



Universidad de Valladolid



**ESCUELA DE INGENIERÍAS
INDUSTRIALES**

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

**Desarrollo e implantación de un sistema de
integración continua para dispositivos
embarcados**

Autor:

Villarroel Arranz, Alejandro

Tutor:

González Sánchez, José Luis

**Ingeniería de sistemas y
automática**

Valladolid, Julio 2019.



Universidad de Valladolid



ESCUELA DE INGENIERÍAS
INDUSTRIALES



Resumen (Abstract)

El presente TFG consiste en la creación de un proyecto para un sistema embarcado provisto de las herramientas necesarias para garantizar su fácil mantenibilidad, incluyendo entre estas tests unitarios y tests de integración.

El proyecto estará dividido en un conjunto de módulos con funcionalidades específicas y bien diferenciadas y que en conjunto formarán el Firmware que permitirá el funcionamiento deseado del dispositivo embarcado.

This TFG consists in the creation of a Project for an on-board system equipped with the necessary tools to guarantee its easy maintainability, including among these unit tests and integration tests.

The Project will be divided in a group of modules with specific and well differentiated functionalities that together will form the Firmware that will allow the desired behaviour of the on-board device.

Palabras clave (Keywords)

Programa	<i>Program</i>
Sistema embarcado	<i>On-board system</i>
Mantenibilidad	<i>Maintainability</i>
Tests unitarios	<i>Unit testing</i>
Tests de integración	<i>Integration testing</i>



Índice de contenido

1.-Introducción y objetivos.....	9
2.-Desarrollo del TFG	13
2.1.-Análisis inicial	13
2.1.1.-Sistema embarcado	13
2.1.2.-Firmware	15
2.2.-Preparación del entorno de trabajo	16
2.2.1.-Entorno de desarrollo (IDE)	16
2.2.2.-Pautas de código limpio.....	18
2.3.-Análisis del proyecto concreto	20
2.3.1.-Dispositivo embarcado	20
2.3.2.-Plataforma web (comunicaciones con central)	23
2.3.3.-Funcionalidades	25
2.3.4.-Modulación del código del proyecto	44
2.4.-Infraestructura para la mantenibilidad y fácil desarrollo del proyecto	48
2.4.1.-Tests unitarios	48
2.4.2.-Tests de integración.....	58
2.4.3.-Automatización del proceso	66
3.-Conclusiones	77
3.1.-Líneas futuras.....	78
4.-Bibliografía	79
5.-Anexos	83
5.1.-ANEXO I: OBD II PIDs	83
5.2.-ANEXO II: El problema del Rollover (GPS)	87
5.3.-ANEXO III: Receptor GPS	89

Tabla de Figuras

Figura 1:Ciclo de integración continua	10
Figura 2:Ventajas de la integración continua	11
Figura 3: Estructura de sistema embebido	14
Figura 4:Ejemplos de sistemas embarcados	14
Figura 5:Esquema explicativo Firmware	15
Figura 6:Iconos de los IDEs Eclipse y Codeblocks	16
Figura 7:Ventana común del IDE Eclipse	17
Figura 8:Procesador tipo AR grado automoción	18
Figura 9: Medida de la calidad del código	19
Figura 10:Dispositivo embarcado del proyecto objeto	20
Figura 11:Tabla estados LED	21
Figura 12:Esquema lateral del dispositivo embarcado	22
Figura 13:Visualización de la web de MOVILOC	24
Figura 14:Resumen funcionalidades dispositivo embarcado	25
Figura 15:Antena GPS+GSM	26
Figura 16:Cobertura 2G en España operador Movistar	27
Figura 17:Cobertura 4G en España operador Movistar	28
Figura 18:Cobertura 3G en España operador Movistar	28
Figura 19:Esquema general OBD	30
Figura 20:Conector DLC y pines	30
Figura 21:Informe detallado de visitas	32
Figura 22:Informe detallado de velocidades	32
Figura 23:Velocímetro y cuentarrevoluciones de un coche	33
Figura 24:Sensor de temperatura TQS3	34
Figura 25:Sensor de apertura de puertas	34
Figura 26:iButton (izq) y lector (dcha)	35
Figura 27:Botón de pánico integrado en un coche	36
Figura 28:Inmovilización remota de un vehículo	36
Figura 29:Informe de alarmas en la plataforma web	37
Figura 30:Zona geofencing y recorrido de un vehículo que entra y sale de la zona	38
Figura 31:Relé usado para inmovilización (x2)	39
Figura 32:Recorrido de un vehículo visualizado en plataforma web	40
Figura 33:Gasolineras marcadas en mapa	40
Figura 34:Efecto Doppler para ondas	41
Figura 35:Icono mantenimiento recomendado	42
Figura 36:Lector RFID con tarjetas identificadoras	43
Figura 37:Algunos de los módulos del proyecto	44
Figura 38: Programación modular	46
Figura 39:Esquema de jerarquía de tests	48
Figura 40:Diferencia entre la técnica de la caja negra y la caja blanca	49
Figura 41:Viñeta cómica sobre el testeo incorrecto	50



Figura 42: Ilustración de la importancia de los tests unitarios	51
Figura 43: Viñeta satírica tests unitarios	58
Figura 44: Esquema de jerarquía de tests	58
Figura 45: Esquema explicativo de la necesidad de tests de integración	59
Figura 46: Diagrama representativo del método Bottom-Up	61
Figura 47: Diagrama representativo método Top-Down	61
Figura 48: Diagrama representativo método híbrido	62
Figura 49: Icono Robot Framework	63
Figura 50: Viñeta satírica tests de integración	65
Figura 51: Manual vs automated testing	66
Figura 52: Gráfica coste temporal del proceso de testeo	67
Figura 53: Man vs computer	68
Figura 54: Ejemplo de herramienta para tests de estrés	68
Figura 55: Esquema de automatización en desarrollo de software	69
Figura 56: Ejemplo de directorio con SVN	70
Figura 57: Ejemplo logs de un repositorio	71
Figura 58: Esquema de uso de un repositorio SVN	72
Figura 59: Aplicación de Branch and Merge a un proyecto	73
Figura 60: Logo de Jenkins	74
Figura 61: Diagrama de automatización con Jenkins y TortoiseSVN	75
Figura 62: Tabla de modos de OBD	83
Figura 63: Formato de las tramas enviadas por la OBD	84
Figura 64: Tabla con información sobre el PID 0C del modo 01	84
Figura 65: Descomposición en bits de la trama de la OBD	84
Figura 66: Imagen de herramienta para receptor GPS	89



Glosario de abreviaturas

ACK	Acknowledgement
CAN	Controller Area Network
DLC	Diagnostic Link Connector
DOP	Dilution Of Precision
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
FMS	Fleet Management System
FW	Firmware
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
HW	Hardware
IDE	Integrated Development Environment
IMSI	International Mobile Subscriber Identity
I/O	Input/Output
ITS	Intelligent Transportation Systems
LED	Light-Emitting Diode
NMEA	National Marine Electronics Association
OBD	On-Board Diagnostics
OS	Operative System
PC	Personal Computer
PIC	Periphereal Interface Controller
PID	Parameter IDentificator



RFID	Radio-Frecuency Identification
ROM	Read-Only Memory
SIM	Subscriber Identity Module
SVN	Subversion
SW	Software
TDD	Test Driven Development
TFG	Trabajo de Fin de Grado

1.-Introducción y objetivos

El objetivo principal de este trabajo es realizar una descripción detallada de las pautas que se deben seguir para crear un proyecto (programa informático) para un dispositivo embarcado de forma que:

- Sea fácil de trabajar, es decir, varias personas distintas puedan trabajar de forma conjunta en el mismo proyecto gracias a que siguen las mismas pautas de programación y reglas de uso. Además debe permitir la fácil incorporación de alguien nuevo al proyecto.
- Sea fácil de mantener, es decir, se puedan encontrar y aislar errores fácilmente, así como implementar parches para solucionarlos de forma sencilla.
- Sea fácil de ampliar, es decir, se puedan añadir nuevos módulos con nuevas funcionalidades sin que entren en conflicto con las ya presentes. Esto se consigue haciendo que unos módulos sean lo más independientes de otros que sea posible.
- Sea difícil introducir código erróneo gracias a los tests unitarios, que comprobarán el correcto funcionamiento del SW de forma celular, y a los tests de integración que comprobarán que el FW funciona correctamente sobre el HW del dispositivo embarcado realizando pruebas mecánicas o eléctricas.

El proyecto en sí seguirá las reglas de adecuada programación pertinentes y dispondrá de una infraestructura de servidores que permitirá la total automatización de creación del Firmware a partir del código. Cada vez que alguien actualice el código del proyecto se pasarán unos tests unitarios y de integración de forma automática, y si todos pasan correctamente se procederá a la compilación y subida del FW en el repositorio o base de datos correspondiente.

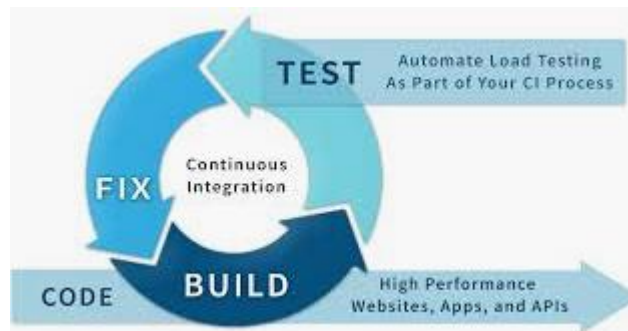


Figura 1: Ciclo de integración continua

El objetivo de crear toda esta estructura de reglas, pautas y tests para compilar un simple programa es en realidad una necesidad, ya que cuando se trabaja sobre un proyecto grande donde hay varias personas programando a la vez es necesario garantizar que el trabajo de unos no pueda verse afectado por culpa de otros, bien porque alguien ha subido un código con errores o bien porque interfiere con otra parte del programa o que directamente no compila.

Además, mediante el seguimiento de unas pautas básicas o personalizadas de programación en las que todos se pongan de acuerdo será mucho más fácil que los programadores entiendan el alcance y limitaciones del código de sus compañeros, ya que será en estructura similar al suyo y por lo tanto más sencillo de entender para todos los integrantes del proyecto.

También es tremendamente beneficiosa la automatización del testeo y compilado. Esto ahorra una gran cantidad de tiempo a los programadores, que solo tendrán que preocuparse de crear un código de calidad, ya que este pasará los tests de forma automática, se compilará y se enviará a la base de datos sin necesidad de que alguien esté pendiente de estas tareas que, por la naturaleza de tratarse de un sistema embarcado, pueden resultar bastante tediosas. Sobre todo a la hora de tener que cargar el FW en el dispositivo cada vez que se haga una modificación mínima y se quiera probar que funcione.



Figura 2:Ventajas de la integración continua

Este conjunto de cosas es lo que se llama sistema de integración continua y conlleva principalmente los 6 beneficios que aparecen en la *Figura 2*.

La calidad del código mejora, ya que para implementar tests unitarios y de integración es necesario hacer código limpio y bien modulado, además se seguirán unas pautas para crear código limpio.

Gracias a los tests la detección de errores es inmediata y nunca habrá que revisar cambios realizados hace más de un día o dos.

Utilizando un software de integración continua (en nuestro caso se usará Jenkins) se reducen las tareas repetitivas y procesos de compilación y testeo manuales.

Gracias a Jenkins también se puede compilar con ciertos parámetros al instante para crear versiones de prueba y con los repositorios no se perderá nada del trabajo ya realizado.



La división en módulos y el uso de tests de integración junto con servidores remotos permiten que la estructura del proyecto sea clara y de fácil visualización.

Los desarrolladores que trabajen en el proyecto lo harán con una mayor seguridad ya que cuentan con el respaldo de los tests unitarios y de integración, que se pasarán tras haber subido el código al repositorio y que avisarán al instante si se detecta algún error, abortando la compilación.



2.-Desarrollo del TFG

2.1.-Análisis inicial

Como se ha explicado ya, el TFG se centra en la creación de un FW para un dispositivo embarcado, así que se comenzará explicando estos términos y las limitaciones que suponen.

2.1.1.-Sistema embarcado

Un sistema embarcado suele estar basado en un microcontrolador y consiste básicamente en un HW al que se le dota de un FW para cumplir una tarea específica y que suele trabajar en tiempo real.

Algunas de sus características son:

- Suelen tener una función específica y bien definida.
- Están diseñados de la forma más eficiente posible, minimizando el coste, el tamaño y la potencia consumida y buscando la máxima velocidad y rendimiento.
- Deben ser capaces de reaccionar a los estímulos del exterior a tiempo ya que un retraso podría causar graves daños dependiendo de la función que realice ese sistema embebido.
- Se suelen basar en un microcontrolador/microprocesador.
- Debe tener una memoria, en la que suele cargarse el software de aplicación (FW) y que suele ser flash para realizar actualizaciones de FW.
- Suelen permitir la conexión de dispositivos de entrada/salida, cuyas lecturas se grabarán en memoria y ante los cuales el FW estará preparado para reaccionar.

En general, la estructura simplificada de un sistema embebido sería la que se muestra a continuación, en la *Figura 3*.

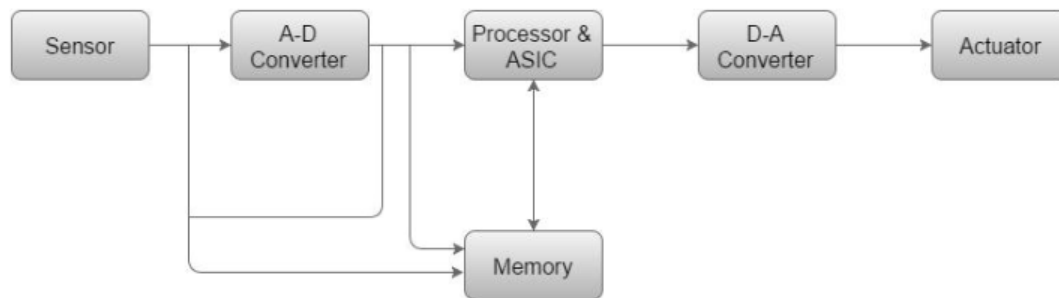


Figura 3: Estructura de sistema embebido

Consta de un conjunto de sensores como pueden ser sensores de temperatura, antenas GPS y GPRS, sistemas de identificación como RFIDs, zumbadores etc. que graban en memoria sus valores. El FW se encarga de leer esos sensores y reaccionar adecuadamente ante sus valores enviando señales de salida para, por ejemplo, activar un zumbador o un inmovilizador de coche.



Figura 4: Ejemplos de sistemas embarcados

En la *Figura 4* aparecen algunos ejemplos de sistemas embarcados actuales. Originalmente solo se aplicaban a sistemas poco complejos como calculadoras o mandos de la televisión por ejemplo, pero actualmente los sistemas embarcados están por todas partes, desde los electrodomésticos que podrían parecer poco complejos a simple vista, hasta aparatos tan cotidianos como el teléfono móvil o la televisión.

Normalmente una de las mayores limitaciones que conllevan los sistemas embarcados es que una vez que el FW está cargado en el dispositivo no se supone que este pueda cambiar ni actualizarse.

Sin embargo, la mayoría de los dispositivos actuales pueden ya no solo actualizar su FW de manera manual sino que además permiten su actualización automática si están conectados a internet.

2.1.2.-Firmware

El Firmware es, básicamente, un software especializado que se carga en un determinado sistema y que está estrechamente relacionado con el HW del mismo.

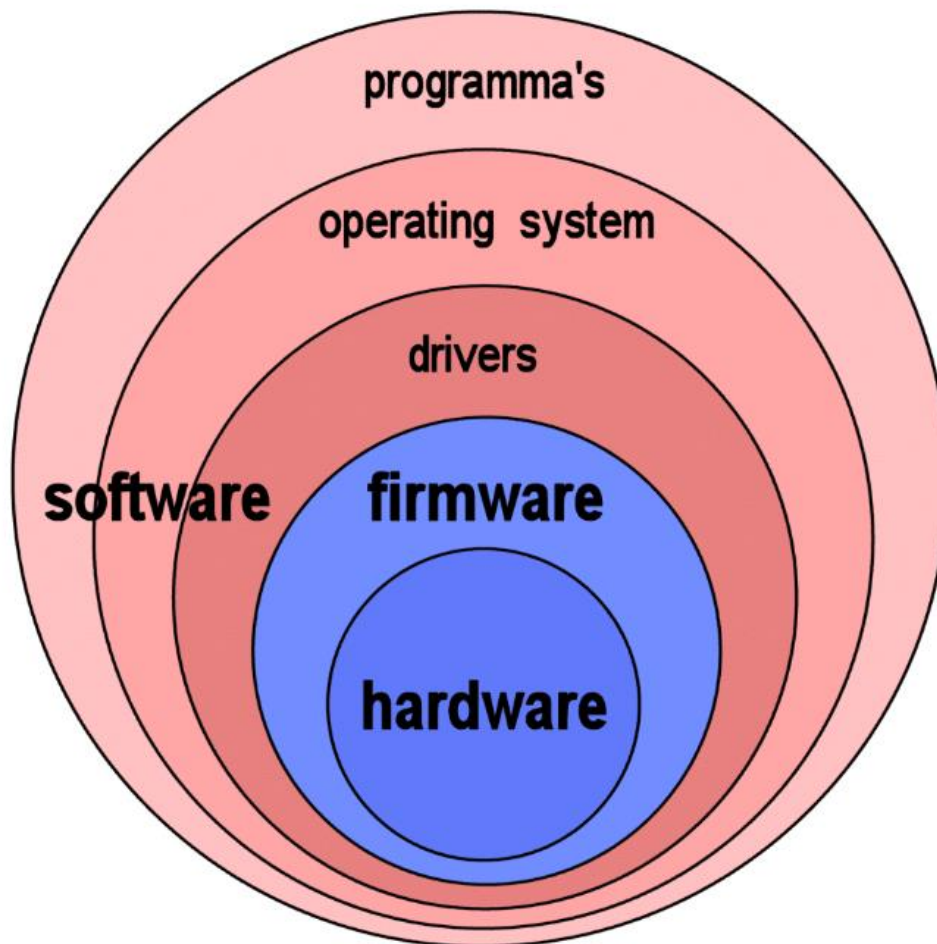


Figura 5:Esquema explicativo Firmware

En la *Figura 5* se aprecia que específicamente el FW está muy relacionado con el HW y es lo que rige el funcionamiento del dispositivo al más bajo nivel,

normalmente se puede incluir el FW como un tipo de SW, aunque este último esté más relacionado con aplicaciones específicas y programas adicionales que pudiese tener el dispositivo.

En el caso que nos atañe el FW consiste en un software diseñado para controlar el HW del que dispone nuestro dispositivo. Por ejemplo unos LEDs y un conjunto de I/O, o también reaccionar adecuadamente a los datos recibidos de las antenas GPS y GPRS o del PIC (peripheral interface controller) interno y darle además distintas funcionalidades y formas de reaccionar a los estímulos en tiempo real.

El FW va almacenado en una memoria flash del mismo, antiguamente se solía cargar el FW en una memoria ROM y no se actualizaba nunca. Esto servía para dispositivos como calculadoras o mandos a distancia por ejemplo, en los que una vez cargado el FW no era necesario modificarlo, actualmente se suele guardar el FW en algún tipo de memoria flash, ya que suele ser necesario su modificación, ya sea para corregir bugs o ampliar funcionalidades, es decir, actualizarlo.

2.2.-Preparación del entorno de trabajo

2.2.1.-Entorno de desarrollo (IDE)

Normalmente el desarrollo sobre sistemas embebidos se realiza en un entorno de desarrollo específico y personalizado, que suele complicar las cosas por la falta de flexibilidad que conlleva. Aunque no sea nada más que un IDE de los clásicos como Eclipse o Codeblocks ligeramente modificado.



Figura 6: Iconos de los IDEs Eclipse y Codeblocks

También se puede usar en determinados momentos un IDE como Codeblocks, que permite crear aplicaciones ejecutables de forma rápida para realizar tests simples sobre ciertas partes del código que puedan parecer problemáticas.

Aparte se suele recurrir a IDEs más específicos dependiendo de sobre qué estemos trabajando, por ejemplo, para trabajar con Arduino necesitaremos el IDE Arduino software y para microcontroladores PIC el IDE MPLAB.

La elección del IDE es muy importante ya que afectará al proyecto en su totalidad. Es importante elegir uno que se adecúe a nuestras necesidades teniendo en cuenta que para poder trabajar correctamente en el software embarcado se necesitaran las siguientes herramientas:

- Editor de texto (Text editor): Permite escribir el código fuente, normalmente en C o C++ y puede ser de gran ayuda si tiene características como el autocompletado, lenguaje personalizado o destacado de palabras.
- Compilador (Compiler): Permite convertir el código fuente (alto nivel) creado en el editor en código objeto (bajo nivel), que ya es entendible directamente por el ordenador.
- Ensamblador (Assembler): Convierte el código objeto a código máquina (binario).
- Depurador (Debugger): Sirve para comprobar si el código tiene errores, puede hacer comprobaciones parciales en busca de errores sintácticos o en tiempo de ejecución y notificar de los mismos.
- Enlazador (Linker): Permite combinar varios códigos objeto y varias librerías, de forma que para crear un proyecto con varios módulos de código y que use varias librerías se necesitará esta herramienta.
- Simulador (Simulator): Ayuda a ver como funcionaría el programa completo en la realidad y permite recrear las I/O para simular sensores.

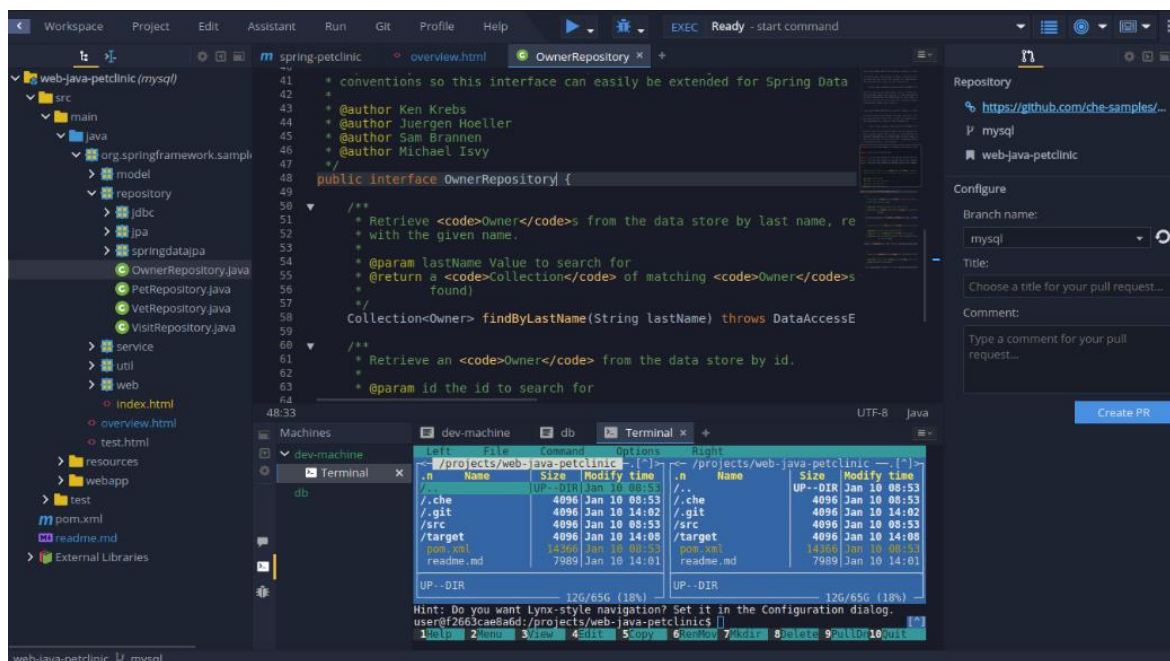


Figura 7: Ventana común del IDE Eclipse

En la *Figura 7* se puede apreciar lo que se visualiza en un IDE normal, desde el explorador de archivos a la izquierda hasta el terminal en la parte inferior, así como resaltado de palabras clave en distintos colores en el código. También se aprecian características más avanzadas como la conexión directa con el repositorio para subir el código directamente.

En el proyecto objeto de este TFG se utilizará un IDE especial que es una variante del clásico Eclipse y proporcionado por Sierra Wireless, proveedor del procesador ARM que está integrado en la placa del dispositivo embarcado y que es un microprocesador serie AR optimizado específicamente para soluciones en automoción y sistemas inteligentes de transporte.



Figura 8: Procesador tipo AR grado automoción

2.2.2.-Pautas de código limpio

A parte de elegir un IDE se deberán tener en cuenta algunas pautas básicas acordadas entre todos los desarrolladores para crear código de calidad.

Estas pautas no están escritas en ningún manual, pero hay algunas reglas básicas en las que la mayoría de la comunidad de desarrolladores se ponen de acuerdo:

- Mantener el código lo más simple posible. Esto es evitar cualquier complejidad no necesaria, al terminar una parte deberíamos preguntarnos si se podría haber conseguido lo mismo de una forma más simple.
- No repetir. Es importante que cada parte del código este claramente definida en un único sitio para evitar ambigüedades y código que se divida en ramas a lo largo de la ejecución, haciendo irrastreable cualquier fallo producido.
- No añadir funcionalidades a no ser que sean necesarias y estén en los requerimientos del cliente. Esto se debería aplicar junto a la refactorización de código, los tests unitarios y los de integración.

- El diseño de tipos se debe basar en para qué sirven más que en qué son realmente ya que a la larga aumenta la flexibilidad y reusabilidad de los mismos.
- Mejorar la legibilidad. Aunque el ordenador pueda entender el código es conveniente hacer algunas modificaciones que ayuden a que otras personas puedan entenderlo también ya que los proyectos suelen tener un carácter colaborativo. Entre otras cosas es recomendable meter en una variable con un nombre claro aquellos números muy utilizados para que no parezcan números mágicos o aumentar el tamaño de los nombres de variables haciéndolos más auto explicativos.
- Consistencia. Una vez que se ha decidido hacer algo de una determinada manera lo mejor es hacerlo así durante todo el proyecto y al menos con entenderlo una vez será suficiente para entenderlo en el resto de los sitios que aparezca.
- Principio de responsabilidad única. Estipula que cada módulo clase o función debería ser responsable de una sola parte de la funcionalidad que provee el software y estar encapsulado enteramente dentro de ese módulo, clase o función.
- La regla del boy scout. Es decir, dejar todo código por el que pases mejor de cómo lo encontraste.

No es necesario seguir todas estas pautas estrictamente, pero el mero hecho de haberlas leído y aplicar las más obvias probablemente mejore mucho la calidad del código de un proyecto.

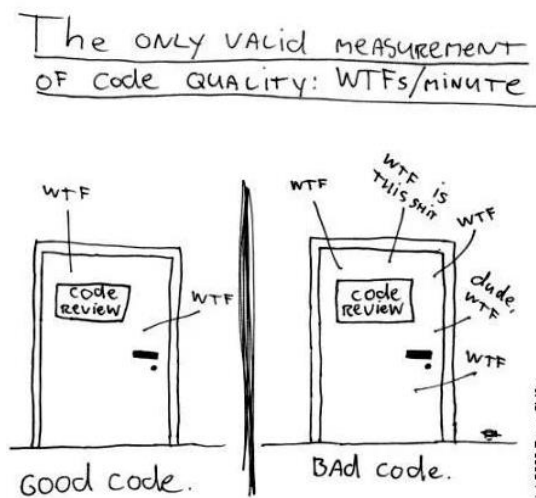


Figura 9: Medida de la calidad del código

2.3.-Análisis del proyecto concreto

El proyecto objeto de este TFG pertenece al sector de ITS (sistemas inteligentes de transporte). Se realiza en una empresa y deberá poder satisfacer una serie de requisitos y necesidades del cliente que se analizarán en este apartado. Para este propósito también será necesario realizar esquemas y diagramas explicativos para resumir y aclarar todas las funcionalidades y características con las que deberá contar el dispositivo embarcado.

2.3.1.-Dispositivo embarcado

El proyecto está orientado a la geolocalización y gestión de flotas de vehículos en tiempo real. Dispone de un dispositivo cuyo diseño y especificaciones técnicas quedan fuera del ámbito de este TFG pero que consiste básicamente en un PIC junto con un módulo simple (procesador ARM más memoria flash). En este módulo se cargará mediante un puerto serie o mediante teleprogramación el FW, es decir, el software que se habrá diseñado para operar en ese dispositivo y que sí es el objeto de este TFG.

Debido a las características del proyecto el dispositivo cumple unas especificaciones que hacen que:

- Tenga el menor tamaño posible, ya que deberá ir en un coche y por lo tanto no deberá molestar al funcionamiento normal del mismo ni a la comodidad del conductor, quedando finalmente contenido (tanto PIC como módulo y memoria) en unas medidas de aproximadamente 15x15x5 cm, en la *Figura 10* se puede ver la apariencia del mismo.



Figura 10:Dispositivo embarcado del proyecto objeto

- También deberá hacer saber al conductor como mínimo si está funcionando correctamente. Para esto se implementa un LED que luce y parpadea de distintas formas indicando el estado de la conexión con la red GPRS (que le permite comunicarse con central) así como el estado de la señal GPS (que le permite conocer su posición exacta).








Color LED	Estado
	Inicializando/Fallo Hardware
	No GPRS No GPS
	Si GPRS No GPS
Parpadeo  	No GPRS Si GPS
Parpadeo  	Si GPRS Si GPS

Figura 11:Tabla estados LED

En la *Figura 11* se pueden distinguir los posibles estados en los que se encuentra el dispositivo con respecto a la red GPS y GPRS.

Si parpadea dispone de señal GPS válida y si está en verde se ha establecido comunicación con central mediante la red GPRS.

En caso de que el LED esté en rojo se ha detectado un problema Hardware o todavía se está iniciando el dispositivo.

- A parte de un conjunto de pines para I/O digitales en los que se podrán conectar periféricos como un zumbador, lector de iButton (otra forma de identificación como RFID), pulsadores y botones a los que se podrá dar el uso que se quiera desde el FW etc. dispone de:

- Un puerto serie para poder conectar directamente con un PC y poder realizar tareas como la depuración, comprobación de tramas, ejecución del FW sobre el dispositivo real o directamente para cargar el FW en el dispositivo.
- Pines de conexión de la fuente de alimentación (12 o 24 V dependiendo del vehículo), gracias a los cuales el FW podrá detectar si el contacto del coche está encendido o apagado y el nivel de la batería del mismo, por si necesita ponerse en modo de protección de batería para no consumirla.
- Los conectores para antenas GPS (azul) y GPRS (morado), que serán clave para el correcto funcionamiento del dispositivo y para que realice las funcionalidades básicas de comunicación y geolocalización.
- Un conector RJ45 para conectar, por ejemplo, el lector RFID.

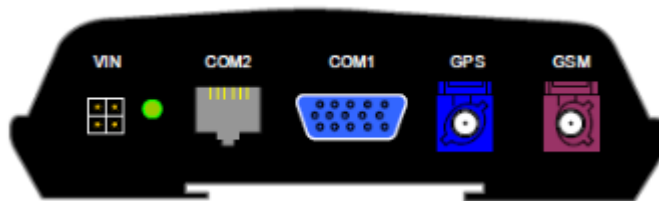


Figura 12: Esquema lateral del dispositivo embarcado

En la *Figura 12* se visualizan los puertos mencionados siendo en orden de izquierda a derecha:

- Los pines de la alimentación.
- El LED de funcionamiento.
- El conector RJ45 (COM2).
- El puerto serie (COM1).
- El conector de la antena GPS.
- El conector de la antena GPRS (GSM).

2.3.2.-Plataforma web (comunicaciones con central)

Mediante la red GPRS el dispositivo establece una conexión con central, donde hay toda una infraestructura de gestores de comunicaciones que procesan las tramas y muestran los resultados de una forma atractiva para el cliente en una página web.

En esta web se puede ver tanto la posición actual del dispositivo como sacar los informes de las rutas realizadas en días anteriores, comprobar alarmas por velocidad excesiva o por entrar en una zona configurada como prohibida y muchas cosas más.

Esta web es además la parte del proyecto más visible para los clientes ya que al tratarse de gestión de flotas será aquí donde recojan los datos que necesitan para mejorar eficiencia y controlar en tiempo real cada uno de sus vehículos.

La plataforma web cuenta con una parte de cara al público en la que los clientes introducen sus credenciales y visualizan su flota, teniendo una capacidad de modificación mínima sobre las funcionalidades del vehículo, aunque se les permite crear zonas prohibidas, registrar conductores para sus vehículos con su correspondiente identificación o elegir que alarmas deben saltar y como son notificados. Desde aquí también se pueden mandar órdenes como actualizar inmediatamente la posición del vehículo, realizar un volcado de las posiciones que no se han enviado porque no había conexión GPRS en el momento y que quedan guardadas en la memoria del vehículo o incluso inmovilizar el vehículo si el cliente tiene contratada esa funcionalidad.

La empresa propietaria del dispositivo embarcado puede acceder además a una plataforma paralela y ligeramente distinta a la que no tienen acceso los clientes y en la que se permite realizar una gestión más minuciosa de los vehículos, de forma que se pueden cambiar parámetros como la IMEI del dispositivo, los valores de tensión que acepta, los sensores que tiene activados y demás características que sí que afectan sensiblemente al funcionamiento del dispositivo y a las funciones que puede realizar.

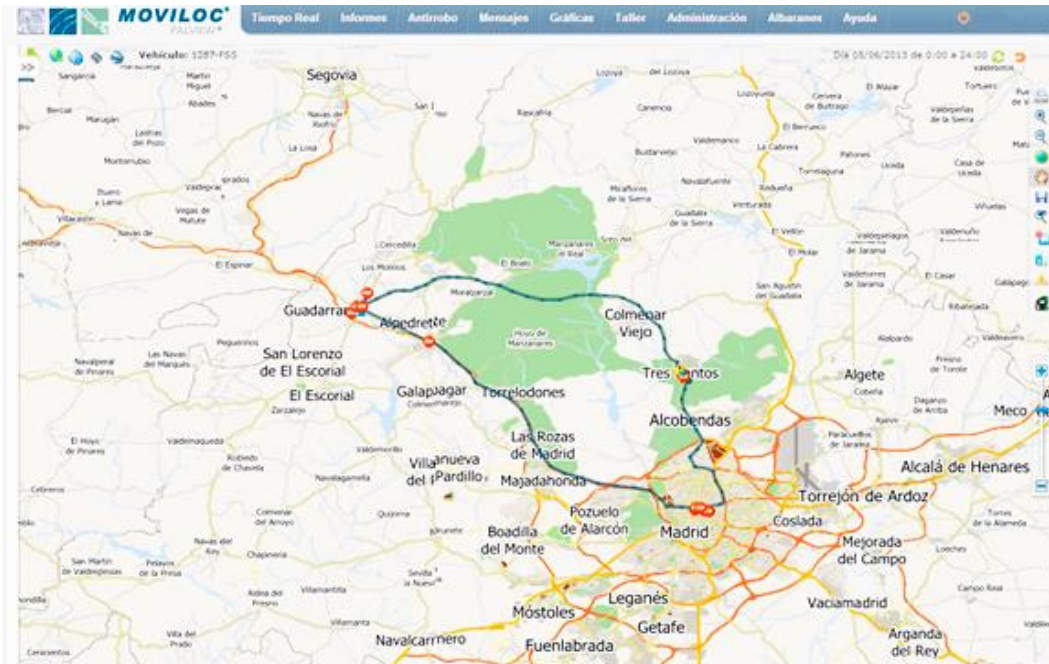


Figura 13: Visualización de la web de MOVILOC

En la *Figura 13* se pueden ver algunas de las características que ofrece la web, en la parte superior se pueden ver las pestañas de tiempo real, informes, gráficos o administración entre otros.

La parte central consiste en un mapa en el que se puede apreciar la ruta realizada por un vehículo, así como los puntos en los que se ha parado y además en los informes se podría visualizar la velocidad a la que fue en cada momento y otra información dependiendo de los sensores que tuviese activados, como temperatura, tensión de la batería, número de reinicios que ha sufrido el dispositivo o si ha saltado alguna alarma.

2.3.3.-Funcionalidades

En la página web de la empresa se puede encontrar la siguiente imagen (*Figura 14*) con las funcionalidades que ofrece su producto, es decir, el dispositivo embarcado con el FW cargado y con el soporte de la plataforma web.



Figura 14: Resumen funcionalidades dispositivo embarcado

Como se puede ver, el producto deberá ser capaz de ofrecer una gran cantidad de servicios, que deberá proveer el FW apoyándose en el dispositivo, que sería su HW y en la plataforma web para facilitar la gestión y visualización, esto significa que la mayor parte del trabajo recae en el desarrollo adecuado del FW.

También hay que tener en cuenta que el servicio que se ofrece es personalizado y por lo tanto no todos los clientes querrán disponer de todas las funcionalidades, aunque todas deban estar implementadas.

Con esto se pasa a explicar las funcionalidades concretas de las que debe disponer el vehículo, así como la solución que se adopta para conseguirla y cuál es la parte que debe ser solucionada mediante software.

2.3.3.1-Gestión y localización

Esta es la principal funcionalidad que ofrece el dispositivo, la geolocalización, puede parecer simple, pero hay muchas cosas a tener en cuenta.

Para obtener un trazado de rutas y posiciones del vehículo en la página web, así como saber la posición actual es necesario enviar un mensaje cifrado a central cada pocos segundos con esta información, esto significa que para saber dónde está el dispositivo en todo momento necesitamos la antena GPS y la GPRS funcionando correctamente a la vez y esto, por temas que no puede solucionar el FW, no es siempre posible.



Figura 15:Antena GPS+GSM

En la *Figura 15* se puede observar una antena similar a la elegida para el proyecto, que une las funcionalidades GPS y GSM de forma que solo se necesita un dispositivo para la geolocalización y comunicación por radiofrecuencia.

La señal GPS no es siempre fiable, si se tiene conexión con pocos satélites podría conseguirse una posición poco exacta y hay que tener en cuenta rebotes, como los causados por garajes o naves industriales, o las pérdidas de señal en parkings subterráneos. También se pueden obtener posiciones incorrectas a muchos kilómetros de la posición real de vez en cuando así que hay que implementar filtros, y además es difícil distinguir posiciones muy cercanas, si el vehículo está moviéndose muy despacio.

Resumiendo, para aceptar una posición GPS se deben tener en cuenta el número de satélites, la calidad de la señal, el DOP (dilution of precisión), umbrales de distancia entre posiciones (tanto muy bajos como muy altos) etc.

En el Anexo III: Receptor GPS se entra un poco más en profundidad en los temas relacionados con el receptor GPS integrado en la placa del dispositivo y la gestión que realiza sobre las tramas GPS para obtener su posición sobre la Tierra.

La señal GPRS también da numerosos problemas, está basada en el standard GSM y en las tecnologías 2G y 3G y todavía existen muchas zonas en las que se pierde la conexión GPRS y por lo tanto el dispositivo queda aislado de la central, también puede pasar que el dispositivo no pueda registrarse en la red por temas de la tarjeta SIM o que al perder la conexión una vez sea incapaz de reconectarse.



Figura 16: Cobertura 2G en España operador Movistar

La red de comunicaciones está evolucionando rápidamente y se está centrando en las nuevas tecnologías de 4G o más, por lo que numerosas zonas quedan completamente abandonadas en cuanto a cobertura 2G se refiere. Para saber la cobertura de una zona concreta basta con entrar en un buscador de internet y localizar una página web de mapas de cobertura, en este caso yo he elegido www.cellmapper.net y buscando el operador Movistar se puede observar la cobertura general en España para las distintas tecnologías como se muestra en las Figuras 16, 17 y 18.

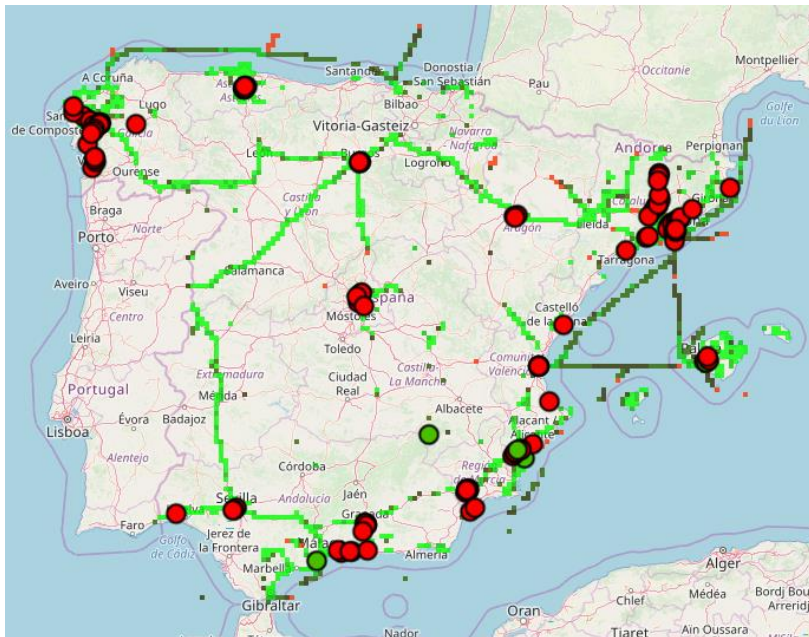


Figura 17: Cobertura 4G en España operador Movistar

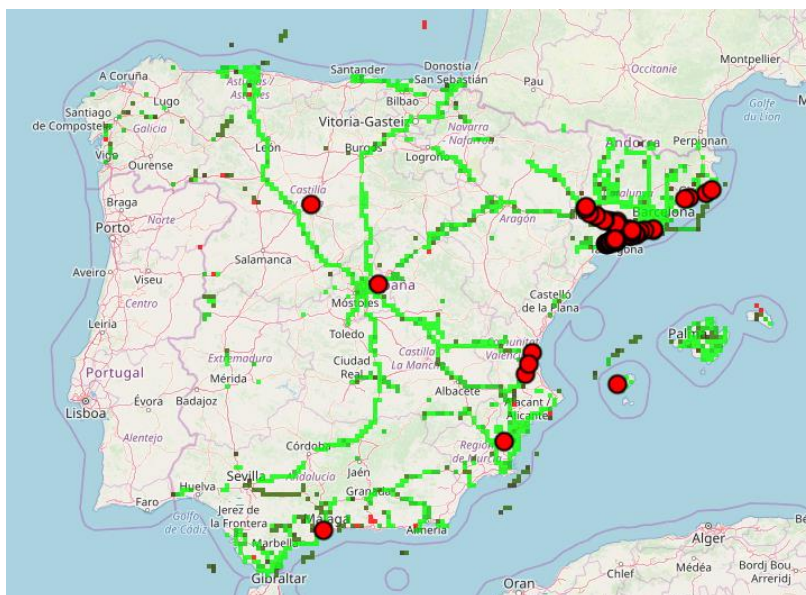


Figura 18: Cobertura 3G en España operador Movistar

Queda claro que la cobertura 2G está bastante olvidada y se está haciendo una transición a las nuevas tecnologías, que por un lado es muy buena señal, pero por otro afecta a los dispositivos basados en tecnologías antiguas.

Sin embargo, ya está prevista una actualización del hardware que permitirá la conexión a la red de telecomunicaciones mediante las nuevas tecnologías.

La tecnología 3G todavía está bastante presente y la 4G está en claro auge.

Todo esto viene a decir que para la señal GPRS también hay que establecer un protocolo de autenticación y de control de señal para detectar cuando la conexión no es correcta o directamente no existe e intentar una reconexión o en un caso peor un reinicio completo del equipo.

El envío de mensajes con la posición del vehículo es la parte central del proyecto y su correcto procesamiento y envío permite a la plataforma web crear todo tipo de informes y mapas con la posición actual o los trazados con los recorridos realizados.

2.3.3.2.-Seguridad

El dispositivo puede conectarse directamente mediante su puerto serie con la OBD del vehículo y obtener una gran cantidad de información sobre el mismo.

La OBD es una característica con la que cuentan prácticamente todos los vehículos desde hace 20 años en adelante, aunque ya existía con anterioridad.

Consiste básicamente en una ECU (Electronic control unit) que recibe la información de sensores situados en numerosas partes del coche y que procesa los datos recibidos mandando señales a unos actuadores para que el coche se comporte como se desea, también es responsable de la infame luz de revise motor y además cuenta con un conector (DLC, Diagnostic link connector) que se utiliza en talleres para poder sacar toda la información sobre el estado del coche mediante una herramienta de escaneo especializada.

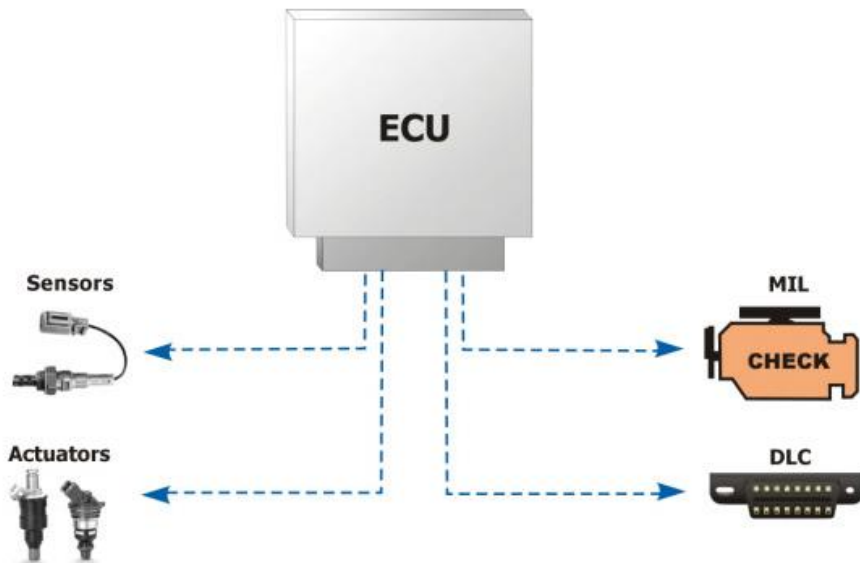


Figura 19: Esquema general OBD

Es gracias a esta última parte (el DLC) que se puede conectar el dispositivo mediante puerto serie (ya que este solo se usa durante el desarrollo y para depuración) para poder obtener toda la información del estado del coche que se quiera procesar y a partir de este momento dispondremos de datos del vehículo como su velocidad, temperatura del motor, niveles de oxígeno, combustible y muchos más, en el Anexo OBD-II PIDS se adjuntan los servicios que da el OBD con los datos que puede recoger del vehículo.

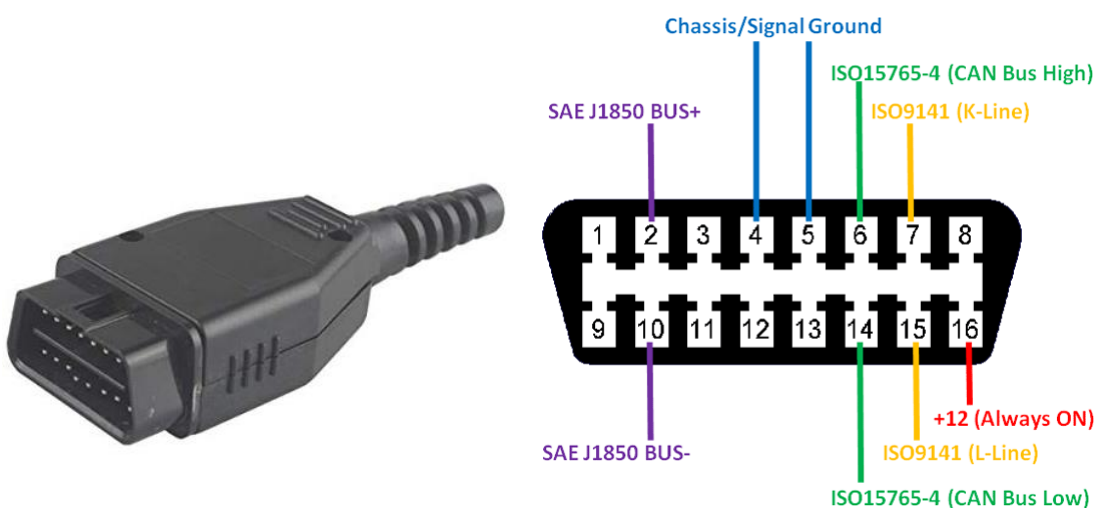


Figura 20: Conector DLC y pines

En la *Figura 20* se puede observar cómo es el conector DLC, que para poder conectar con el dispositivo del proyecto debería tener en su otro extremo un conector serie (se pueden utilizar adaptadores intermedios especiales, llamados interpretes) y además a la derecha aparece el uso de los pines.

El procesado de las tramas de la OBD por parte del dispositivo es bastante estándar, aunque hay que tener un poco de cuidado ya que los vehículos pueden seguir protocolos distintos y algunos pueden emitir mediante el protocolo CAN y otros seguir el especificado en el SAE J1979 u otro distinto dependiendo de la marca y antigüedad del vehículo.

En definitiva, mediante el correcto procesado de las tramas de la OBD en el FW del dispositivo embarcado se pueden configurar alarmas en caso de fallos mecánicos, necesidad de revisión del vehículo, robos e incidencias de todo tipo.

2.3.3.3.-Eficiencia y beneficios

Esta funcionalidad va más relacionada con la plataforma web y la forma de crear informes y gráficas útiles a partir de los datos conseguidos que con el FW en sí.

Lo único de lo que debe asegurarse el FW es de enviar todos los datos necesarios a central y referenciarlos al vehículo adecuado con una fecha correcta para que central pueda almacenar todos los datos recibidos bajo la ID del dispositivo y crear informes y gráficas temporales con los datos del mismo.

Esto se consigue con varios identificadores, cada dispositivo cuenta con:

- Una ID numérica de 5 cifras única
- Una IMEI propia de la SIM y que también es única de cada SIM
- Un nombre de dispositivo que facilita su búsqueda y que es al que tiene acceso el cliente y puede modificar
- Un campo que identifica la versión exacta de FW cargada en el vehículo, así como la versión para el PIC.

Gracias a todo esto es casi imposible identificar erróneamente un dispositivo y además cada vez que el dispositivo envía un mensaje al centro de control también manda en la trama su ID numérica para que central pueda procesar y relacionar los datos correctamente.

Asegurando la transmisión de datos correctos, identificados y fechados termina el trabajo del FW para esta funcionalidad, después será la plataforma web la que deba analizar y juntar los datos para crear mapas, informes y gráficas.

MOVILOC PALVIEW | Tiempo Real | Informes | Antirrobo | Mensajes | Gráficas | Taller | Administración | Albaranes | Ayuda

INFORME DE VISITAS DE VEHÍCULOS

Fecha de Consulta: 13/05/2013 0:00 - 13/05/2013 23:59

Vehículo:

13/05/2013	Hora	Descripción	Duración	Visita Nº
9:14:57	Alpedrete - Cliente [Autopista del Noroeste, s/n - San Lorenzo de El Escorial (Madrid) [España]]	5m.	1	
9:19:57	Majadahonda - cliente [N-505, s/n - Rozas de Madrid (Las) (Madrid) [España]]	0s.	1	
9:39:32	Majadahonda [Calle de Valgrande, s/n - Majadahonda (Madrid) [España]]	27m. 36s.	1	
9:39:32	Majadahonda - cliente [Calle de la Sacedilla, s/n - Majadahonda (Madrid) [España]]	0s.	2	
10:08:08	LIBRERIA 2 [Avenida de Pozuelo, s/n - Alcorcón (Madrid) [España]]	36m. 20s.	1	
10:24:08	LIBRERIA 1 [Avenida de Europa, 11 - Pozuelo de Alarcón (Madrid) [España]]	6m. 4s.	1	
10:24:08	Majadahonda - cliente [Avenida del Príncipe de Asturias, s/n - Majadahonda (Madrid) [España]]	5m.	3	
11:00:28	Majadahonda [Calle de Valgrande, s/n - Majadahonda (Madrid) [España]]	0s.	2	
13:19:53	Majadahonda - cliente [Avenida de Juan Carlos I, 14 - Majadahonda (Madrid) [España]]	1m.	4	
13:25:57	Majadahonda [Calle de Valgrande, s/n - Majadahonda (Madrid) [España]]	2h. 24m. 33s.	3	
14:24:17	Majadahonda - cliente [Avenida del Príncipe de Asturias, s/n - Majadahonda (Madrid) [España]]	2m. 16s.	5	
14:29:17	Alpedrete - Cliente [Avenida de Juan Carlos I, s/n - Collado Villalba (Madrid) [España]]	2m.	2	

Resumen completo

» Alpedrete - Cliente	7m. 3m. 30s.*	2
» LIBRERIA 1	6m. 4s. 6m. 4s.*	1
» LIBRERIA 2	36m. 20s. 36m. 20s.*	1
» Majadahonda	2h. 52m. 9s. 57m. 23s.*	3
» Majadahonda - cliente	8m. 16s. 1m. 39s.*	5
» Número de puntos visitados:		5

Figura 21:Informe detallado de visitas

MOVILOC PALVIEW | Tiempo Real | Informes | Mensajes | Gráficas | Taller | Administración | Ayuda | Administrador

Informe Detallado Completo

14:33:07	Autovía del Mediterráneo, s/n - Tarragona (Tarragona) [España]	100,00	90,00
14:34:07	N-340, s/n - Tarragona (Tarragona) [España]	100,00	93,00
14:35:07	N-340, s/n - Rima de Gaià (la) (Tarragona) [España]	100,00	86,00
14:36:48	T-214, s/n - Torredembarra (Tarragona) [España]	60,00	40,00
14:37:07	Autopista del Mediterráneo, s/n - Torredembarra (Tarragona) [España]	40,00	21,00
14:38:07	AP-7, s/n - Poble de Montornès (la) (Tarragona) [España]	120,00	123,00
14:39:07	Autopista del Mediterráneo, s/n - Creixell (Tarragona) [España]	120,00	111,00
14:40:07	Autopista del Mediterráneo, s/n - Roda de Barà (Tarragona) [España]	120,00	125,00
14:41:07	Autopista del Mediterráneo, s/n - Roda de Barà (Tarragona) [España]	120,00	122,00
14:42:07	Autopista del Mediterráneo, s/n - Vendrell (el) (Tarragona) [España]	120,00	132,00
14:43:07	Autopista de Pablo Casals, s/n - Vendrell (el) (Tarragona) [España]	50,00	51,00
14:44:07	Autopista de Pablo Casals, s/n - Vendrell (el) (Tarragona) [España]	120,00	108,00
14:45:07	Autopista de Pablo Casals, s/n - Vendrell (el) (Tarragona) [España]	120,00	122,00
14:46:07	Autopista de Pablo Casals, s/n - Calafell (Tarragona) [España]	120,00	117,00
14:48:07	Autopista de Pablo Casals, s/n - Cunit (Tarragona) [España]	100,00	100,00
14:49:07	Autopista de Pablo Casals, s/n - Cubelles (Barcelona) [España]	90,00	111,00
14:50:07	Autopista de Pablo Casals, s/n - Cubelles (Barcelona) [España]	120,00	104,00
14:51:07	Autopista de Pablo Casals, s/n - Cubelles (Barcelona) [España]	120,00	131,00
14:52:07	Autopista de Pablo Casals, s/n - Vilanova i la Geltrú (Barcelona) [España]	120,00	127,00
14:53:07	Autopista de Pablo Casals, s/n - Vilanova i la Geltrú (Barcelona) [España]	120,00	114,00
14:54:07	Autopista de Pablo Casals, s/n - Sant Pere de Ribes (Barcelona) [España]	120,00	113,00
14:55:17	Autopista de Pablo Casals, s/n - Sant Pere de Ribes (Barcelona) [España]	120,00	130,00
14:56:07	Autopista de Pablo Casals, s/n - Sant Pere de Ribes (Barcelona) [España]	120,00	110,00
14:57:07	Autopista de Pablo Casals, s/n - Sant Pere de Ribes (Barcelona) [España]	120,00	125,00
14:58:07	Autopista de Pablo Casals, s/n - Sitges (Barcelona) [España]	90,00	100,00
14:59:07	Autopista de Pablo Casals, s/n - Sitges (Barcelona) [España]	70,00	100,00
15:00:07	Autopista de Pablo Casals, s/n - Sitges (Barcelona) [España]	90,00	100,00

Nota Legal - MOVILOC. Powered by PALVIEW® v3.7 (© GRV Sistemas SRA)

Figura 22:Informe detallado de velocidades

En las *Figuras 21 y 22* se puede observar un ejemplo de informes visualizables por el cliente en la plataforma web, con estas herramientas los clientes pueden establecer una serie de prioridades y mejoras sobre su flota para mejorar la eficiencia de su negocio gracias a datos fiables y variados.

2.3.3.4.-Periféricos

Gracias a las I/O digitales de las que dispone el dispositivo se pueden conectar numerosos periféricos cuyos datos procesará el FW para dar nuevas funcionalidades personalizadas a los clientes que las requieran.

Los periféricos más comunes son:

- Conexión CAN BUS-FMS

Como ya se ha explicado se pueden obtener datos relevantes del vehículo conectándose a la OBD, en este caso se utiliza un analizador FMS para extraerlos, los datos en tiempo real permiten analizar el comportamiento del conductor, aumentar la eficiencia del vehículo, ahorrar combustible, reducir emisiones, disminuir costes de mantenimiento etc.

Algunos datos extraíbles serían por ejemplo el nivel de combustible, la distancia recorrida, la temperatura del motor, las revoluciones por minuto, los excesos de velocidad o el tiempo de conducción entre otros.



Figura 23:Velocímetro y cuentarrevoluciones de un coche

Los datos básicos que aparecen en la *Figura 23* sobre un coche, como la velocidad, las rpm, la temperatura o el nivel del depósito pasan directamente al dispositivo mediante el conector DLC.

- Sensor de temperatura digital

Permite conocer la temperatura donde se sitúe el sensor, es especialmente útil para camiones frigoríficos que quieren evitar interrumpir la cadena de frío mediante alarmas configuradas para activarse al llegar a unos límites establecidos.

El sensor usado en el proyecto será el TQS3 (Figura 24), con una alta precisión y que sigue el estándar de comunicación RS-485, cuyas tramas serán procesadas por el FW.

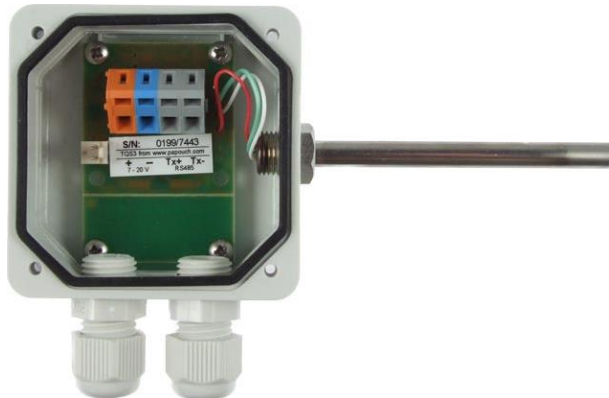


Figura 24: Sensor de temperatura TQS3

- Sensor de apertura de puertas

Se pueden colocar en las puertas de un vehículo con carga trasera para garantizar el control y seguridad de la carga transportada ya que envían señales directamente al dispositivo embarcado.



Figura 25: Sensor de apertura de puertas

En la Figura 25 se muestra el sensor de apertura de puertas utilizado en el proyecto, la mayoría de los sensores de este tipo trabajan mediante un interruptor por campo magnético y un imán,

cada una de estas partes va situada en uno de los lados de forma que estén casi juntos cuando las puertas estén cerradas y se separarán cuando se abran, lo que hará que se abra el circuito y por tanto se detecte la apertura de la puerta, de forma que se enviará esta información al dispositivo y este actuará de manera acorde.

- Identificador de conductor (iButton)

Consta de una pequeña unidad de lectura conectada al dispositivo y unas llaves de identificación con forma de botón que envían un código único al pasar por el lector y que permite identificar al conductor que está utilizando el vehículo.

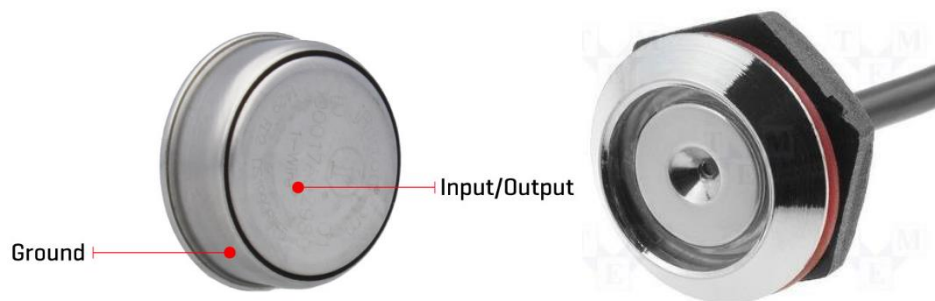


Figura 26:iButton (izq) y lector (dcha)

En la *Figura 26* se aprecia como el iButton no es más que un pequeño botón metálico en el que en realidad la superficie actúa como interfaz para comunicar información.

Cuando tanto el lateral del iButton (tierra) como su parte inferior (I/O) entran en contacto con las partes correspondientes del lector se inicia el protocolo 1-Wire de comunicación por el cual se transmite la información del conductor hasta el dispositivo, donde de nuevo el FW se encarga de procesar las tramas 1-Wire para conseguir toda la información necesaria.

- Botón de pánico

Es un botón discreto instalado en el vehículo que activa una alarma inmediata al pulsarlo avisando a los servicios de emergencias, quedando el lugar fecha y hora en la que se ha producido la alarma registrados.



Figura 27: Botón de pánico integrado en un coche

Es importante establecer una clara prioridad para el envío de la señal de pánico dentro del FW ya que dentro del mismo hay muchas condiciones que pueden causar que no se envíen tramas a central y esta señal específica debería tener la más alta prioridad y enviarse siempre que se use el pulsador a no ser que sea imposible, por ejemplo si se ha perdido la señal GPRS y por lo tanto la comunicación con central.

- Inmovilización remota

Se puede conectar un sistema de inmovilización al puerto serie del dispositivo que permite la completa inmovilización del vehículo dadas unas determinadas circunstancias como la identificación de un conductor no registrado o directamente una orden desde central o desde la plataforma web por parte del cliente.



Figura 28: Inmovilización remota de un vehículo

Este apartado se explica más en detalle en el punto 2.3.3.6, Inmovilización remota.

2.3.3.5.-Gestión de alarmas

Esta funcionalidad está más ligada a central que al FW del dispositivo, se pueden generar alarmas por numerosos motivos como:

- Detección de un exceso de velocidad, el dispositivo sabe la velocidad máxima permitida en la zona en la que se encuentra gracias a una base de datos y puede conseguir la velocidad real a partir de la señal GPS o directamente de la OBD del vehículo. La central simplemente lanza una alarma cuando detecta que se supera la permitida en la zona actual.
- Entrada/salida de una zona restringida, desde la plataforma web se pueden configurar zonas de “geofencing” que después se cargan al dispositivo y este se encarga de procesar en cada momento si se encuentra dentro o fuera de la zona mediante filtros y algoritmos implementados en el FW. Cuando se detecta que el estado del dispositivo respecto de la zona ha cambiado (dentro/fuera) se envía un mensaje a central y esta manda la alarma correspondiente.
- Parada o parada en ralentí demasiado larga, el dispositivo puede detectar mediante el FW el estado del motor y del contacto del coche a partir de las tensiones de entrada y mandar esos datos a central, que hace saltar una alarma si se detecta una situación como las anteriores durante un período largo.

Todas estas alarmas se pueden configurar para que sean enviadas al propietario o gestor de la flota mediante correo electrónico o SMS automáticamente.

Hora	Evento	Descripción	Valor
11:44:14	Geofencing	zzPruebas traiva leivansan, ENTRADA en zzPruebas_traiva_POI_ [zzPruebas_traiva_POI_ [Calle de Barcelona, 26 - Arroyomolinos (Madrid) [España]]]	
12:16:05	Geofencing	zzPruebas traiva leivansan, SALIDA de zzPruebas_traiva_POI_ [zzPruebas_traiva_POI_ [Avenida del Mediterráneo, s/n - Arroyomolinos (Madrid) [España]]]	
12:19:04	Geofencing	zzPruebas traiva leivansan, ENTRADA en zzPruebas_traiva_POI_ [zzPruebas_traiva_POI_ [R-5, s/n - Arroyomolinos (Madrid) [España]]]	
12:20:14	Geofencing	zzPruebas traiva leivansan, SALIDA de zzPruebas_traiva_POI_ [zzPruebas_traiva_POI_ [R-5, s/n - Arroyomolinos (Madrid) [España]]]	
12:24:23	Geofencing	zzPruebas traiva leivansan, ENTRADA en zzPruebas_traiva_POI_ [zzPruebas_traiva_POI_ [Calle de Barcelona, 87 - Arroyomolinos (Madrid) [España]]]	

Resumen completo	Valor
Parada Larga	0
Parada Larga Ralentí	0
Geofencing	5
Alarma Antirrobo	0

Figura 29:Informe de alarmas en la plataforma web

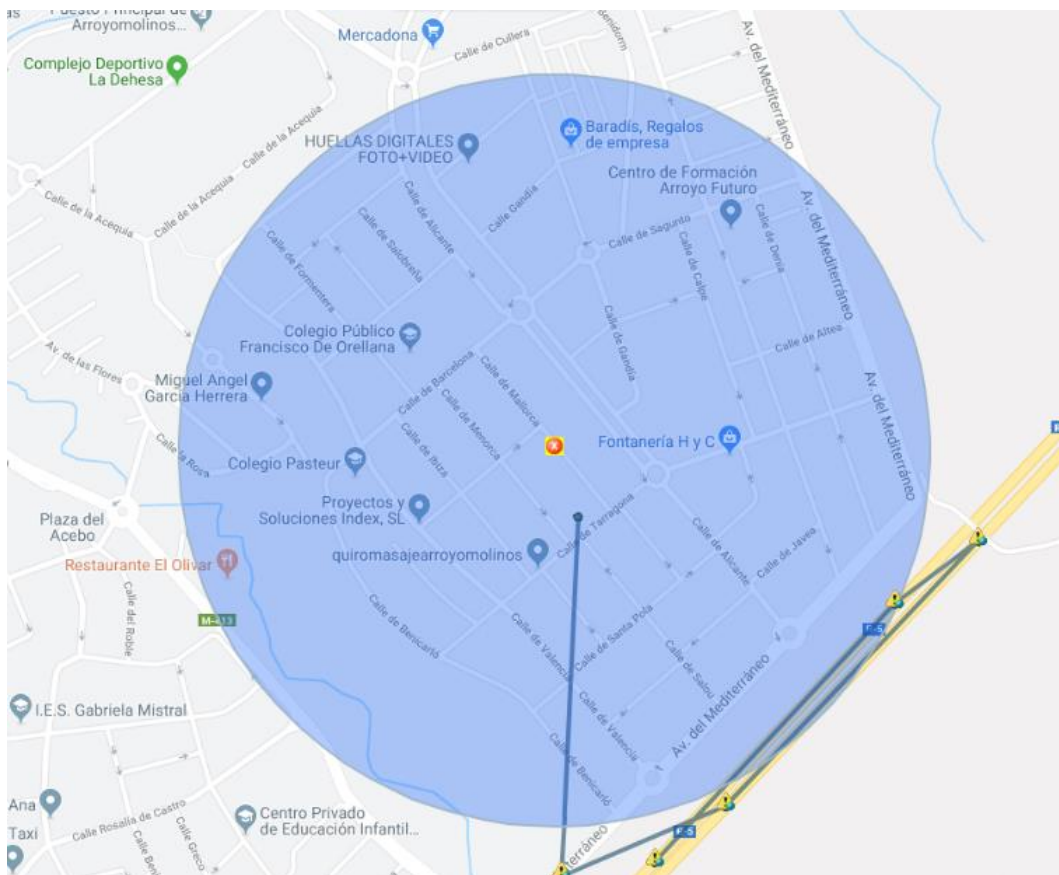


Figura 30: Zona geofencing y recorrido de un vehículo que entra y sale de la zona

En la *Figura 29* se aprecia como en la plataforma web aparecen las alarmas correspondientes a cuando el vehículo se sale de la zona establecida para el mismo y en la *Figura 30* cómo en la pantalla de informes se pueden ver todas las alarmas que han saltado con datos como la hora y el lugar exacto, así como el motivo de la alarma.

2.3.3.6.-Inmovilización remota

Como ya se ha mencionado uno de los periféricos que se puede acoplar al dispositivo para dotarlo de más funcionalidades es el inmovilizador, en este caso se usa un sistema de dos inmovilizadores con lógica negativa para reducir el consumo que puedan causar.

Uno de los inmovilizadores se alimenta directamente del contacto del vehículo y conecta al puerto serie del mismo y el segundo inmovilizador se alimenta con la salida del anterior y conecta la batería del coche para permitir su arranque.

De esta forma el coche estará inmovilizado siempre a no ser que el contacto esté encendido (dando paso a la señal a través del primer inmovilizador) y además la señal del puerto serie sea negativa (ya que una señal positiva empujaría el relé e impediría el arranque del motor).

Esta funcionalidad se puede activar directamente desde central con el envío de un mensaje al dispositivo o mediante el FW si se detectan unas condiciones específicas como encendido tras protección de batería o si no está logado el conductor designado.



Figura 31:Relé usado para inmovilización (x2)

2.2.3.7.-Recorridos realizados

Esta funcionalidad es casi enteramente de la central, el dispositivo solo debe mandar posiciones datadas y a partir de las posiciones recibidas para cada vehículo crear un mapa de recorridos.

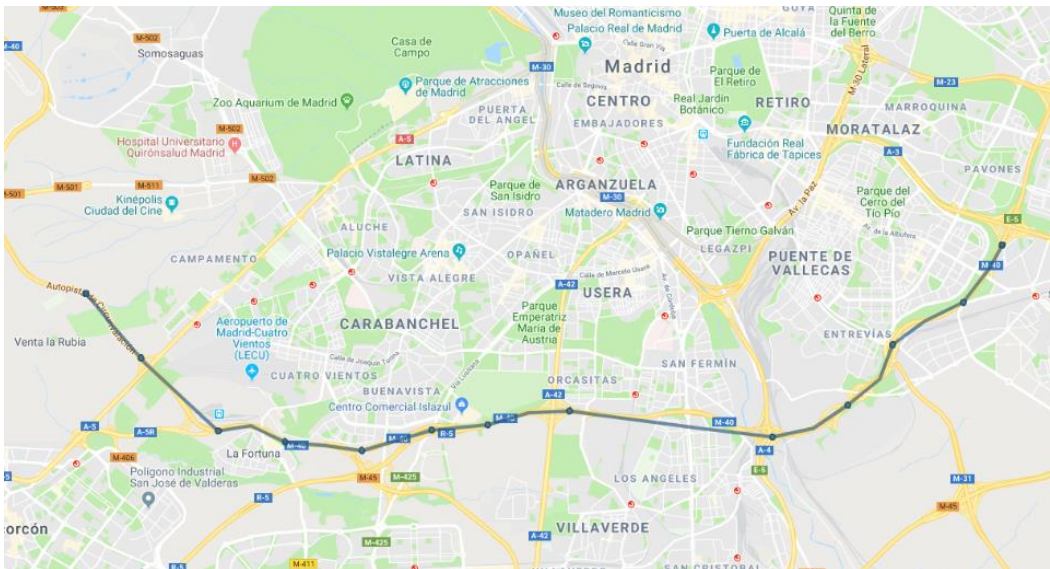


Figura 32: Recorrido de un vehículo visualizado en plataforma web

En la Figura 32 se puede ver el recorrido que ha realizado un coche, trazado uniendo las posiciones que ha enviado el mismo.

2.3.3.8.-Cercanía a un lugar

También recae casi completamente sobre central, el dispositivo no cuenta con una base de datos con las gasolineras o restaurantes cercanos, sería un desaprovechamiento de memoria, esa información la consigue central y la relaciona con la posición actual del vehículo para que el cliente pueda visualizar los lugares cercanos que sean de su interés.

En la Figura 33 aparecen por ejemplo las gasolineras en los alrededores de Valladolid.

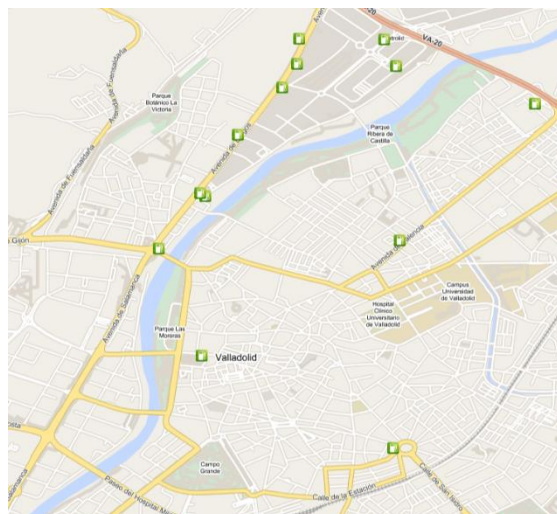


Figura 33: Gasolineras marcadas en mapa

2.3.3.9.-Consulta detallada de velocidades

La velocidad del vehículo se conoce en tiempo real y es calculada de forma precisa directamente por el receptor GPS utilizando la posición, el tiempo y el efecto Doppler relativo a cada satélite con el que establece comunicación la antena GPS, también se puede obtener directamente del vehículo mediante la OBD, pero no suele ser necesario ya que la primera práctica es más sencilla para el dispositivo, al que llega una velocidad y no se tiene que preocupar de nada más.



Figura 34: Efecto Doppler para ondas

Una vez conocido ese dato se envía a central junto con la posición y allí ya se crean los informes detallados de velocidades en cada minuto y cuando se ha superado el límite concreto de una zona aparece la velocidad para ese momento en naranja o rojo, dependiendo de por cuanto se haya superado, pudiendo esto causar que salte también alguna alarma.

2.3.3.10.-Acceso a los datos CAN BUS-FMS

Como ya se ha mencionado uno de los periféricos más comunes establece una conexión entre el dispositivo y la OBD del vehículo que suele ser utilizada solo para realizar escaneos del vehículo en busca de fallos en los talleres pero que en realidad cuenta con una enorme cantidad de información que se puede utilizar para numerosos propósitos.

La única dificultad que tiene el dispositivo es ser capaz de pedir a la OBD los datos que necesita utilizando los protocolos de comunicación específicos del vehículo (hay muchos y no todos son necesarios) y procesar las tramas respuesta adecuadamente para extraer los valores de los distintos sensores y almacenarlos para realizar cálculos o enviarlos directamente a central.

2.3.3.11.-Avisos de mantenimiento de vehículos

De esta parte también se encarga central ya que el dispositivo puede recabar la información directamente de la OBD y enviarla para que la plataforma web muestre el aviso en el vehículo pertinente.

En este caso específico el FW mandará a la OBD un mensaje siguiendo el protocolo CAN o el pertinente en cada vehículo en el que solicita los datos que quiere, para el mantenimiento del vehículo se debe solicitar el servicio 03, que devuelve los DTCs (Diagnostic trouble codes) que se hayan activado en el vehículo en una trama que sigue el protocolo de encapsulamiento ISO 15765-2. Si hay más de 3 DTCs se envían en tramas separadas.

El FW se encarga del completo procesamiento de tramas entre la OBD y el dispositivo para poder mandar al centro de control los datos relevantes. que haya “traducido”.

En el centro de control también se lleva cuenta de las veces que ha pasado un vehículo por el taller y aparecen notificaciones con recomendaciones para la próxima fecha por la que debería pasar por mantenimiento.



Figura 35:Icono mantenimiento recomendado

2.3.3.12.-Identificador de conductor

Esta funcionalidad se implementa mayormente en el FW, que se encarga de detectar cuando se ha identificado un conductor, ya sea mediante RFID (lector más tarjeta) o iButton (lector más botón identificador), registrar su número único de identificación y realizar una comprobación cada vez que se intenta identificar un nuevo usuario de forma que si es el mismo que ya estaba logado se produzca un deslogado y si es distinto se produzca un deslogado del anterior seguido de un logado del nuevo.



Figura 36: Lector RFID con tarjetas identificadoras

En la *Figura 36* se puede observar el sistema de identificación del tipo RFID, que es distinto al ya explicado con anterioridad del iButton.

Esta dualidad y la posibilidad de que haya otros métodos de identificación aparte crea un problema relativamente complejo para el FW ya que en todos los casos se debe detectar un conductor con su identificador correcto, pero en cada caso se debe hacer de una forma ya que las tramas recibidas son distintas y por tanto requieren un procesado diferente y además los números de identificación no tienen por qué tener ni siquiera el mismo tamaño.

También se establece una comunicación con la base de datos de la plataforma web en la que los clientes pueden registrar conductores nuevos para sus vehículos, así cada vez que se intenta identificar un conductor el FW manda su número a central que verifica que éste se encuentre registrado como conductor válido para ese vehículo, impidiendo el logado del usuario si no es el caso.

2.3.4.-Modulación del código del proyecto

El código del proyecto no se puede mostrar como tal ya que es propiedad intelectual de la empresa y además ha pasado por muchos otros desarrolladores, se muestran, sin embargo, como se ha realizado la división en módulos los más independientes posibles y la división en funciones con propósitos diferenciados.

En la *Figura 37* se puede apreciar como el FW se ha dividido en muchos módulos orientando cada uno de ellos a una funcionalidad concreta, además cabe destacar el desglosado en funciones en cada módulo con su correspondiente propósito y explicación del mismo.

Algunos módulos que destacar serían:

- El de GPS, que procesa todo lo relacionado con las tramas de la antena GPS, así como los cálculos de fechas y horas que se relacionan con cada posición, transformaciones de formatos GPS, detección de posiciones no válidas, detección de estación del año, o solución de problemas como el rollover (Se trata en el ANEXO II).

La aceptación de la posición solo se produce cuando hay un número suficiente de satélites detectados por el receptor GPS, la calidad de la señal es superior a un umbral y el DOP está dentro de unos límites.

- El de Geofencing, que se encarga de configurar en el vehículo todas las zonas que se delimiten como “especiales” por el usuario desde la plataforma web, ya sean círculos de un determinado radio alrededor de un punto o zonas poligonales como la creada en la zona central de Madrid para evitar que entren los coches con emisiones altas.

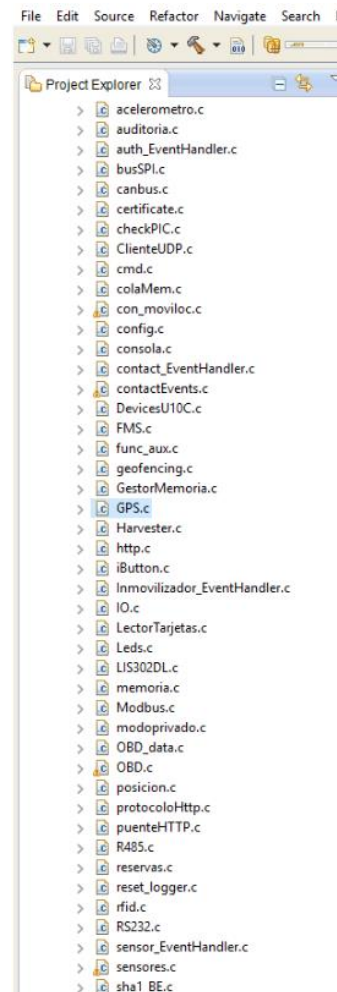


Figura 37:Algunos de los módulos del proyecto

El FW recibe de central los segmentos o parámetros que componen la zona y los junta para crearla, además permite la realización de un chequeo para validar que se ha cargado correctamente e implementa varios algoritmos que permiten conocer si la posición actual del vehículo se encuentra dentro o fuera de alguna de las varias zonas que se pueden cargar en el vehículo, pudiendo así detectarse entradas o salidas de la misma y hacer que central envíe las alarmas correspondientes.

- El de iButton y el de RFID, los dos son formas de realizar una identificación de usuario y normalmente se usa únicamente uno de los dos métodos, pero en el FW se crea una sola variable que guarda el número de identificación del conductor para facilitar las cosas y por si en un futuro apareciesen nuevas formas de identificación de usuario.

En ambos casos el FW se encarga de extraer el número de identificación pertinente cuando se aproxima un identificador de usuario al lector correspondiente, procesando la trama recibida y guardando en una variable el número que le interesa y que comparará con los que hay en la base de datos de conductores para ese vehículo que se encuentra en el centro de control.

También se contemplan casos para realizar deslogados automáticos después de varias horas o si se entra en el estado de protección de batería o tras algún tipo de reinicio concreto.

Con estos datos se puede llevar cuenta del kilometraje realizado por un solo conductor, las velocidades a las que ha ido o si ya lleva demasiado tiempo en carretera y debería realizar un descanso.

- El de temporizadores, este es un módulo general en el que se inicializan numerosos temporizadores propios del modem y a los que se suscriben procesos de otros módulos ya sea para enviar las posiciones a central cada cierto tiempo, para esperar ACKs o para restablecer conexiones perdidas.

Los temporizadores se pueden lanzar de forma cíclica o única, se establece un tiempo y además un Handler que establezca las acciones a realizar si el temporizador se completa.

En algunos se suscribe al temporizador con la intención de que se lleve a cabo lo que establece el Handler cada cierto tiempo mientras que en

otros el Handler trata la excepción y solo ocurre cuando no ha ocurrido el evento esperado, que habría cancelado el temporizador desuscribiéndose del mismo.

- El de Cliente UDP, se encarga de abrir un canal de comunicación entre el centro de control y el dispositivo, primero registrándose en la red GPRS con la IMSI única de su SIM y después abriendo el canal hacia la IP y puerto correspondiente del servidor de central.

En este módulo también se realiza una monitorización del estado de la red y se programan acciones específicas ante casos como pérdida de conexión GPRS o no detección de mensajes de central durante un tiempo determinado.

El modem detecta internamente si se pierde la antena GPRS y en este módulo se llevan a cabo acciones en caso de que se detecte este evento, sin embargo, hay veces que el dispositivo no se consigue comunicar con central y el estado de la antena está, según el módem, perfectamente.

Para esto se suscribe a temporizadores que intentar restablecer la conexión si pasa mucho tiempo sin recibir ningún mensaje de central o si no se recibe el ACK que sigue a cada envío de posición varias veces seguidas.

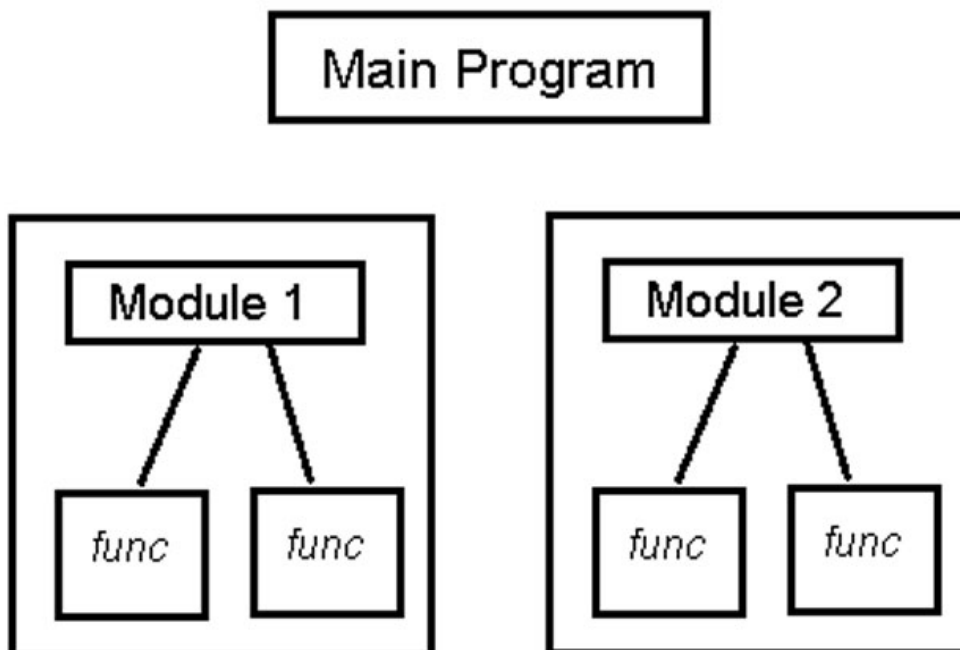


Figura 38: Programación modular



A la práctica de dividir el proyecto en numerosos módulos lo más independientes posible que a su vez contienen funciones se la conoce con el nombre de programación modular y presenta numerosas ventajas:

- Las funciones que realizan acciones parecidas suelen incluirse en un mismo módulo de forma que son más fáciles de entender y localizar.
- Al realizar una separación por funcionalidades el código puede volverse a usar para un proyecto distinto que requiera una funcionalidad concreta ya diseñada.
- Permite que varios desarrolladores trabajen a la vez en un mismo proyecto.
- Mejorar la mantenibilidad del código imponiendo límites lógicos entre los módulos.
- Si los módulos son lo suficientemente independientes no se requiere conocimiento del resto de módulos para entender el funcionamiento de uno de ellos.
- Se minimiza el alcance de las variables si se evitan o minimizan las de carácter global.
- La localización de errores se acelera.

2.4.-Infraestructura para la mantenibilidad y fácil desarrollo del proyecto

A parte de lo que es el código en sí, se pretende la implementación de un conjunto de características que facilitarán enormemente el desarrollo futuro y mantenibilidad del proyecto.

Las tres principales características para este propósito son los tests unitarios, los tests de integración y el uso de servidores remotos.

2.4.1.-Tests unitarios

Dentro de la estructura de testeo de software los tests unitarios son el principio de la cadena, sin ellos es difícil asegurar la validez del resto de tests y además son la primera línea de defensa contra código defectuoso.

Un proyecto sin tests unitarios no se beneficia demasiado de disponer de ningún otro tipo de tests ya que aunque este detectase un fallo sería muy complicado encontrar la causa.

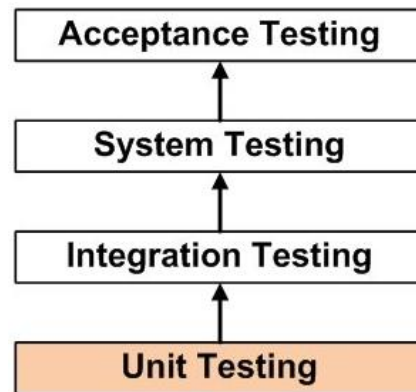


Figura 39:Esquema de jerarquía de tests

2.4.1.1.-Introducción a los tests unitarios

Los tests unitarios son un tipo de tests a nivel de software que se basan en realizar comprobaciones y validaciones a las mínimas unidades funcionales, es decir, se divide el software lo máximo posible en unidades con una función claramente definida y acotada como por ejemplo una sola función dentro de un módulo y se realizan los tests necesarios sobre esa función o unidad funcional para asegurarse de que cumple con todos los requisitos con los que se supone que debe cumplir.

Estas unidades funcionales mínimas suelen tener como mucho un par de entradas y una sola salida.

Un ejemplo podría ser una función simple dentro del módulo GPS que convirtiese la hora en un formato GPS (entrada) a otro formato distinto (salida).

Para realizar unos tests unitarios sobre esta función se deberá pasar unos argumentos a la función y comprobar que la salida de la misma es la esperada.

Es de especial interés a la hora de testear comprobar que todo funciona correctamente en casos fuera de lo normal, en este caso podrían ser años bisiestos o cambios de hora y también casos extremos, como pasos de un día a otro (23:59:59 — 00:00:00) o de un año a otro.

Los tests unitarios están muy relacionados con la técnica de *White Box Testing*, que consiste básicamente en el hecho de que cuando el desarrollador prepara los tests conoce a la perfección lo que está testeando y por lo tanto en el caso de una función sabe perfectamente todo lo que hay dentro y cuáles son las salidas a esperar ante unas determinadas entradas.

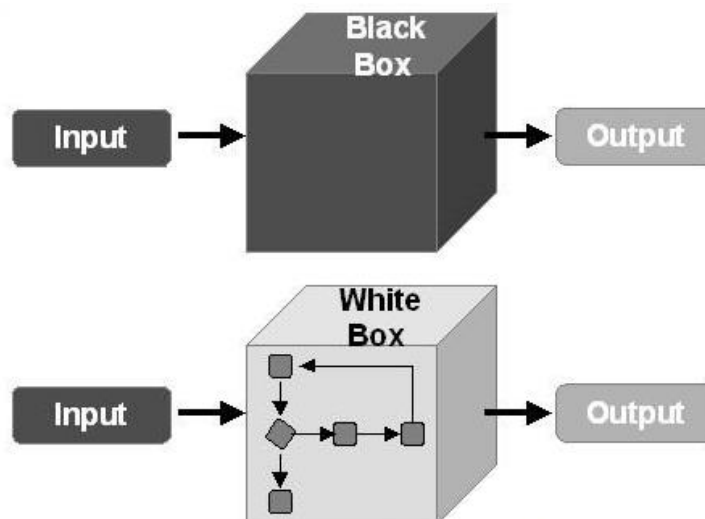


Figura 40: Diferencia entre la técnica de la caja negra y la caja blanca

Como se parecía en la *Figura 40* es básicamente lo opuesto a la técnica de la caja negra, en la que se supone que la función es un bloque opaco del que no se conoce nada más que las entradas que le damos y las salidas que devuelve.

Normalmente es recomendable crear el código desde un principio pensando en implementar tests unitarios ya que esto hace que el código se vuelva mucho más limpio y modulado como consecuencia y obviamente facilita la implementación de tests unitarios.

La otra opción es intentar implementar los tests unitarios directamente en un código ya creado de antemano pero esto puede complicar innecesariamente la tarea de testeo ya que si el código no se creó con la idea en mente de estar modulado y diferenciado al realizar los tests unitarios se puede caer en interrelaciones con otras funciones o módulos y puede llegar a no salir rentable la implementación de tests unitarios.

En muchos casos los tests unitarios no se llegan a implementar con excusas como que no son imprescindibles, que llevan demasiado tiempo o simplemente que el cliente no los pidió pero en el 90% de los casos el ligero sobresfuerzo inicial de crear estos tests ahorrará una cantidad de problemas futuros sorprendentes, sobre todo en proyectos grandes y con varios desarrolladores.



Figura 41: Viñeta cómica sobre el testeo incorrecto

En la *Figura 41* se muestra una viñeta satírica que critica un problema común a la hora de desarrollar código.

Suponiendo un caso corriente en el que al probar una versión se detecta un pequeño bug y se comunica al desarrollador correspondiente, este tenderá a intentar quitárselo de encima rápidamente implementando una solución que podría causar algún problema en otra parte del código.

Creyendo que el cambio es mínimo y que soluciona el bug objetivo nadie se percata de que ahora hay otra funcionalidad que ha dejado de funcionar y nadie ha validado hasta que se termina detectando en producción, causando muchísimos más problemas que si se hubiese detectado con un simple test unitario nada más que el desarrollador intentase implementar la solución.

2.4.1.2.-Ventajas del uso de tests unitarios

El uso de tests unitarios en general da una gran confianza a la hora de desarrollar software ya que permite avanzar en un proyecto sabiendo que existe un respaldo que hará que salte una alarma en caso de que se introduzca código defectuoso, de forma que no hay que estar pendiente continuamente de no meter la pata. Como se muestra en la *Figura 42* equivaldría a la cuerda que asegura a los escaladores en caso de que sufran un resbalón.

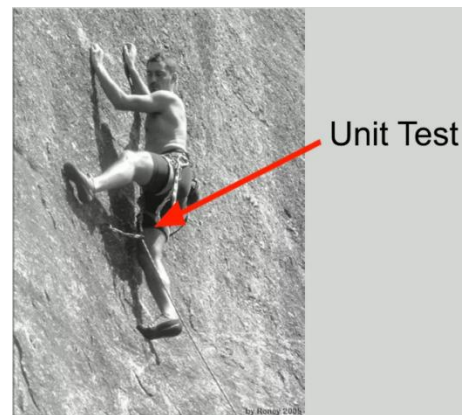


Figura 42: Ilustración de la importancia de los tests unitarios

También ayuda el hecho de que para realizar tests unitarios el código creado deberá ser lo menos interdependiente posible y por lo tanto en el raro caso de que se cuele algún error afectaría poco al resto de módulos del proyecto.

Al crear código más modulado y realizar tests unitarios sobre el mismo aumenta enormemente la reusabilidad del código, de forma que si creamos y testeamos una función que, por ejemplo, transforma los números en una string a formato float o una función que actúa como filtro paso bajo de doubles tenemos la posibilidad de extraer esa función con sus tests y usarlas de nuevo en un proyecto completamente distinto.

Aunque en un principio pueda parecer lo contrario, los tests unitarios también ahorran mucho tiempo, es cierto que su implementación aumenta el tiempo de desarrollo pero el tiempo que requieren las fases de testeo y validación se puede reducir enormemente ya que no es necesario que el desarrollador inicie el programa entero y realice sus pruebas para comprobar cada posible input y output y además durante la validación los errores se pueden detectar y localizar mucho más fácilmente, por otro lado el hecho de detectar un error mediante tests unitarios es mucho más beneficioso y menos costoso de solucionar que tener que esperar hasta los niveles más altos de testeo para encontrarlo, o lo que sería todavía peor, que sea el cliente final el que lo encuentre.

2.4.1.3.-Herramientas para implementación de tests unitarios (Testing frameworks)

Las dos principales herramientas a las que se ha echado mano durante este proyecto han sido Google Test y Ceedling, ambas comparten la mayoría de las funcionalidades y características, pero están optimizadas para ámbitos distintos.

Ceedling

Es un framework que junta las funcionalidades de los proyectos de código abierto CMock, Unity y CException y está orientado a al desarrollo con tests en lenguaje C.

Como el proyecto objeto de este TFG consiste en un desarrollo embarcado en C específicamente esta será la opción elegida.

Google Test

Es un framework también orientado hacia el desarrollo con tests que incluye las mismas funcionalidades que el anterior y que es además una de las soluciones más comunes elegidas a la hora de implementar tests unitarios en un proyecto. Sirve tanto para C como para C++, sin embargo, está más centrado en lo relacionado con clases y aspectos de C++ que no está disponibles en C y por lo tanto no se eligió esta solución.

Ambos frameworks se basan en unas cuantas características similares a la hora de crear tests unitarios:

- **Stubs:** Los stubs (talones) son objetos creados para ayudar a implementar tests unitarios y que devuelven una respuesta prefijada cuando se los llama de una determinada forma, además no suelen responder fuera de esos parámetros preprogramados.
- **Mocks:** Los mocks (imitaciones) se utilizan principalmente para la creación de tests unitarios para aislar el comportamiento de la unidad que deseamos testear en caso de que dependa de otros objetos o funciones complejas. Esto significa que todas las unidades funcionales complejas que intervengan en la unidad que se está testeando se mockearan creando unidades similares que solamente simulen el comportamiento de esas unidades reales.

La principal diferencia entre mocks y stubs es que los primeros esperan que se realicen ciertas llamadas a funciones durante el test y por lo tanto pueden causar el fallo del test unitario, mientras que los segundos solo se usan para que la unidad testeada reciba los argumentos que necesita para funcionar devolviendo respuestas fijas y por lo tanto no suelen ser la causa de fallo de un test unitario.

Las funciones principales que permitirán realizar tests unitarios serán la función ASSERT (y sus variaciones) y la función EXPECT (y sus variaciones).

Assert hace que el test falle y pare completamente cuando no se llama a la función que pide assert o no se entra en la misma con los argumentos esperados. En principio se usa cuando no tiene sentido que el test continúe si no se ha cumplido la condición del assert.

Expect hace que el test falle cuando no se recibe la output esperada de una función, pero no interrumpe su ejecución por lo que una vez finalizados todos los tests aparecería un mensaje diciendo que han fallado X tests porque no se ha cumplido su condición de expect.

2.4.1.4.- Ejemplo de tests unitarios

A continuación se muestran unos breves ejemplos para exponer en general la metodología de los tests unitarios de forma más visual que solo mediante texto.

El primer ejemplo consiste en una función que calcula un checksum de una cadena de caracteres.

```
int checksum(void *p, int len)
{
    int accum = 0;
    unsigned char* pp = (unsigned char*)p;
    int i;
    for (i = 0; i <= len; i++)
    {
        accum += *pp++;
    }
    return accum;
}
```

En esta función el puntero '*p*' que se da como parámetro de entrada a la función es la cadena de la que se quiere obtener el checksum y '*len*' sería la longitud de dicha cadena.

Lo que hace la función es sumar el valor de los caracteres de la cadena casteados como unsigned chars en un acumulador, que es lo que devuelve al final la función checksum.

Sin embargo, en esta función se ha cometido un error muy común y que es el perfecto ejemplo de por qué los tests unitarios son necesarios.

Dentro del bucle *for* se suman en total los *len+1* primeros caracteres de la cadena debido a que el bucle *for* va desde *i = 0* hasta *i <= len*, repitiéndose una vez de más.

En este caso el error podría pasar desapercibido muy fácilmente ya que si la cadena contiene palabras el carácter extra que se coja será un `\0` y por lo tanto no cambiará el checksum, pero en cuanto no se dé este caso el checksum fallará.

Es por esta razón que se debe crear un test unitario que permita comprobar que esta función realiza lo que se pretende correctamente en varios casos.

A continuación se muestra el test unitario en el que fallaría esta función:

```
void test_checksum()
{
    srand(time(NULL));
    char buf[1024];

    for (int i = 0; i < 1024; i++)
    {
        buf[i] = (char)rand();
    }

    for (i = 0; i <= 9; i++)
    {
        buf[i] = i;
    }

    int testval = checksum(buf, 10);
    // if (testval == 45)
    {
        printf("Passed!\n");
    }
    else
    {
        printf("Failed! Expected 45, got %d\n", testval);
    }
}
```

Se comienza por crear una semilla para crear números aleatorios y declarar un buffer, que será la cadena que se pasará el test.

Después se llena la cadena de caracteres aleatorios, pero también se da valor a los diez primeros caracteres, siendo estos los números del 0 al 9.

Presumiblemente el checksum debería dar 45, ya que la suma de los diez primeros caracteres de la cadena es 45 ($0+1+2+3+4+5+6+7+8+9$), sin embargo, si dejamos la función checksum como estaba un poco más arriba el resultado no será 45 en casi ningún caso (se puede dar la casualidad de que el carácter después del 9 sea un cero) ya que esa función sumaría $0+1+2+3+4+5+6+7+8+9+'?'$ siendo '?' un carácter con valor aleatorio después del 9.

Al pasar el test se obtendría el mensaje de **Failed!** y se pasaría a buscar el error.

Como nota cabe apuntar que los tests unitarios deberían ser lo más deterministas posibles pero en algunos casos es recomendable usar números aleatorios ya que en algún momento se podría producir una situación que no se había tenido en cuenta como conflictiva y el test podría fallar, detectando un bug que de otra forma habría sido imposible de encontrar.

La mejor forma de realizar tests unitarios es, sin embargo, realizar antes los tests que el propio código, esto puede parecer un poco contradictorio a primera vista pero se supone que cuando se comienza a desarrollar una función parte de un módulo se obtienen una lista de requisitos y se puede prever cual es el propósito de la función antes de crearla, de esta forma, lo primero que se deberá hacer es crear un test sobre la función (que todavía no está implementada) y que vaya chequeando que se cumplan todos los requisitos de la función.

En un ejemplo muy simple y ya en el entorno de desarrollo en C con Ceedling podríamos tener lo siguiente:



```
#include "unity.h"
#include "point.h"

void setUp(void)
{
}

void tearDown(void)
{
}

void test_MakePoint_Returns_A_New_Point(void)
{
    struct point pt = MakePoint(3, 4);
    TEST_ASSERT_EQUAL_INT(3, pt.x);
    TEST_ASSERT_EQUAL_INT(4, pt.y);
}
```

Donde *unity.h* es una librería auxiliar que nos permite usar funciones como *assert* y *expect*, *point.h* sería el header del módulo en el que se encuentra la función que se va a probar y las funciones *setUp* y *tearDown* se ejecutarían antes y después de cada test en ese orden.

El primer test en este caso sería *test_MakePoint_Returns_A_New_Point*, que realiza una llamada a la función *MakePoint* con los valores 3 y 4 y después usa *TEST_ASSERT_EQUAL_INT* para esperar que el valor 'X' del punto 'pt' valga 3 y el valor 'Y' 4, como debe ser ya que son los valores que se han dado a la función *MakePoint*.

Al pasar este test de primeras está claro que fallará, y esa es la intención de hacer los tests antes que la función, ya que así se sabrá que cuando el test pase será que se ha implementado correctamente la función, mientras que si pasa antes de haber hecho nada queda claro que no se está testeando bien lo que se debe.

Esta sería la forma que tendría el header *point.h*

```
#ifndef point_H
#define point_H

struct point {
    int x;
    int y;
};

struct point MakePoint(int, int);

#endif // point_H
```


Donde se declara la estructura *'point'*, formada por dos enteros (*'X'* e *'Y'*) y aparece la definición de la función *MakePoint*, a la que se le pasan dos enteros y devuelve una estructura tipo *point*.

Finalmente en el módulo *point.c* se crearía la función *MakePoint*

```
#include "point.h"

struct point MakePoint(int x, int y)
{
    struct point pt;
    pt.x = x;
    pt.y = y;
    return pt;
}
```

Que simplemente mete los dos valores de entrada en una estructura tipo *point* y devuelve la estructura.

Una vez creado *point.h* y *point.c* e implementada la función *MakePoint* el test pasará sin errores y quedará comprobado que se cumple correctamente la funcionalidad diseñada para dicha función.

Obviamente estos dos ejemplos han sido bastante simples y las funciones reales son mucho más complejas, sin embargo, si se estructura el trabajo de forma ordenada y se divide el código en unidades funcionales lo suficientemente pequeñas, es decir, si se trabaja ya pensando que se van a implementar tests unitarios, o si se introducen antes de comenzar, el trabajo queda bastante simplificado y además ofrece mucha seguridad a la hora de trabajar ya que se dispone de la red de seguridad que son los tests unitarios.



Figura 43:Viñeta satírica tests unitarios

En la *Figura 43* aparece una función básica que recoge el momento del día y devuelve que es por la mañana cuando obviamente es de noche y por lo tanto el test falla, no hay un ejemplo más sencillo de lo que son en esencia los tests unitarios.

2.4.2.-Tests de integración

Los tests de integración son el segundo nivel de la pirámide de tests, una vez que se ha asegurado la validez de las unidades funcionales mediante los tests unitarios se comprueba la cohesión y compatibilidad entre unidades funcionales con los tests de integración.

En resumen, los tests de integración prueban la interfaz entre dos unidades de software.

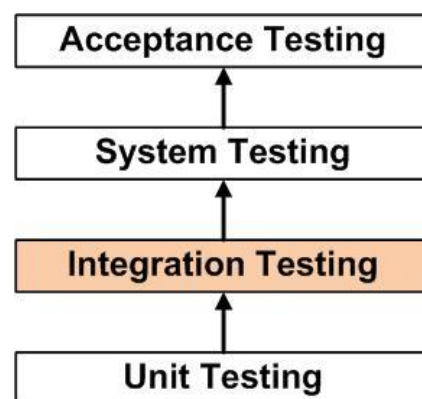


Figura 44:Esquema de jerarquía de tests

2.4.2.1.-Introducción a los tests de integración

Los tests de integración están un nivel por encima de los unitarios y consisten en tomar varias unidades funcionales y testear cómo interactúan entre ellas, es decir, no se busca probar que cada unidad hace lo que le corresponde si no comprobar que todas las unidades son capaces de relacionarse y depender de otras correctamente, es decir, se busca testear las interfaces entre unidades funcionales.

Sin estos tests se puede caer en el error de crear módulos llenos de funciones perfectamente válidas pero que son incapaces de funcionar como un sistema conexo ya que no pueden interactuar entre sí correctamente, la *Figura 44* explica de forma bastante clara por qué los tests unitarios no son suficiente.







	Unit Testing	Integration Testing
		
		

Figura 45:Esquema explicativo de la necesidad de tests de integración

Esta situación no es para nada extraña ya que un módulo en general es diseñado por un solo desarrollador que puede crear código perfecto pero que puede entender la lógica del proyecto al que se enfrenta de forma distinta a los desarrolladores que están trabajando en el resto de los módulos de forma que cuando se intenta juntar todos los módulos en un solo programa aparecen fallos que se podrían haber evitado con tests de integración.

La principal tarea de los tests de integración es testear interfaces, esto significa que no solo deben comprobar que el software funciona correctamente si no también que el software interactúa correctamente con el hardware, cosa bastante importante en los sistemas embebidos y que por tanto suele ser el motivo del uso de tests de integración en proyectos como el que se trata en este TFG.



2.4.2.2.-Metodologías para la realización de tests de integración

Método Big Bang

Consiste en unir todo el proyecto de una sola tacada de forma que se obtenga el programa completo y se testee en su totalidad.

Es especialmente adecuado para proyectos pequeños ya que si se realiza un test de integración con todas las unidades funcionales unidas a la vez se puede asegurar que en las interacciones entre unidades en el programa final van a ser correctas.

Sin embargo, cuando el proyecto sobrepasa un cierto tamaño aparecen problemas como que la localización de errores se vuelve difícil, se puede pasar por alto alguna interacción entre unidades y por lo tanto no se realiza el test correspondiente, es necesario que todos los módulos del proyecto estén terminados y en muchos casos no se finalizan a la vez y además no se puede establecer una prioridad de los módulos a testear en caso de que se quiera hacer mayor hincapié en alguno que sea crítico y menos en otros que puedan no interactuar con el usuario final.

Método Bottom-Up

Es un método de tipo incremental que consiste en testear las relaciones de las unidades funcionales de más bajo nivel primero y luego las relaciones de estas con las de más alto nivel paso a paso hasta que se testean todos los módulos.

Ofrece ventajas como la fácil localización de errores, así como que no es necesario disponer de todo el software para comenzar los tests de integración, pero puede provocar que defectos en los módulos críticos de alto nivel se detecten muy tarde.

Se necesitarán herramientas como los stubs usados en los tests unitarios para simular las unidades de alto nivel en caso de que no estén disponibles cuando se quieran realizar los tests.

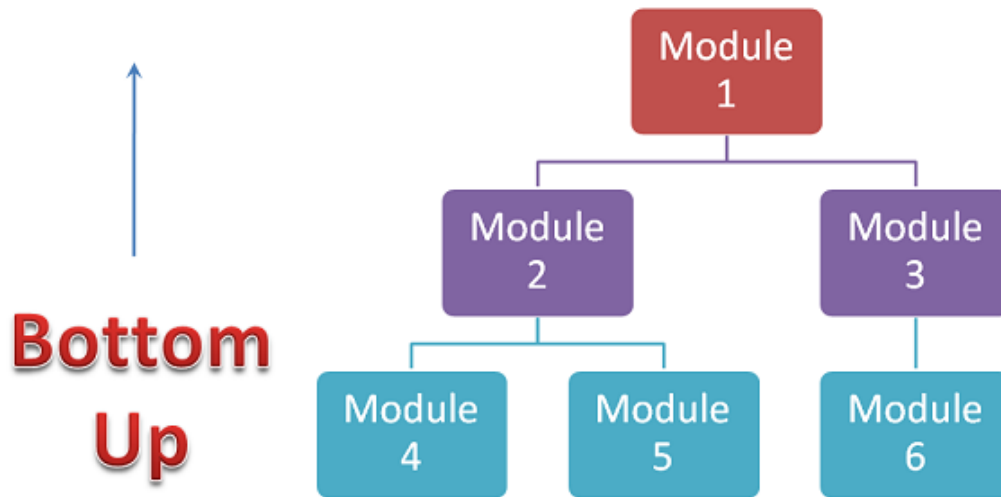


Figura 46:Diagrama representativo del método Bottom-Up

Método Top-Down

Es un método incremental que sigue la lógica opuesta al anterior, las unidades funcionales de más alto nivel que suelen ser críticas y que rigen el flujo del programa son testeadas antes evitando o detectando rápidamente posibles fallos de diseño y después se testean las unidades de más bajo nivel paso a paso.

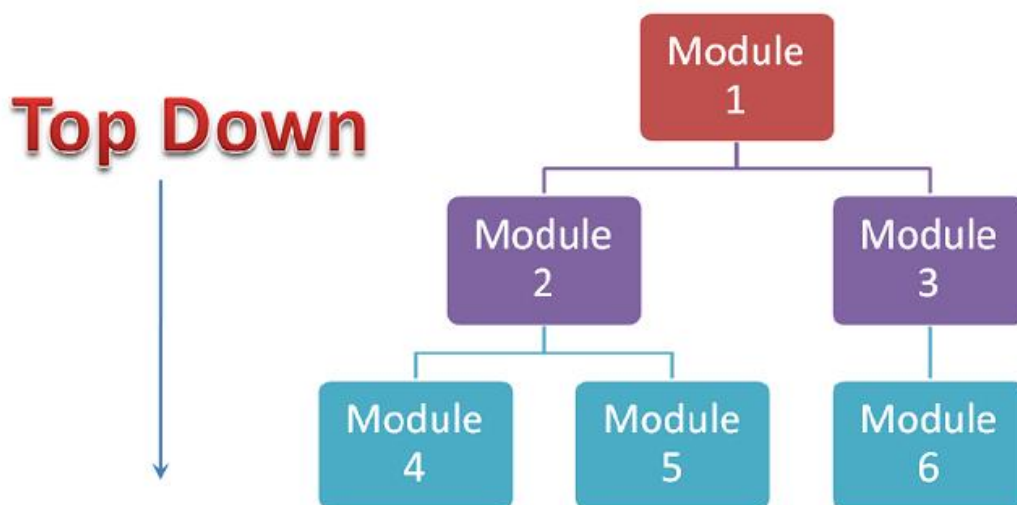


Figura 47:Diagrama representativo método Top-Down

Este método también permite la fácil localización de errores y además permite la obtención de un prototipo inicial del programa así como la priorización de testeo a módulos críticos.

Sin embargo, al realizar tests sobre las unidades de alto nivel se suelen necesitar numerosos stubs para simular las unidades de bajo nivel que todavía no estarán diseñadas en esta fase del proyecto y además los tests sobre las unidades de bajo nivel no suelen ser adecuados.

Método Híbrido o Sandwich

También es un método de tipo incremental que consiste en una mezcla de los dos anteriores, los módulos de alto nivel se testean con los de bajo nivel y a la vez se prueba la integración de los de bajo nivel con los de alto nivel.

Hace uso de stubs también y suele necesitarse el proyecto completo para realizar los tests, pero es bastante riguroso a la hora de cubrir todos los fallos que deberían controlar los tests.

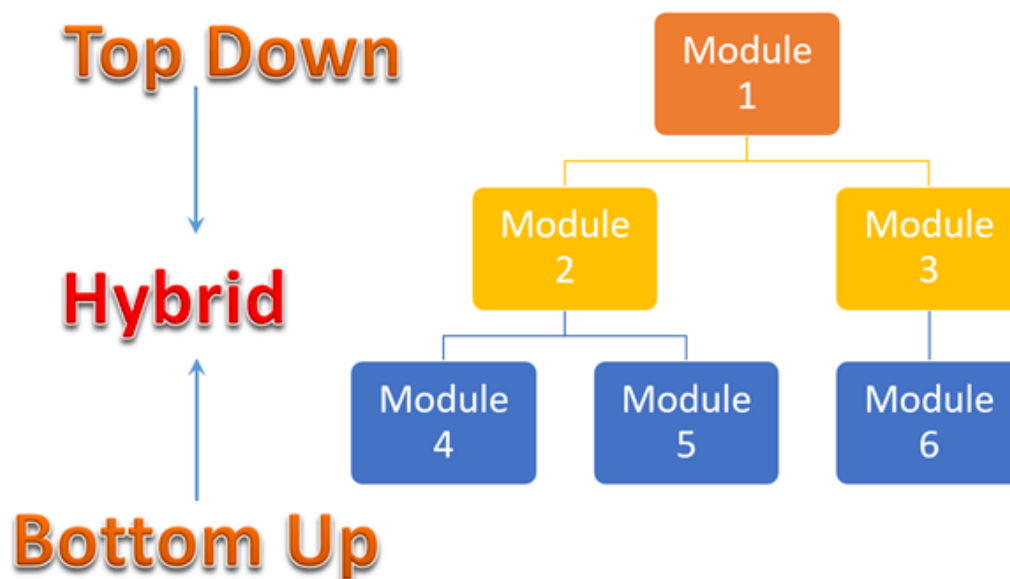


Figura 48: Diagrama representativo método híbrido

Lo primero es elegir un método que se deberá seguir en todo el proyecto para realizar los tests de integración.

Una vez hecho esto se deberán poder diferenciar los módulos críticos para concentrar los tests en ellos y además se deben haber definido previamente las interfaces entre todas las unidades relativas a bases de datos, software y hardware ya que estas serán objeto de tests y podría olvidarse alguna si se intenta definir las mismas durante la fase de testeo en vez de con antelación.

Hay que asegurarse de que cada unidad haya pasado los tests unitarios antes de pasar los de integración y hay que tener preparados los datos falsos que servirán para realizar los tests también antes de la fase de testeo.

2.4.2.3.-Herramienta para implementación de tests de integración

Como en este caso el principal problema al que se enfrenta el proyecto es el hecho de que el FW controle correctamente el HW del equipo los tests de integración implementados cubren sobre todo la interfaz entre el Software desarrollado y el HW del dispositivo del que se dispone.

Por este motivo se trata de unos tests de integración un poco especiales y se utilizará Robot Framework.

Robot Framework es una herramienta de código abierto orientada a la creación de tests de aceptación para proyectos TDD (Test Driven Development), incluye numerosas librerías que facilitan la creación y automatización de tests y está escrito en Java y Python, siendo su uso completamente independiente del sistema operativo y de la aplicación que queramos testear.



Figura 49:Icono Robot Framework

En el proyecto objeto del TFG se utiliza esta herramienta para realizar tests de integración que consisten en ir modificando variables eléctricas como el voltaje de alimentación del dispositivo y comprobar que responde adecuadamente, en este caso entrando en protección de batería.

Se realizan también comprobaciones sobre tensiones internas del circuito integrado y sobre el funcionamiento del LED gracias a estos tests de integración.

2.4.2.4.-Ejemplos de tests de integración

En este apartado no se tratará código real de tests de integración ya que en el caso de este proyecto solo se aplica para testear la interfaz entre el FW y el HW del dispositivo embarcado y no se dispone de un ejemplo lo suficientemente simple y claro como para hacer una demostración.

Se pasa por lo tanto a explicar un ejemplo genérico sobre el testeo de interfaces o tests de integración.

Suponiendo un proyecto ejemplo basado en una compañía dedicada a anunciar productos en la web, que requiere saber donde se han expuesto sus anuncios, cuanta gente los ha visto y cuantos han clicado en ellos, se puede dividir el proyecto en 3 capas que se suelen denominar genéricamente Front layer, Middle layer y Back layer.

La primera se encarga de la interacción con el usuario y de la parte gráfica y visible, la segunda de procesado de datos, cálculos del negocio, validaciones, informes etc. y por último la última capa se encarga de llevar la base de datos.

Si se intenta aplicar tests de integración a este proyecto, el propósito de los mismos será comprobar el correcto flujo de datos entre las diferentes capas y esto significa que se deben cumplir numerosas condiciones y hay que realizar comprobaciones como:

- Los módulos de la capa intermedia sean capaces de recibir y comprender la información que entra por la capa superior.
- Los datos que se procesan y llegan hasta la capa intermedia sean correctos.
- Los datos que llegan a la capa intermedia lo hagan en el formato adecuado dependiendo del módulo al que se dirigen.
- Las peticiones de servicios de todas las capas se entiendan y respondan correctamente dependiendo del servicio requerido.
- Los datos de los informes sean correctos y se generen adecuadamente y además vuelvan a la capa superior también correctamente.
- Los datos almacenados en la base de datos se recojan en el orden y momento correctos y además sean válidos.
- El nivel superior pueda ofrecer servicios correctos al cliente tras ser capaz de recibir información desde la capa inferior o base de datos.

En conclusión, para realizar los tests de integración en un proyecto como el ejemplo basta con asegurar que todas las comunicaciones se producen correctamente y no en cada una cumpla su función por separado, ya que esto se comprobó en su momento mediante tests unitarios.

Si se aplican tests de integración a el proyecto objeto de este TFG la aproximación varía, ya no se concentra tanto en comprobar que las comunicaciones entre módulos se realizan correctamente si no más en que la interfaz entre el Hardware y el FW del equipo es sólida, es decir, que se detectan correctamente todos los cambios de estado en los parámetros del dispositivo, como son la tensión de alimentación o la detección de antenas conectadas.

Los tests de integración en este caso son bastante lentos y pueden llegar a durar horas, por lo que se automatizan para que se ejecuten durante la noche y el dispositivo va cambiando valores internos para ver si el FW reacciona adecuadamente.

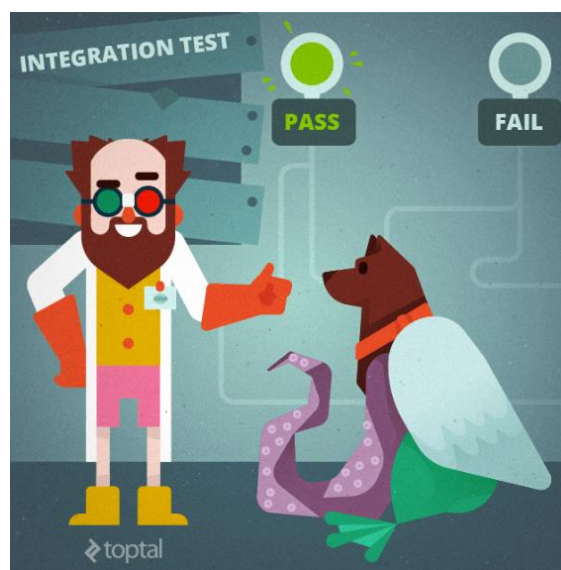


Figura 50:Viñeta satírica tests de integración

En la *Figura 50* aparece una viñeta satírica de el resultado de pasar tests de integración a módulos de distintos desarrolladores, si todo se ha hecho correctamente las partes se pueden llegar a comunicar bien, incluso teniendo cada desarrollador su propio estilo y forma de hacer las cosas.

2.4.3.-Automatización del proceso

Una de las partes más importantes a la hora de mejorar la integración y mantenibilidad de un proyecto es la automatización.



Figura 51: Manual vs automated testing

Si se consigue que el desarrollador solo tenga que preocuparse del código se puede ahorrar mucho tiempo y esfuerzo, por esto, una vez que se han implementado las medidas de seguridad necesarias (tests unitarios y de integración) lo más recomendable es automatizar el proceso de compilación del proyecto.

Esta fase consiste en hacer que el desarrollador solamente tenga que subir su trabajo al final del día a un repositorio determinado y que al día siguiente pueda ver si todo ha ido correctamente o ha aparecido algún error y tiene que revisar lo que hizo el día anterior.

De esta forma cualquier error introducido en el código será detectado bien por los tests unitarios o bien por los de integración y serán mucho más fáciles de solucionar que si se detectan al validar una versión completa que se ha tardado varios meses en sacar ya que para encontrar y solucionar el error es probable que haya que revisar trabajo de muchos días atrás y esto puede resultar increíblemente complicado.

2.4.3.1.-Ventajas de la automatización del proceso de testeo

La inversión temporal necesaria para implementar el proceso de automatización es bastante mayor que si simplemente se realizan los tests de forma manual, sin embargo, está claro que a la larga el ganador en cuestión de ahorro temporal es el que ha elegido la automatización.

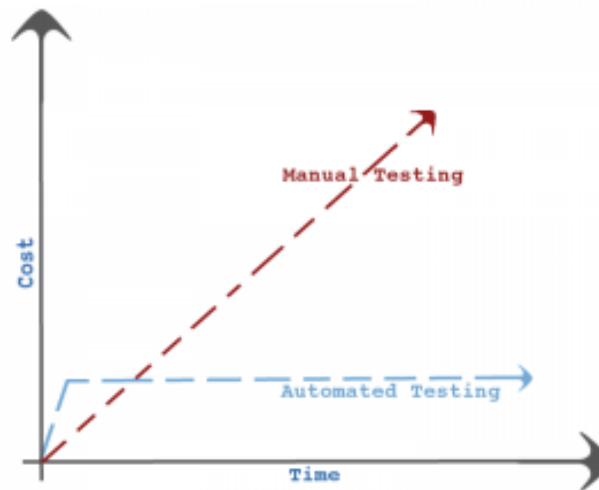


Figura 52: Gráfica coste temporal del proceso de testeo

Otro beneficio que se puede encontrar en la automatización es la posibilidad de realizar los tests desde cualquier parte del mundo y en cualquier momento que se desee, ya que no solo es posible realizar un programa para que ejecute los tests todos los días a una determinada hora sino que además es posible tener un acceso remoto que permita ejecutar los tests en cualquier momento y lugar con, por ejemplo, una aplicación móvil.

También se reduce la cantidad de gente necesaria para una tarea ya que en vez de necesitar que varios desarrolladores hagan tests manualmente a su código basta con tener a una sola persona especializada realizando la tarea de automatización para varios proyectos a la vez.

Los scripts necesarios para la automatización de tests son muy similares entre sí y por lo tanto la reusabilidad es un importante factor a tener en cuenta ya que además suelen ser independientes del sistema operativo.

Las máquinas cometen menos errores que los humanos y esto es especialmente evidente cuando se habla de la ejecución de tests estándar repetitivos que no se pueden saltar. Un humano podría fácilmente terminar cometiendo un error al realizar la misma tarea aburrida tantas veces o saltarse algún paso ocasionalmente, con los tests automáticos esto no es posible.



Figura 53: Man vs computer

Gracias a que los tests no requieren que haya un humano pulsando botones en ningún momento se presenta la posibilidad de realizar tests simultáneos en máquinas distintas de forma que se puedan comparar parámetros y características ya que podemos hacer que en ambas máquinas se ejecute al mismo tiempo el script que automatiza los tests.

Los informes creados tras cada ejecución de tests permiten al resto de desarrolladores conocer el estado del proyecto, es decir, echando un vistazo a los tests realizados y sus resultados pueden saber que funcionalidades se han implementado, que bugs se han encontrado y hasta cuales se han solucionado.

La automatización también permite realizar un tipo de tests que no suelen estar disponibles para el testeo manual, estos son los tests de estrés, que buscan llegar hasta los límites de la aplicación y de la estructura operacional, esto suele requerir cambios muy rápidos y modificaciones de variables muy específicas que solo pueden conseguirse programándolo previamente.

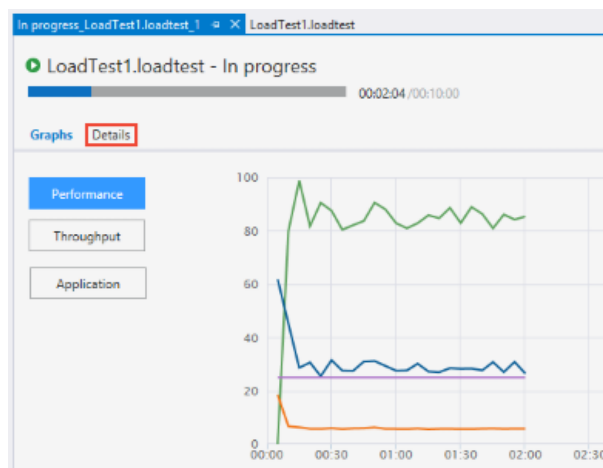


Figura 54: Ejemplo de herramienta para tests de estrés

Aumenta enormemente el volumen de testeo, normalmente los desarrolladores comprueban el funcionamiento correcto de su producto en uno o unos pocos dispositivos de pruebas que se encuentran en sus instalaciones y bajo

condiciones controladas, mediante la automatización de tests esto se puede llevar al siguiente nivel de forma que pulsando un botón se ejecuten tests en todos los dispositivos que se quiera a la vez, aunque lo normal será acotar esto a los dispositivos que hayan salido todavía a cliente.

En definitiva la automatización de tests permitirá obtener un software de mayor calidad al final del día, se ahorrará en tiempos de desarrollo y en despliegue de nuevas versiones y se utilizarán menos recursos para conseguirlo.

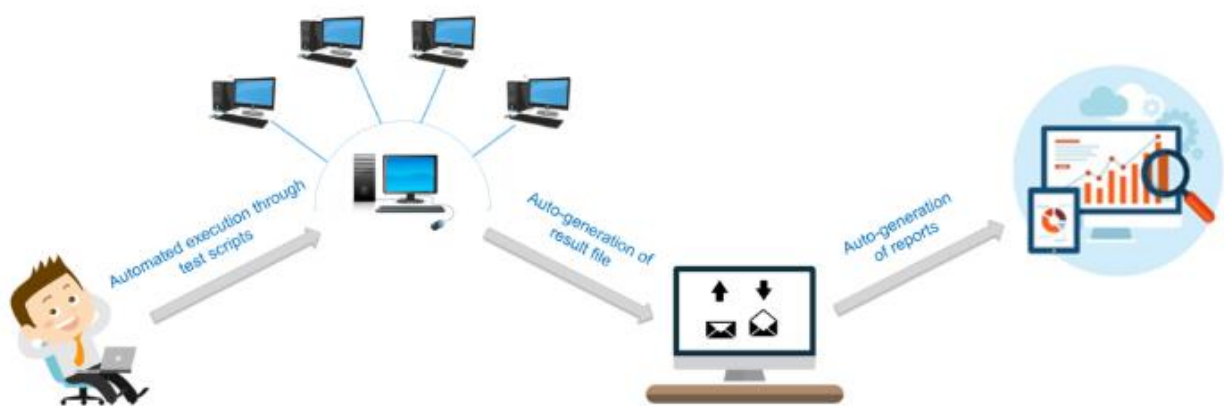


Figura 55: Esquema de automatización en desarrollo de software

Cabe destacar también algunas cosas que la automatización de tests no puede solucionar, por ejemplo, problemas específicos de algún usuario, casos que no se hayan tenido en cuenta en los tests, crashes que requieren un reseteo manual de la aplicación o que no se esté utilizando el dispositivo correctamente.

2.4.3.2.-Repositorios

Los repositorios son, en esencia, un conjunto de archivos y directorios localizados en una base de datos especial que permite llevar un registro de todos los cambios realizados sobre estos archivos en cualquier momento.

Es común hablar de SVN repositories, siendo SVN una abreviatura de Subversion, que es un proyecto open-source que consiste en un sistema de control de versiones, una aplicación ampliamente extendida de este tipo es TortoiseSVN.



Lo más común es crear repositorios que contengan un solo proyecto completo o como mucho un conjunto de proyectos que estén relacionados.

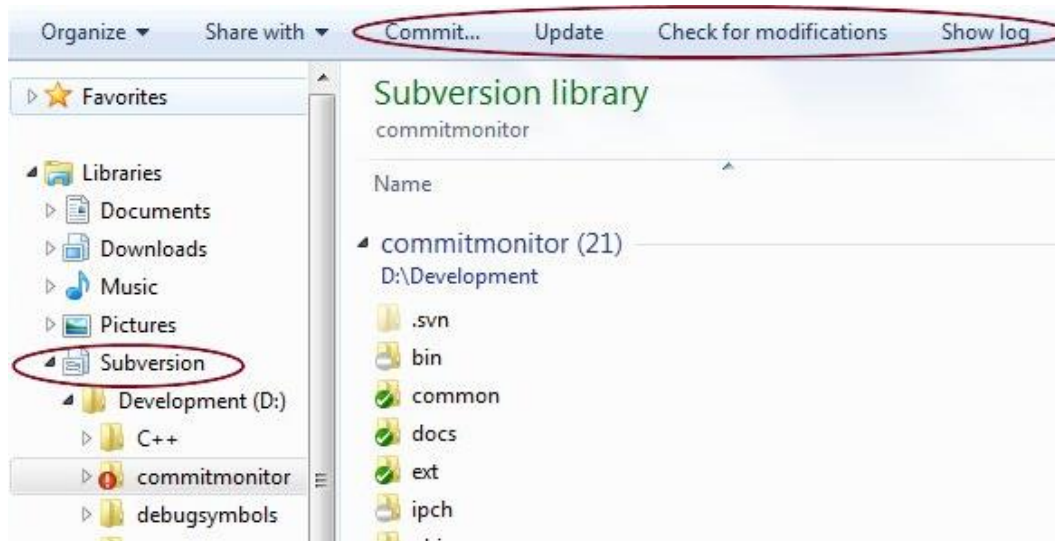


Figura 56: Ejemplo de directorio con SVN

En la *Figura 56* se puede observar la apariencia de un directorio adherido a un repositorio, aparecen con un tick verde todos los archivos que estén actualizados a la última versión que se encuentre en el repositorio y con un tick rojo aquellos de los que existe una nueva versión, probablemente porque algún compañero la ha subido al repositorio recientemente.

La gran ventaja de estos repositorios consiste en el registro que llevan sobre los cambios a los archivos ya que un proyecto en desarrollo puede modificarse numerosas veces y si en algún momento aparece un fallo que no se consigue solucionar fácilmente se puede ver el historial de cambios realizados y, en última instancia, revertir los cambios volviendo a una versión estable del proyecto.

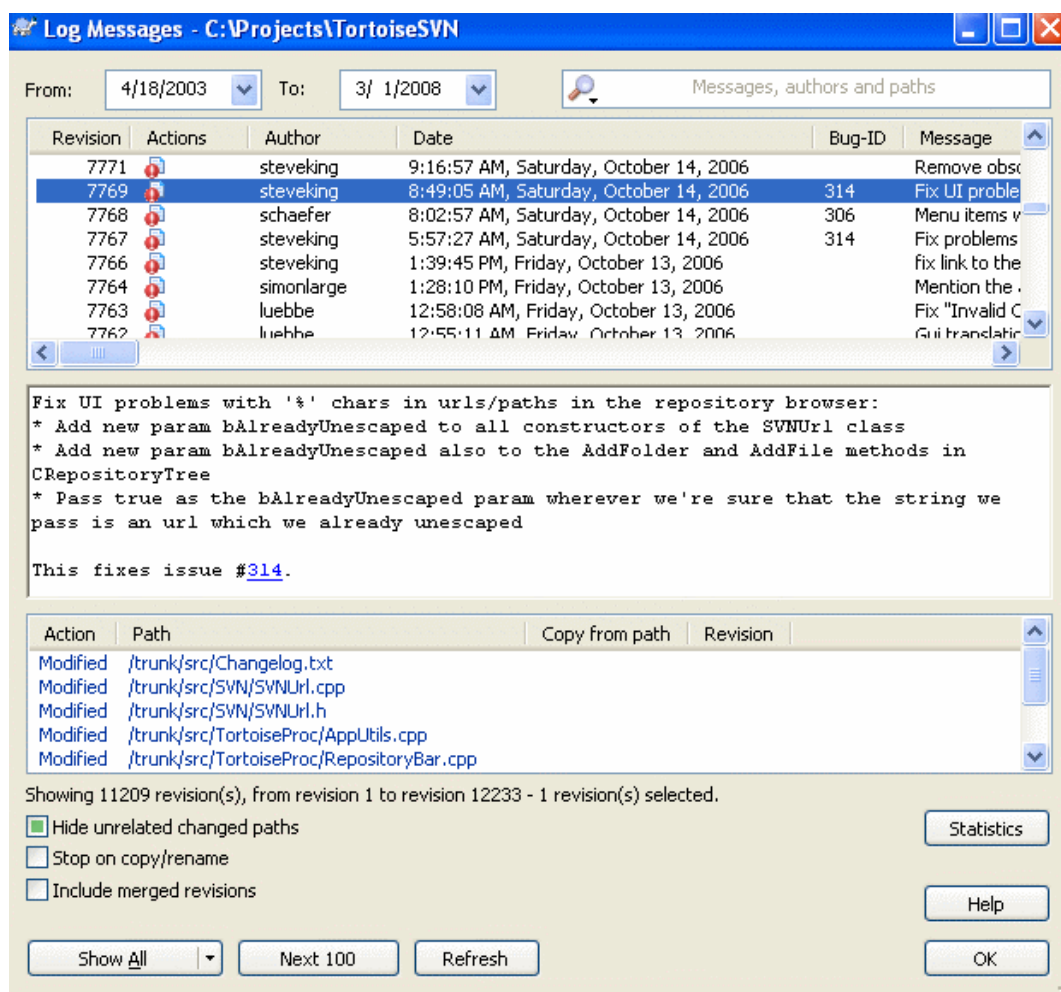


Figura 57: Ejemplo logs de un repositorio

En la *Figura 57* se pueden observar cómo serían los logs de un archivo de un repositorio.

En la parte superior aparecen las revisiones que ha habido, es decir, cada modificación realizada dentro de esa carpeta ya sea modificación de contenido, eliminación o adición de un archivo.

Justo debajo aparece un mensaje que habrá redactado quién haya subido la revisión correspondiente. Suelen ser explicaciones sobre los cambios realizados.

Finalmente abajo aparecen los cambios específicos realizados en los archivos y en las opciones se pueden realizar acciones como actualizar, retroceder a una revisión anterior, crear una copia, juntar dos archivos de repositorios distintos etc.

Otra gran ventaja es que estos repositorios se crean en servidores remotos y por lo tanto es posible que varios desarrolladores accedan a ellos desde sus ordenadores personales, de esta forma cada vez que alguien vaya a trabajar sobre un proyecto solo tiene que actualizarlo en su ordenador cargando los cambios que hayan realizado sus compañeros y así dispondrá de la última versión de ese proyecto para trabajar sobre ella y evitar problemas de compatibilidad o la corrección de un mismo error varias veces.

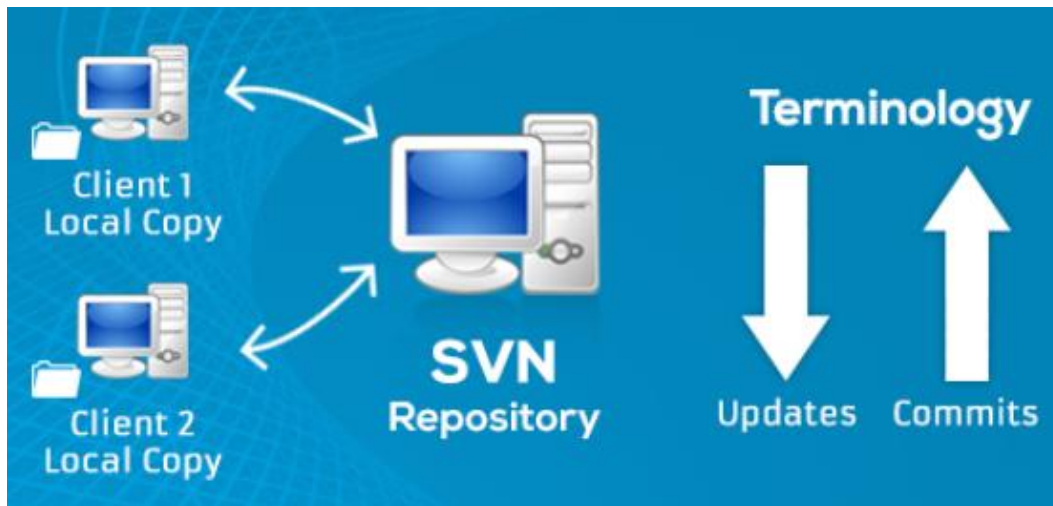


Figura 58: Esquema de uso de un repositorio SVN

Los repositorios guardan una copia del estado del proyecto después de cada cambio para poder volver en caso necesario, pero no guardan los archivos completos si no solo las diferencias entre archivos, ahorrando mucho espacio.

Una ventaja más de los repositorios es que por el mero hecho de existir ya se dispone de una copia de seguridad del proyecto, impidiendo que este se pierda por algún motivo inesperado.

Finalmente, otra característica útil de los repositorios es la capacidad de crear ramas y de volverlas a juntar al repositorio principal, esto permite que durante un proyecto en el que se quiere implementar una nueva funcionalidad alguien cree una rama de ese proyecto, que es básicamente una copia separada del mismo y que trabaje sobre ella, de forma que una vez que haya terminado la funcionalidad y haya comprobado que todo funciona correctamente sin haber estorbado a nadie durante el desarrollo (porque estaba trabajando sobre una rama separada) pueda volver a juntar la rama al proyecto principal.

Branch and Merge

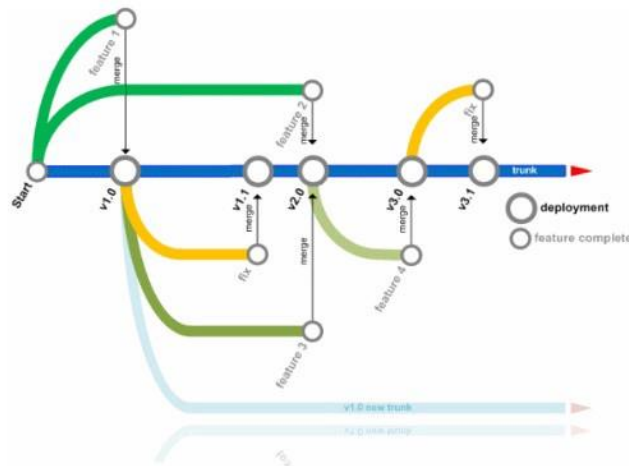


Figura 59: Aplicación de Branch and Merge a un proyecto

En la *Figura 59* se aprecia el proceso de ramificación y unión de un proyecto, siendo la línea azul el Trunk, o parte principal del proyecto, y las Branches o ramas que salen del mismo las copias creadas en el repositorio a partir del Trunk y en las que se trabaja en un funcionalidad o problema en concreto, hasta que se termina y se produce un Merge, que consiste en volver a unir la rama creada con el problema objetivo solucionado al Trunk del proyecto.

2.4.3.3.-Servidores remotos

Tras haber explicado el proceso de automatización las tareas del servidor remoto quedan claras.

Debe estar conectado a un repositorio SVN de forma que pueda acceder en todo momento a la última versión del proyecto que se quiera automatizar y tener fijada una hora a la que empezar a trabajar, es decir, a una determinada hora (normalmente durante la madrugada) se iniciará el proceso de testeo y compilación, que consistirá en:

- Coger la última versión subida al repositorio.
- Pasar unos tests básicos relacionados con el lenguaje de programación (para evitar errores de compilación).
- Pasar los tests extra que se desee sobre la calidad del código (eliminación de warnings, código más entendible etc.).

- Pasar los tests unitarios programados y crear un informe de los errores encontrados, pudiendo ser comunicados directamente mediante e-mail a los desarrolladores del proyecto.
- Pasar los tests de integración, que a veces pueden durar mucho tiempo y que suele necesitar que el ordenador esté conectado al dispositivo embarcado para realizar las pruebas de hardware correspondientes.
- Realizar la compilación con la herramienta específica del proyecto elegida dependiendo del dispositivo embarcado.
- Subir los archivos creados a un nuevo repositorio de Firmware para que puedan ser cargados directamente desde ahí en los dispositivos que se desee.

Un servidor de automatización muy popular es Jenkins, que permite la construcción, despliegue y automatización de cualquier proyecto mediante cientos de plugins.

Es fácil de instalar y ayuda mucho en la tarea de integración continua que se pretende para este proyecto.

A continuación se enumeran algunas ventajas que ofrece el uso de un servidor remoto especializado como Jenkins.



Figura 60: Logo de Jenkins

El envío de comandos repetitivos puede ser sustituido por un solo botón si se juntan y ordenan adecuadamente las tramas que se quieren enviar.

Permiten realizar ciertos tests de integración sobre las unidades funcionales creadas gracias a la ejecución secuencial y en paralelo.

Las salidas de la aplicación (stdout y stderr) son registradas automáticamente por el servidor creando logs de las mismas.

Permite la opción de realizar tests manuales ya que puede que los tests pasen en la máquina local pero no en una general.

Implementan automáticamente tests sobre la calidad del código.

Se registran las modificaciones en las versiones y permiten entregarlas directamente a un entorno de producción si es necesario, además solo realizan el merge a la rama principal si se han pasado exitosamente todos los tests.

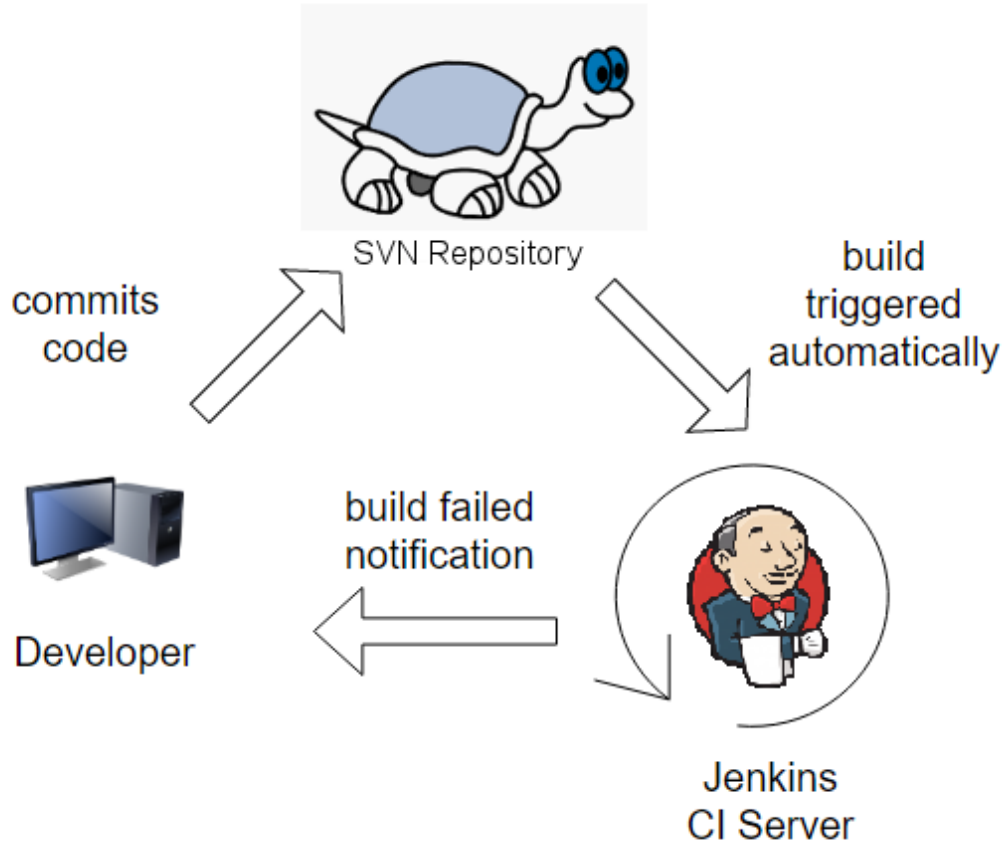


Figura 61: Diagrama de automatización con Jenkins y TortoiseSVN

Como se muestra en la *Figura 61* mediante TortoiseSVN y un servidor Jenkins es posible la automatización del proceso de compilación y testeo de un proyecto.



3.-Conclusiones

A lo largo de este proyecto se han podido repasar las pautas básicas necesaria a la hora de crear un proyecto que se tiene pensado mantener de cara al futuro y de un tamaño lo suficientemente grande como para necesitar de varios desarrolladores trabajando de forma conjunta.

Mediante pautas como las del código limpio y la elección de un entorno de trabajo adecuado y común para todo el equipo de trabajo se facilita mucho el trabajo en equipo, ya que, aunque los desarrolladores trabajen en módulos distintos a la larga vendrá bien que hayan seguido unas mismas normas y hayan trabajado sobre el mismo IDE.

Además, si todos ellos integran tests unitarios y de integración en sus correspondientes módulos y los pasan antes de subir nuevo código al repositorio se ahorrará mucho tiempo buscando responsables y localizando errores.

Lo ideal sería que cada desarrollador introdujese sus propios tests ya que son los que mejor conocen su propio código y están más capacitados para testearlo, sin embargo, es posible añadir otro desarrollador especializado que realice todos los tests y que se encargue también de diseñar el sistema de automatización con servidor remoto, realizando todas las configuraciones necesarias en el mismo e informando a los desarrolladores de en qué repositorio tienen que subir su nuevo código.

Otras veces se organiza el proceso de forma que cada desarrollador trabaje en los módulos que le correspondan, realizando sus tests unitarios sobre los mismos y una vez que los han superado pasen a otra persona que se encargue de realizar los tests de integración para unir los distintos módulos de varios desarrolladores, habiendo todos ellos pasado los tests unitarios previamente.

También es común que cuando los tests de integración estén más orientados a hardware que a unir módulos los realice todos junto con los unitarios el mismo desarrollador y deje que el sistema de automatización realice las pruebas pertinentes durante las noches, dejando conectado el hardware con el FW que se quiere probar conectado y listo para pasar los tests.

En conclusión, es normal que se pasen por alto todos estos aspectos que se han comentado y que mejoran notablemente la vida útil de un proyecto ya sea debido a ignorancia sobre sus beneficios, prisas en las entregas (aunque a la larga saldrían ganando) o pereza del desarrollador ante tener que aprender como trabajar con todas las herramientas de test.



Sin embargo queda clara la necesidad de todo esto en cuanto se habla de un proyecto de un calibre medianamente importante, siempre saldrá mucho más rentable a la larga adoptar unas pautas de código conjuntas, elegir un IDE adecuado, implementar tests unitarios y de integración y por último automatizar el proceso de ejecución de los tests y de compilación del proyecto usando repositorios y servidores remotos.

Este TFG en sí ha sido altamente beneficioso como material formativo ya que aunque no se vayan a realizar todas las recomendaciones aquí expuestas es conveniente al menos conocerlas antes de comenzar un proyecto para así saber que se está haciendo bien o mal y como se podría mejorar.

3.1.-Líneas futuras

Como el alcance de este TFG está bastante limitado se analizarán brevemente y para terminar los posibles temas que podrían ser objeto de investigación de trabajos futuros.

El tema de la integración continua en general está en auge y aparecen cada poco tiempo nuevos métodos o herramientas realmente sorprendentes para la automatización en general.

Estas podrían ser herramientas para la creación sencilla de tests unitarios y de integración o que tengan la complejidad y robustez suficiente como para soportar el peso de un proyecto de programación realmente grande, siendo estas herramientas y su funcionamiento un tema que merecería la pena estudiar.

También podría ser objeto de investigación algún estándar no mencionado en este TFG y que se suele seguir a la hora de programar, como por ejemplo el uso del convenio de codificación de PHP, que trata aspectos como la nomenclatura de ficheros y variables, el uso de comillas, la concatenación, uso de arrays y cadenas etc. incluso hace recomendaciones como el uso de la herramienta Doxygen para la creación de documentación.

También se puede ahondar bastante en el tema de los servidores remotos y sus herramientas, similares a Jenkins, ya que permiten un abanico de posibilidades enorme y llevan el proceso de automatización a otro nivel.

4.-Bibliografía

1.-Introducción y objetivos:

- Chávez, Gisell. (3 julio 2019) “6 Beneficios De La Integración Continua ‘Cl.’” SmartNodus, www.smartnodus.cl/integracion-continua/

2.1.- Análisis inicial:

- Tutorialspoint.com. (3 julio 2019) “Embedded Systems Overview.” Www.tutorialspoint.com, www.tutorialspoint.com/embedded_systems/es_overview.htm.
- Carol Fernandes. (3 julio 2019) “Sistemas Operacionais Embarcados – Carol Fernandes.” Publicado 11 Julio 2018, medium.com <https://medium.com/@carol.fernandes/sistemas-operacionais-embarcados-o-sistema-embarcado-tamb%C3%A9m-chamado-de-sistema-embutido-%C3%A9-um-9e4695923ff3>
- (3 julio 2019) “Importance of Firmware Update: Boost Your Device Performance.” TechNorms, 25 Aug. 2017, www.technorms.com/64686/firmware-update-improve-device-performace.<https://whatis.techtarget.com/definition/PIC-microcontrollers>
- (3 julio 2019) “Firmware.” Wikipedia, Wikimedia Foundation, 27 June 2019, <https://en.wikipedia.org/wiki/Firmware>.
- (3 julio 2019) “What Is Firmware? - Definition from Techopedia.” Techopedia.com, www.techopedia.com/definition/2137/firmware.

2.2.-Preparación del entorno de trabajo

- (3 julio 2019) “Embedded Systems Software Development Tools.” The Engineering Projects, 31 de octubre. 2017, www.theengineeringprojects.com/2016/11/top-10-embedded-systems-software-development-tools.html.
- Moor, Thomas De. (3 julio 2019) “A Few Principles of Clean Code.” X, X-Team - The World's Most Energizing Community for Developers, 22 Mar. 2019, <https://x-team.com/blog/principles-clean-code/>.
- (3 julio 2019) “Single Responsibility Principle.” Wikipedia, Wikimedia Foundation, 1 Feb. 2019, https://en.wikipedia.org/wiki/Single_responsibility_principle.

- (3 julio 2019) “Embedded Solutions - LPWA, 4G LTE, 3G, 2G, Wi-Fi, Bluetooth & GNSS Wireless Modules.” Sierra, www.sierrawireless.com/products-and-solutions/embedded-solutions/.

2.3.-Análisis del proyecto concreto

- (3 julio 2019) “Sistemas De Localización y Gestión De Flotas De Vehículos Por GPS.” MOVILOC®, <https://moviloc.com/>.
- Perdue, Lisa. (3 julio 2019) “The GPS 2019 Week Rollover - What You Need to Know.” Orolia, 18 Dec. 2017, www.orolia.com/resources/blog/lisa-perdue/2017/gps-2019-week-rollover-what-you-need-know.

2.3.3.1.-Gestión y localización

- (3 julio 2019) “What Is GPRS (General Packet Radio Services)? - Definition from WhatIs.com.” SearchMobileComputing, <https://searchmobilecomputing.techtarget.com/definition/GPRS>
- (3 julio 2019) “GSM.” Wikipedia, Wikimedia Foundation, 4 July 2019, <https://en.wikipedia.org/wiki/GSM>
- CellMapper. (3 julio 2019) “Cellular Tower and Signal Map.” CellMapper, www.cellmapper.net/map.
- (3 julio 2019) GALILEO, <https://galileognss.eu/gps-week-number-rollover-april-6-2019/>.
- Korosec, Kirsten. (3 julio 2019) “GPS Rollover Is Today. Here's Why Devices Might Get Wacky.” TechCrunch, TechCrunch, 6 Apr. 2019, https://techcrunch.com/2019/04/06/gps-rollover-is-today-heres-why-devices-might-get-wacky/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=TLAn33hr0AvNqwFFPkXYg.

2.3.3.2.-Seguridad

- (3 julio 2019) “What Is OBD?” OBD Solutions, www.obdsol.com/knowledgebase/on-board-diagnostics/what-is-obd/.
- (3 julio 2019) “OBD-II PIDs.” Wikipedia, Wikimedia Foundation, 25 June 2019, https://en.wikipedia.org/wiki/OBD-II_PIDs.

2.3.3.4.-Periféricos

- (3 julio 2019) “Sistemas Periféricos De Gestión y Localización.” MOVILOC®, <https://moviloc.com/sistemas-perifericos-localizacion/>.
- (3 julio 2019) “How Does a Door Sensor Work?” SafeWise, 20 May 2019, www.safewise.com/home-security-faq/how-door-sensors-work/.

- (3 julio 2019) “What Is an IButton Device?” Maxim Integrated, www.maximintegrated.com/en/products/ibutton/ibuttons/index.cfm.

2.3.3.9.-Consulta detallada de velocidades

- (3 julio 2019) “GPS Forums.” GPS Forums, www.gps-forums.com/.
- (3 julio 2019) “Doppler Effect.” Wikipedia, Wikimedia Foundation, 2 July 2019, https://en.wikipedia.org/wiki/Doppler_effect.

2.3.4.-Modulación del código del proyecto

- (3 julio 2019) “What Is Modular Programming? - Definition from Techopedia.” Techopedia.com, www.techopedia.com/definition/25972/modular-programming.

2.4.1.-Tests unitarios

- (3 julio 2019) “Unit Testing.” Software Testing Fundamentals, 3 Mar. 2018, <http://softwaretestingfundamentals.com/unit-testing/>.
- /@rafaelasguerra. (3 julio 2019) “Basics of Unit Testing - AndroidPub.” Medium, AndroidPub, 31 Jan. 2018, <https://android.jlelse.eu/basics-of-unit-testing-affdd2273310>.
- (3 julio 2019) “White Box Testing.” Software Testing Fundamentals, 3 Mar. 2018, <http://softwaretestingfundamentals.com/white-box-testing/>.
- (3 julio 2019) “Ceedling.” Throw The Switch, www.throwtheswitch.org/ceedling.
- Cooper, Rob CooperRob. (3 julio 2019) “What Is a ‘Stub’?” Stack Overflow, <https://stackoverflow.com/questions/463278/what-is-a-stub>.
- masoud ramezanimasoud (3 julio 2019) “What Is Mocking?” Stack Overflow, <https://stackoverflow.com/questions/2665812/what-is-mocking>.
- (3 julio 2019) “Unity.” Throw The Switch, www.throwtheswitch.org/unity.
- DrAIDrAI “Good Unit Test Examples for Embedded C Developers.” Software Engineering Stack Exchange, <https://softwareengineering.stackexchange.com/questions/79310/good-unit-test-examples-for-embedded-c-developers/80717>.
- (3 julio 2019) “Test-Driven C with Ceedling.” Blog by Carbon Five, 23 Jan. 2016, <https://blog.carbonfive.com/2012/10/14/test-driven-c-with-ceedling/>.



2.4.2.-Tests de integración

- (3 julio 2019) “Integration Testing.” Software Testing Fundamentals, 3 Mar. 2018, <http://softwaretestingfundamentals.com/integration-testing/>.
- (3 julio 2019) “Integration Testing: What Is, Types, Top Down & Bottom Up Example.” Meet Guru99 - Free Training Tutorials & Video for IT Courses, www.guru99.com/integration-testing.html.
- (3 julio 2019) “Robot Framework.” Robot Framework, <https://robotframework.org/>.
- Krishna, et al. (3 julio 2019) “What Is Integration Testing (Tutorial with Integration Testing Example).” Software Testing Help, 2 July 2019, www.softwaretestinghelp.com/what-is-integration-testing/.

2.4.3.-Servidores remotos

- (3 julio 2019) “What Is a SVN Repository?” ProjectHut, 30 Sept. 2013, www.projecthut.com/what-is-svn-repository/.
- (3 julio 2019) “Top 10 Benefits of Automated Testing.” Sauce Labs, <https://saucelabs.com/blog/top-10-benefits-of-automated-testing>.
- (3 julio 2019) “Uses of Jenkins: Top 10 Best Uses Of Jenkins In Real World.” EDUCBA, 16 Apr. 2019, www.educba.com/uses-of-jenkins/.

5.-Anexos

5.1.-ANEXO I: OBD II PIDs

Los OBD-II PIDs (**O**n-**B**oard **D**iagnosics **P**arameter **I**Ds) son los códigos utilizados para extraer información de la OBD del vehículo, que registra datos de todo tipo acerca del estado, movimiento, mantenimiento etc. del vehículo.

Para poder comunicarse con la OBD del vehículo de forma efectiva se le debe mandar un código especial indicando la información que se requiere, dependiendo del este la OBD responderá con unos datos u otros.

En este anexo se recogen los distintos modos, que contendrán sus respectivos PIDs que ofrecen todas las OBDs según el estándar SAE J1979.

Lista de modos

Modo (hex)	Descripción
01	Muestra los parámetros disponibles
02	Muestra los datos almacenados por evento
03	Muestra los códigos de fallas de diagnóstico (D iagnostic T rouble C odes, DTC)
04	Borra los datos almacenados, incluyendo los códigos de fallas (DTC)
05	Resultados de la prueba de monitoreo de sensores de oxígeno (solo aplica a vehículos sin comunicación C ontroller A rea N etwork, CAN)
06	Resultados de la prueba de monitoreo de componentes/sistema (resultados de la prueba de monitoreo de sensores de oxígeno en vehículos con comunicación CAN)
07	Muestra los códigos de fallas (DTC) detectados durante el último ciclo de manejo o el actual
08	Operación de control de los componentes/sistema a bordo
09	Solicitud de información del vehículo
0A	Códigos de fallas (DTC) permanentes (borrados)

Figura 62: Tabla de modos de OBD

Dentro de cada modo se establecen unos PIDs y los fabricantes no están obligados a incluir todos, de hecho pueden incluir algunos propios si lo desean.

La notación en bits que seguirán los mensajes se esquematiza en la *Figura 63* que permitirá utilizar las fórmulas de procesado para obtener el valor real del dato que se busca.

A								B								C								D							
A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0	C7	C6	C5	C4	C3	C2	C1	C0	D7	D6	D5	D4	D3	D2	D1	D0

Figura 63: Formato de las tramas enviadas por la OBD

La cantidad de PIDs es muy elevada y no se van a enumerar todos aquí, la información completa se puede encontrar en:

https://en.wikipedia.org/wiki/OBD-II_PIDs

Aquí se probarán algunas tramas como ejemplo para mostrar el uso de las fórmulas de procesado.

PID (hex)	PID (dec)	Bytes de datos respuesta	Descripción	Min valor	Max valor	Unidades	Fórmula
0C	12	2	RPM motor	0	16383.75	rpm	$\frac{256A + B}{4}$

Figura 64: Tabla con información sobre el PID 0C del modo 01

En la *Figura 64* se muestra la información sobre un PID del servicio 01, más en concreto uno que devuelve las RPM actuales del motor, de forma que si se manda a la OBD el mensaje 01 0C (servicio 01 PID 0C) devolverá una cadena de 16 bits (2 bytes) de la que hay que sacar el valor real usando la fórmula.

Como se ve en la *Figura 64* al llamar al PID 0C la OBD devuelve 2 bytes de datos por lo que se podría esperar una respuesta como por ejemplo la de la *Figura 65*.

A	B
00110110	10110111

Figura 65: Descomposición en bits de la trama de la OBD

De forma que A = 54 y B = 183 y por lo tanto siguiendo la fórmula las RPM del vehículo serían:

$$RPM = \frac{256 * 54 + 183}{4} = 3501.75 \text{ rpm}$$

Este proceso es el que hay que seguir para procesar todos los PIDs que se quiera con sus respectivas formulas, en el FW del dispositivo solo se implementa el procesamiento de los que tengan algún interés ya que implementar un procesado para todos los PIDs sería muy costoso.



Aparte de estos PIDs estandarizados muchas compañías utilizan PIDs propios que no suelen coincidir con los estandarizados y de los que existe muy poca información, además algunos vehículos ni siquiera siguen el estándar de comunicación de 4 bytes y usan el formato CAN (de 11 bytes), lo que puede complicar enormemente los procesados de tramas y en general la comunicación con la OBD, más en concreto si no se sabe cuál de los protocolos está usando cada coche y se necesita hacer un Firmware generalizado.



5.2.-ANEXO II: El problema del Rollover (GPS)

El sistema de posicionamiento global (GPS) tiene una forma de definir el momento exacto del tiempo en el que nos encontramos de una forma peculiar y es por eso que cada cierto tiempo pueden surgir problemas.

Para medir el tiempo y la fecha los GPS utilizan dos variables, una que guarda el número de semanas que han pasado desde el comienzo del GPS (que empezó a contar el 6 de enero de 1980) y otra que guarda el tiempo en segundos que ha transcurrido de esa semana.

Esto significa que el tiempo GPS viene dado por dos números que ofrecen tanta exactitud en la fecha como el sistema al que estamos acostumbrados (dd/mm/aa + hh:mm:ss).

Un ejemplo de equivalencia entre el tiempo en formato GPS y el normal se muestra a continuación (WN = Week Number y WT = Week Time):

$$\text{Formato GPS} \rightarrow \text{WN} = 784 \quad \text{WT} = 28456$$

Esto significa que se trata de la semana 784 desde el 6 de enero de 1980 y que dentro de esa semana han pasado 28456 segundos. Realizando los cálculos correspondientes se puede ver que esta fecha corresponde exactamente con el 15 de enero de 1995, a las 07:54:16.

Se puede usar la siguiente página web para calcular las correspondencias entre formatos automáticamente:

<https://www.labsat.co.uk/index.php/en/gps-time-calculator>

Una vez presentado el método por el cual los GPS miden el tiempo y visto que tienen tanta exactitud como el sistema al que estamos acostumbrados se presenta el problema del rollover.

Los primeros receptores GPS implementaron el número de semana con una variable de 10 bits, lo que quiere decir que su valor máximo es 1024 y que por lo tanto, cuando pasan 1024 semanas desde la inicial la variable vuelve a ponerse a cero, de forma que las tramas que envía el GPS en la semana 1 y en la semana 1025 son indistinguibles, ya que debido al rollover volverá a aparecer como semana 1 en la variable.

Ya se han producido dos rollover desde el 6 de enero de 1980, el primero el 21 de agosto de 1999 y el segundo hace bastante poco, el 6 de abril de 2019.



El rollover en sí no es difícil de solucionar para los receptores GPS ya que basta con establecer una fecha de referencia, como podría ser la fecha de creación de ese dispositivo) y hacer unos cálculos simples para saber en qué periodo de 1024 semanas estamos, en estos momentos nos encontramos ya en el tercer periodo de 1024 semanas, ya que se han producido dos rollovers a lo largo de la historia.

Sin embargo, es fundamental que los sistemas críticos sean supervisados antes y después del rollover para garantizar su correcto funcionamiento y es recomendable que se contacte con el fabricante del receptor GPS para saber si tiene previsto implementar una solución al rollover antes de que se produzca.

El mayor problema viene en los equipos antiguos o descatalogados que ya no reciben actualizaciones de Firmware y que por lo tanto es posible que fallen a partir del rollover, y esto es realmente problemático porque hay muchos dispositivos que utilizan el tiempo del receptor GPS como referencia temporal debido a su gran exactitud.

Actualmente se están implementando nuevas tramas para las comunicaciones GPS con variables de más bits, en el caso del número de semana se ha empezado a utilizar una variable de 13 bits, que permitiría alcanzar las 8192 semanas sin realizar un rollover.

5.3.-ANEXO III: Receptor GPS

Algunos de los dispositivos embarcados utilizados en el proyecto disponen de un receptor GPS que ha sufrido un fallo de Software tras el rollover del 6 de abril de 2019 y que hace que la calidad de la señal de posición se vaya deteriorando con el paso de las horas hasta que queda totalmente inservible a las 3 o 4 horas.

Esto se debe a que el almanaque del receptor no se actualiza correctamente y por lo tanto se pierden los datos necesarios sobre la constelación de satélites.

El almanaque del GPS contiene información vital para el receptor como el estado de todos los satélites de la constelación, datos de la órbita de cada satélite, calibración de relojes, correcciones respecto a la ionosfera etc.

Esto permite que el receptor localice rápidamente los satélites que puede usar para obtener una posición válida.

La solución a este problema es una actualización del FW del receptor, que se podrá comprobar que funciona correctamente con la aplicación del fabricante en la que se pueden visualizar numerosos datos.

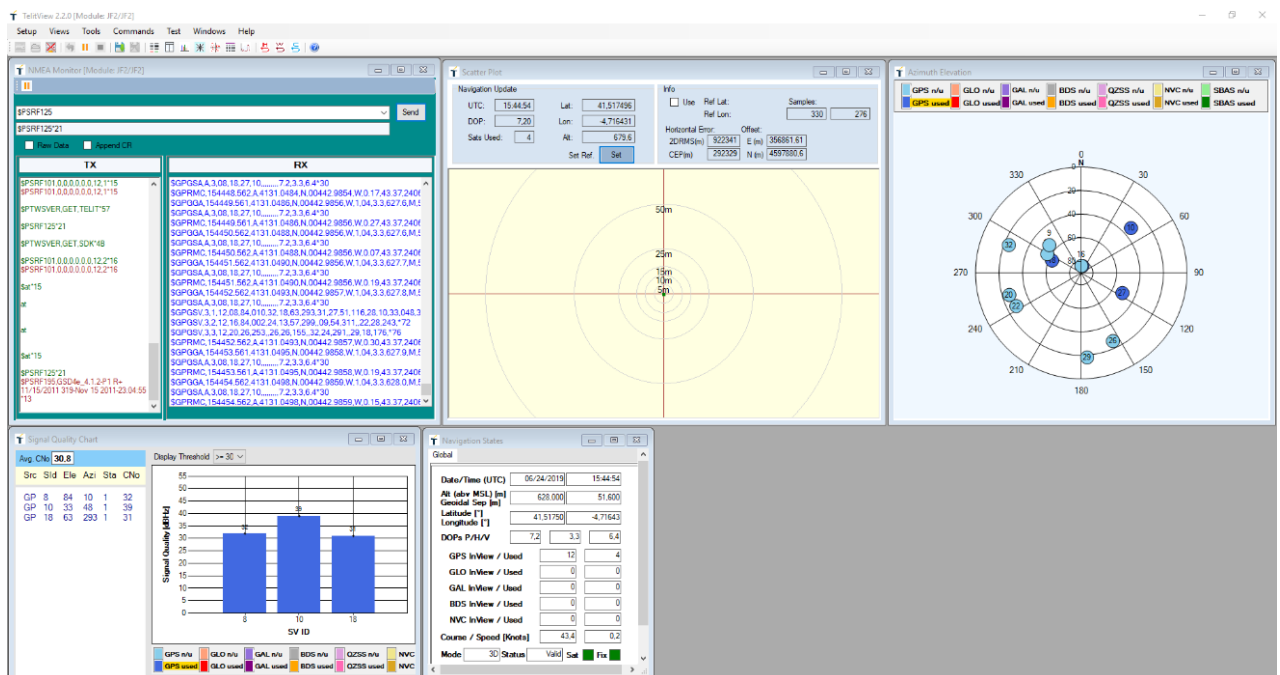


Figura 66: Imagen de herramienta para receptor GPS



En la *Figura 66* aparecen distintas ventanas en las que se pueden ver:

- Las tramas crudas que está recibiendo el receptor (NMEAS) así como una consola que permite mandar comandos al receptor (izquierda arriba).
- Los satélites que está utilizando y sus parámetros como calidad y propietario (GPS, GLONASS, Galileo, BDS etc.) a la izquierda abajo.
- La validez de la señal en un diagrama con el error en metros (centro arriba).
- Más parámetros de la señal de posición (centro abajo).
- Mapa con la posición de los satélites y cuales se están utilizando para calcular la posición (derecha arriba).