuario a aplazar el tiempo de vida de la batería, notificándole por medio de una aplicación móvil, si el dejo algún faro de su automóvil encendido, de manera que el los pueda apagar desde la aplicación.

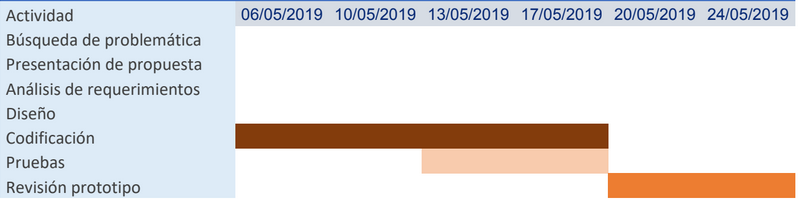
## 2.3 Cronograma











## 2.4 Metodología

Esta metodología proporciona una manera para diseñar paso a paso sistemas y programas detallados. Cabe mencionar que unos pasos involucran el análisis, otros el desarrollo del diseño y otros más la medición y la mejora de la calidad del diseño. La principal herramienta generada en el diseño estructurado es el “diagrama de estructura” donde muestra los componentes de procedimientos del programa, su ordenación jerárquica y los datos conectados a ellos. El diagrama de estructura es un diagrama de árbol o jerárquico que, en términos generales, define la arquitectura global de un programa que muestra los procedimientos y sus interrelaciones. En dicho diagrama se utilizan bloques básicos, como son cajas que representan los componentes de procedimientos y las flechas que muestran cómo se conectan. Yourdon en su metodología propone en cuatro pasos el proceso de diseño. A continuación, se explicará cada uno.

### 2.4.1 Trazar el diagrama de estructura

En este punto se desea representar el diseño del programa como una jerarquía de componentes de procedimiento. El diagrama de estructura se deriva del diagrama de flujo de datos obtenido previamente. El diseño estructurado proporciona dos estrategias de diseño para guiar la transformación respectiva, las cuales son: los análisis de transformación y los análisis de transacción. Estas dos estrategias nos ayudan a dirigir el diseño jerárquico, así como un proceso paso a paso de transformación por cada estrategia.

### 2.4.2 Análisis de transformación

Este modelo de flujo de información divide al diagrama de flujo de datos (DFD) en tres partes: la entrada que recibe el nombre de rama aferente; el proceso lógico llamado transformación central; y la salida, denominada rama eferente.

### 2.4.3 Análisis de transacción

Este modelo se utiliza cuando se diseñan programas con proceso de transacciones. El diagrama de estructura general para un programa con procesos de transacciones se representa en la parte superior por el módulo de la transacción central y en la parte inferior hay varios módulos de transacciones para cada tipo distinto de transacción.

### 2.4.4 Evaluación del diseño

En este punto la medición de la calidad de diseño es fundamental, para ello se utilizan dos técnicas ya conocidas, como son el acoplamiento y la cohesión.

El acoplamiento mide el grado de independencia entre los componentes de los procedimientos (módulos) en el diagrama de estructura.

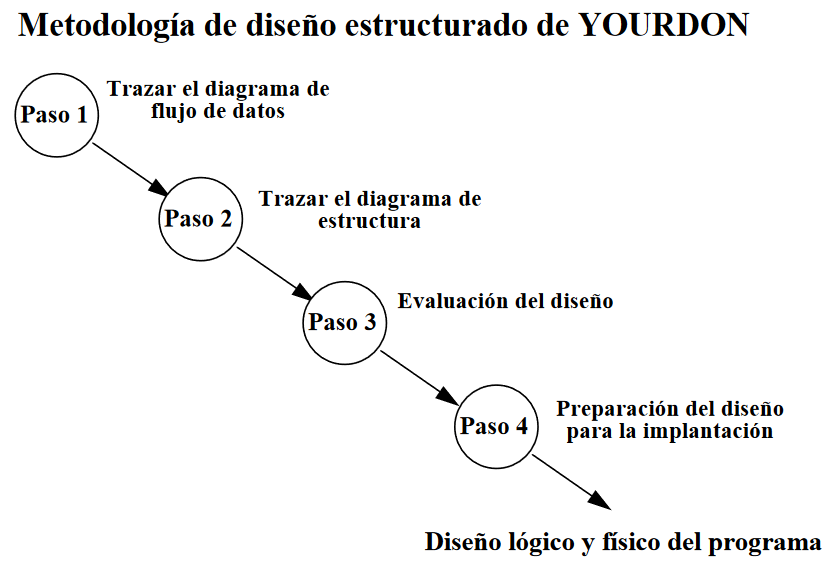
La cohesión mide la fuerza de las relaciones entre los elementos dentro de un módulo.

Lo ideal es tener un bajo acoplamiento y un alto grado de cohesión.

### 2.4.5 Preparación del diseño para la implantación

Esta parte también es conocida como empaquetar el diseño. Empaquetar es el proceso de dividir el diseño del programa lógico en unidades físicas de implantación llamadas unidades de carga. De hecho, es un diseño físico del programa.

En la siguiente figura se muestra los pasos básicos del diseño de Yourdon.



**Figura 2.1 Metodología estructurada**

## 2.5 Métrica de estimación

El Análisis de Puntos Función nos proporcionará una medida objetiva de la funcionalidad de nuestra aplicación software, ayudándonos en la evaluación, planificación, gestión y control de los procesos de desarrollo software. Inicialmente propuesto por Allan Albrecht en 1979, está basado en la teoría de que las funciones de una aplicación software son la mejor medida de su tamaño. Se trata de un método utilizado en la medición del tamaño del software, desde el punto de vista de los requisitos funcionales que debe implementar. Por esta razón, se considera este método para tratar de determinar el número total de puntos función del sistema.

### 2.5.1 Puntos Función sin Ajustar (PFsA).

Después de un análisis al sistema que se está desarrollando se encontraron los siguientes componentes de este:

* Entradas externas: 3.
* Salidas externas: 3.
* Consultas externas: 3.
* Ficheros lógicos internos: 0.
* Ficheros externos de interfaz: 0.

En la siguiente tabla se muestra como fue considerada la complejidad de los componentes para obtener los puntos de función sin ajustar del sistema.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Complejidad del componente | | | | |
| Componente | Baja | Media | Alta | Total |
| Entradas externas | 0 x 3=0 | 1 x 4 = 4 | 3 x 6 = 18 | 22 |
| Salidas Externas | 0 x 4= 0 | 0 x 5 = 0 | 3 x 7 = 21 | 21 |
| Consultas Externas | 0 x 3 = 0 | 3 x 4 = 12 | 0 x 6 = 0 | 12 |
| Ficheros Lógicos Internos | 0 x 7 = 0 | 0 x 10 = 0 | 0 x 15 = 15 | 0 |
| Ficheros Lógicos Externos | 0 x 5 = 0 | 0 x 7 = 21 | 0 x 10 = 0 | 0 |
| Total | | | | 55 |

**Tabla 2.1 Complejidad del componente**

### 2.5.2 Cálculo del Factor de Ajuste (VAF).

El VAF está basado en 14 características generales del sistema (General System Characteristics ó GSC’s) que evalúan la funcionalidad general de la aplicación que se está midiendo. Cada GSC tiene asociada una serie de cuestiones o preguntas acerca de la misma, cuya respuesta ayuda a determinar su grado de importancia dentro del sistema en función de una escala que va de cero (sin influencia) a cinco (esencial), según se muestra en la siguiente tabla:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Valor | 0 | 1 | 2 | 3 | 4 | 5 |
| Significado | Sin influencia | Incidental | Moderado | Medio | Significativo | Esencial |

**Tabla 2.2 VAF**

En la siguiente tabla se muestran las 14 características generales del sistema, debidamente punteadas para el proyecto realizado.

|  |  |  |  |
| --- | --- | --- | --- |
| Características generales del sistema | | Cuestiones | Influencia |
| 1 | Comunicación de datos | ¿Qué necesidades de comunicación  requiere el sistema para transferencia o  intercambio de información? | 5 |
| 2 | Procesamiento de datos distribuidos | ¿Existen funciones de procesamiento  distribuido?  ¿Cómo son manejados los datos  distribuidos? | 3 |
| 3 | Rendimiento | ¿Es importante el tiempo de respuesta?  ¿Es crítico el rendimiento? | 5 |
| 4 | Uso del hardware existente | ¿En qué medida se está utilizando la  plataforma hardware en donde se  ejecutará la aplicación? | 5 |
| 5 | Transacciones | ¿Con qué frecuencia se ejecutan las  transacciones? (diariamente,  semanalmente, mensualmente, etc..) | 5 |
| 6 | Entrada de datos interactiva | ¿Requiere el sistema entrada de datos  interactiva?  ¿Cuánta información se captura on-line? | 5 |
| 7 | Eficiencia | ¿Se diseñó la aplicación pensando en  que fuera eficiente y fácilmente utilizable  por el usuario? | 5 |
| 8 | Actualizaciones on-line | ¿Cuántos ficheros Lógicos Internos se  actualizan interactivamente (por medio  de transacciones on-line)? | 4 |
| 9 | Complejidad de procesamiento | ¿Existe mucha carga en cuanto a  procesamiento lógico y/o matemático?  ¿Es complejo el procesamiento interno? | 3 |
| 10 | Reusabilidad | ¿Se desarrolló la aplicación para cumplir  las necesidades de más de un usuario?  ¿Se ha diseñado el código para ser  reutilizable? | 3 |
| 11 | Facilidad de conversión e instalación | ¿Cómo son de difíciles la conversión y la  instalación?  ¿Se ha incluido en el diseño la  conversión y la instalación? | 3 |
| 12 | Facilidad de operación | ¿Requiere el sistema copias de  seguridad y de recuperación fiables?  ¿Cómo son de efectivos y qué grado de  automatización tienen los procesos de  arranque, copia de seguridad y  recuperación de datos? | 2 |
| 13 | Múltiples Instalaciones | ¿Se diseñó y desarrolló el sistema para  soportar múltiples instalaciones en  diferentes organizaciones? | 3 |
| 14 | Facilidad de mantenimiento | ¿Se diseñó y desarrolló el sistema  pensando en facilitar el posterior  proceso de mantenimiento? | 3 |
| Suma total | | | 54 |

**Tabla 2.3 Cuestiones del sistema**

Una vez que se ha determinado la influencia de cada GSC en el sistema (valor entre 0 y 5), se utiliza la siguiente fórmula para obtener el valor del VAF:

Con lo cual nuestro valor de ajuste quedo determinado de la siguiente manera.

### 2.5.3 Puntos de función ajustados.

Una vez obtenidos los puntos de función no ajustados y el factor de ajuste, el obtener los puntos de función ajustados es bastante sencillo solo es necesario realizar la siguiente operación:

La cual nos da como resultado que los puntos de función de nuestro sistema son de 65.45, con lo cual ahora nos es posible determinar las líneas de código de nuestro proyecto que se presenta en la siguiente sección.

### 2.5.4 Cálculo de líneas de código del sistema.

Conociendo los puntos de función totales del sistema es posible estimar el número total de líneas de código para su desarrollo, según QSM (Quantitative Software Management) el número de líneas de código promedio para un punto de función de un sistema de software es de 63 LDC en el lenguaje JavaScript, para fines de la estimación nosotros consideraremos 70 LDC por no ser expertos en el lenguaje.

Por lo cual al realizar la operación mostrada a continuación de obtiene que el número estimado de líneas de código para el desarrollo del proyecto.

Este resultado es importante, porque con base en el se realizan los cálculos necesarios para estimar otros factores en el desarrollo en la siguiente sección.

### 2.5.5 Modelo Constructivo de Costes (Constructive Cost Model)

El Modelo Constructivo de Costes COCOMO -Constructive Cost Model– es utilizado en proyectos de software para estimar los costes de este en función de tres submodelos: básico, intermedio y detallado.

En este documento se hará uso de la estimación de este modelo utilizando su submodelo básico, esta decisión es tomada con base en los datos conocidos y la siguiente descripción del submodelo:

“El modelo básico estima el coste del proyecto –pequeño o mediano- en función de número de líneas de código estimadas. En este modelo, el algoritmo COCOMO establece varios criterios de desarrollo, dependiendo el nivel de dificultad no del nivel de experiencia de los desarrolladores –que se supone- sino de posibles dificultades que se pueden encontrar en el desarrollo o limitaciones del hardware usado en el desarrollo del software.”

A continuación, se describirán las estimaciones realizadas con este submodelo.

### 2.5.6 Modelo básico de estimación

Se utiliza para obtener una primera aproximación rápida del esfuerzo, cada submodelo también se divide en modos que representan el tipo de proyecto, y puede ser:

* **modo orgánico**: un pequeño grupo de programadores experimentados desarrollan software en un entorno familiar. El tamaño del software varía desde unos pocos miles de líneas (tamaño pequeño) a unas decenas de miles (medio).
* **modo semilibre o semi encajado**: corresponde a un esquema intermedio entre el orgánico y el rígido; el grupo de desarrollo puede incluir una mezcla de personas experimentadas y no experimentadas.
* **modo rígido o empotrado**: el proyecto tiene fuertes restricciones, que pueden estar relacionadas con la funcionalidad y/o pueden ser técnicas. El problema por resolver es único y es difícil basarse en la experiencia, puesto que puede no haberla.

Hace uso de la siguiente tabla de constantes para calcular distintos aspectos de costes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modo | A | B | C | D |
| Orgánico | 2.40 | 1.05 | 2.50 | 0.38 |
| Semi orgánico | 3.00 | 1.12 | 2.50 | 0.35 |
| Empotrado | 3.60 | 1.20 | 2.50 | 0.33 |

**Tabla 2.4 Modelo de estimación**

Estos valores son para las fórmulas:

* Personas necesarias por mes para llevar adelante el proyecto (MM) = a\*(Klb)
* Tiempo de desarrollo del proyecto (TDEV) = c\*(MMd)
* Personas necesarias para realizar el proyecto (CosteH) = MM/TDEV
* Costo total del proyecto (CosteM) = CosteH \* Salario medio entre los programadores y analistas.

Nuestro por las características de líneas de código y un poco de experiencia en desarrollo de software, nos basaremos en el modelo orgánico de estimación. Nuestro valor de líneas de código se obtuvo de 4581.5 LDC, para la estimación COCOMO se utilizó de 4.58 KLDC y un salario promedio de $30 000 MXN. A continuación, se muestran los valores obtenidos.

# Capítulo 3: Análisis

Para el análisis del proyecto se utilizará un modelo de prototipos. El modelo de proceso a utilizar es de prototipo, porque permite realizar un análisis y diseño rápido con lo cual se puede desarrollar un sistema prototipo que reduce el riesgo de construir un producto que no satisfaga las necesidades del usuario, una vez aprobado el prototipo se pasa al desarrollo tomando como base el prototipo y realizando un análisis y diseño más riguroso para cubrir vulnerabilidades y/o faltas de funcionalidad.

## 3.1 Reglas de negocio

|  |  |  |
| --- | --- | --- |
| ID | Nombre | Descripción |
| RDN1 | Controlador de faros | El usuario permitirá que se realicen modificaciones en su automóvil para la implementación del controlador de los faros |
| RDN2 | Bluetooth | El celular del usuario deberá de contar con conexión bluetooth para la comunicación entre la aplicación y el controlador |
| RDN3 | Sensor ultrasónico | El usuario permitirá la modificación del automóvil para colocar el sensor en donde sea conveniente |
| RDN4 | Rango de funcionamiento1 | El rango de distancia para activar la notificación será de 40 cm a 250 cm |
| RDN5 | Instalación | Se le proporcionara al usuario el APK de la aplicación para que la instale en su celular |

**Tabla 3.1 Reglas de negocio**

## 3.2 Requerimientos funcionales

|  |  |  |
| --- | --- | --- |
| ID | Nombre | Descripción |
| RF1 | Verificación de estado de faros | El modulo conectado al carro será capaz de verificar si los faros están encendidos en caso de que no se detecte un conductor |
| RF2 | Generación de notificaciones | El módulo conectado al carro será capaz de generar notificaciones en caso de que los faros del automóvil estén encendidos |
| RF3 | Visualizador de notificaciones | El usuario será capaz de visualizar las notificaciones en su teléfono |
| RF4 | Cambio de estado de los faros | El usuario será capaz de apagar los faros desde la aplicación |
| RF5 | Recepción de respuestas | El modulo conectado al carro será capaz de recibir la respuesta del usuario |
| RF6 | Control de faros | El modulo conectado al carro será capaz de controlar el estado del faro dependiendo de la respuesta del usuario |

**Tabla 3.2 Requerimientos funcionales**

## 3.3 Requerimientos no funcionales

|  |  |  |
| --- | --- | --- |
| ID | Nombre | Descripción |
| RNF1 | Interfaz de la aplicación | Se diseñará una interfaz intuitiva para comodidad del usuario |
| RNF2 | Diseño del módulo externo | Se diseñará un modulo que no ocupe demasiado espacio sin restar la eficacia de este |
| RNF3 | Lenguaje de programación | Se utilizará java para Android para el desarrollo de la aplicación y se ocupará C++ para la implementación de Arduino |
| RNF4 | Tiempo de respuesta | El tiempo de respuesta entre el controlador y la aplicación no deberá de tardar más de 2 minutos, de lo contrario se volverá a hacer la solicitud |

**Tabla 3.3 Diagrama de la batería**

## 3.4 Gestión de riesgos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Riesgo | Categoría | Probabilidad | Impacto |
| 01 | Fallo en el modulo de control de faros | Riesgo técnico | 70% | 5 |
| 02 | Planificación demasiado optimista | Riesgo de gestión del proyecto | 85% | 2 |
| 03 | Falta de capacidad por parte del usuario al manejar la aplicación | Riesgo externo | 65% | 3 |
| 04 | Cambios de requerimientos | Riesgo técnico | 60% | 3 |
| 05 | Compatibilidad | Riesgo técnico | 10% | 4 |
| 06 | Fallo en la comunicación controlador-aplicación | Riesgo técnico | 50% | 5 |

**Tabla 3.4 Gestión de riesgos**

# Capítulo 4: Diseño

## 4.1 Diagrama de casos de uso

Durante la especificación del diagrama de casos de usos se encontraron 2 actores, los cuales estarán interactuando con el sistema y los cuales se presentan en las siguientes tablas:

|  |  |
| --- | --- |
| Actor | Controlador |
| Descripción | El controlador será el encargado de checar el estado de los faros del automóvil, en caso de que encuentre los faros encendidos y no detecte ningún conductor cerca, generara una notificación la cual se enviara al celular del conductor, también leerá una respuesta y será capaz de cambiar el estado de los faros dependiendo de la respuesta del usuario. |
| Responsabilidades | Verificar estado de faros  Generar notificaciones  Envió de notificaciones  Control de faros |

**Tabla 4.1 Actor Controlador**

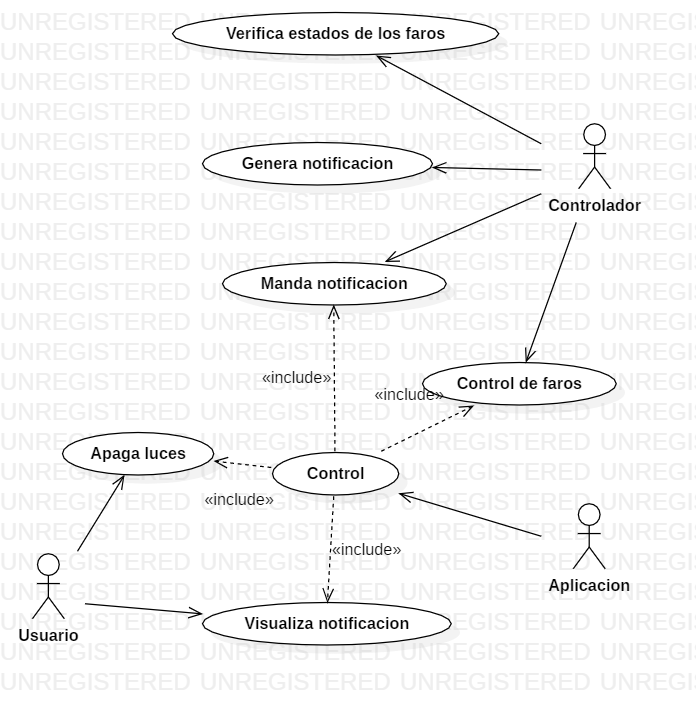
|  |  |
| --- | --- |
| Actor | Usuario |
| Descripción | El usuario será capaz de visualizar las notificaciones generadas por el controlador en una aplicación, en caso de que deje los faros de su automóvil encendidos, este será capaz mandar una respuesta de apagado desde la aplicación. |
| Responsabilidades | Visualizador de notificaciones  Apagar faros |

**Tabla 4.2 Actor Usuario**

|  |  |
| --- | --- |
| Actor | Aplicación |
| Descripción | La aplicación se encargará de controlar la comunicación entre el usuario y el controlador |
| Responsabilidades | Control de comunicación |

**Tabla 4.3 Actor Aplicación**

A continuación, se presenta el diagrama de casos de uso del proyecto.



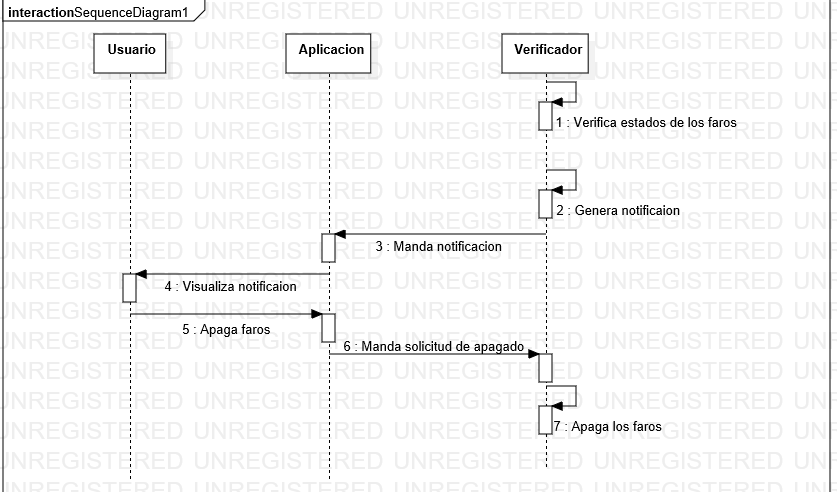
**Figura 4.1 Diagrama Caso de usos**

|  |  |
| --- | --- |
| Caso de Uso | CU.1 Funcionamiento |
| Actor | Controlador  Usuario  Aplicación |
| Descripción | El controlador verificara el estado de los faros, en caso de estar encendidos, este generara una notificación y la enviara a la aplicación, una vez que la aplicación reciba la notificación esta podrá ser visualizada por el usuario, el será capaz de apagar las luces desde la app con lo cual generara una respuesta que se enviara a la aplicación, una vez que la aplicación tenga la respuesta esta será mandada al controlador, el cual será capaz de cambiar el estado del faro de encendido a apagado. |
| Flujo básico | **1. Verifica estado de las luces** En este paso el controlador verifica el estado de los faros, en caso de que este encendido cualquier faro y no detecte ningún conductor cerca. **2. Generador de notificaciones**  En este paso se genera una notificación en caso de que el controlador haya detectado algún faro encendido y no se detecte algún conductor cerca.  **3. Mandar notificación**  Si el controlador genero alguna notificación, este paso se encarga de enviarla a la aplicación para que el usuario pueda visualizar la información.  **4. Control**  El control es como se conoce la aplicación, donde estará administrando el envió de notificaciones y respuestas, este paso es el encargado de la interacción entre el controlador y el usuario.  **5. Visualizar notificación**  Este paso es el encargado de mostrarle al usuario la notificación que genero el controlador  **6. Apagar luces**  En este paso el usuario manda la respuesta al controlador de que quiere apagar las luces  **7. Control de faros**  El controlador interpreta la respuesta del usuario y dependiendo de ella, puede cambiar el estado del faro de encendido a apagado. |
| Pre-condiciones | **1. Inicio de la generación de notificaciones** Para que el controlador pueda generar una notificación debe de estar encendido al menos un faro del automóvil y no se debe detectar algún usuario cerca del automóvil.  **2. Recepción de notificaciones**  Para poder realizar la recepción de las notificaciones el usuario deberá mantener encendido el bluetooth de su smartphone. |
| Post-condiciones | **1. Finalización de las notificaciones** Para que el controlador deje de mandar notificaciones deberá de detectar que todos los faros del automóvil estén apagados. |

**Tabla 4.4 Descripción del caso de usos**

## 4.2 Diagrama de secuencia

A continuación, se presenta el diagrama de secuencia del proyecto.

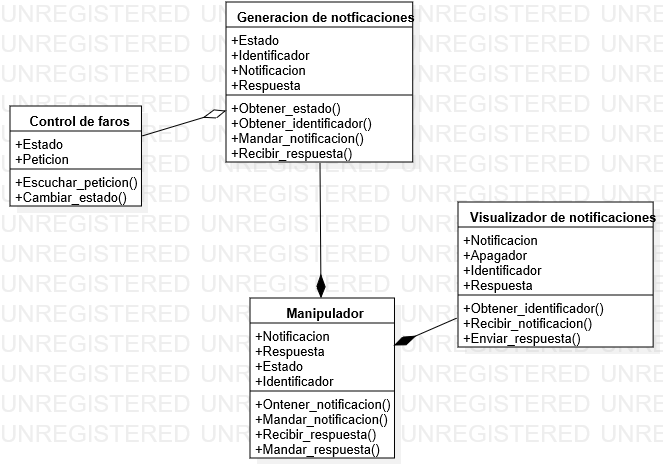


**Figura 4.2 Diagrama de secuencias**

En el siguiente diagrama de secuencia se muestra los tiempos de proceso de nuestro sistema. Como podemos observar el proceso comienza cuando el usuario apaga el automóvil, en ese momento si no se detecta ningún conductor dentro del auto, el verificador checara el estado de los faros, si los faros están encendidos, el verificador generara una notificación la cual mandara a la aplicación, la aplicación mostrara dicha notificación al usuario, el usuario será capaz de apagar los faros desde dicha aplicación la cual generara una respuesta para el verificador, la respuesta pasara primero por la aplicación y de la aplicación pasara al verificador, el verificador checara la respuesta y con dicha respuesta será capaz de apagar los faros.

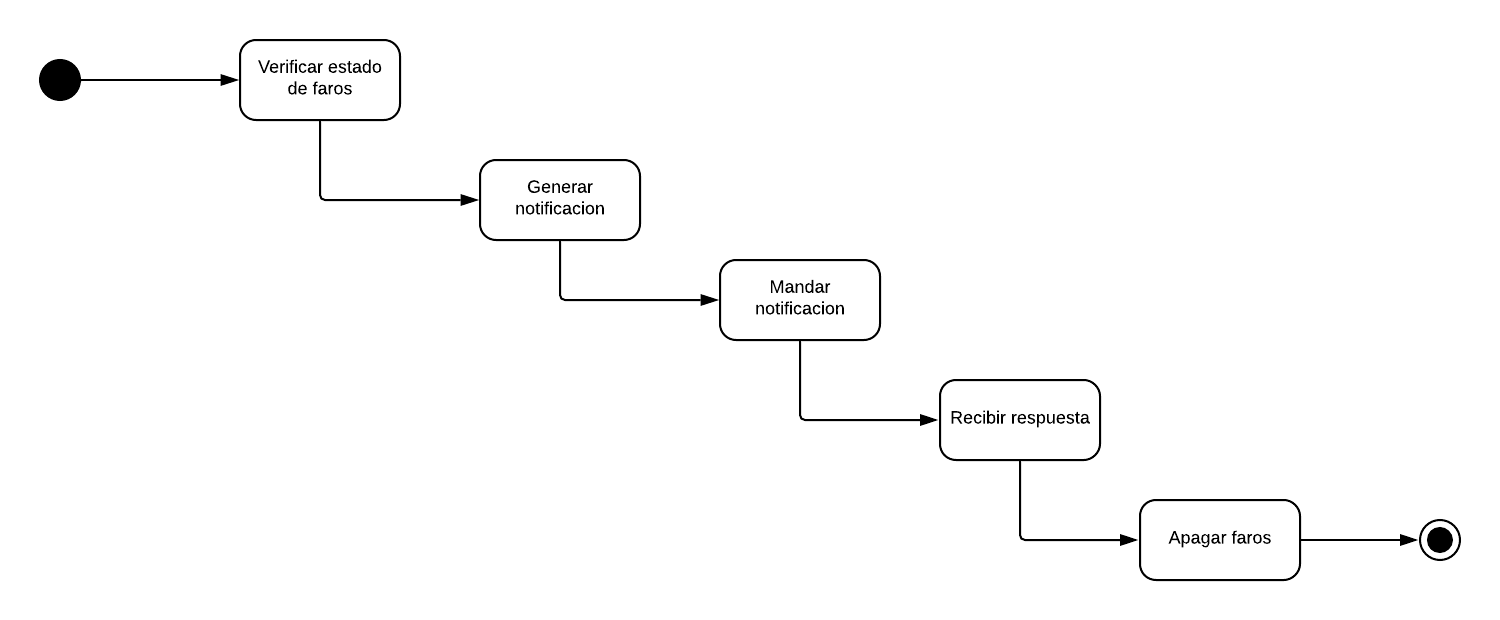
## 4.3 Diagrama de clases

A continuación, se presenta el diagrama de clases de la aplicación.

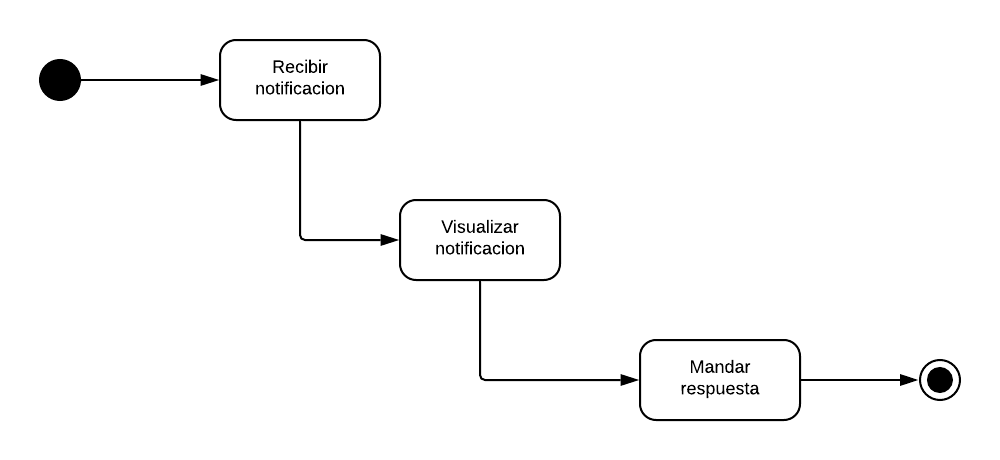


**Figura 4.3 Diagrama de clases**

## 4.4 Diagrama de actividades

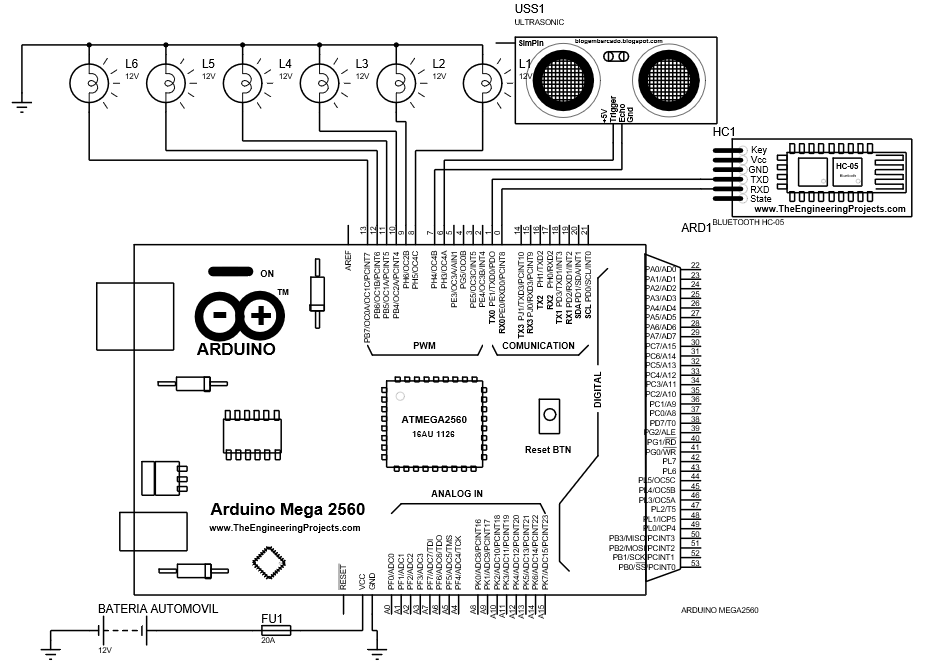


**Figura 4.4 Diagrama de actividades controlador**



**Figura 4.5 Diagrama de actividades del usuario**

## 4.5 Diagrama de modulo externo



**Figura 4.6 Diagrama físico del modulo externo**

# Capítulo 5: Codificación

En este apartado se mostrará la codificación que se implemento para el desarrollo de este proyecto: En este caso se hizo uso de dos IDE’s. Se hizo uso de Android Studio para el desarrollo de la aplicación, y Arduino IDE para la programación del módulo externo que se instalara en el automóvil.

## 5.1 Dispositivos Bluetooth disponibles

Con el siguiente código se muestra en la aplicación una lista con todos los dispositivos bluetooth con los que el celular se a vinculado anteriormente.

1. package com.example.app\_agador;
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
6. import android.bluetooth.BluetoothAdapter;
7. import android.bluetooth.BluetoothDevice;
8. import android.content.Intent;
9. import android.util.Log;
10. import android.view.View;
11. import android.widget.AdapterView;
12. import android.widget.ArrayAdapter;
13. import android.widget.ListView;
14. import android.widget.TextView;
15. import android.widget.Toast;
17. import java.util.Set;
19. public class DispositivosBT extends AppCompatActivity {
21. // Depuración de LOGCAT
22. private static final String TAG = "DispositivosBT"; //<-<- PARTE A MODIFICAR >->->
23. // Declaracion de ListView
24. ListView IdLista;
25. // String que se enviara a la actividad principal, mainactivity
26. public static String EXTRA\_DEVICE\_ADDRESS = "device\_address";
28. // Declaracion de campos
29. private BluetoothAdapter mBtAdapter;
30. private ArrayAdapter mPairedDevicesArrayAdapter;
32. @Override
33. protected void onCreate(Bundle savedInstanceState) {
34. super.onCreate(savedInstanceState);
35. setContentView(R.layout.activity\_dispositivos\_bt);
36. }
38. @Override
39. public void onResume()
40. {
41. super.onResume();
42. //---------------------------------
43. VerificarEstadoBT();
45. // Inicializa la array que contendra la lista de los dispositivos bluetooth vinculados
46. mPairedDevicesArrayAdapter = new ArrayAdapter(this, R.layout.nombre\_dispositivos);//<-<- PARTE A MODIFICAR >->->
47. // Presenta los disposisitivos vinculados en el ListView
48. IdLista = (ListView) findViewById(R.id.IdLista);
49. IdLista.setAdapter(mPairedDevicesArrayAdapter);
50. IdLista.setOnItemClickListener(mDeviceClickListener);
51. // Obtiene el adaptador local Bluetooth adapter
52. mBtAdapter = BluetoothAdapter.getDefaultAdapter();

55. // Obtiene un conjunto de dispositivos actualmente emparejados y agregua a 'pairedDevices'
56. Set <BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
58. // Adiciona un dispositivos previo emparejado al array
59. if (pairedDevices.size() > 0)
60. {
61. for (BluetoothDevice device : pairedDevices) { //EN CASO DE ERROR LEER LA ANTERIOR EXPLICACION
62. mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
63. }
64. }
65. }
67. // Configura un (on-click) para la lista
68. private AdapterView.OnItemClickListener mDeviceClickListener = new AdapterView.OnItemClickListener() {
69. public void onItemClick(AdapterView av, View v, int arg2, long arg3) {
71. // Obtener la dirección MAC del dispositivo, que son los últimos 17 caracteres en la vista
72. String info = ((TextView) v).getText().toString();
73. String address = info.substring(info.length() - 17);
75. // Realiza un intent para iniciar la siguiente actividad
76. // mientras toma un EXTRA\_DEVICE\_ADDRESS que es la dirección MAC.
77. Intent i = new Intent(DispositivosBT.this, MainActivity.class);//<-<- PARTE A MODIFICAR >->->
78. i.putExtra(EXTRA\_DEVICE\_ADDRESS, address);
79. startActivity(i);
80. }
81. };
83. private void VerificarEstadoBT() {
84. // Comprueba que el dispositivo tiene Bluetooth y que está encendido.
85. mBtAdapter= BluetoothAdapter.getDefaultAdapter();
86. if(mBtAdapter==null) {
87. Toast.makeText(getBaseContext(), "El dispositivo no soporta Bluetooth", Toast.LENGTH\_SHORT).show();
88. } else {
89. if (mBtAdapter.isEnabled()) {
90. Log.d(TAG, "...Bluetooth Activado...");
91. } else {
92. //Solicita al usuario que active Bluetooth
93. Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);
94. startActivityForResult(enableBtIntent, 1);
96. }
97. }
98. }
99. }

## 5.2 Interfaz principal

Con el siguiente código lo que hacemos es iniciar la interfaz principal, con la que interactuara el usuario.

1. **package** com.example.app\_agador;
3. **import** android.bluetooth.BluetoothAdapter;
4. **import** android.bluetooth.BluetoothDevice;
5. **import** android.bluetooth.BluetoothSocket;
6. **import** android.content.Intent;
7. **import** android.os.Handler;
8. **import** android.support.v7.app.AppCompatActivity;
9. **import** android.os.Bundle;
10. **import** android.view.View;
11. **import** android.widget.Button;
12. **import** android.widget.TextView;
13. **import** android.widget.Toast;
15. **import** java.io.IOException;
16. **import** java.io.InputStream;
17. **import** java.io.OutputStream;
18. **import** java.util.UUID;
20. **public** **class** MainActivity **extends** AppCompatActivity {
22. //1)
23. Button IdApagar,IdDesconectar, IdVinculo;
24. TextView IdBufferIn;
25. //-------------------------------------------
26. Handler bluetoothIn;
27. **final** **int** handlerState = 0;
28. **private** BluetoothAdapter btAdapter = **null**;
29. **private** BluetoothSocket btSocket = **null**;
30. **private** StringBuilder DataStringIN = **new** StringBuilder();
31. **private** ConnectedThread MyConexionBT;
32. // Identificador unico de servicio - SPP UUID
33. **private** **static** **final** UUID BTMODULEUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
34. // String para la direccion MAC
35. **private** **static** String address = **null**;
36. //-------------------------------------------
38. @Override
39. **protected** **void** onCreate(Bundle savedInstanceState) {
40. **super**.onCreate(savedInstanceState);
41. setContentView(R.layout.activity\_main);
42. //2)
43. //Enlaza los controles con sus respectivas vistas
44. IdVinculo = (Button) findViewById(R.id.IdVinculo);
45. IdApagar = (Button) findViewById(R.id.IdApagar);
46. IdDesconectar = (Button) findViewById(R.id.IdDesconectar);
47. IdBufferIn = (TextView) findViewById(R.id.IdBufferIn);
49. bluetoothIn = **new** Handler() {
50. **public** **void** handleMessage(android.os.Message msg) {
51. **if** (msg.what == handlerState) {
52. String readMessage = (String) msg.obj;
53. DataStringIN.append(readMessage);
55. **int** endOfLineIndex = DataStringIN.indexOf("#");
57. **if** (endOfLineIndex > 0) {
58. String dataInPrint = DataStringIN.substring(0, endOfLineIndex);
59. IdBufferIn.setText("Notificacion: " + dataInPrint);
60. DataStringIN.delete(0, DataStringIN.length());
61. }
62. }
63. }
64. };
66. btAdapter = BluetoothAdapter.getDefaultAdapter(); // get Bluetooth adapter
67. VerificarEstadoBT();
69. // Configuracion onClick listeners para los botones
70. // para indicar que se realizara cuando se detecte
71. // el evento de Click
73. IdVinculo.setOnClickListener(**new** View.OnClickListener() {
74. **public** **void** onClick(View v) {
75. MyConexionBT.write("0");
76. }
77. });
79. IdApagar.setOnClickListener(**new** View.OnClickListener() {
80. **public** **void** onClick(View v) {
81. MyConexionBT.write("1");
82. }
83. });
85. IdDesconectar.setOnClickListener(**new** View.OnClickListener() {
86. **public** **void** onClick(View v) {
87. **if** (btSocket!=**null**)
88. {
89. **try** {btSocket.close();}
90. **catch** (IOException e)
91. { Toast.makeText(getBaseContext(), "Error", Toast.LENGTH\_SHORT).show();;}
92. }
93. finish();
94. }
95. });
96. }
98. **private** BluetoothSocket createBluetoothSocket(BluetoothDevice device) **throws** IOException
99. {
100. //crea un conexion de salida segura para el dispositivo
101. //usando el servicio UUID
102. **return** device.createRfcommSocketToServiceRecord(BTMODULEUUID);
103. }
105. @Override
106. **public** **void** onResume()
107. {
108. **super**.onResume();
109. //Consigue la direccion MAC desde DeviceListActivity via intent
110. Intent intent = getIntent();
111. //Consigue la direccion MAC desde DeviceListActivity via EXTRA
112. address = intent.getStringExtra(DispositivosBT.EXTRA\_DEVICE\_ADDRESS);
113. //Setea la direccion MAC
114. BluetoothDevice device = btAdapter.getRemoteDevice(address);
116. **try**
117. {
118. btSocket = createBluetoothSocket(device);
119. } **catch** (IOException e) {
120. Toast.makeText(getBaseContext(), "La creacción del Socket fallo", Toast.LENGTH\_LONG).show();
121. }
122. // Establece la conexión con el socket Bluetooth.
123. **try**
124. {
125. btSocket.connect();
126. } **catch** (IOException e) {
127. **try** {
128. btSocket.close();
129. } **catch** (IOException e2) {}
130. }
131. MyConexionBT = **new** ConnectedThread(btSocket);
132. MyConexionBT.start();
133. }
135. @Override
136. **public** **void** onPause()
137. {
138. **super**.onPause();
139. **try**
140. { // Cuando se sale de la aplicación esta parte permite
141. // que no se deje abierto el socket
142. btSocket.close();
143. } **catch** (IOException e2) {}
144. }
146. //Comprueba que el dispositivo Bluetooth Bluetooth está disponible y solicita que se active si está desactivado
147. **private** **void** VerificarEstadoBT() {
149. **if**(btAdapter==**null**) {
150. Toast.makeText(getBaseContext(), "El dispositivo no soporta bluetooth", Toast.LENGTH\_LONG).show();
151. } **else** {
152. **if** (btAdapter.isEnabled()) {
153. } **else** {
154. Intent enableBtIntent = **new** Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);
155. startActivityForResult(enableBtIntent, 1);
156. }
157. }
158. }
160. //Crea la clase que permite crear el evento de conexion
161. **private** **class** ConnectedThread **extends** Thread
162. {
163. **private** **final** InputStream mmInStream;
164. **private** **final** OutputStream mmOutStream;
166. **public** ConnectedThread(BluetoothSocket socket)
167. {
168. InputStream tmpIn = **null**;
169. OutputStream tmpOut = **null**;
170. **try**
171. {
172. tmpIn = socket.getInputStream();
173. tmpOut = socket.getOutputStream();
174. } **catch** (IOException e) { }
175. mmInStream = tmpIn;
176. mmOutStream = tmpOut;
177. }
179. **public** **void** run()
180. {
181. **byte**[] buffer = **new** **byte**[256];
182. **int** bytes;
184. // Se mantiene en modo escucha para determinar el ingreso de datos
185. **while** (**true**) {
186. **try** {
187. bytes = mmInStream.read(buffer);
188. String readMessage = **new** String(buffer, 0, bytes);
189. // Envia los datos obtenidos hacia el evento via handler
190. bluetoothIn.obtainMessage(handlerState, bytes, -1, readMessage).sendToTarget();
191. } **catch** (IOException e) {
192. **break**;
193. }
194. }
195. }
196. //Envio de trama
197. **public** **void** write(String input)
198. {
199. **try** {
200. mmOutStream.write(input.getBytes());
201. }
202. **catch** (IOException e)
203. {
204. //si no es posible enviar datos se cierra la conexión
205. Toast.makeText(getBaseContext(), "La Conexión fallo", Toast.LENGTH\_LONG).show();
206. finish();
207. }
208. }
209. }
210. }

## 5.3 Programación del Arduino

El siguiente código se utilizo para programar el Arduino del modulo externo que se instalaría en el automóvil.

1. **int** leds[]={8, 9, 10, 11, 12, 13}; //Arreglo de leds
3. **int** pot; //Valor del Potenciometro
4. **int** n=0;
6. **const** **int** Trigger = 6; //Trigger del Sensor
7. **const** **int** Echo = 7; //Echo del sensor
9. **char** VarChar = '0';
11. **void** setup(){
12. Serial.begin(9600);
13. pinMode(Trigger, OUTPUT); //pin como salida
14. pinMode(Echo, INPUT);  //pin como entrada
15. digitalWrite(Trigger, LOW);//Inicializamos el pin con 0
17. **for**(n = 0; n < 6; n++) {
18. pinMode(leds[n],OUTPUT);
19. digitalWrite(leds[n], LOW);
20. }
21. }
23. **void** loop(){
25. //Control de luces
26. pot = analogRead(A0);// variable pot entrada analógica en A0
27. //Serial.print("Valor del potenciometro");
28. //Serial.println(pot);
29. **if** (VarChar == '1'){
30. **for**(n = 0; n < 6; n++) {
31. digitalWrite(leds[n], LOW);
32. }
33. }
34. **else** **if**(pot > 0 && pot <= 165){
35. **for**(n = 0; n < 6; n++) {
36. digitalWrite(leds[n], LOW);
37. }
38. digitalWrite(leds[0], HIGH);
39. }
40. **else** **if**(pot >= 165 && pot <= 330){
41. **for**(n = 0; n < 6; n++) {
42. digitalWrite(leds[n], LOW);
43. }
44. digitalWrite(leds[1], HIGH);
45. }
46. **else** **if**(pot >= 330 && pot <= 495){
47. **for**(n = 0; n < 6; n++) {
48. digitalWrite(leds[n], LOW);
49. }
50. digitalWrite(leds[2], HIGH);
51. }
52. **else** **if**(pot >= 495 && pot <= 660){
53. **for**(n = 0; n < 6; n++) {
54. digitalWrite(leds[n], LOW);
55. }
56. digitalWrite(leds[3], HIGH);
57. }
58. **else** **if**(pot >= 660 && pot <= 825){
59. **for**(n = 0; n < 6; n++) {
60. digitalWrite(leds[n], LOW);
61. }
62. digitalWrite(leds[4], HIGH);
63. }
64. **else** **if**(pot >= 825 && pot <= 1000){
65. **for**(n = 0; n < 6; n++) {
66. digitalWrite(leds[n], LOW);
67. }
68. digitalWrite(leds[5], HIGH);
69. }
70. **else** **if**(pot >= 1000){
71. **for**(n = 0; n < 6; n++) {
72. digitalWrite(leds[n], LOW);
73. }
74. }
76. //Distancia
77. **long** t; //timepo que demora en llegar el eco
78. **long** d; //distancia en centimetros
80. digitalWrite(Trigger, HIGH);
81. delayMicroseconds(10);          //Enviamos un pulso de 10us
82. digitalWrite(Trigger, LOW);
84. t = pulseIn(Echo, HIGH); //obtenemos el ancho del pulso
85. d = t/59;             //escalamos el tiempo a una distancia en cm
87. delay(100);          //Hacemos una pausa de 100ms
89. //Comunicacion Bluetooth
90. **if**(Serial.available() > 0){
91. **int** bandera = 0;
92. Serial.println("Modulo Conectado");
93. Serial.println("#");
94. delay(1000);
96. //Verifica si alguna luz esta prendida
97. **for**(n = 0; n < 6; n++){
98. **int** estado = digitalRead(leds[n]);
99. **if**(estado == HIGH && d >= 50){
100. bandera = 1;
101. }
102. }
104. **if**(bandera == 1){
105. Serial.println("Luz encendida");
106. //Serial.println("#");
107. Serial.println("Usuario no detectado");
108. Serial.print("Distancia: ");
109. Serial.print(d);      //Enviamos serialmente el valor de la distancia
110. Serial.print("cm");
111. Serial.println("#");
112. delay(1000);
113. VarChar = Serial.read();
114. **if**(VarChar == '1'){
115. Serial.print("Luces apagadas");
116. Serial.println("#");
117. }
118. }
120. **else**{
121. Serial.println("Usuario cerca detectado");
122. Serial.print("¿Desea apagar sus luces?");
123. Serial.println("#");
124. delay(1000);
125. VarChar = Serial.read();
126. **if**(VarChar == '1'){
127. Serial.print("Luces apagadas");
128. Serial.println("#");
129. }
130. }
131. }
132. }

# Conclusión

Al desarrollar este proyecto uno se da cuenta de la importancia de todo el análisis que se debe de realizar, ya que es de gran ayuda y nos sirve para poder explicar el posible alcance que deseamos lograr. Este análisis realizado ayudo a identificar una problemática, la cual este proyecto piensa resolver. Al realizar el diseño de este proyecto se hizo uso de una metodología estructura, decidí ocupar esta metodología porque la aplicación requería de un módulo externo para poder funcionar, y la metodología estructura nos ayuda a trabajar por bloques y funciones, para ello se probó el funcionamiento bloque por bloque antes de integrar por completo el proyecto.

Otra cosa que me ayudo en el desarrollo del proyecto fue el uso de los diagramas de caso de usos ya que gracias a ellos pude definir las clases que iba a utilizar para el desarrollo de la aplicación. El diagrama de actividades me ayudo a saber como iba a ir el proceso del proyecto completo y a identificar las posibles fallas o incongruencias que podría tener el proyecto.

Otra cosa que se concluye con este trabajo es el saber como calcular el posible costo que te llevara desarrollar este proyecto, al igual que la cantidad mínima de personas que se necesita para que el proyecto se realice en un tiempo en específico, así como las líneas aproximadas de código que podrían salir al final del proyecto.

También se pudo observar el análisis de riesgos que podría ocasionar ciertas fallas en nuestros sistemas, y si son factibles de solucionar o en que tiempo podría ocurrir, con ello se pudo analizar los tiempos de actualizaciones, mantenimiento y otro tipo de mejoras que se le podrían hacer al proyecto. Y así dependiendo de nuestro análisis de riesgos buscar la mejor manera de desarrollo del proyecto para evitarlas.

# Bibliografía

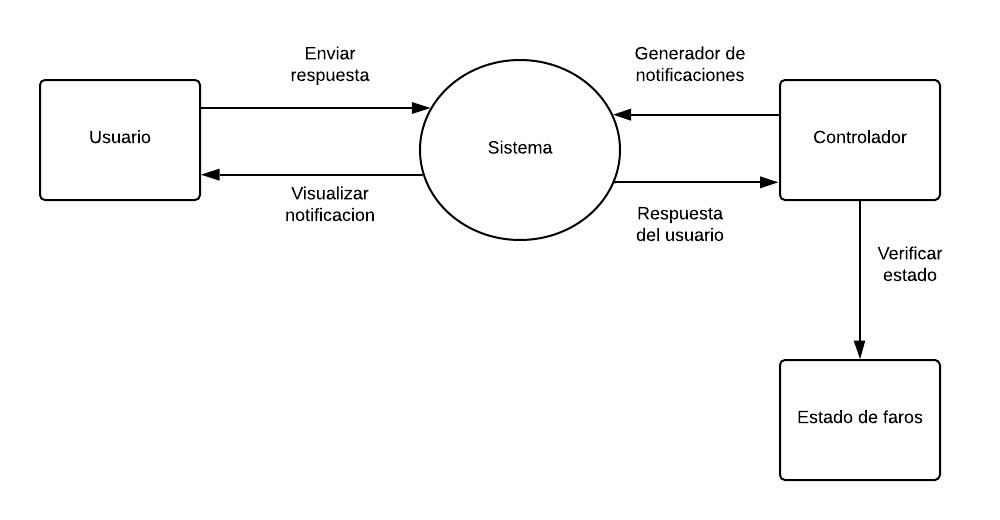
* La batería del coche: cómo funciona y cuánto dura. (2019). Retrieved from https://www.ro-des.com/mecanica/bateria-del-coche-como-funciona-y-cuanto-dura/
* En qué consiste el sistema de alumbrado del coche. (2019). Retrieved from https://www.ro-des.com/mecanica/sistema-alumbrado-del-coche-que-es/Anexos
* La batería del coche: cómo funciona y cuánto dura. (2019). Retrieved from https://www.ro-des.com/mecanica/bateria-del-coche-como-funciona-y-cuanto-dura/
* (2019). Retrieved from https://es.slideshare.net/KeinerValerio/metodologia-de-prototipos
* (2019). Retrieved from http://profesores.fi-b.unam.mx/heriolg/Apa\_1apa.pdf
* HC-05, M. (2019). Módulo Bluetooth HC-05 BricoGeek HC-05-BT | BricoGeek.com. Retrieved from https://tienda.bricogeek.com/modulos-bluetooth/800-modulo-bluetooth-hc-05.html
* Diosdado, R. (2019). Zona Maker - Ultrasonido HC-SR04. Retrieved from https://www.zonamaker.com/arduino/modulos-sensores-y-shields/ultrasonido-hc-sr04
* Arduino Mega: Características, Capacidades y donde conseguirlo en Panamá | Panama Hitek. (2019). Retrieved from <http://panamahitek.com/arduino-mega-caracteristicas-capacidades-y-donde-conseguirlo-en-panama/>

# Anexos

## Análisis Estructurado

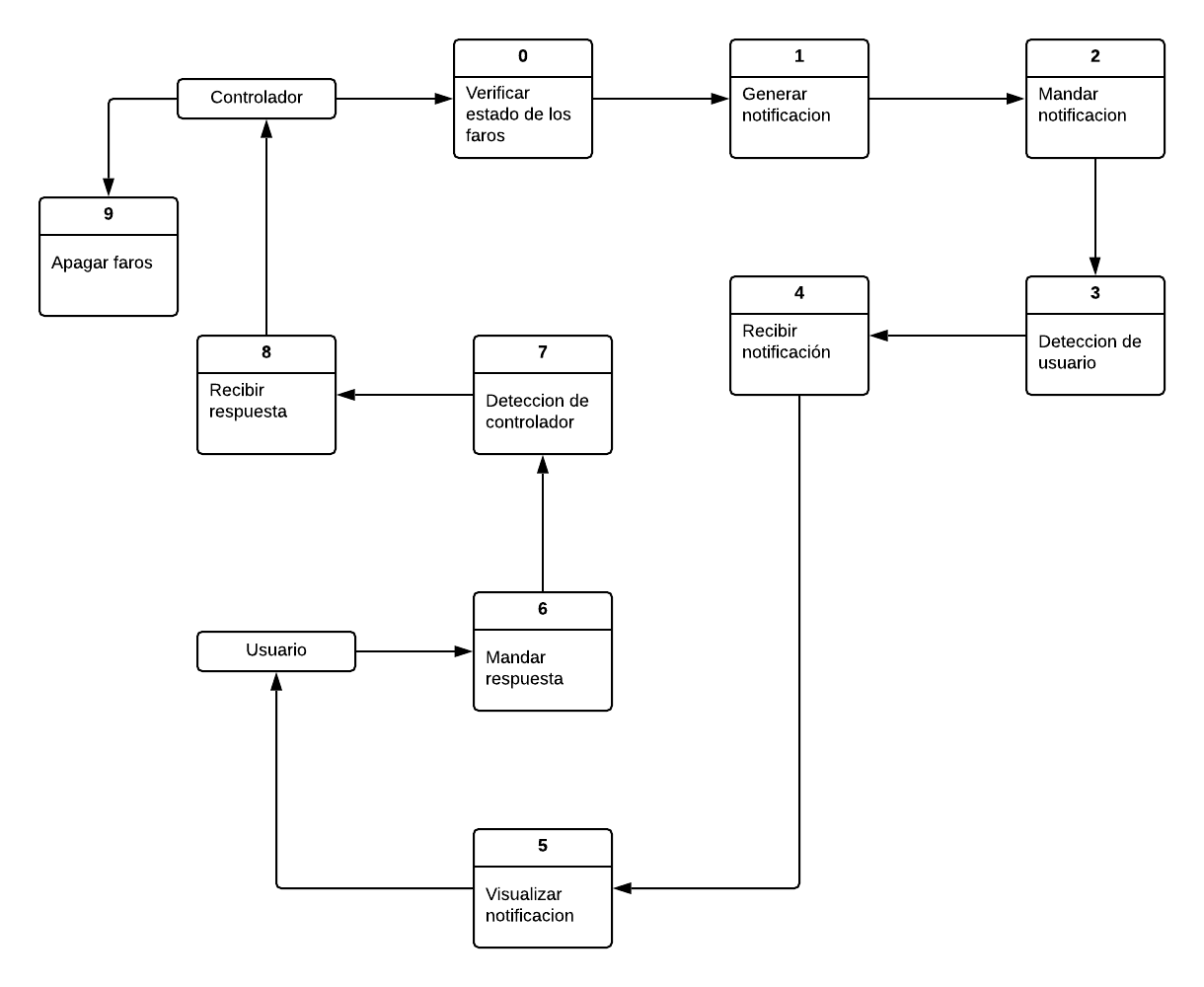
El análisis estructurado se realizó usando la metodología de Gane & Searson, el cual será presentado a continuación.

### Diagrama de contexto



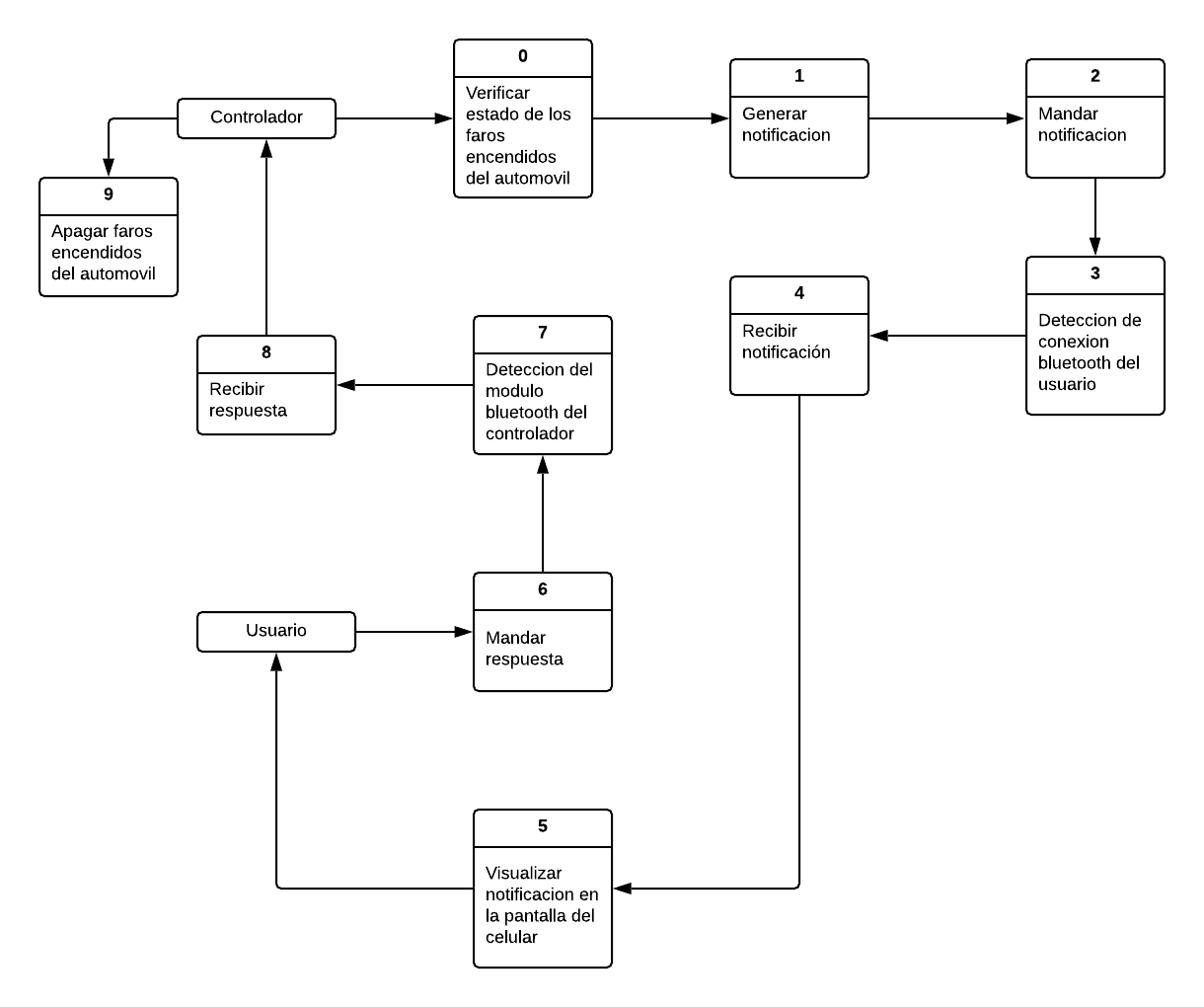
**Figura A.1 Diagrama de contexto**

### Diagrama de flujo de datos lógico



**Figura A.2 Diagrama de flujo de datos lógico**

### Diagrama de flujo de datos físicos



**Figura A.3 Diagrama de flujo de datos físico**

## Análisis Orientado a objetos

El análisis orientado a objetos se realizó usando la metodología de Coad & Yourdon, el cual será presentado a continuación.

### Narrativa

El controlador verifica el estado de los faros del automóvil, en caso de que identifique algún faro encendido y no identifique algún conductor cerca, generara una notificación la cual se enviara a la aplicación instalada en el celular del conductor. El usuario visualizara dicha notificación y tendrá la opción de apagar los faros desde la aplicación, generando una respuesta que se le enviara al controlador, el controlador interpretara la respuesta, y dependiendo de ella cambiara el estado de los faros de encendido a apagado.

### Identificación de objetos

Los objetos identificados en un principio para presentar en el proyecto son los siguientes:

* Generador de notificaciones: Este objeto se encargará de generar las notificaciones en caso de que se detecte algún faro encendido y no se detecte un conductor cerca.
* Control de faros: Este objeto se encargara de controlar el estado de los faros, dando la capacidad de cambiar el estado del faro de encendido a apagado
* Manipulador: Este objeto se encargará de lograr la comunicación entre el controlador y la aplicación, por este objeto pasaran las notificaciones y las respuestas.
* Visualizador de notificaciones: Este objeto será con el que interactuará el usuario directamente, permitiéndole ver la notificación y cambiar el estado de los faros encendidos.

### Análisis de métodos y atributos

Generador de notificaciones:

* Métodos:
  + Estado: Almacena el estado de los faros
  + Identificador: Almacena el identificador bluetooth que permitirá la comunicación entre el controlador y la aplicación
  + Notificación: Almacena la notificación generada
  + Respuesta: Almacena la respuesta recibida
* Operaciones:
  + Obtener\_estado(): Obtiene el estado de los faros del automóvil
  + Obtener\_identificador(): Obtiene el identificador bluetooth que permite la comunicación entre controlador y aplicación
  + Generar\_notificacion(): Genera la notificación que se le enviara al usuario
  + Mandar\_notificacion(): Manda la notificación generada
  + Recibir\_respuesta(): Recibe la respuesta por parte del usuario

Visualizador de notificaciones:

* Métodos:
  + Notificación: Almacena la notificación recibida.
  + Apagador: Permite mandar un cero para apagar el faro
  + Identificador: Almacena el identificador bluetooth del controlador
  + Respuesta: Almacena la respuesta a mandar
* Operaciones:
  + Enviar\_respuesta(): Envía la respuesta al controlador dependiendo del estado del apagador
  + Recibir\_notificacion(): Recibe la notificacion enviada por el controlador
  + Obtener\_identificador(): Obtiene el identificador bluetooth del controlador

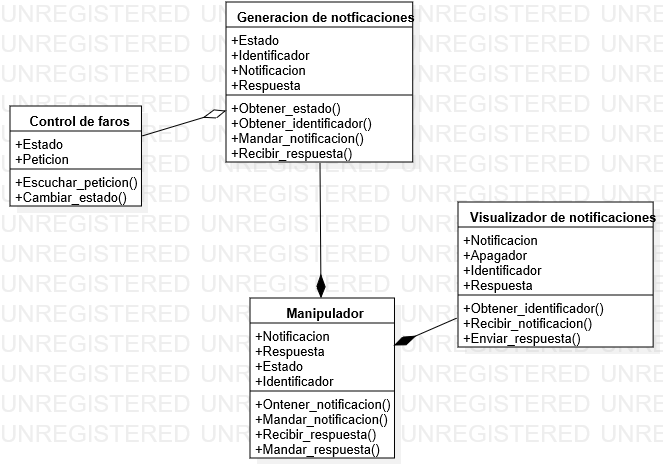
Manipulador:

* Métodos:
  + Notificación: Almacena la notificación generada por el controlador
  + Respuesta: Almacena la respuesta del usuario
  + Estado: Almacena el estado del faro
  + Identificador: Almacena los identificadores bluetooth del usuario y del controlador.
* Operaciones:
  + Obtener\_notoficacion(): Obtiene la notificación generada por el controlador
  + Mandar\_notificación(): Manda la notificación al usuario
  + Recibir\_respuesta(): Recibe la respuesta del usuario
  + Mandar\_notificacion(): Envía la respuesta al controlador.

Control de faros:

* Métodos:
  + Estado: Almacena el estado actual del faro
  + Peticion: Alamcena la petición de cambio de estado
* Operaciones:
  + Escuchar\_peticion(): Escucha la petición de cambio de parte del controlador
  + Cambiar\_estado(): Cambia el estado del faro.

### Diagrama de clases



**Figura A.4 Diagrama de clases**