



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

---

FACULTAD DE INGENIERÍA

“IMPLEMENTACIÓN DE UN SOFTWARE DE  
COMUNICACIÓN DE DATOS PARA VALIDAR  
AUTOMÁTICAMENTE LOS CASOS DE PRUEBA DE  
LOS MÓDULOS QUE SOPORTAN COMUNICACIÓN  
DE DIAGNÓSTICO ISO 14229”

## **REPORTE DE APLICACIÓN DE CONOCIMIENTOS**

QUE PARA OBTENER EL TÍTULO DE  
**INGENIERO EN COMPUTACIÓN**

PRESENTA:

**DANIEL MEJÍA HERNÁNDEZ**

ASESOR:

**DR. MARCELO ROMERO HUERTAS**

TOLUCA, MÉXICO; FEBRERO DE 2017



UAEM

Universidad Autónoma  
del Estado de México

Facultad de Ingeniería

DEPTO. DE EVALUACIÓN PROFESIONAL

No. Oficio: 004/2017

Ciudad Universitaria, Toluca, Méx. a 10 de febrero del 2017

C. DANIEL MEJÍA HERNÁNDEZ  
PASANTE DE INGENIERÍA EN COMPUTACIÓN  
P R E S E N T E.

En respuesta a su solicitud, a continuación transcribo el tema aprobado por esta Dirección, que propuso el **DR. MARCELO ROMERO HUERTAS**, con el fin de que lo desarrolle en la modalidad de **REPORTE DE APLICACIÓN DE CONOCIMIENTOS**, le informo que se autoriza la impresión de su trabajo para presentar su Evaluación Profesional.

*"IMPLEMENTACIÓN DE UN SOFTWARE DE COMUNICACIÓN DE DATOS PARA VALIDAR AUTOMÁTICAMENTE LOS CASOS DE PRUEBA DE LOS MÓDULOS QUE SOPORTAN COMUNICACIÓN DE DIAGNÓSTICO ISO 14229".*

	RESUMEN
	INTRODUCCIÓN
CAPÍTULO 1	ANTECEDENTES
CAPÍTULO 2	DISEÑO DE IMPLEMENTACIÓN
CAPÍTULO 3	PRUEBAS DE ACEPTACIÓN DE LA SOLUCIÓN
	CONCLUSIONES Y TRABAJOS A FUTURO
	ANEXOS
	REFERENCIAS

Ruego a usted tomar nota de que en cumplimiento a lo especificado por la Ley de Profesiones, deberá prestar Servicio Social durante un tiempo mínimo de seis meses, como requisito indispensable para sustentar su Evaluación Profesional.

Asimismo, para la elaboración del **REPORTE DE APLICACIÓN DE CONOCIMIENTOS** y demás trámites, deberá sujetarse a la reglamentación respectiva de esta Universidad.

ATENTAMENTE  
PATRIA, CIENCIA Y TRABAJO  
"2017, Año Del Centenario de la Promulgación de la Constitución Política de los Estados Unidos Mexicanos".  
ESTADOS UNIDOS MEXICANOS  
FACULTAD DE INGENIERÍA  
U. A. E. M.  
M. EN I. RAÚL VERA NOGUEZ  
DIRECTOR

\*\*/Saha

Cerro de Coatepec S/N, Ciudad Universitaria; Toluca México  
Tel. (722) 2-14-08-55 / 2-15-13-51

www.uaem.mx



Toluca, México a 14 de Febrero de 2017

Espacio Académico, Subdirector Académico, Coordinador de Programa  
Presente

**Carta de excepción para publicación en el RI**

**Declaración de autoría original**


Quien firma al calce declara que: soy autor intelectual del original titulado "IMPLEMENTACIÓN DE UN SOFTWARE DE COMUNICACIÓN DE DATOS PARA VALIDAR AUTOMÁTICAMENTE LOS CASOS DE PRUEBA DE LOS MÓDULOS QUE SOPORTAN COMUNICACIÓN DE DIAGNÓSTICO ISO 14229" y que estoy de acuerdo con la totalidad de su contenido.

Que el/la REPORTE DE APLICACIÓN DE CONOCIMIENTOS presentado es original y se encuentra en proceso de dictaminación o embargo en:

del cual se adjunta captura de pantalla como evidencia y en su caso número de folio.

Quien abajo firma solicita que el trabajo titulado "IMPLEMENTACIÓN DE UN SOFTWARE DE COMUNICACIÓN DE DATOS PARA VALIDAR AUTOMÁTICAMENTE LOS CASOS DE PRUEBA DE LOS MÓDULOS QUE SOPORTAN COMUNICACIÓN DE DIAGNÓSTICO ISO 14229", sea incluido con fines de evidencia y disseminación únicamente con portada, capitulado o índice, resumen y datos de contacto del autor, en alguna de las colecciones del Repositorio Institucional. Así mismo permito que la Oficina de Conocimiento Abierto realice lo propio para la preservación y difusión de la obra.

Sin otro particular.

  
DANIEL RESTÁ HERNÁNDEZ  
Nombre y firma  
No de cuenta: 0715218

Conozco y acepto los términos de privacidad de la Universidad Autónoma del Estado de México  
[http://web.uaemex.mx/avisos/Aviso\\_Privacidad.pdf](http://web.uaemex.mx/avisos/Aviso_Privacidad.pdf)

Toluca, México a 14 de Febrero de 2014

**Hoja de datos del autor**

Nombre: DANIEL MESÍA HERNÁNDEZ

Número de cuenta: 0715218

Grado académico obtenido: LICENCIATURA

Programa educativo de procedencia: INGENIERÍA EN COMPUTACIÓN

Correo electrónico: danielmh-msn.com



DANIEL MESÍA HERNÁNDEZ  
Nombre y firma

Esta información es recabada con fines administrativos para el proceso de titulación del Espacio Académico que suscribe.

Conozco y acepto los términos de privacidad de la Universidad Autónoma del Estado de México  
[http://web.uaemex.mx/avisos/Aviso\\_Privacidad.pdf](http://web.uaemex.mx/avisos/Aviso_Privacidad.pdf)

## ***Agradecimientos***

*A dios:*

*Por haberme permitido cumplir con una de mis metas más importantes de mi vida, titularme.*

*A mi asesor:*

*Dr. Marcelo Romero Huerta, por brindarme todo el apoyo y confianza para culminar este reporte.*

*A mi mentor Vic y compañeros de trabajo:*

*Por enseñarme los conocimientos básicos necesarios para poder realizar este proyecto. Además, gracias, porque siempre me han estrechado la mano cuando más lo he necesitado y nunca me han dejado solo.*

*A mi familia:*

*Porque siempre han estado detrás de mi apoyándome en todas las circunstancias de la vida. No puedo pedirle más a dios de lo que ya me ha regalado, teniéndolos a mi lado.*

*A mi novia:*

*Gaby, por tu apoyo, paciencia y comprensión incondicional en todo el desarrollo de este reporte. Gracias a ti he crecido culturalmente de una forma increíble. Siempre estaré agradecido contigo.*

*A mi mamá y a mi hermana:*

*Ustedes mis mujercitas (que son mi vida), porque juntos hemos pasado momentos únicos (buenos y malos), pero ahora sé que juntos siempre saldremos adelante y podremos tener momentos que nos llenen el alma de alegría.*

*A ti papá:*

*Que junto con mi mamá plantaste una semilla la cual el día de hoy puedes ver crecer. Gracias por todo lo que me enseñaste y diste por mí. Sé que cuando te necesite simplemente miraré al cielo y sabré que tú me guiarás hacía el camino correcto.*

## RESUMEN

El presente documento describe el trabajo realizado durante mi estancia en el Centro Regional de Ingeniería de General Motors ubicado en la ciudad de Toluca donde, como resultado, fue implementado un nuevo proceso automatizado para la validación de casos de prueba de los módulos que soportan la comunicación de diagnóstico en los automóviles *ISO 14229*. Todo esto posible en virtud de la integración de un software de comunicación de datos con un complemento unificador de formatos de información que simplifica el proceso de validación, reduciendo significativamente el tiempo destinado para dicha actividad.

En esencia, este reporte incluye conceptos que son necesarios para el entendimiento apropiado del ambiente de trabajo (protocolos y herramientas de comunicación) sobre el cual fue desarrollado el proyecto. Asimismo, se detalla la problemática que tenía anteriormente el equipo de comunicación de diagnóstico de la misma compañía y se presentan las alternativas de solución con base en los requerimientos demandados por el cliente.

También, se documenta el uso de una metodología de desarrollo de *Design For Six Sigma* en conjunto con una metodología clásica que permitieron la creación y desarrollo de sistemas *híbridos*. Es así como el uso de *In-house software* y del *test procedure generator* contribuyeron al cumplimiento de los requerimientos en lapsos de tiempo cortos y a bajos costos.

La herramienta de *software test procedure generator* se caracteriza por estar construida con un lenguaje de programación *Java* por lo que puede ser utilizada en cualquier plataforma (Windows, IOS, Linux), además de proporcionar una ejecución y empleo sencillo, unificando de esta forma las fuentes de información del proceso de validación del software de comunicación de diagnóstico embebido en los controladores de un vehículo.

Las pruebas de aceptación, realizadas por el cliente, así como la implantación del nuevo proceso, resaltan la efectividad estimada en la ejecución de pruebas de validación del software de diagnóstico, pues existe una reducción de tiempo considerable de hasta un 84.2% con respecto al proceso que ejecutaban anteriormente.

# Índice

---

<b>INTRODUCCIÓN .....</b>	<b>- 9 -</b>
<b>CAPÍTULO I. ANTECEDENTES.....</b>	<b>- 14 -</b>
1.1 UNIDAD DE CONTROL ELECTRÓNICA .....	- 14 -
1.1.1 Arquitectura de un ECU.....	- 16 -
1.2 ESTÁNDARES DE COMUNICACIÓN .....	- 18 -
1.2.1 ISO 11898-1:2003.....	- 20 -
1.2.2 Red de Controladores de Área (CAN).....	- 20 -
1.2.3 ISO 15765-2:2011.....	- 27 -
1.2.4 Protocolo de control.....	- 27 -
1.2.5 Protocolo de transporte .....	- 32 -
1.2.6 ISO 14229-1:2013.....	- 35 -
1.2.7 Protocolo de aplicación UDS.....	- 36 -
1.3 INTERFAZ DE COMUNICACIÓN DE DIAGNÓSTICO.....	- 41 -
1.3.1 Assembly Line Data Link.....	- 41 -
1.3.2 NeoVi.....	- 42 -
<b>CAPÍTULO II. DISEÑO E IMPLEMENTACIÓN.....</b>	<b>- 44 -</b>
2.1 PROBLEMÁTICA ACTUAL.....	- 44 -
2.1.1 Identificación del problema.....	- 44 -
2.1.2 Análisis del proceso actual.....	- 47 -
2.2 REQUERIMIENTOS.....	- 61 -
2.2.1 Definición de requerimientos.....	- 61 -
2.2.2 Comparación de software actual contra requerimientos .....	- 64 -
2.2.3 Análisis y modelado de requerimientos .....	- 65 -
2.3 ANÁLISIS DE ALTERNATIVAS DE SOLUCIÓN .....	- 73 -
2.3.1 Vehicle Spy – Function blocks.....	- 74 -
2.3.2 Vector Cantech - CANoe DiVa .....	- 75 -
2.3.3 Herramienta de ejecución automática de pruebas (In-house software).....	- 77 -
2.3.4 Comparación del software existente .....	- 81 -
2.4 DESARROLLO DEL CONCEPTO .....	- 87 -
2.4.1 Diseño.....	- 87 -
2.4.2 Implementación .....	- 99 -
2.5 PRUEBAS.....	- 114 -
2.5.1 Pruebas de caja blanca.....	- 114 -
2.5.2 Pruebas de integración.....	- 125 -
<b>CAPÍTULO III. PRUEBAS DE ACEPTACIÓN DE LA SOLUCIÓN.....</b>	<b>- 128 -</b>
3.1 PRUEBAS DE VALIDACIÓN CON EL USUARIO FINAL Y OPTIMIZACIÓN.....	- 128 -
3.2 IMPLANTACIÓN DEL NUEVO PROCESO .....	- 132 -
<b>CONCLUSIONES Y TRABAJO FUTURO .....</b>	<b>- 135 -</b>
CONCLUSIONES.....	- 135 -
TRABAJO FUTURO.....	- 136 -
COMPETENCIAS Y APRENDIZAJES ADQUIRIDOS .....	- 137 -
<b>ANEXO A – FORMATO DE VALIDACIÓN DE REQUERIMIENTOS.....</b>	<b>- 139 -</b>
<b>GLOSARIO .....</b>	<b>- 142 -</b>

<b>REFERENCIAS .....</b>	<b>- 147 -</b>
<b>ÍNDICE DE FIGURAS.....</b>	<b>- 150 -</b>
<b>ÍNDICE DE TABLAS .....</b>	<b>- 152 -</b>



# Introducción

---

Durante las últimas décadas, la velocidad de los avances tecnológicos en la industria automotriz ha ido aumentando a un ritmo vertiginoso, aunque muchas veces imperceptible. El impacto que estos avances han tenido sobre los vehículos automotores se puede ilustrar claramente en un consumo eficiente de combustible con base en la inyección electrónica, un mejor cambio de velocidades para transmisiones automáticas e incluso en tecnología de info-entretenimiento, brindando así un mejor confort para el cliente final. Todo esto es posible gracias a los dispositivos electrónicos de control del vehículo, conocidos como *Unidades de Control Electrónica (ECUs*, por sus siglas en inglés *Electronic Control Unit*).

El sistema de comunicación de un vehículo automotor está compuesto de *ECUs* permitiendo, mediante sensores y actuadores, conocer el estado actual del automóvil durante su uso. La interacción de dichas unidades de control se realiza mediante canales de comunicación de datos que permiten la transmisión y recepción de información con la finalidad de obtener un mejor desempeño en el funcionamiento del vehículo, así como detección de las fallas que se generen.

Debido a los estándares de seguridad con los que cuenta la industria automotriz para el desarrollo y producción de vehículos automotores, se tiene un riguroso y robusto esquema de validación en todos y cada uno de los elementos que componen a un vehículo, permitiendo, de esta forma, probar la funcionalidad de dichos componentes que se estén validando, así como también la detección de errores de desarrollo. Un ejemplo de estas validaciones es ilustrado claramente, en la comprobación del funcionamiento de las *ECUs*, es decir, verificar que el software embebido cumpla con una funcionalidad específica bajo estándares de desarrollo específicos.

Actualmente, existe una empresa líder en el sector automotriz de la ciudad de Toluca que realiza dichas pruebas de validación. Este Centro Regional de Ingeniería de Toluca es el único lugar, dentro de la misma empresa a nivel Latinoamérica, donde se realizan este tipo de pruebas.

El presente proyecto de aplicación de conocimientos muestra la implementación de un software de comunicación de datos que permite validar de forma automática y mediante casos de prueba la funcionalidad del software contenido en *ECUs* que soportan la comunicación de diagnóstico *ISO 14229*.

Este proyecto de aplicación de conocimientos fue desarrollado por el sustentante del reporte como parte de la estancia como becario dentro de la empresa. La toma de decisiones para el cumplimiento del objetivo y requerimientos fue realizada con base en evidencia mostrada al cliente para que, en conjunto, se determinara la mejor solución para su implementación y verificación (validación y pruebas).

Para el desarrollo de este proyecto se utiliza una metodología llamada *Design For Six Sigma (DFSS)* la cual permitió el análisis del problema, así como el desarrollo y validación de las soluciones. Dentro de una de las fases de la metodología de *DFSS* se utilizó una metodología en cascada, creando de esta forma un híbrido de metodologías que permitió robustecer el desarrollo del proyecto con el objetivo de cumplir con los requerimientos del cliente.

Como resultado de este proyecto, se implementó un nuevo proceso automatizado que incluye dos herramientas de software que, en conjunto, permiten realizar la transmisión y recepción de tramas, comparando la respuesta recibida de la *ECU* con una respuesta esperada previamente analizada y determinada en documentos con formato *XSLX* referentes a libros de trabajo de *Excel*.

La primera herramienta de *software* es un desarrollo realizado por la empresa automotriz, la cual permite realizar la transmisión y recepción de mensajes de diagnóstico *UDS* (por sus siglas en inglés *Unified Diagnostic Services*) en un bus utilizando el protocolo *CAN* (por sus siglas en inglés *Controller Area Network*). Esta herramienta permite comparar la respuesta que transmite la *ECU*, con base en un requerimiento (mensaje) enviado por el *tester* (especialista), contra una respuesta esperada previamente diseñada por el ingeniero de validación.

La segunda herramienta de *software* permite generar archivos a partir de documentos que el ingeniero de validación ha desarrollado con la finalidad de probar la funcionalidad de comunicación de diagnóstico embebido en las *ECUs*. La conversión de estos formatos de archivos (*XLSX* a *TXT*) fue realizada con la finalidad de evitar el re-trabajo que los ingenieros de validación tienen al transcribir los mensajes que ya están establecidos en los documentos hacia las herramientas de software de comunicación.

La implementación de estas dos herramientas, permite reducir el tiempo de ejecución de los casos de prueba y evitar errores por la incorrecta transcripción de las tramas, contribuyendo en el uso eficiente de recursos como es el tiempo asignado en los bancos de prueba disponibles, así como en la entrega a tiempo de resultados de las

validaciones del software de las *ECUs*, proporcionando información pertinente a los directivos del área de Diagnóstico.

### **Objetivo general del reporte de aplicación de conocimientos**

Implementar un *software* de comunicación de datos para realizar la validación automática de los casos de prueba de las *Unidades de Control Electrónicas* que soportan comunicación de diagnóstico *ISO 14229*.

### **Alcances y limitaciones**

Dentro de la empresa donde se desarrolló el presente proyecto de titulación, existen varios grupos encargados de la validación del software embebido de las *ECUs*. Específicamente se colaboró con el área de *Comunicación de Diagnóstico*, debido a la disponibilidad de especificaciones para generar *procedimientos de prueba (test procedures)*, así como del número de *casos de prueba* desarrollados.

El alcance de este proyecto fue determinado mediante los requerimientos y necesidades del cliente resumidos en la transmisión y recepción de mensajes (tramas o mensajes de diagnóstico *UDS*) sobre un bus con protocolo *CAN* extendido (tramas con *encabezados* de 29 bits).

La plataforma operativa de la empresa es homogénea, por lo tanto, las aplicaciones implementadas tuvieron que ser diseñadas para un sistema operativo específico. Debido a las restricciones, estándares y políticas de la empresa, no es posible implementar dichas herramientas en un ambiente de servidores o web, por lo que se determinó implementar una aplicación de escritorio.

Dentro del análisis de componentes, la empresa requirió un diseño para una única interfaz de comunicación serial entre el *ingeniero* y los *bancos de prueba*, denominada *NeoVi*.

### **Directriz del reporte de aplicación de conocimientos**

En el presente reporte se utilizó terminología técnica especializada en el idioma inglés, con el fin de evitar ambigüedades en la traducción al idioma español, por lo que se especifican en el glosario los términos de origen anglosajón.

Los conocimientos empleados para la implementación del *software* de este reporte están relacionados al área de redes en vehículos automotores, así como también en el diseño de software de aplicación.

Dentro del área de redes se enfatiza el uso del estándar *ISO 11898-1:2003 Road vehicles – Controller Area Network (CAN)*, del estándar *ISO 15765-2:2011 Road vehicles – Diagnostic Communication over Controller Area Network (DoCAN)*, y del estándar *ISO 14229-1:2013 Road vehicles – Unified Diagnostic Services (UDS)*. Estos estándares están considerados dentro del contexto de las siete capas del modelo *OSI (Open System Interconnection)*.

Para los estándares antes mencionados se toma en cuenta la creación de tramas en los protocolos de transporte y de aplicación.

Para el área de *software de aplicación*, el presente proyecto tiene relación concreta con el desarrollo de aplicaciones de escritorio, en la cual se realiza la lectura y escritura de archivos en diferentes formatos, por ejemplo *XSLX* y *TXT*.

Para facilitar la lectura y comprensión de este documento, se ha cuidado el uso de terminología especializada. Sin embargo, es recomendable que el lector tenga conocimientos en protocolos de comunicación de datos, así como también conocimientos en programación de aplicaciones de escritorio y conocimientos mínimos en cuanto a la arquitectura de los controladores electrónicos del ambiente automotriz. Por lo tanto, se pretende que este trabajo de titulación sea de utilidad para aquellas personas interesadas en la creación de herramientas que permitan la validación del software embebido en los controladores que soporten el estándar *ISO 14229-1:2013*.

Por motivos de confidencialidad de la empresa, no se describen detalles técnicos ni se revelan identidades que puedan poner en riesgo la integridad de la información y por consiguiente el posicionamiento de la empresa contra sus competidores en el mercado del mismo giro. Por tal motivo, el presente reporte de aplicación de conocimientos tiene únicamente una naturaleza descriptiva en algunas secciones.

## Organización del documento

El contenido de este reporte de aplicación de conocimientos está estructurado de la siguiente forma:

*El Capítulo I. Antecedentes*, muestra una descripción detallada de las herramientas, protocolos y recursos necesarios para comprender el ambiente sobre el cual fueron implementadas las herramientas de *software*.

*El Capítulo II. Diseño e implementación*, presenta las condiciones iniciales sobre las cuales se desarrolló este proyecto; así como las diferentes propuestas de solución y su evaluación con base en los requerimientos planteados por el cliente, definiendo la viabilidad de la implementación para cada propuesta mostrada y determinando una ganadora. Por lo que se refiere a la implementación, se proporciona una descripción general del sistema desarrollado y de las pruebas realizadas.

*El Capítulo III. Pruebas de aceptación*, documenta los resultados de las pruebas de aceptación realizadas por los ingenieros del área de comunicación de diagnóstico así como las mejoras implementadas a partir de estos resultados. También, se especifica la implantación del nuevo proceso de automatización y los recursos entregados para el mantenimiento de las herramientas de *software*.

Finalmente, en las conclusiones se mencionan los resultados y beneficios que se obtuvieron con el desarrollo de este proyecto. Además, se puntualizan los hallazgos relevantes obtenidos en la implementación de la solución documentada en este reporte de titulación, seguido de un glosario de términos y la lista de referencias consultadas para el desarrollo de este reporte. Por último, se despliega el anexo A para un mejor entendimiento de puntos específicos del reporte.

# Capítulo I. Antecedentes

---

*En este capítulo se muestra una descripción detallada de las herramientas, protocolos y recursos necesarios para comprender el ambiente sobre el cual fueron implementadas las herramientas de software.*

## 1.1 Unidad de Control Electrónica

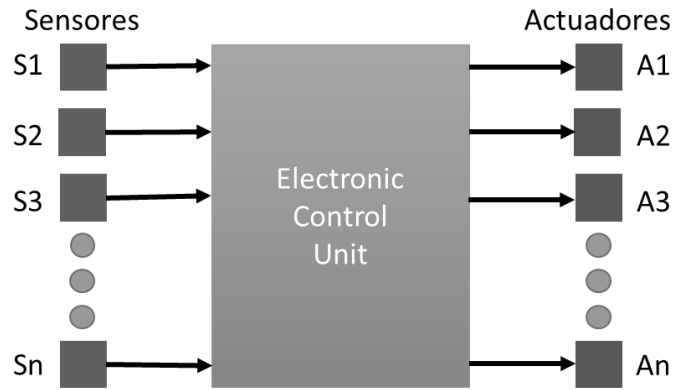
Hoy en día, aquellas personas que adquieren un vehículo automotor esperan que éste les brinde un funcionamiento de excelencia, es decir, un alto rendimiento, la mayor comodidad posible y tecnología de punta.

Las empresas líderes en el sector automotriz que quieren competir en el mercado, no se pueden quedar atrás, por lo que en las últimas décadas han utilizado la tecnología como un arma a su favor. Este es el caso particular del desarrollo de componentes eléctricos/electrónicos, los cuales permiten, mediante diferentes funcionalidades, ampliar las fronteras de las aplicaciones tradicionales (funcionamiento mecánico vehicular). En consecuencia, el desarrollo de software que permita manipular, administrar y controlar dichos componentes es una labor clave para la mejora y perfeccionamiento de los vehículos automotores que se producen actualmente.

Antes de que surgieran las *Unidades de Control Electrónicas (ECU*, por sus siglas en inglés *Electronic Control Unit*), comúnmente conocidas como computadoras de los vehículos, toda la funcionalidad del vehículo se ejecutaba de manera eléctrico/mecánica trayendo consigo inconvenientes reflejados en el bajo rendimiento y consumos excesivos de combustible, es decir, el automóvil realizaba la funcionalidad esperada, pero lo hacía de tal manera que requería una gran cantidad de recursos.

Los sistemas eléctrico-electrónicos que hoy en día se encuentran integrados en algunos vehículos automotores, tienen la capacidad de leer datos alrededor de su ambiente mediante sensores, los cuales, recaban toda la información de las entidades (otras computadoras interconectadas en la misma red del vehículo) o del entorno (condiciones de algún elemento del auto que se debe de estudiar, así como momentos

o situaciones) y que son necesarios registrar para poder tomar acciones determinadas a partir de los datos analizados (Figura 1).



**Figura 1** - Black box de una Unidad de Control Electrónica.

Una *Unidad de Control Electrónica* es un dispositivo que controla uno o más sistemas eléctricos en un vehículo, dando instrucciones a los actuadores de lo que deben de hacer (tiempos de ejecución y sincronización, así como las operaciones a realizar) a partir de datos recabados por los sensores. Estas *ECUs* están diseñadas para soportar condiciones extremas, como por ejemplo cambios súbitos e inesperados de temperatura, vibraciones, humedad y cambios de voltaje.

Dependiendo de la configuración de la *ECU* así como de la importancia de los datos, el monitoreo podría ejecutarse en ciclos de milisegundos (tiempo real), con el fin de obtener información lo más precisa posible, pues la seguridad e integridad de las personas o clientes finales depende del correcto funcionamiento de todas las partes que componen a un vehículo, y más si éstas se encuentran ligadas a temáticas de motor, transmisión, seguridad, etc.

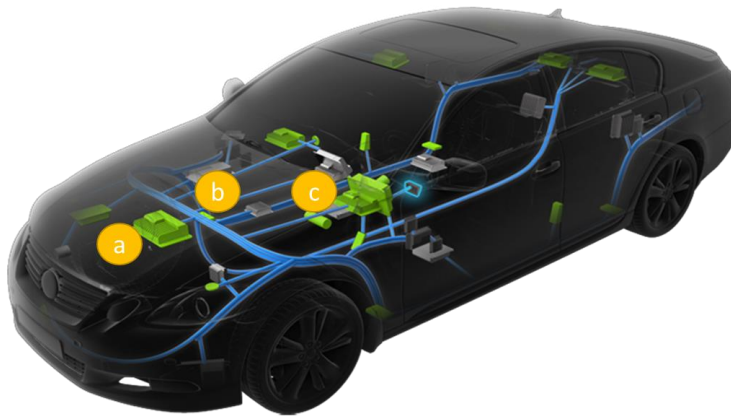
En la actualidad, las condiciones socio-económicas y ambientales de algunos países, como lo son Estados Unidos y México entre muchos otros, se encuentran influenciadas mediante leyes de protección ambiental o altos precios de combustibles, por lo tanto, es necesario generar un óptimo desempeño en los vehículos automotores para que cumplan estas normas y además brinden mayor eficiencia en cuanto al manejo de recursos.

Hoy en día, dentro de la industria automotriz los vehículos están compuestos por más de 80 unidades de control diferentes, todos comunicados entre sí para que, en conjunto, se obtenga un mejor desempeño del vehículo.

Como ya se ha mencionado previamente, la *ECU* se alimenta de diferentes señales captadas por los sensores, las cuales son transformadas para ser utilizadas en la realización de los cálculos que determinarán las señales de salida que interpretarán los actuadores.

Los algoritmos que permiten administrar los diferentes procesos o señales de entrada se encuentran almacenados en una memoria especial y son ejecutados por un microcontrolador.

Las *ECUs* más conocidas que componen un vehículo son: a) *ECM* (*Engine Control Module*), encargado de administrar componentes del motor como la mezcla de combustible, el tiempo de ignición, control de emisiones, etc.; b) *TCM* (*Transmission Control Module*), para vehículos con transmisiones automáticas, encargado de administrar la presión del aceite y generando así los cambios de velocidades; c) *BCM* (*Body Control Mode*), encargado de cubrir la parte de confort y seguridad como pueden ser las ventanas y seguros eléctricos, las luces interiores y exteriores, así como los accesorios: el estéreo, calefacción, aire acondicionado, entre otros. (Figura 2).



**Figura 2** - *ECUs* que componen a un vehículo (ChipsAway, 2016).

### 1.1.1 Arquitectura de un ECU

Los elementos a nivel hardware que componen a un *ECU* y que permiten realizar toda la interacción entre la computadora y su exterior son (Figura 3):



*Unidad de procesamiento central:* de forma similar a una computadora de escritorio o una laptop. Un vehículo contiene una unidad de control principal así como una unidad aritmética y lógica (ALU).

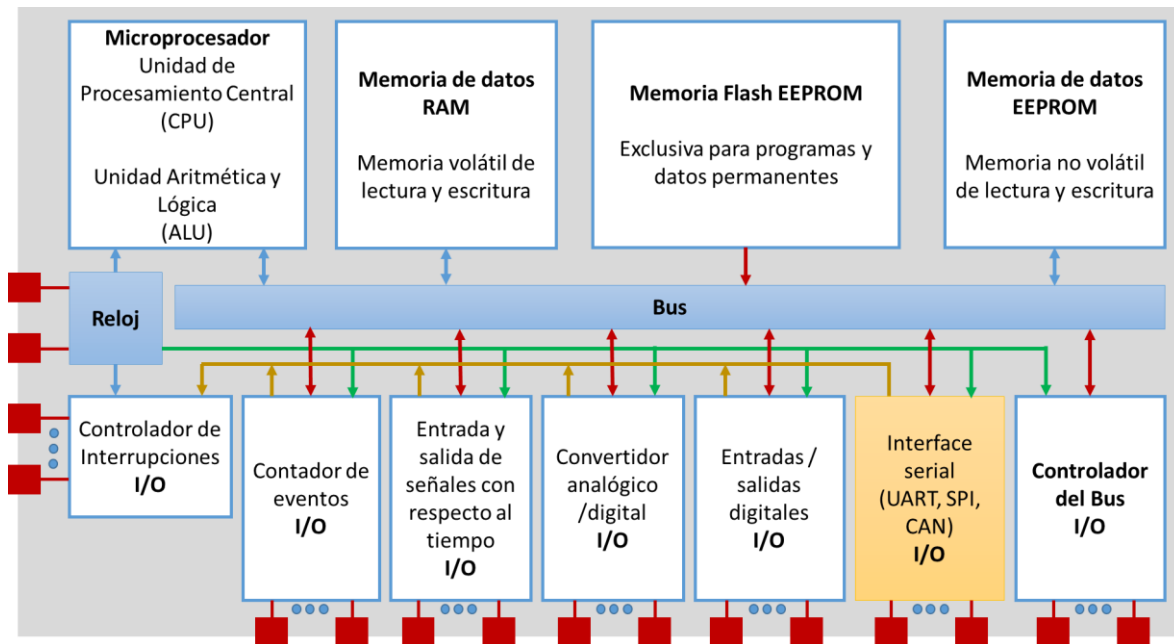
*Dispositivos de entrada y salida:* dispositivos que manejan el intercambio de datos.

*Memoria Flash EEPROM:* en esta memoria se encuentra almacenado el programa de ejecución en forma de código máquina creado por el ingeniero de desarrollo de software.

*Memoria principal (RAM):* es una memoria de lectura y escritura donde se almacenan los datos que se están procesando a tiempo de ejecución.

*Buses:* conectan los elementos de la ECU con el microcontrolador permitiendo la transferencia de datos.

*Reloj:* genera la frecuencia de ejecución de los comandos o instrucciones.



**Figura 3** – Arquitectura típica del microcontrolador de un vehículo (Automotive Mechatronics, 2015a).

Una ECU tiene embebido en la *memoria Flash EEPROM* el código con el funcionamiento que será ejecutado por el microprocesador.

Dentro de las rutinas o funciones del código embebido se pueden encontrar declaraciones de variables, que son creadas a tiempo de ejecución y por lo tanto deben ser instanciadas en un espacio de memoria (*RAM*), la cual puede ser sobrescrita en cualquier momento de la ejecución del código.

Los valores de entrada que tiene el código son obtenidos mediante señales que previamente han sido convertidas a valores digitales, sin embargo, el programa a ejecutar también puede contener datos previamente establecidos que permitan la obtención de valores correctos de entrada, así como valores límites para la salida.

Dentro de la información embebida en un controlador existen datos que identifican tanto a la *ECU* como al mismo vehículo automotor, así como datos que no deben ser eliminados en ninguna situación (por ejemplo, al apagar el vehículo o desconectar la batería de alimentación), por lo que esta información también es almacenada en la *memoria flash EEPROM*.

Las señales que los sensores transmiten hacia la *ECU* pueden ser analógicas o digitales. Las señales analógicas indican el nivel de voltaje, debido a que la máxima resolución para estas señales es de 5mV a través de un rango de 0 a 5V (rangos de temperatura, presión de fluidos). Las señales digitales solamente tienen dos estados lógicos (0s y 1s). Las señales analógicas son convertidas a valores digitales mediante un conversor analógico-digital.

## 1.2 **Estándares de Comunicación**

Debido a la fehaciente necesidad de conocer el estado completo de un vehículo automotor, la industria automotriz, en compañía con empresas expertas en el sector eléctrico/electrónico, han desarrollado diferentes protocolos de comunicación capaces de permitir la comunicación intra-vehicular.

Un protocolo de comunicación puede ser descrito como un conjunto de reglas o normas que permiten establecer una comunicación entre dos o más dispositivos, en el caso de los vehículos automotores, los protocolos permiten conectar dos o más *ECUs* aprobando el intercambio de información y brindando así, un correcto funcionamiento del mismo vehículo, además de la identificación de posibles anomalías que pueden surgir.

Existen protocolos que han sido diseñados y desarrollados con base en estándares que son propuestos por organizaciones como *ISO* (*International*

*Organization for Standardization*) o *SAE (Society of Automotive Engineers)*. Estos estándares pueden ser representados a través del modelo de capas *OSI (Open System Interconnection)* identificando el impacto y la función que brindan en torno a cada capa (Tabla 1).

Actualmente, una empresa líder en el sector automotriz en la ciudad de Toluca, cuenta con estándares *ISO* que permiten la comunicación de las *ECUs* que integran los vehículos que desarrolla. Estas normas son regidas, de entre muchos otros, bajo los estándares *ISO 11898-1:2003*, *ISO 15765-2:2011* e *ISO 14229-1:2013*.

**Tabla 1** – Representación de estándares aplicados a las capas del modelo OSI (ISO, 2013a).

Capa del modelo OSI	Estándares ISO
<b>Aplicación</b>	ISO 14229-1, ISO 14229-3, ISO 14229-4, ISO 14229-5, ISO 14229-6, ISO 14229-7, entre otros estándares.
<b>Presentación</b>	Especificado por el fabricante del vehículo.
<b>Sesión</b>	ISO 14229-2.
<b>Transporte</b>	ISO 15765-2.
<b>Red</b>	
<b>Enlace de datos</b>	ISO 11898-1, ISO 11898-2.
<b>Física</b>	

Debido al impacto que tienen estos tres estándares mencionados anteriormente para el desarrollo de este reporte de aplicación de conocimientos, se dará una descripción más detallada de ellos, sin embargo, se debe hacer hincapié que para cada capa del modelo *OSI* existen estándares que permiten la comunicación de datos. De la misma forma, se debe tomar en cuenta que el estándar *ISO 11898-1:2003* se encuentra alineado a las primeras dos capas del modelo *OSI* (capa física y de enlace de datos), el estándar *ISO 15765-2:2011* hace referencia a la capa de red y de transporte, mientras que el estándar *ISO 14229-1:2013* hace referencia a la capa de aplicación, meramente identificado en aplicación de diagnóstico.

No todas las capas en el modelo *OSI* son necesarias en un sistema de comunicación simple (Automotive Mechatronics, 2015b) por consiguiente, las capas que toman mayor relevancia en los sistemas de redes en vehículos automotores son: física, enlace de datos, red, transporte y aplicación.

### 1.2.1 ISO 11898-1:2003

De acuerdo al estándar *ISO 11898-1:2003 Road vehicles – Controller Area Network*, se especifica la señalización de la red de área de controlador (*CAN*) dentro de las capas física (*physical layer*) y de enlace de datos (*data link layer*) del modelo *OSI* (ISO, 2003a), mediante un protocolo de comunicación serial que soporte el control y la multiplexación (habilidad de transmitir datos que provienen de diversos emisores y receptores en un medio físico único) distribuida en tiempo real, para el uso de vehículos de carretera. Además, brinda especificaciones para configurar el intercambio de información digital entre módulos que implementan este estándar.

Las partes que componen al estándar *ISO11898:2003*, son:

- Parte 1: Señalamiento de la capa física y de enlace de datos.
- Parte 2: Unidad de acceso al medio de alta velocidad.
- Parte 3: Baja velocidad, interfaz dependiente del medio con tolerancia a fallos.
- Parte 4: Comunicación en tiempo desencadenado.

Para el presente proyecto de aplicación de conocimientos se trabajó exclusivamente con la parte 1 del estándar *ISO11898:2003* con la finalidad de comprender el protocolo *CAN*, debido a que la comunicación de diagnóstico de los vehículos automotores se realiza mediante este protocolo.

### 1.2.2 Red de Controladores de Área (CAN)

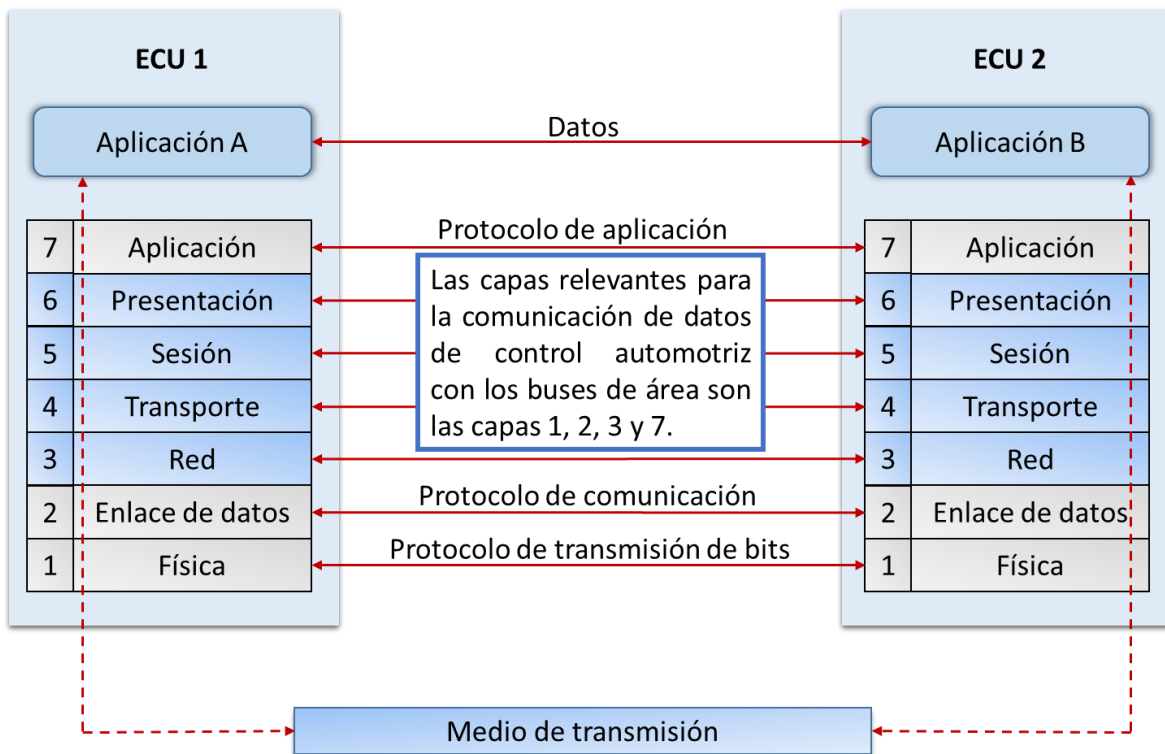
*CAN* es un estándar para comunicaciones distribuidas con enfoque en el manejo de errores, creado por la empresa alemana Bosch, para aplicarlo, inicialmente, en vehículos automotores con el objetivo de habilitar una comunicación serial para permitir una filtración, priorización y fragmentación de mensajes, así como el manejo y detección de errores.

En el estándar *ISO 11898-1:2013* se considera a *CAN* dentro del contexto de las siete capas del modelo *OSI*, enfocándose en la capa de enlace de datos y los efectos que tiene con el resto de las capas (Figura 4).

Una de las características con las que cuenta el protocolo *CAN*, es que tiene dos cables dedicados para la comunicación, *CAN high* y *CAN low*, ambos con 2.5 volts en un estado *inactivo* o de *stand by*, pero cuando comienza la transmisión de datos,

*CAN high* puede llegar a tomar un voltaje de 3.75v mientras que *CAN low* toma 1.25v (Figura 5), generando un contraste de voltaje de 2.5v entre las líneas de comunicación creando un diferencial de tensión que no es sensible a picos inductivos, campos eléctricos u otros ruidos (AXIOMATIC, 2006).

La importancia del uso de este protocolo en los vehículos automotores radica en que, el protocolo *CAN* es usado para conectar *ECUs* implicadas en el funcionamiento del *tren motor (powertrain)*, donde la obtención y análisis de datos así como la ejecución de funciones o actividades, deben ser desarrolladas en intervalos de tiempo realmente cortos, con la finalidad de mostrar respuestas, a situaciones que se manifiestan de manera indefinida. En otras palabras, todos los cálculos y procedimientos que ejecuten las *ECUs* comunicadas dentro de este protocolo (*CAN*) deben ser transmitidos en tiempo real (milisegundos).



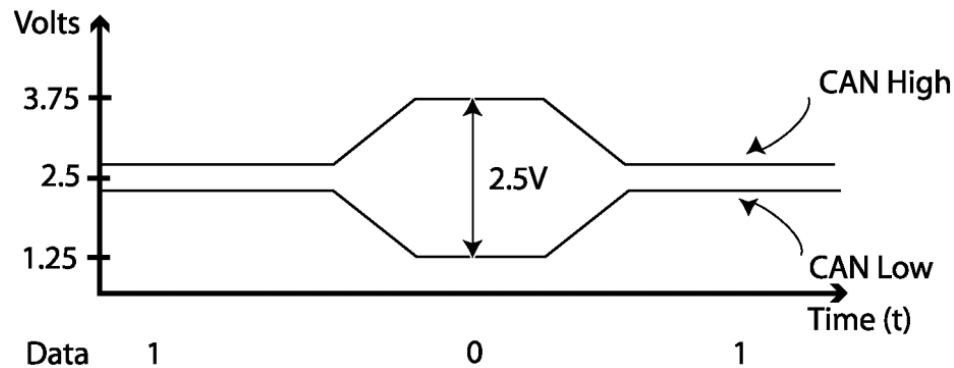
**Figura 4** – Interacción de las capas del modelo OSI.

Como se mencionó anteriormente, el estándar *ISO 11898:2003* cuenta con dos rangos de velocidades, *high speed* y *low speed* especificados en los apartados 2 y 3 respectivamente. Para la ejecución de este proyecto solamente fue utilizado el rango de *high speed* del protocolo *CAN* debido a que la problemática del mismo proyecto se encontraba enfocada a *ECUs* de motor (*ECM*) y transmisión (*TCM*). Sin embargo, se

resaltan las comparaciones entre estas dos velocidades para que el lector se dé una idea de los rangos de transmisión.

*Low speed CAN* maneja rangos de transferencia en máximos de 125 kb/s, y su uso está enfocado a la comunicación en unidades de confort, es decir, a datos mostrados en el *cluster* que se encuentra en el tablero del auto, luces tanto exteriores como interiores, seguros eléctricos, info-entretenimiento, etc., donde, a pesar de que la velocidad es baja en comparación con el rango *high speed*, esta velocidad es imperceptible para la persona que se encuentra conduciendo el vehículo.

En contraparte los rangos máximos de transferencia *high speed CAN* oscilan 1 Mb/s, lo que permite la transferencia de datos en tiempo real entre unidades de control enfocadas al manejo del vehículo y seguridad de los pasajeros (sistemas de frenos anti-bloqueo, bolsas de aire, etc.).



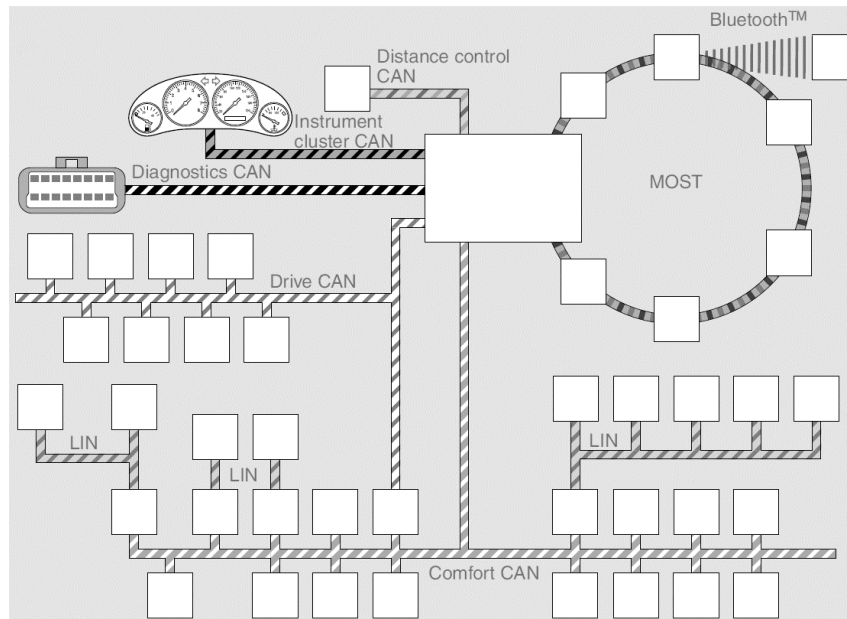
**Figura 5** – Estados dominante y recesivo del protocolo *CAN* (AXIOMATIC, 2006).

Existen otros protocolos dentro de una red de comunicación de un vehículo automotor como lo son *LIN* (*Local Interconnect Network*) con aplicaciones similares a las de *low speed CAN*, sin embargo los rangos de transferencia son inferiores (20kb/s). También existe el protocolo *MOST* (*Media Oriented Systems Transport*) que, aunque sus velocidades son mayores que las del protocolo *CAN* (22Mb/s), el costo de implementación es elevado, por lo que actualmente sólo los vehículos de gama alta lo tienen incorporado para sus sistemas de info-entretenimiento.

Como se puede ver en la Figura 6, existen diferentes protocolos en una misma red de comunicación de un vehículo, y en ocasiones, los datos que una *ECU* puede obtener y analizar, son transmitidos a otra *ECU* con diferente protocolo de comunicación. Para estas situaciones existe una *ECU* que funge el rol de *gateway*, permitiendo este tipo de comunicación entre *ECUs*.

Dentro de la capa de enlace de datos del modelo *OSI*, la unidad de intercambio de información (*PDU*, *Protocol Data Unit* por sus siglas en inglés) son las tramas, las cuales se pueden encontrar en el canal de comunicación.

El protocolo *CAN* permite la comunicación entre dos o más *ECUs* sin necesidad de que exista una jerarquía maestro-esclavo, permitiendo que cualquier *ECU* pueda transmitir mensajes, siempre y cuando el bus de comunicación se encuentre libre.



**Figura 6** – Topología de red de un vehículo de lujo (Automotive Mechatronics, 2015c).

La propiedad anterior se puede obtener debido a que *CAN* no se enfoca en las direcciones de los *ECUs* para realizar la comunicación (la topología de este protocolo es el bus), en cambio realiza el direccionamiento de la transmisión con base en los mensajes (Figura 7), es decir, dentro de las tramas que son creadas por las *ECUs* (en la capa de aplicación) se determina el(los) nodo(s) receptor(es), mientras que la *ECU* receptora cuenta con una lista de posibles *ECUs* emisoras de las cuales se puede aceptar el mensaje recibido (identificadores).

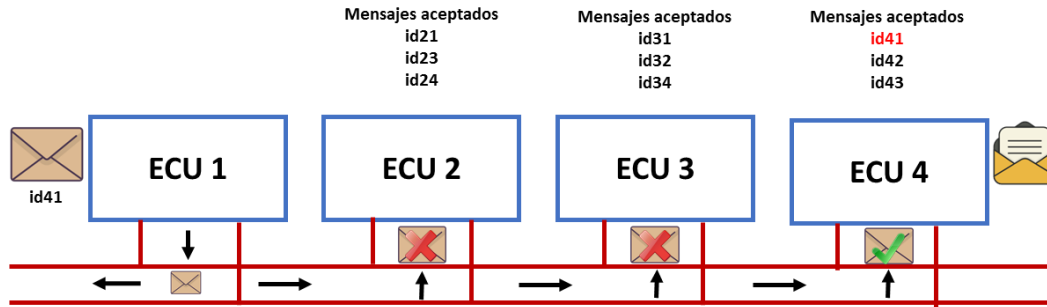


Figura 7 – Protocolo CAN basado en mensajes.

La información empaquetada en la trama puede ser recibida por la *ECU* descrita dentro del identificador del mensaje (*physical request*), o por todas las *ECUs* que se encuentran conectadas (*functional request, broadcast*), donde solamente las *ECUs* que soportan la información empaquetada en el mensaje podrán analizarla y, si es el caso (lógica a nivel de aplicación), generar una respuesta que será transmitida al bus de comunicación, donde, el emisor de la solicitud podrá obtener la respuesta enviada.

El protocolo CAN ha ido evolucionando desde su creación, agregando requerimientos y surgiendo nuevas versiones (Tabla 2).

Tabla 2 - Versiones de CAN (Reuss H. Christian, 1993).

Versión	Año	Actualización
1.0	1985	Estándar original
1.1	1987	Re-especificación de requisitos de temporización de bits
1.2	1990	Aumento de tolerancia del oscilador
2.0A	1991	Es similar a la versión 1.2
2.0B	1991	Agrega la creación de frames extendidos de manera opcional
FD	2011	Incrementa la velocidad de transmisión a 3.7 Mb/seg

Para el presente proyecto de aplicación de conocimientos fue utilizada la versión 2.0 B como parte de los requerimientos del cliente (ingeniero de validación).

Para el protocolo CAN 2.0 existen dos tipos de formatos de tramas, las que tienen 11 bits para el identificador (trama estándar, 2.0A) y las que tienen 29 bits (trama extendida, 2.0B). Su principal diferencia es que el primero permite transmitir 2048 mensajes únicos en todo el bus, que van desde 0x000 hasta 0x7FF, mientras que las de 29 bits soportan hasta 536 millones de mensajes únicos. Esto se ve reflejado en una mayor cantidad de información transmisible.



### 1.2.2.1 Estructura de un mensaje CAN

Realizando una paráfrasis del protocolo *CAN* de Bosch (Bosch, 1991) el formato de una trama extendida (identificador de 29 bits) es el siguiente (Figura 8):

**Bit de comienzo (1)** señala el comienzo de la trama con un valor dominante (0).

**Bits de identificación de mensaje CAN (32)** compuesto por:

- a) Bits de nivel de prioridad (3):
  - Un valor de cero tiene la más alta prioridad (000).
  - Un valor de ocho tiene la más baja prioridad (111).
- b) Bit de página de datos extendida (1), bit reservado para uso a futuro, su valor es dominante (0).
- c) Bit de página de datos (1), bit selector de página de datos, usualmente con valor dominante (0).
- d) Bits de formato del *PDU* (6), determinan si el mensaje es transmitido hacia un único nodo o hacia todos:
  - $< 0xF0$ , el mensaje es transmitido hacia un nodo.
  - $\geq 0xF0$ , el mensaje es transmitido a todos los nodos (*broadcast*).
- e) Bit de solicitud remota sustituta o *SRR* (1), que funge como marcador de referencia entre mensajes *CAN 2.0A* y extendidos *CAN 2.0B*, con un valor recesivo (1).
- f) Bit de identificador de extensión o *IDE* (1), indica que los siguientes bits continuarán siendo parte del identificador del mensaje, teniendo un valor recesivo (1).
- g) Bits de formato del *PDU* restantes (2).
- h) Bits de parte específica del *PDU* (8), basado en una dependencia con el formato del *PDU*, es decir, de los ocho bits anteriores:
  - Si el formato *PDU* es menor a  $0xF0$  entonces la parte específica del *PDU* representará la dirección destino.
  - Si el formato *PDU* es mayor o igual a  $0xF0$  entonces la parte específica del *PDU* representará una extensión de dirección del grupo destino permitiendo, así, expandir el número de grupos de difusión que pueden ser representados y de los cuales se esperará respuesta.
- i) Bits de dirección de controlador (8), contienen la dirección del controlador o dispositivo que transmite el mensaje.
- j) Bit de transmisión remota o *RTR* (1), para el estándar *J1939* de *SAE* este valor siempre es dominante (0), es decir que la información es requerida por otro nodo.

**Bits del campo de control (6)** descrito como:

- a) Bits reservados (2), ambos con valor recesivo (1).

b) Bits de especificación (4), del tamaño de los datos (en bytes) que serán transmitidos (*Data Length Code* o *DLC*).

**Bits que contienen información (0-64)**, conformados por los datos que serán transmitidos por la trama tomando en cuenta que de cada ocho bits el primero en ser transmitido será el más significativo (*Most Significant Bit – Less Significant Bit*).

**Bits del Código de Redundancia Cíclica o CRC (15)**, que permite la detección de errores mediante el *checksum* o número de bits que fueron transmitidos. Además, al final de estos 15 bits se encuentra un bit como delimitador entre el *CRC* y el *ACK*.

**Bit delimitador del CRC (1)**, permite identificar el final del *CRC* mediante un valor recesivo (1).

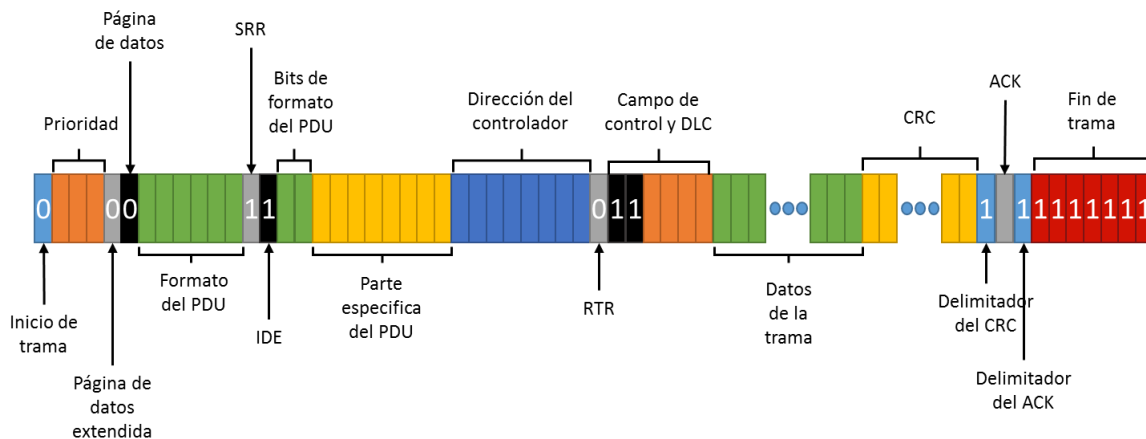
**Bit de reconocimiento o ACK (1)**, determina si existe conexión con el receptor o no para la transmisión de la información. Este campo es previamente llenado por el transmisor con un valor recesivo (1) y sobrescrito por el receptor con un valor dominante (0) una vez que se ha recibido el mensaje de manera correcta.

**Bit delimitador del ACK (1)**, es un segundo bit delimitador, recesivo (1).

**Bits de fin de trama (7)**, representan el final de la trama o mensaje.

La estructura de un mensaje *CAN 2.0B* (formato extendido con identificador de 29 bits) cuenta con un máximo de 150 bits.

Se debe de tomar en cuenta que a nivel de aplicación (referente al modelo *OSI*), el software que se utilice puede realizar la configuración de algunos bits con valores ya preestablecidos.



**Figura 8** – Estructura de un mensaje *CAN* con *header* de 29 bits.

### 1.2.3 ISO 15765-2:2011

El estándar *ISO 15765-2:2011* hace referencia al protocolo de capa de red del protocolo *CAN*. Constituido bajo el título de *Road vehicles – Diagnostic Communication over Controller Area Network (DoCAN)* (ISO, 2011a), está compuesta de los siguientes apartados:

- Parte 1: Información general y definición de casos de uso.
- Parte 2: Protocolo de transporte y servicios de la capa de red.
- Parte 3: Implementación de servicios de diagnóstico unificado.
- Parte 4: Requerimientos para sistemas relacionados con emisiones.

La principal consideración de este estándar es que especifica el protocolo de red usado sobre el protocolo *CAN* para la transmisión de tramas.

Debido a la existencia de mensajes o tramas con más de ocho bytes de datos (mínimo número de bytes soportados por un único mensaje *CAN*) es necesario que se establezcan los criterios de transmisión y recepción entre dos o más *ECUs*. Dicho de otra manera, este estándar permite la segmentación de la información que puede ser empaquetada en mensajes de un tamaño máximo de ocho bytes manteniendo comunicación entre *ECUs* mediante herramientas de control de flujo de información.

### 1.2.4 Protocolo de control

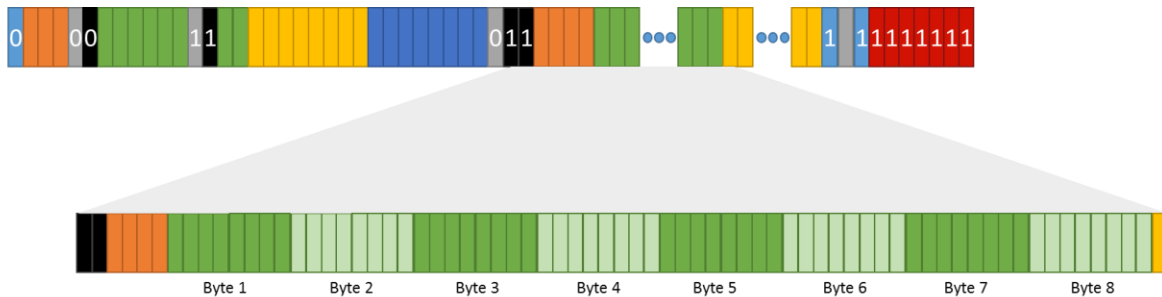
La correcta transmisión de información entre dos o más nodos conectados dentro de una misma red es el resultado de una comunicación detallada y estructurada. Dicha comunicación es establecida por dos entidades o *ECUs*, que fungen como emisor y receptor, así como un mensaje generado por el emisor que contendrá información que puede ser útil para el receptor. La interacción entre estos dos entes se realiza bajo un canal de comunicación, en este caso un bus *CAN* que sirve como medio de transmisión de los mensajes. Sin embargo, debe de existir un apropiado entendimiento o conjunto de reglas entre el emisor y el receptor para poder interpretar y establecer una correcta comunicación de la información que ha sido transmitida.

Para poder realizar una comunicación exitosa entre dos *ECUs* mediante un protocolo *CAN*, se toma en consideración la estructura básica de un mensaje *CAN* y se genera (mediante el estándar *ISO 15765-2:2011*) de tal forma que dentro del contenido de la trama se pueda ver reflejado el código o las reglas que permitan la interacción.

### 1.2.4.1 Tramas CAN

Habiendo especificado la composición de las tramas, se debe hacer mención que existen diferentes tipos de mensajes de acuerdo a la cantidad de información que va a ser transmitida, es decir, si existe la necesidad de mandar o recibir información que exceda el límite de una trama (ocho bytes), entonces habrá que segmentar la información.

Una trama que es transmitida bajo el protocolo *CAN 2.0B* está compuesta de ocho bytes que contienen información relevante para el emisor (datos). Además, dentro de estos ocho bytes se encuentra información como el tipo de trama que es transmitida, así como el tamaño (en bytes) de la información a transferir. El tipo de trama señala un punto clave en el protocolo de comunicación, pues permite identificar si la información a transmitir tendrá como máximo siete bytes, siendo ésta una transmisión de una sola trama o excederá este límite y por consiguiente serán necesarias más tramas para poder transferir toda la información. La representación de los ocho bytes de datos se muestra en la Figura 9.



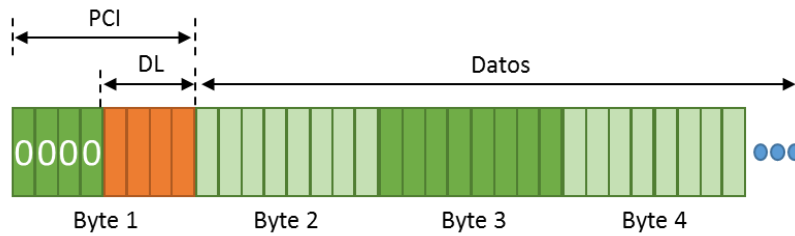
**Figura 9** – Representación del contenido de un mensaje *CAN 2.0B*.

#### Trama simple (0x0h)

Este tipo de trama denominada *single frame* (Figura 10) representa la transmisión de contenido no mayor a 56 bits, es decir, la información que será enviada no excederá este límite por lo cual se podrá transmitir en una sola trama.

Descrito en el primer byte de información (del bit más significativo al menos significativo) con el valor de 0000<sub>b</sub>. Los siguientes cuatro bits del primer byte

corresponden al tamaño del contenido (en bytes) que será transmitido (*Data Length* o *DL*), este primer byte es conocido como *Protocol Control Information* o *PCI*.



**Figura 10** – Trama simple.

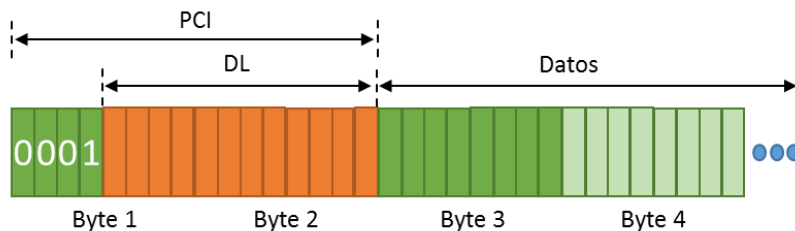
Se debe hacer énfasis que, al tener cuatro bits en el campo de *DL*, se puede llegar a tener un valor de cero hasta 15 bytes transmisibles (valores  $0_h$  o mayor a  $7_h$ ), sin embargo, las tramas que contengan estos valores no serán tomadas en cuenta por el receptor.

### Primer trama ( $0x1_h$ )

Como su nombre lo indica, es la trama inicial. Este tipo de trama es utilizada para indicar cuando una serie de tramas serán enviadas, esto debido a que el contenido a transmitir excede los 56 bits de una sola trama (Figura 11).

Su identificador corresponde al valor  $0001_b$  situados en el mismo lugar que la trama simple (en el *PCI*).

Debido a que el contenido que será transmitido excede siete bytes, es determinado que, para representar el tamaño del contenido, se podrán utilizar los siguientes cuatro bits así como el siguiente byte del contenido (byte #2) a transmitir.



**Figura 11** – Primer trama.

De la misma forma que la trama con la configuración de una simple (*single frame*), esta configuración cuenta con limitantes para el apartado de *DL* pues no está permitido agregar valores menores a ocho bytes, es decir que el *DL* puede llegar a tomar valores desde  $8_h$  hasta  $FFF_h$  (4095 bytes de información transmissible).

### Trama consecutiva (0x2<sub>h</sub>)

Este tipo de trama indica una trama consecutiva a partir del tipo *first frame*, es decir, después de haber sido enviada la primera trama y cuyo objetivo es transmitir información de manera consecutiva contemplando el valor que fue asignado en el campo de *DL*.

Su identificador está definido en los dos primeros bits (del bit más significativo al menos significativo) dentro del campo de control con el valor 0010<sub>b</sub>. Los siguientes cuatro bits corresponden al identificador de secuencia *SN* (*Sequence Number*) que determina el número de mensaje que está siendo transmitido comenzando con el valor 0x0Y<sub>h</sub> (Y puede tomar los valores de 1<sub>h</sub>-F<sub>h</sub>) pues, para esta trama consecutiva después de la *primer trama* o *first frame* tendría un identificador de secuencia 0x01<sub>h</sub>. Si el tamaño de información a transmitir necesita una o varias tramas más, entonces éstas tomarán el siguiente identificador de secuencia (en este caso 0x02<sub>h</sub>) y así sucesivamente. No obstante, si el número de tramas consecutivas excede el identificador de secuencia F<sub>h</sub>, se dará un reinicio del identificador, pero esta vez desde un valor 0<sub>h</sub>.

Como se puede visualizar en la Figura 12, esta configuración no cuenta con el campo de *DL* debido a que fue previamente calculado en el *first frame*, tomando en cuenta todos los bytes a transmitir.

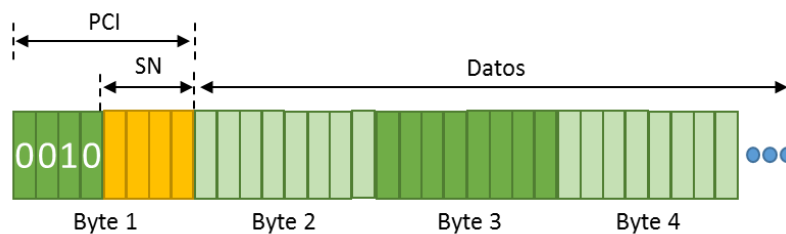


Figura 12 – Trama consecutiva.

### Trama de control de flujo (0x3<sub>h</sub>)

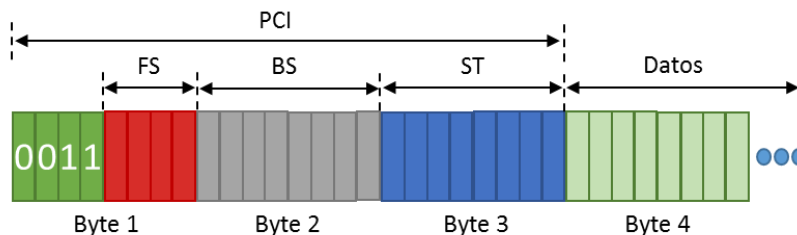
Este tipo de trama representa la respuesta del receptor al haber recibido una primera trama (*first frame*). Es una trama que transmite el receptor al emisor para notificar que se encuentra disponible para recibir todas las tramas consecutivas (*consecutive frames*) que sean necesarias en la transmisión de información.

Esta trama tiene por identificador el valor  $0x3_h$  que se encuentra en los dos primeros bits (del bit más significativo al menos significativo) dentro del campo de control.

Los siguientes cuatro bits representan el estatus de flujo o *FS* (*Flow Status*), indicando al transmisor de la *primera trama* (*first frame*) si puede o no continuar con la transmisión de tramas consecutivas (*consecutive frames*). Si dentro de estos cuatro bits se encuentra un  $0x0_h$  en el estatus de flujo, significa que el transmisor puede mandar las tramas consecutivas (*consecutive frames*). Si se encuentra un  $0x1_h$  en el estatus de flujo, significa que existe una condición de pausa por lo que el transmisor no puede mandar las tramas consecutivas (*consecutive frames*) hasta que el receptor envíe una trama de control de flujo con el valor de  $0x0_h$ . Si se tienen un valor de  $0x2_h$  en el estatus de flujo, entonces el receptor está indicando que su espacio en memoria no es tan amplio para poder almacenar toda la información, terminando así la recepción de tramas.

Dentro de este tipo de trama los siguientes ocho bits después del *FS* representan el tamaño de bloque (*BS*, *Block Size*) o número de tramas consecutivas (*consecutive frames*) que pueden ser enviadas antes de que sea necesario que el receptor vuelva a mandar otra trama de control de flujo. Si el valor de estos ocho bits es de  $0x00_h$ , entonces el receptor podrá recibir todas las tramas consecutivas (*consecutive frames*) sin necesidad de mandar otra trama de control de flujo, en otro caso los valores que este campo toma son de  $01_h$  a  $FF_h$ , lo que indica el número máximo de tramas consecutivas (*consecutive frames*) soportadas hasta que se mande otra trama de control de flujo (*flow control*).

Los últimos ocho bits, indican el tiempo de separación mínimo o *ST* (*Separation Time*) que está permitido entre la transmisión de las tramas consecutivas (*consecutive frames*, medido en milisegundos o microsegundos, por ejemplo, para el valor  $0x05_h$ , se deberán transmitir las tramas consecutivas (*consecutive frames*) cada cinco milisegundos (Figura 13).



**Figura 13** – Trama de control de flujo.

Los rangos en milisegundos que se encuentran definidos bajo el estándar *ISO 15765-2:2011* van desde  $0_h$  hasta  $7F_h$ , pero, los valores que van desde  $80_h$  hasta  $F0_h$  así como de  $FA_h$  a  $FF_h$  se encuentran reservados por este estándar, por lo tanto, el rango de  $F1_h$  a  $F9_h$  representan unidades de 100 microsegundos ( $F1_h = 100\mu s$ ).

El lector puede apreciar que dentro del *PCI*, la configuración del tipo de trama emplea solamente dos bits de los cuatro utilizables, de modo que los dos bits más significativos se encuentran reservados por el estándar *ISO 15765-2:2011*.

### 1.2.5 Protocolo de transporte

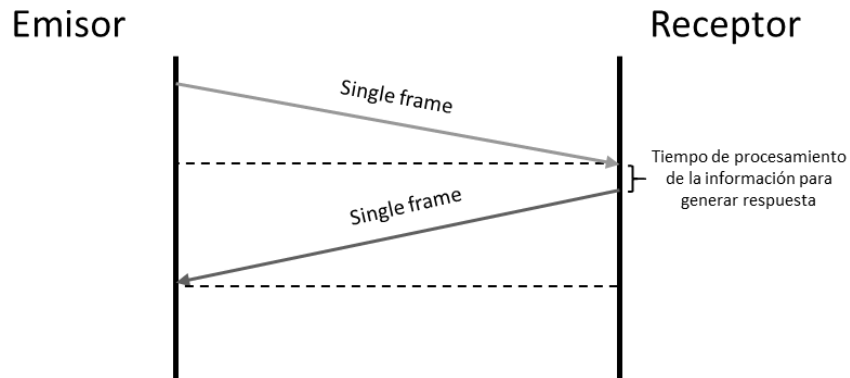
Para aquellas transmisiones que están compuestas de más de ocho bytes, es decir, que la transmisión realizada consta de más de una trama (comúnmente denominada transmisión *multi-frame*), es necesario contar con un protocolo de transporte. Este protocolo de transporte, se encuentra especificado en el estándar *ISO 15765-2:2011*.

Entre las tareas que realiza este protocolo se encuentra la de segmentar y re ensamblar los bloques de datos que puedan ser transmitidos en un mensaje, es decir, sólo ocho bytes por trama. Si la información a transmitir ocupa un mayor espacio que la cantidad anterior, será necesario tener un control del flujo de datos para completar dicha transmisión.

En primera instancia, es representada la transmisión simple con la finalidad de que el lector pueda reconocer las diferencias entre transmisiones simples y compuestas.

En una comunicación simple el emisor genera una trama de datos con no más de ocho bytes que es transmitida al bus *CAN* y recibida por aquellas *ECUs* que puedan soportar dicha petición. Si existe una respuesta y ésta no excede los ocho bytes de datos, el receptor será ahora el emisor, generando una trama simple (*single frame*) que recibirá el emisor (Figura 14).



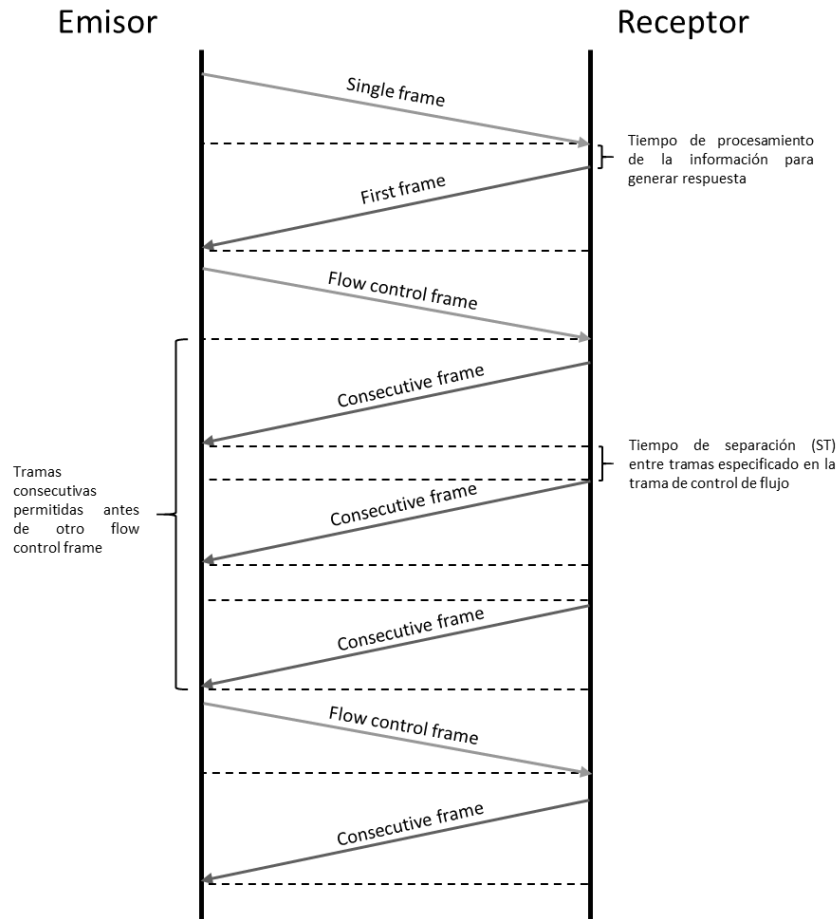


**Figura 14** – Transmisión simple.

La comunicación compuesta o *multi-frame* se realiza en dos casos:

- Cuando el emisor transmite información que excede siete bytes.
- Cuando el receptor manda respuesta a alguna petición del emisor y la información de esa respuesta excede los siete bytes.

En cualquiera de los dos casos es necesario transmitir las diferentes tramas previamente descritas donde (en el segundo caso) el emisor solicita información al receptor mediante una trama simple (*single frame*), pero la información solicitada es mayor a 56 bits por lo que el emisor debe generar una primer trama (*first frame*) indicando el tamaño exacto de la respuesta. Al ser recibido esta trama por el emisor, genera una respuesta o *flow control*, indicando el estatus de la comunicación. Dicho de otra forma, los parámetros como el número de mensajes soportados como el tiempo en que tienen que ser enviados uno detrás de otro, son especificados y enviados el receptor. A partir de este punto, de no existir restricción alguna, el receptor debe de transmitir toda la información o tramas consecutivas (*consecutive frames*) que fueron indicados en la trama de control de flujo (*flow control*), soportados por el transmisor (Figura 15).



**Figura 15** – Transmisión compuesta receptora.

Para el caso contrario (Figura 16), cuando el emisor manda una solicitud de envío o almacenamiento de información generada por el mismo emisor, entonces la comunicación será inversa, puesto que el emisor generará un *first frame*; el receptor, después de haber recibido esa trama, generará el *flow control* para que el emisor pueda transferir la información restante mediante tramas consecutivas (mismo caso que el anterior).

Se deben de tomar en cuenta los tiempos antes mencionados porque son importantes para el protocolo de transporte, pues si en una transmisión de *multi-frames* el tiempo de envío entre frames es superior al asignado, la *ECU* determinará que no existe más contenido que el que ya fue transmitido y comenzará con el ensamble de los mensajes que haya recibido de manera correcta hasta esa instancia, lo que posiblemente resulte en una respuesta negativa o incorrecta de la *ECU* al no contar con toda la información.

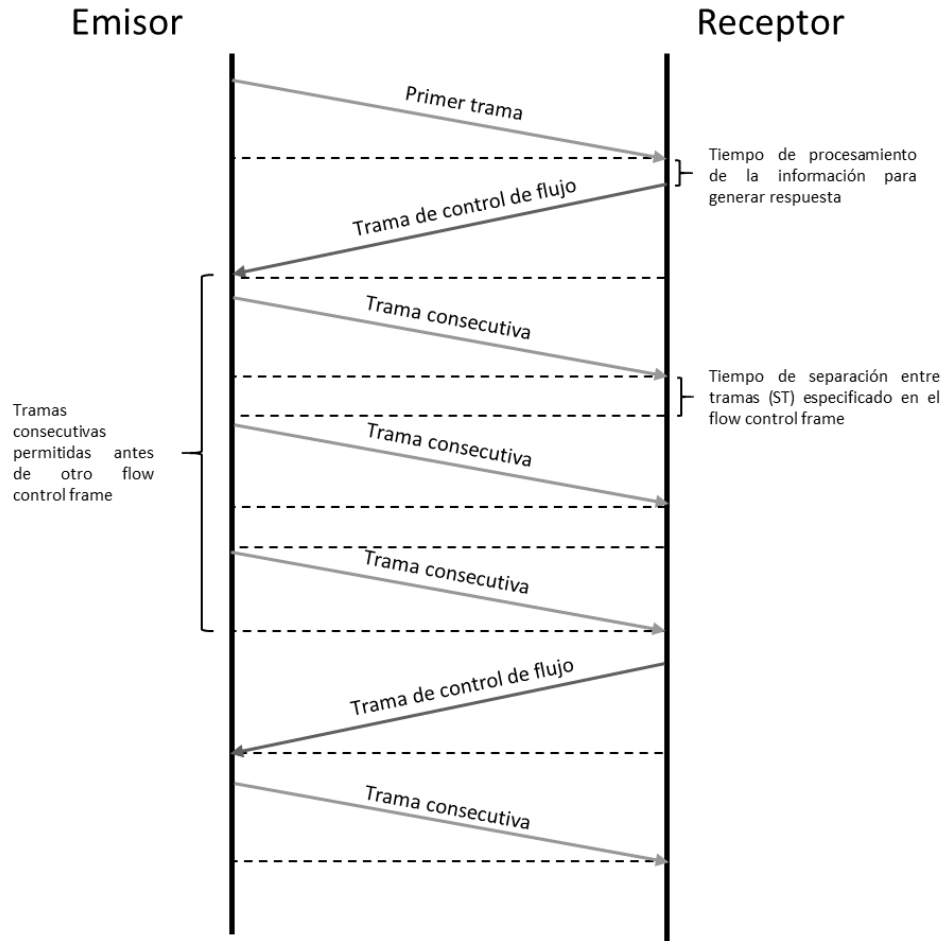


Figura 16 – Transmisión compuesta emisora.

### 1.2.6 ISO 14229-1:2013

El estándar *ISO 14229-1:2013* tiene como objetivo dar a conocer las especificaciones o requerimientos necesarios para poder brindar servicios de diagnóstico de datos seriales del vehículo automotor, los cuales permiten al ingeniero de validación controlar funciones de diagnóstico de una o varias *ECUs*.

El estándar *ISO 14229-1:2013* no es aplicable para la comunicación de mensajes que no sean de diagnóstico transmitidos entre dos *ECUs*, sin embargo, no está restringida la implementación de validaciones mediante un tercero (ingeniero en validación) con el fin de utilizar los servicios de diagnóstico en los canales de comunicación del vehículo, para así realizar un intercambio de datos de manera bidireccional (ISO 14229-1. Second Edition. 2013:1).

Las partes que componen al estándar *ISO* (2013b), bajo el título de *Road vehicles – Unified Diagnostic Services (UDS)*, son:

- Parte 1: Especificación y requerimientos.
- Parte 2: Servicios de la capa de sesión.
- Parte 3: Servicios de diagnóstico unificado sobre *CAN*.
- Parte 4: Servicios de diagnóstico unificado sobre *FlexRay*.
- Parte 5: Servicios de diagnóstico unificado sobre *IP*.
- Parte 6: Servicios de diagnóstico unificado sobre *K-Line*.
- Parte 7: Servicios de diagnóstico unificado sobre *LIN*.

Cabe resaltar, que en este proyecto de aplicación de conocimiento, exclusivamente se trabajará con la Parte 1 *especificación y requerimientos*, debido a que es la parte requerida por el centro de ingeniería de México, lugar donde se realizó la colaboración.

Además, el estándar *ISO 14229-1:2013* brinda una interfaz de programación de aplicaciones simple para el envío y recepción de mensajes; este protocolo abstrae detalles técnicos y complejidades asociadas con la sincronización de mensajes y de interconexión de red *CAN*, explicada en el estándar *ISO 11898-1*.

### 1.2.7 Protocolo de aplicación UDS

El protocolo *UDS* ofrece una serie de funcionalidades necesarias para talleres de reparación, los desarrolladores y *testers*, para que puedan, por ejemplo, leer o escribir datos en la memoria de la *ECU*, el programa de la memoria flash y crear un comportamiento específico para una *ECU* (Assawinjaipetch P., Heeg M., Gross D., Kowalewski S., 2014).

Con la finalidad de conocer el estado actual de un vehículo automotor así como la detección oportuna de fallas del mismo, en las últimas décadas, las compañías fabricantes de vehículos han desarrollado algoritmos que permiten identificar posibles anomalías en cuanto al desempeño del automóvil. Sin embargo, cada una de estas empresas tenía (hasta la aparición del estándar *ISO*) su propio lenguaje y código para el reconocimiento de los errores, creando así, una amplia variedad de respuestas para una misma falla entre diferentes marcas.

El protocolo *UDS* ha sido desarrollado con la finalidad de que la implementación de los sistemas de diagnóstico de los vehículos automotores se

unifique para que todas las empresas manufactureras de vehículos (aquellas que quieran apegarse a este esquema) utilicen el mismo protocolo.

Debido al creciente uso de las *ECUs* en los automóviles para administrar funcionalidades mecánicas y eléctricas, ha sido conveniente y necesario desarrollar sistemas que permitan monitorear el estado del automóvil, permitiendo reportar errores en tiempo real o bajo circunstancias reales. Estos sistemas son denominados como servicios de diagnóstico.

Los sistemas de diagnóstico se encuentran regidos bajo un modelo cliente/servidor donde cada una de las *ECUs* que se encuentran conectadas a un mismo bus toma el rol de servidor, generando funcionalidad y respondiendo a peticiones que realiza el cliente, siendo éste el *tester* de diagnóstico (ingeniero de validación).

*UDS* es un protocolo de mensajes confirmados, lo que indica que por cada petición transmitida por parte del *tester*, una respuesta debe ser generada y transmitida por la(s) *ECU(s)* (respuesta positiva o negativa) que soporten dicha solicitud, tomando en cuenta casos particulares de direccionamiento funcional.

La forma en la que trabaja *UDS* es a través de servicios, los cuales son descritos e implementados independientemente del protocolo de comunicación que utilicen las *ECUs*. Con la ayuda de estos servicios, implementados en la capa de aplicación, el *tester* puede controlar funciones de diagnóstico sobre una *ECU*. Los servicios soportados por *UDS* son divididos en grupos que describen su funcionalidad como se puede ver en la Tabla 3.

**Tabla 3** – Clasificación de servicios UDS por tipo de funcionalidad.

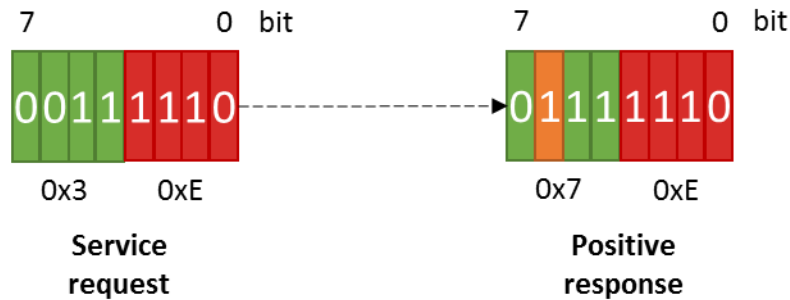
Tipo de unidad funcional	Descripción de funcionalidad	Servicios
<b>De administración de diagnóstico y comunicación</b>	Definen parámetros referentes a la administración del diagnóstico, el acceso de usuarios y la transmisión de información.	-Diagnostic session control -Tester present -Security Access -ECU reset -Communication control -Control DTC setting -Access timing parameters -Link control -Respose on event service
<b>De transmisión de datos</b>	Realizan la lectura o escritura de datos que son de acceso inmediato, debido al cambio constante que puede llegar a tomar sus valores.	-Read data by identifier -Read memory by address -Write data by identifier -Write data by address -Read data by periodic identifier -Dynamically define data identifier
<b>De transmisión de datos almacenados</b>	Ejecutan actividades similares a la unidad funcional anterior pero con respecto a información de diagnóstico que se encuentra dentro de memoria que no es volátil.	-Clear diagnostic information -Read DTC information
<b>De control de entradas y salidas</b>	Controlan actuadores, sensores y dispositivos ligados al controlador.	-Input output control by identifier
<b>De activación a distancia de rutinas</b>	Este grupo contiene un servicio que influye en el comportamiento de la ECU.	-Routine control
<b>De carga y descarga</b>	Transfieren de datos entre el cliente y el servidor en forma de carga o descarga de información.	-Request download -Request upload -Transfer data -Request transfer exit

La estructura que deben tener los mensajes que contienen a los servicios está dada por un identificador o *SID* (*Service Identifier*), el cual permite la distinción de cada uno de los servicios soportados bajo el estándar *ISO 14229-1:2013*. El rango de valores que pueden tomar los *SIDs* van desde 0x00<sub>h</sub> a 0xFF<sub>h</sub> ya que se encuentra compuesto de ocho bits (Tabla 4).

**Tabla 4** – Rangos del identificador de servicios (ISO, 2013c).

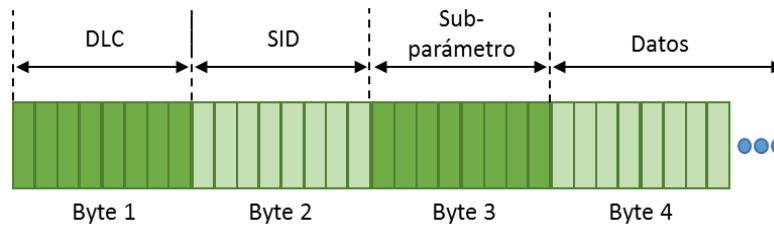
Service Identifier	Function
0x00 <sub>h</sub> – 0x3E <sub>h</sub>	Service request
0x3F <sub>h</sub>	Not applicable
0x50 <sub>h</sub> – 0x7E <sub>h</sub>	Positive service responses
0x7F <sub>h</sub>	Negative response
0x80 <sub>h</sub> – 0x82 <sub>h</sub>	Not applicable
0x83 <sub>h</sub> – 0x88 <sub>h</sub>	Service request
0x89 <sub>h</sub> – 0xB9 <sub>h</sub>	Not applicable
0xBA <sub>h</sub> – 0xBE <sub>h</sub>	Service request
0xBF <sub>h</sub> – 0xC2 <sub>h</sub>	Not applicable
0xC3 <sub>h</sub> – 0xC8 <sub>h</sub>	Positive service response
0xC9 <sub>h</sub> – 0xF9 <sub>h</sub>	Not applicable
0xFA <sub>h</sub> – 0xFE <sub>h</sub>	Positive service response
0xFF <sub>h</sub>	Not applicable

Dentro de la especificación de rangos de *SID*, la forma de obtener una respuesta positiva es mediante uno de los bits que comprenden el *SID* cuando se realiza la construcción de una respuesta. Todo los *service requests* tienen en el sexto bit (tomado del menos significativo al más significativo) un valor de 0<sub>b</sub> con la finalidad de que, al crear una respuesta positiva, este bit cambie a un valor de 1<sub>b</sub> lo que generaría una respuesta comprendida entre los rangos de *positive service response* (Figura 17).



**Figura 17** – Positive service response.

Cada uno de los servicios descritos puede tener sub-parámetros o sub-funciones que permitan realizar una misma actividad (referente al tipo de servicio) pero de diferente forma, agregando información específica en la creación de la trama para poder configurar un comportamiento más detallado (Figura 18). Estas sub-funciones dependen definitivamente de cada uno de los servicios y se encuentran claramente especificados en el estándar *ISO 14229-1:2013*.



**Figura 18** – Estructura de los mensajes UDS.

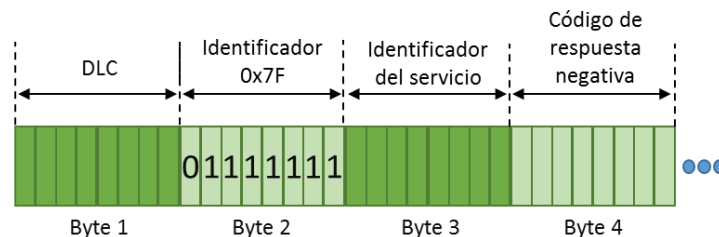
Una vez que el servicio ha sido transmitido desde el *tester* hacia el servidor a través del bus *CAN*, si la *ECU* tiene la capacidad lógica de responder a la petición solicitada lo hará.

Pueden existir casos donde la información enviada es incorrecta, incompleta o el servidor se encuentra ocupado. El mismo estándar *ISO 14229-1:2013* define tales respuestas como *negative responses* las cuales tienen como identificador el valor  $0x7F_h$ . Sin embargo, así como las *positive responses*, este tipo de respuestas contienen información adicional que identifican el tipo error en la transmisión del mensaje (manejando esta información como un sub-parámetro).

Existen rangos ya establecidos por el estándar *ISO 14229-1:2013* que definen el tipo de respuesta negativa que se puede generar, así como la posible causa:

- Para los rangos de  $0x00_h - 0x7F_h$  la respuesta negativa se relaciona a problemas de comunicación.
- Para los rangos de  $0x80_h - 0xFF_h$  la respuesta negativa podría ser referida a condiciones incorrectas.

Las respuestas negativas más comunes se pueden visualizar en la Tabla 5 así como la estructura de una respuesta negativa es desplegada en la Figura 19.



**Figura 19** – Estructura de una respuesta negativa.



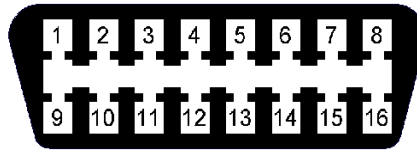
**Tabla 5** – Respuestas negativas más comunes de los servicios de *UDS*.

Identificador hexadecimal	Respuesta negativa	Descripción
<b>0x11</b>	<i>Service not supported</i>	Este tipo de error se genera debido a que el servidor no puede reconocer el tipo de servicio que fue solicitado.
<b>0x12</b>	<i>Sub-function not supported</i>	Similar al tipo de respuesta negativa anterior, el servidor no reconocer el sub-parámetro de acuerdo al servicio que fue solicitado.
<b>0x13</b>	<i>Incorrect message length or invalid format</i>	Esta respuesta negativa es generada debido a que el tamaño del mensaje es diferente al descrito en el DLC o porque el servicio solicitado requiere un tamaño de información diferente al que fue transmitido.
<b>0x22</b>	<i>Conditions not correct</i>	Muchas veces para poder cumplir con ciertos comportamientos es necesario tener precondiciones configuradas en la ECU, por lo que, si éstas no fueron generadas previamente, el servidor mandará este tipo de respuesta negativa al enviar un mensaje que requiera dicha configuración.
<b>0x31</b>	<i>Request out of range</i>	Esta respuesta negativa es generada a causa de valores, dentro de los datos enviados por el <i>tester</i> , los cuales exceden los límites soportados por la configuración lógica del servicio en cuestión.
<b>0x33</b>	<i>Security access denied</i>	Debido a que mucha de la información cuenta con niveles de seguridad, si no se han abierto los candados de seguridad y se manda un mensaje a ese tipo de información, una respuesta negativa de este tipo será generada.

## 1.3 Interfaz de comunicación de diagnóstico

### 1.3.1 Assembly Line Data Link

Con la finalidad de poder monitorear el conjunto de redes que se encuentran en comunicación dentro de un vehículo automotor, los automóviles cuentan con un conector que permite la entrada de interfaces para la interacción con un *tester* y, de esta forma, poder conocer el estado actual del vehículo, identificando posibles fallas o comportamientos. Este conector es conocido como *ALDL* (por sus siglas en inglés *Assembly Line Data Link*) comúnmente localizado por debajo del tablero y volante de lado del conductor (Figura 20).



**Figura 20** – Pines del conector ALDL (OBDTester, 2015).

Uno de los estándares sobre los que se rige el diseño del conector *ALDL*, y sobre el cual se trabajó en el presente proyecto de titulación, es el estándar *ISO 15031-3 Diagnostic connector and related electrical circuits, specification and use*, el cual muestra la configuración específica para los pines de salida (Intrepid Control Systems, Inc, 2016d):

- Pin 2: Es usado como línea *high* del estándar *SAE J1850*.
- Pin 10: Es usado como línea *low* del estándar *SAE J1850*.
- Pin 7 y 15: Son líneas ligadas a los estándares *ISO 9141-2* e *ISO 14230-4*.
- Pin 6 y 14: Usados para señales *CAN\_H* (high) y *CAN\_L* (low) respectivamente bajo el estándar *ISO 15765*.
- Pin 1, 3, 8, 9, 11, 12 y 13: Se dejan a disposición de la empresa manufacturera.
- Pin 4: Funge como tierra del vehículo (chasis).
- Pin 5: Es la señal de tierra.
- Pin 16: Voltaje de la batería.

### 1.3.2 NeoVi

*NeoVi* es una herramienta diseñada por IntrepidCS, que permite el monitoreo de redes *CAN*, y realiza funciones de diagnóstico similares a las que realiza un escáner en un taller automotriz (Figura 21).

Para el presente proyecto de aplicación de conocimientos fue utilizada la herramienta de comunicación *NeoVi FIRE*, por requerimiento del cliente específicamente para las actividades de monitoreo y comunicación.



**Figura 21** – NeoVi Fire (HongKe, 2014).

Gracias a la herramienta de comunicación *NeoVi Fire*, la transmisión de mensajes es posible debido a que utiliza el protocolo *SAE J2534*, el cual, a pesar de ser desarrollado prioritariamente para especificaciones de reprogramación de las *ECUs*, permite realizar comunicación de diagnóstico a través de una *API (Application Programming Interface)*.

La interfaz de comunicación *NeoVi Fire* funciona a partir de la conexión de una de las terminales (*conector OBD II*) con la entrada *ALDL* del automóvil, mientras que la terminal opuesta (conexión *USB*) debe ir conectada a la computadora (Figura 22).



**Figura 22** – Conexión del NeoVi Fire.

## Capítulo II. Diseño e implementación

---

*En este capítulo se expone el desarrollo del proyecto iniciando con la etapa de identificación del problema y el análisis del proceso actual permitiendo reconocer y analizar los requerimientos del cliente con el fin de generar la propuesta de solución que fue implementada.*

### 2.1 Problemática actual

#### 2.1.1 Identificación del problema

Hoy en día, uno de los grupos de diagnóstico de una empresa líder en el sector automotriz de la ciudad de Toluca realiza pruebas de validación del software de diagnóstico embebido en *ECUs* de los vehículos automotores (*TCM* y *ECM*); en concreto, los ingenieros de validación construyen los mensajes que serán transmitidos *nibble* por *nibble* (cuatro bits) a partir de un software llamado *VehicleSpy*.

*VehicleSpy* es un software destinado completamente al diagnóstico y monitoreo de *ECUs*, bajo protocolos como *CAN*, *LIN*, *ISOs*, *FlexRay*, entre otros. Una característica importante que brinda esta herramienta es la visualización de la comunicación de mensajes en tiempo real. De igual manera, *VehicleSpy* permite la generación de mensajes transmisibles a través de un bus de red vehicular.

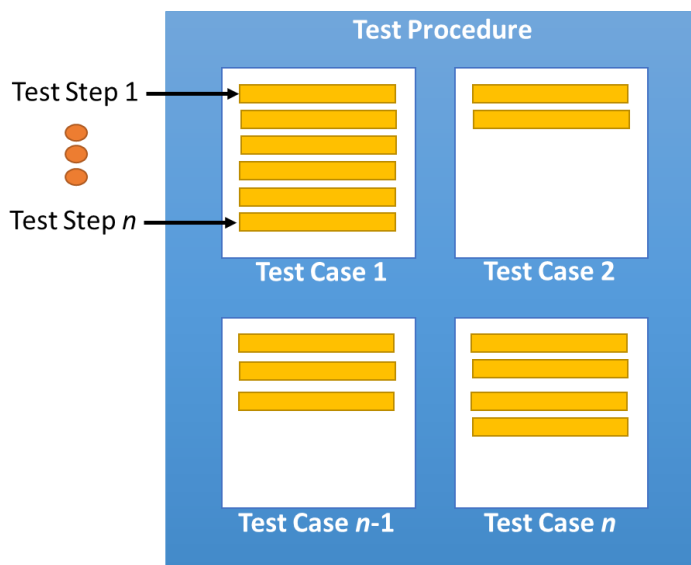
Los mensajes que se transmiten son generados a partir de documentos en donde se estipulan los pasos (mensajes) que deberán transmitirse para poder comprobar una funcionalidad de diagnóstico específica (*test procedures*). Estos *test procedures* son diseñados por el ingeniero de validación del área mediante el estándar *ISO 14229-1:2013*.

El proceso de validación del software embebido involucra una cantidad significativa de recursos, tiempo y herramientas de desarrollo para poder ejecutar las pruebas. Además, los recursos y herramientas utilizadas para dicha ejecución son compartidas con los demás grupos del área quienes realizan distintas pruebas referentes al software embebido en las *ECUs*.

Por otra parte, la verificación del código embebido en los controladores cuenta con una agenda para su ejecución, pues se tiene un periodo de tiempo determinado para poder realizar todas y cada una de las pruebas de validación. Por cuestión de

confidencialidad, solamente, se dará un ejemplo del tiempo requerido para realizar esta actividad, así como del tiempo objetivo para cumplir con las pruebas de validación, siendo éste de cuatro semanas.

Hasta el momento, el área de diagnóstico cuenta con 16 *test procedures* referentes a los servicios de diagnóstico del estándar *ISO 14229-1*, los cuales están conformados por casos de prueba (*test cases*) que verifican una funcionalidad en específico y que a su vez se encuentran integrados por pasos (*test steps*) que son los mensajes que el *tester* (ingeniero de validación) deberá transmitir hacia la *ECU* (Figura 23).



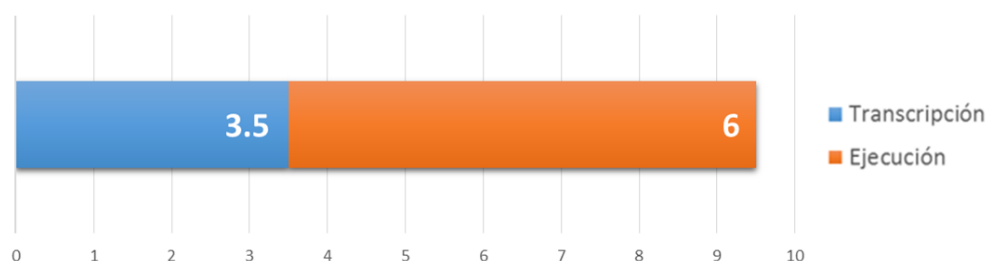
**Figura 23** – Estructura de un *test procedure*.

Un *test step* está compuesto por una descripción acerca del mensaje que se va a transmitir, así como del mensaje (en base hexadecimal) que será transmitido. De la misma forma, un *test step* cuenta con la descripción de la respuesta esperada que debe recibir el ingeniero de validación por parte de la *ECU*, seguido del mensaje esperado en base hexadecimal.

Los *test procedures* tienen un cociente de 34 *test cases* que a su vez están integrados, en promedio, de 13 a 15 *test steps*.

En la actualidad, el ingeniero de validación genera manualmente los *mensajes de diagnóstico* a partir de los documentos que contienen los *test procedures*. Dicha actividad requiere un promedio de tres horas y media, solamente para transcribir los mensajes de los documentos a la herramienta de software que se utiliza. Sin embargo, el tiempo para ejecutar un *test procedure* de manera completa (*step por step* en la

herramienta de software) es de seis horas, lo que conlleva a un gran total de nueve horas y media por *test procedure* (Figura 24).

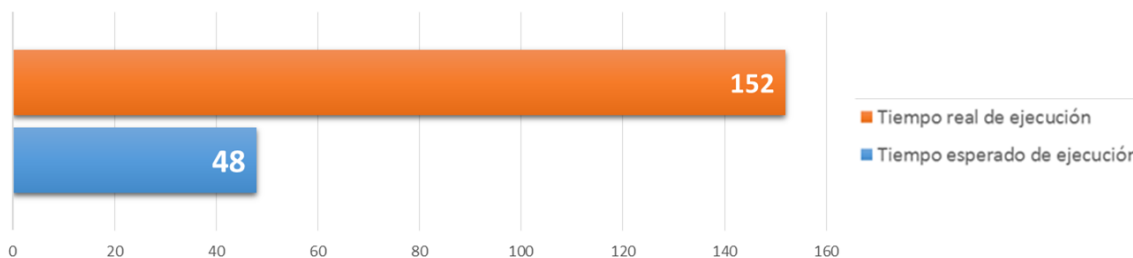


**Figura 24** – Horas necesarias por Test Procedure.

Uno de los recursos limitados son las horas disponibles en los *bancos de prueba (benches)* para poder realizar las pruebas de validación de diagnóstico. Estas pruebas son ejecutadas en lapso de 12 horas por semana divididos en bloques de cuatro horas por día (tres días a la semana).

Tomando en cuenta la restricción anterior, el tiempo que tomaría realizar la completa ejecución de los 16 *test procedures* con los que actualmente se cuenta sería de 152 horas. No obstante, el tiempo definido (y con el que se dispone) para poder realizar dichas pruebas es de 48 horas (12 horas a la semana durante un mes), por lo que se estaría excediendo en un 316% el tiempo destinado a esta actividad (Figura 25). Lo cual, se vería reflejado en tiempos de retraso en las pruebas de validación de controladores como *TCM* y *ECM*, teniendo como consecuencia el aplazamiento (tiempo) de puntos clave en la construcción (significativamente en la parte de ECUs) de los autos que contengan dicho software. Incluso, la realización de estas pruebas permite la detección temprana y oportuna de errores que pudieran llegar a suscitarse en la ejecución del código embebido.

Dicho de otra manera, la ejecución de estas pruebas permite desarrollar un código más robusto reduciendo significativamente los niveles de errores o bugs que, si no se llegan a detectar en ninguna parte del proceso de construcción del vehículo automotor, pueden llegar hasta el cliente final, generando, muy probablemente, *recalls* (llamadas de la agencia a autos para la resolución de algún problema en específico) y como resultado una insatisfacción del cliente, pérdidas económicas y de reputación para la compañía.



**Figura 25** – Comparación entre tiempo objetivo esperado y real de ejecución.

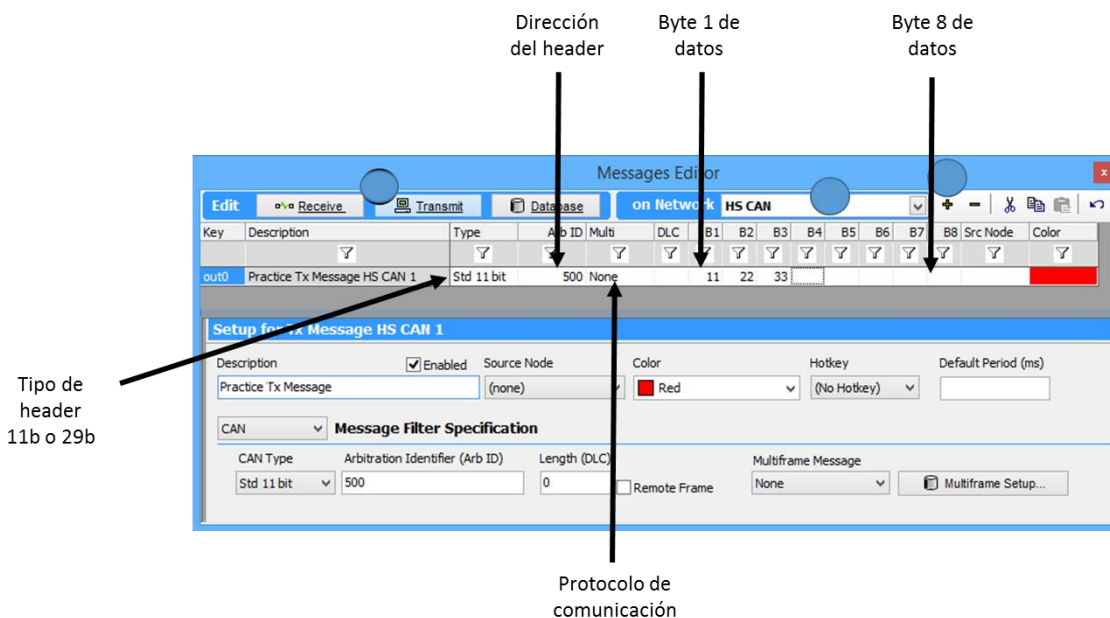
### 2.1.2 Análisis del proceso actual

La empresa automotriz, donde se realizó el presente reporte de aplicación de conocimientos, cuenta con las licencias de *Intrepid Control Systems, Inc.* para el uso del software *Vehicle Spy* como herramienta de comunicación de tramas en un bus de protocolo *CAN*, así como un *sniffer* del bus de comunicación.

*Vehicle Spy* es una herramienta ingenieril de clase mundial para el diseño, verificación y análisis de redes de vehículos automotores de hoy y mañana (Intrepid Control Systems, Inc., 2016)

Dentro del proceso actual se encuentra la extracción de mensajes (tramas) a partir del *test procedure* que las contiene. Para lo anterior es necesario que elementos del mensaje como el *header*, el *DLC* y el contenido del mensaje, sean ingresados de forma manual al *software Vehicle Spy*. En otras palabras, el ingeniero de validación debe agregar la información de una trama byte por byte (Figura 26) en dicho software, lo que se traduce en un consumo excesivo de tiempo (3.5 horas en promedio).

La generación de los *test steps* dentro del *software Vehicle Spy* debe realizarse de forma obligatoria para la primera ejecución del *test procedure*; sin embargo, estos *steps* pueden almacenarse en archivos de datos del mismo *software* para futuras ejecuciones (formato *VS3*), lo que permite reducir el tiempo en la generación de los mensajes y evitar el re-trabajo. No obstante, existen situaciones en las que los *test procedures* deben actualizarse, ya sea en el contenido de los *steps*, o modificando la estructura de ejecución de los *test cases* (adición o eliminación de *steps*) lo que representaría una modificación en los mensajes ya creados dentro de *Vehicle Spy*.



**Figura 26** – Construcción de tramas en *Vehicle Spy* (Intrepid Control Systems, Inc., 2016c).

*Vehicle Spy*, hasta la versión 3.7.1.83 – 04/19/2016 (descargable desde la página de *IntrepidCS* <https://www.intrepidcs.com/>), no cuenta con ninguna opción que permita extraer información de archivos *XSLX* (formato del archivo de un *test procedure*) para la creación de mensajes. Solamente cuenta con una función de importación de archivos con formato propio de *Vehicle Spy* (*VS3*).

El monitoreo de los mensajes transmitidos y recibidos puede realizarse mediante una función con la que cuenta *Vehicle Spy* como se muestra en la Figura 27. Esta función permite visualizar los mensajes en tiempo real como pueden aquellos generados a partir de la comunicación de la *bench* con el *ECU*, así como los mensajes que el *tester* transmite hacia alguna *ECU* conectada en el bus de comunicación.

Dado que el ingeniero de validación ha transmitido un mensaje, rápidamente tiene que comprobar la respuesta recibida. De la misma forma, la verificación de estas respuestas se realiza de forma manual mediante la visualización de los datos transferidos en el bus *CAN* que se pueden visualizar en la pestaña de *Messages* de *Vehicle Spy*. Este proceso se realiza para cada uno de los *test steps* contenidos en los *test cases* que se deben ejecutar.



Filter	Line	Time	Tx	Er	Description	Arbid/Header	DataBytes	Network
o/v	1077	24.328 ms			Test Message	110	0F 98 52 0F 65 2C 26 41	HS CAN
o/v	1078	0 µs			Practice Tx Message	500	11 22 33	HS CAN
o/v	1079	13.000 ms			SW CAN \$240	240	45 34 53 03	SW CAN
o/v	1080	6.910 ms			J1850 VPW FF:32:43	FF 32 43	42 33 EE 47	J1850 VPW
o/v	1081	12.413 ms			J1850 PWM 04:F1:F2	04 F1 F2	23 43 03 79 23 03	J1850 PWM
o/v	1082	45.395 ms			Test Message	110	03 EA 90 B8 5A CA 80 77	HS CAN
o/v	1083	0 µs			Practice Tx Message	500	11 22 33	HS CAN
o/v	1084	13.000 ms			ISO9141/KWZK 23:32:23	23 32 23	44 03 DB 9A	ISO9141/KWZK
o/v	1085	14.417 ms			MS CAN \$123	123	23 23 23 BD 1C	MS CAN
o/v	1086	125 µs			MS CAN \$110	110	12 01	MS CAN

**Figura 27** – Mensajes en tiempo real en *Vehicle Spy* (Intrepid Control Systems, Inc., 2016a).

Si al realizar la verificación de los mensajes, el ingeniero de validación encuentra alguna inconsistencia en la respuesta (posibles errores al transmitir los mensajes o inexistencia de mensajes cuando se esperaba alguna respuesta por parte de la *ECU*), entonces, tiende a revisar y, si es meritorio, modificar el o los *steps* que se encuentren involucrados.

*Vehicle Spy* permite, una vez transmitidos los mensajes y (si es el caso) recibido respuestas, guardar toda la información que se encuentra contenida en el *buffer de datos*. Esta información puede ser visualizada mediante algún *software* que soporte la lectura de archivos con formato *CSV*, *PCAP*, *VSF*, *ASC*, *BLF* o *IMG*.

El ingeniero de validación, contando con la información correspondiente a los resultados de la ejecución del *test procedure*, puede determinar cuáles *steps*, comparados con la respuesta esperada, pasaron y cuáles no. Dicho proceso se realiza de forma manual puesto que la comparación se elabora mediante una comparación visual de los mensajes.

La información obtenida es proporcionada a los directivos del área de diagnóstico para su análisis e investigación, concluyendo de esta forma una de las iteraciones del proceso de validación que, como se mencionó anteriormente, debe realizarse periódicamente.

Una vez comprendido el entorno sobre el cual se realiza la validación del software de diagnóstico, es posible construir un modelo utilizando diagramas para describir el proceso de negocio del cliente, es decir, determinar cuáles son los requerimientos iniciales del software a implementar (Schach, 2006).

En muchos casos, no hay necesidad de crear una representación gráfica de un escenario de uso; sin embargo, la representación con diagramas facilita la comprensión (Pressman, 2010). Por ello, a partir de este apartado se incluirán algunos diagramas con el fin de proporcionar mayor información acerca de los procesos.

En la parte posterior, las Figuras 28 y 29, brindan una referencia ilustrativa de los elementos utilizados en el proceso actual de validación de los *test procedures*.



**Figura 28** – Diagrama de bloques del proceso de validación actual.

A partir del *test procedure*, el ingeniero de validación extrae los *test steps* que componen a un *test case* y los agrega al *software Vehicle Spy*. Una vez creados los mensajes dentro del *software* mencionado, el ingeniero de validación comienza a transmitir los mensajes (uno a la vez). Este proceso es posible gracias a la comunicación que realiza la interfaz *NeoVI*, entre el *PC* donde se encuentra instalado el *software* y la *bench*.

La *bench*, por su parte, es un simulador en tiempo real que crea un ambiente de pruebas para sistemas embebidos, generando y recibiendo señales.

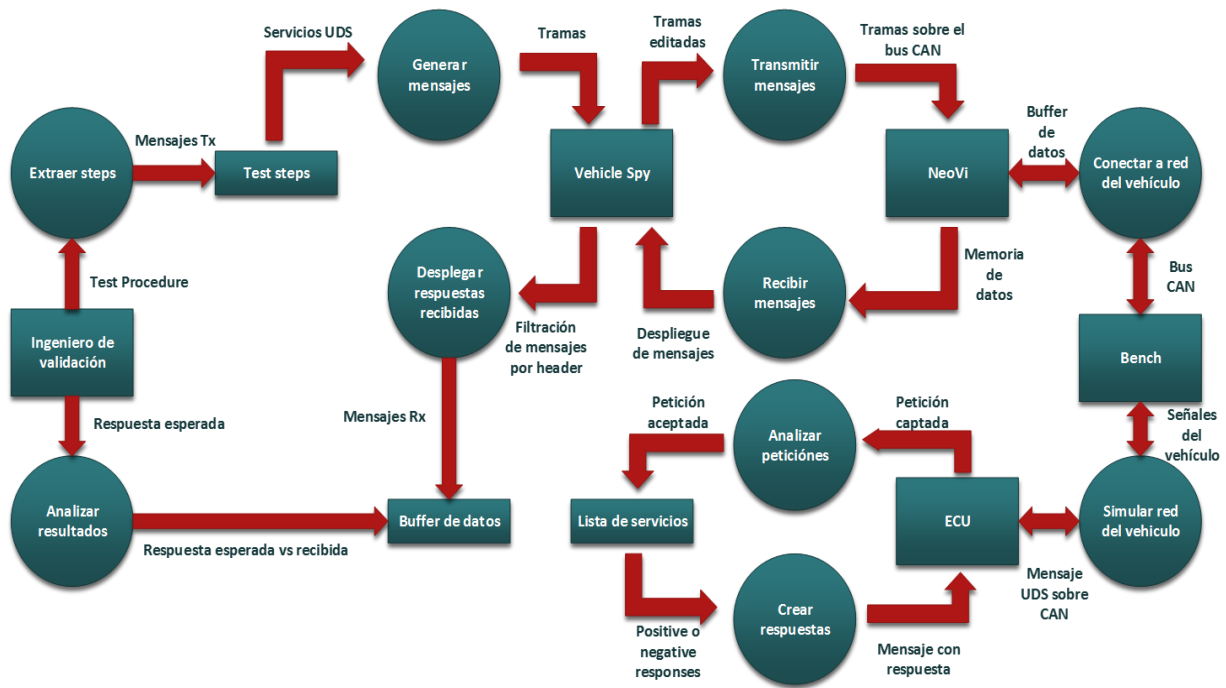
Por último, la *ECU*, recibe las señales emitidas por la *bench*, y, mediante el código embebido, determinará la respuesta a transmitir hacia el *tester* o, en este caso, el ingeniero de validación.

El ingeniero de validación realiza una inspección visual de los mensajes recibidos y colocará en el documento *XLSX (test procedure)* el resultado o mensaje recibido como respuesta por parte de la *ECU*, indicando si, en comparación con la respuesta esperada, el *test step* pasó o falló.

Con la finalidad de dar seguimiento a los resultados obtenidos, se guardan los mensajes que han sido creados en el *software Vehicle Spy*, para la ejecución posterior de los mismos o, si es el caso, su modificación y transmisión. De este modo, el ingeniero de validación exporta los mensajes que fueron transmitidos y recibidos, contenidos en el *buffer de datos* del *software*, en un documento de *Excel (CSV)*, para

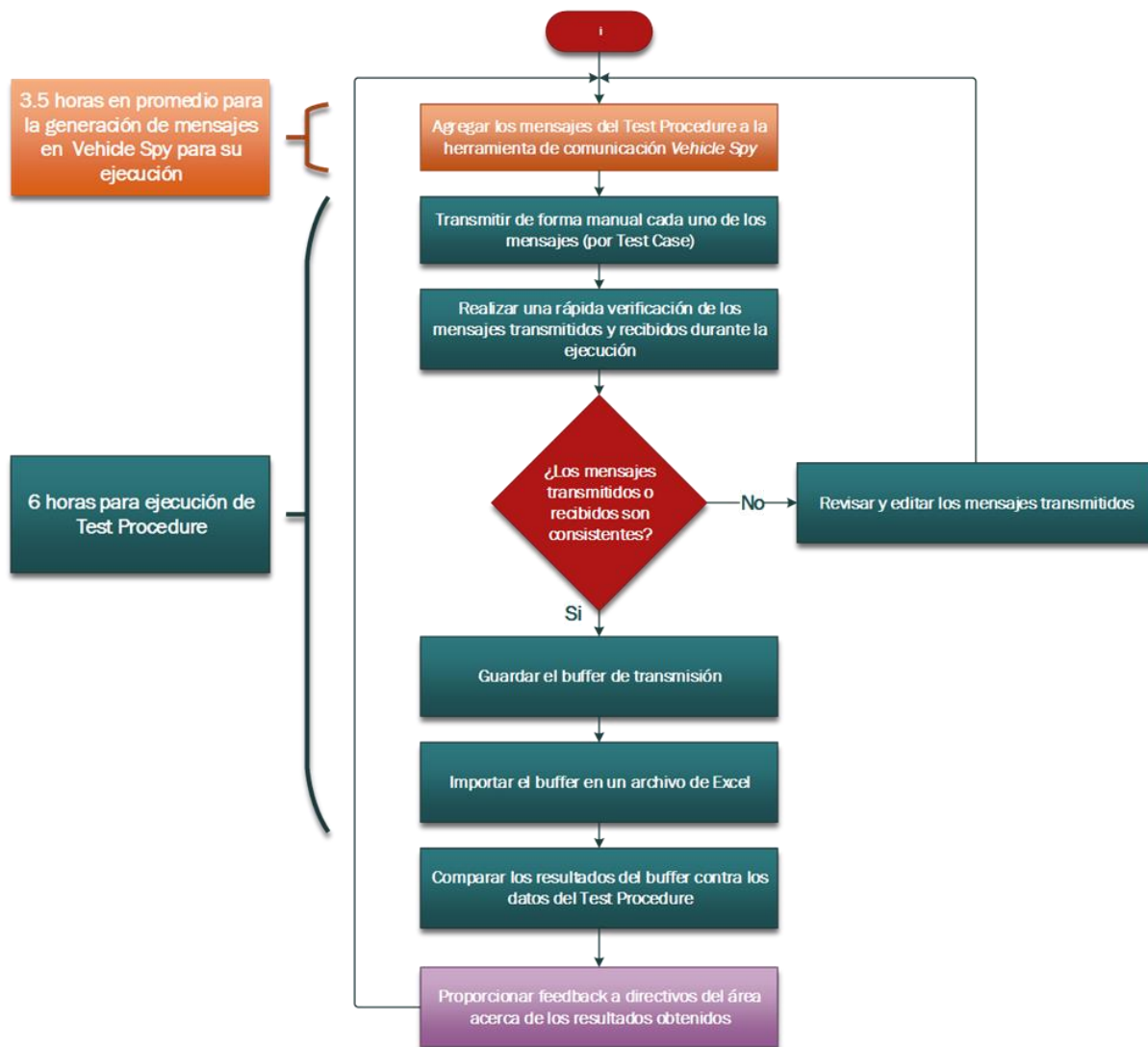
la verificación de resultados y, si existiera el caso de alguna respuesta errónea, identificar las posibles causas.

La Figura 29 ilustra de manera simplificada el proceso actual de validación de los *test procedures* que va desde la generación, ejecución, revisión de los mensajes para validar el código embebido referente a diagnóstico de las ECUs.



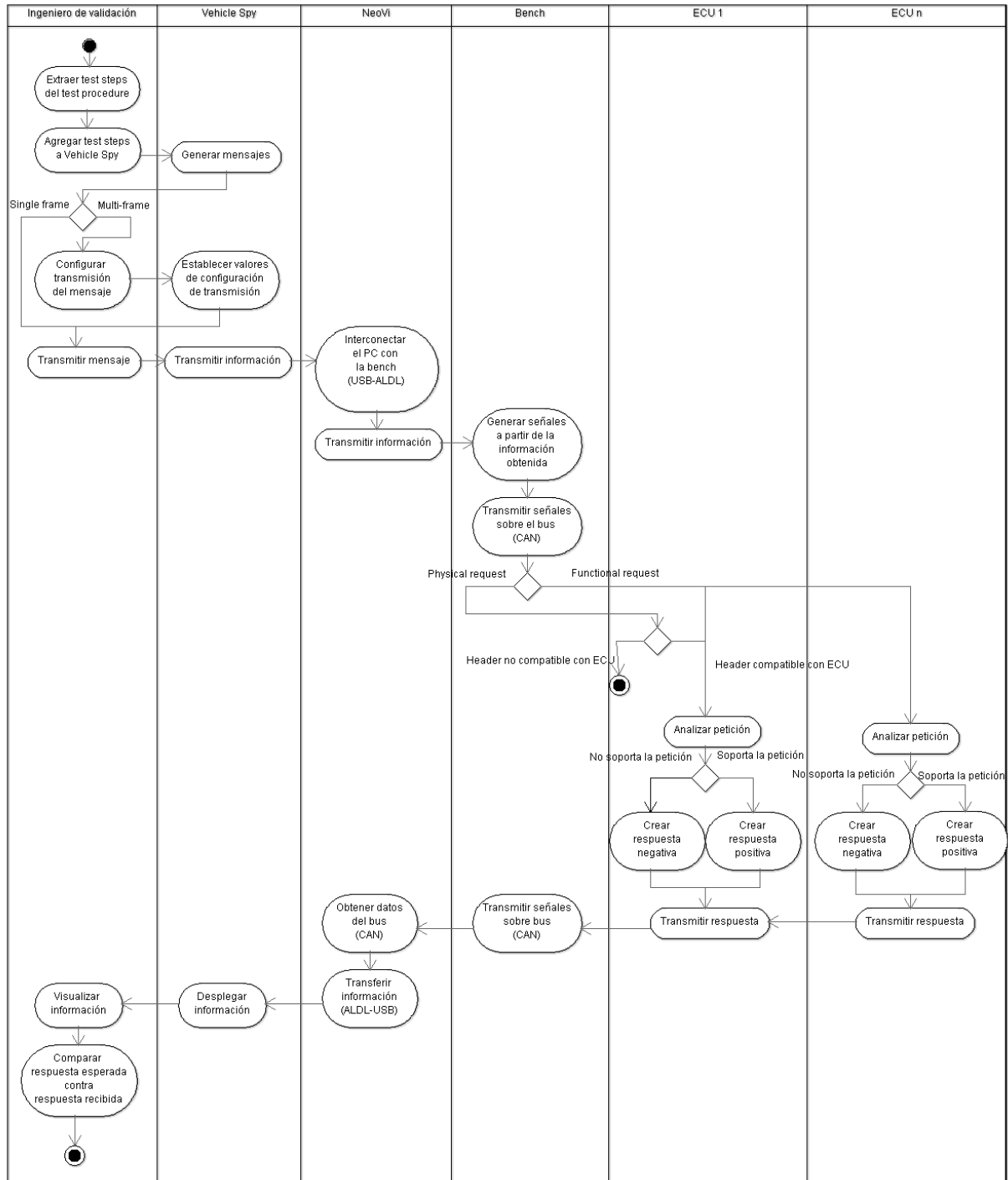
**Figura 29** – Diagrama de contexto del proceso de validación actual.

La Figura 30 muestra el diagrama con la representación simplificada para la visualización de las actividades en la ejecución de las pruebas de validación del software de diagnóstico; asimismo, se identifican las horas destinadas a dichas acciones.



**Figura 30** – Diagrama de flujo del proceso de validación actual.

Con el objetivo de representar las actividades involucradas en el proceso actual de validación de software de diagnóstico mediante *Vehicle Spy*, se ha generado un diagrama de actividades con la secuencia de pasos referente a este proceso (Figura 31). El lector notará que las actividades como el almacenamiento de los mensajes creados o el *buffer de datos* no se muestran en el diagrama de actividades, debido a que éstas son adicionales a la problemática existente.



**Figura 31** – Diagrama de actividades del proceso de validación actual.

En la Figura 32 se ejemplifican las actividades que ejecuta el ingeniero de validación del área de diagnóstico con *Vehicle Spy*, este diagrama ramifica las respectivas descripciones del proceso.

Debe entenderse que el uso de un modelo de caso es, simplemente, una representación del sistema actual, permitiendo identificar a los actores, al ambiente y las actividades realizadas, complementando los diagramas anteriores.



**Figura 32** – Diagrama de caso de uso del proceso de validación actual.

Como menciona Weitzenfeld (2005) la parte fundamental del modelo de casos de uso es la descripción textual detallada de cada uno de los actores y la funcionalidad

del sistema identificado, puesto que son documentos sumamente críticos debido a que a partir de ellos se desarrolla, o en este caso, se analiza el sistema existente.

A continuación, se expone una breve descripción por cada caso de uso que ha sido identificado en el *software* actual *Vehicle Spy* (Tabla 6 a 11).

**Tabla 6** – Descripción del caso de uso administrar mensajes.

<b>Nombre del caso de uso:</b>	Administrar mensajes.	
<b>Actores:</b>	Ingeniero de validación.	
<b>Descripción:</b>	Dado un <i>test procedure</i> previamente creado, el ingeniero de validación debe ejecutar los <i>test cases</i> mediante la transcripción de los mensajes ( <i>test steps</i> ) a transmitir ( <i>Tx</i> ).	
<b>Pre-condiciones:</b>	Debe existir al menos un <i>test procedure</i> que cuente con un <i>test case</i> compuesto de, por lo menos, un <i>test step</i> que contenga información referente al mensaje que será transmitido ( <i>Servicio UDS</i> ).	
<b>Flujo normal:</b>		<b>Flujo alternativo:</b>
1- El ingeniero de validación accede al apartado <i>Messages Editor</i> que se encuentra en la opción <i>Spy Networks</i> en la barra de herramientas.		
2- El ingeniero de validación da clic en el botón <i>Transmit</i> .	2. Si el ingeniero de validación no da clic en el botón <i>Transmit</i> , el sistema dejará la opción de <i>Receive</i> que está por defecto, creando mensajes para ser recibidos y no transmitidos. Termina el caso de uso.	
3- El ingeniero de validación da clic en el botón con signo ‘+’ que significa <i>Add Message</i> .	3. Si el ingeniero de validación no da clic en el botón con signo ‘+’ ( <i>Add Message</i> ) un nuevo renglón no será creado. Termina el caso de uso.	
4- El sistema crea un renglón con celdas señaladas como: <i>Key</i> , <i>Description</i> , <i>Type</i> , <i>Arb ID</i> , <i>Multi</i> , <i>Len</i> , <i>B1</i> , <i>B2</i> , <i>B3</i> , <i>B4</i> , <i>B5</i> , <i>B6</i> , <i>B7</i> , <i>B8</i> , <i>Src Node</i> y <i>Color</i> .		
5- El ingeniero de validación introduce los datos del mensaje		
a. El ingeniero de validación ingresa la descripción o nombre del mensaje.	a- Si el ingeniero de validación no ingresa una descripción del mensaje, el sistema asignará una descripción por defecto.	
b. El ingeniero de validación elige, en la celda del tipo de mensaje, <i>CAN Xtd 29 bit</i> .	b- Si el ingeniero de validación no elige el tipo de mensaje <i>CAN Xtd 29 bit</i> , el sistema asignará por defecto el tipo <i>CAN Std 11bit</i> limitando el número de bits que pueden ser ingresados en el campo de <i>header</i> . Termina el caso de uso.	
c. El ingeniero de validación ingresa el <i>header</i> del mensaje en formato hexadecimal ( <b>Ingresar header</b> ).	c- Si el ingeniero de validación no ingresa un <i>header</i> en el campo de <i>Arb ID</i> , el sistema establecerá un cero como <i>header</i> por defecto.	
d. El ingeniero de validación ingresa el <i>DLC</i> ( <b>Ingresar DLC</b> ) en el byte de datos 1 ( <i>B1</i> ).	d- Si el ingeniero de validación no ingresa ningún valor en el campo de <i>DLC</i> , el	

	sistema calculará el número de bytes usados por el mensaje y se lo asignará automáticamente.
e. El ingeniero de validación ingresa el servicio ( <b>Ingresar servicio UDS</b> ) en el byte de datos 2 ( <i>B2</i> ).	
f. El ingeniero de validación ingresa el sub-parámetro ( <b>Ingresar sub-parámetro</b> ) del servicio en el byte de datos 3 ( <i>B3</i> ) si es que el <i>servicio UDS</i> lo requiere.	f- Puede existir o no, un sub-parámetro en el mensaje, dependiendo si es requerido por el <i>servicio UDS</i> , si no existe entonces el byte destinado para el sub-parámetro puede ser utilizado para almacenar información del contenido del mensaje.
g. El ingeniero de validación agrega los datos correspondientes al contenido del mensaje a transmitir a partir del byte de datos cuatro al ocho ( <i>B4 - B8</i> ) ( <b>Ingresar contenido</b> ).	g- Si el mensaje a transmitir está compuesto por más de cuatro bytes de datos para el contenido, el ingeniero de validación selecciona en la casilla de <i>B8</i> del mensaje la opción <i>Extra Bytes...</i> , el sistema desplegará una tabla con celdas numeradas de 0 a 9 y de 0x a 929x, donde el primer elemento es el dato del byte ocho ( <i>B8</i> ), entonces el ingeniero de validación podrá agregar el contenido restante del mensaje dentro de cada una de las celdas como en los bytes <i>B4</i> al <i>B8</i> .
<b>Post-condiciones:</b>	
	El mensaje debe contener: una descripción, el tipo de transmisión, un <i>Arb ID</i> o <i>header</i> , un <i>DLC</i> , un <i>servicio UDS</i> , un sub-parámetro (si el <i>servicio UDS</i> lo requiere) y el contenido del mensaje.
	El tipo de transmisión debe ser <i>CAN Xtd 29 bit</i> .
	El <i>Arb ID</i> debe ser de 29 bits.
<b>Excepciones:</b>	
	Error al crear un nuevo renglón.

**Tabla 7** – Descripción del caso de uso transmitir mensajes.

<b>Nombre del caso de uso:</b>	Transmitir mensajes.
<b>Actores:</b>	Ingeniero de validación.
<b>Descripción:</b>	El ingeniero de validación debe transmitir los mensajes, que previamente fueron generados, hacia el <i>ECU</i> .
<b>Pre-condiciones:</b>	Debe existir por lo menos un <i>servicio UDS</i> previamente generado en el apartado <i>Messages Editor – Transmit</i> .
<b>Flujo normal:</b>	<b>Flujo alternativo:</b>
1- El ingeniero de validación accede al apartado <i>Messages Editor</i> que se encuentra en la opción <i>Spy Networks</i> en la barra de herramientas.	
2- El ingeniero de validación da clic en el botón <i>Transmit</i> .	2. Si el ingeniero de validación no da clic en el botón <i>Transmit</i> , no podrá ver los mensajes que fueron creados previamente puesto que el sistema desplegará por defecto la vista de mensajes <i>Receive</i> . Termina el caso de uso.
3- El sistema despliega los mensajes generados.	



4- El ingeniero de validación visualiza los mensajes previamente generados.	
5- El ingeniero de validación selecciona el mensaje a transmitir.	5. Si el ingeniero de validación no selecciona ningún mensaje, el sistema seleccionará por defecto el primer mensaje que se encuentre enlistado.
6- El ingeniero de validación se dirige al apartado <i>Setup for</i> (descripción del mensaje) donde se encuentra la configuración de <i>Multiframe Message</i> y selecciona la opción <i>ISO 15765-2 (Manejar mensajes multi-frame)</i> .	6. Si el ingeniero de validación no realiza la configuración del <i>Multiframe Message</i> , el sistema solo transmitirá <i>single frames</i> eliminando información adicional a los ocho bytes.
7- El ingeniero de validación da clic en el botón <i>Multiframe Setup...</i>	
8- El sistema despliega una nueva ventana para la configuración de la transmisión <i>multiframe</i> .	
9- El ingeniero de validación selecciona la pestaña <i>Flow Control Filter</i> y coloca el <i>Arbitration ID</i> que se utilizará para el <i>Flow Control Message (Manejar mensajes flow control)</i> , siendo el mismo <i>header</i> del mensaje que se va a transmitir.	9. Si el ingeniero de validación no realiza la configuración del <i>Flow Control Filter</i> , el sistema transmitirá el <i>flow control frame</i> con el <i>header</i> por defecto el cual es cero. Termina el caso de uso.
10- El ingeniero de validación da clic en el botón de <i>OK</i> para efectuar la configuración.	10. Si el ingeniero de validación no da clic en el botón <i>OK</i> , el sistema configurará los valores por defecto o los que fueron guardados anteriormente.
11- El ingeniero de validación accede al apartado <i>Tx Panel</i> que se encuentra en la opción <i>Spy Networks</i> en la barra de herramientas.	
12- El sistema despliega el (los) mensajes(s) previamente generados.	
13- El ingeniero de validación visualiza los mensajes previamente generados.	
14- El ingeniero de validación da clic en el botón <i>Tx</i> que se encuentra después de la descripción del mensaje a transmitir.	14. Si el ingeniero de validación no da clic en el botón <i>Tx</i> , el sistema no transmitirá el mensaje. Termina el caso de uso.
<b>Post-condiciones:</b>	Mensaje transmitido sobre el bus con protocolo <i>CAN</i> a la red intra-vehicular.
<b>Excepciones:</b>	Error interno al desplegar los mensajes generados. Volver al paso 1.
<b>Notas:</b>	Este caso de uso es ejecutado cada vez que se deba transmitir un <i>test step</i> o mensaje, esto a partir del paso 13 del caso de uso y una vez configurado el <b>manejo de multi-frames y flow control</b> por cada <i>test step</i> .

**Tabla 8** – Descripción del caso de uso generar archivo de datos.

<b>Nombre del caso de uso:</b>	Generar archivo de datos.	
<b>Actores:</b>	Ingeniero de validación.	
<b>Descripción:</b>	Un archivo con extensión <i>CSV</i> es generado con los mensajes que se encuentre en el <i>buffer de datos</i> del sistema.	
<b>Pre-condiciones:</b>	Debe existir por lo menos un mensaje (transmitido o recibido) en el <i>buffer de datos</i> .	
	La pantalla del sistema debe estar situada en el apartado de <i>Messages</i> .	
<b>Flujo normal:</b>		<b>Flujo alternativo:</b>
1-	El ingeniero de validación accede al apartado <i>Messages</i> .	
2-	El sistema despliega los mensajes contenidos en el buffer ( <b>Leer buffer de datos</b> ).	
3-	El ingeniero de validación da clic en el botón <i>Save</i> dentro del apartado <i>Messages</i> .	
4-	El sistema despliega una ventana que solicita el nombre del archivo con el cual será guardada la información, así como <i>checkboxes</i> que indican el formato del archivo, señales a guardar o si el archivo contendrá en su nombre la fecha y hora actual.	
5-	El ingeniero de validación elige el tipo de formato de archivo <i>CSV</i> para el análisis posterior.	5. Si el ingeniero de validación elige un tipo de formato distinto a <i>CSV</i> , el sistema guardará el archivo con el formato elegido.
6-	El ingeniero de validación da clic en el botón <i>Save</i> .	6. Si el ingeniero de validación no da clic en el botón <i>Save</i> , el sistema no generará ningún archivo.
7-	El sistema crea el archivo en la ruta por defecto.	
<b>Post-condiciones:</b>	Un archivo con extensión <i>CSV</i> es generado con la información correspondiente al <i>buffer de datos</i> .	
<b>Excepciones:</b>	Error al generar el archivo. Volver a ejecutar el caso de uso.	

**Tabla 9** – Descripción del caso de uso visualizar mensajes.

<b>Nombre del caso de uso:</b>	Visualizar mensajes.
<b>Actores:</b>	Ingeniero de validación.
<b>Descripción:</b>	Los mensajes que se encuentren circulando por la red intra-vehicular con protocolo <i>CAN</i> son leídos por el sistema y desplegados para su visualización.
<b>Pre-condiciones:</b>	La interfaz de comunicación <i>NeoVi</i> debe estar conectado al <i>CPU</i> donde se está ejecutando el sistema.
	La interfaz de comunicación <i>NeoVi</i> debe estar conectado en su otro extremo a una terminal <i>ALDL (bench)</i> .
	Deben existir mensajes (señales) en el bus con protocolo <i>CAN</i> .
<b>Flujo normal:</b>	<b>Flujo alternativo:</b>
1- El sistema, a partir del <i>NeoVi</i> , captura toda la información (señales) que se encuentra en el bus con protocolo <i>CAN</i> de la red intra-vehicular ( <b>Leer buffer de datos</b> ).	
2- El sistema despliega todos los mensajes, en tiempo real, divididos por <i>header</i> en la pantalla de <i>Messages</i> .	
3- El ingeniero de validación visualiza los mensajes desplegados.	
<b>Post-condiciones:</b>	El mensaje desplegado debe contener: un contador con el número de veces que un mensaje es recibido o transmitido (dividido por <i>header</i> ), el tiempo que existe de diferencia entre el penúltimo y el último mensaje capturado, la descripción del mensaje, el <i>Arb Id</i> o <i>header</i> , el <i>DLC</i> , el contenido del mensaje y la red sobre la que se obtuvo la información ( <i>SW CAN</i> o <i>HS CAN</i> ).
<b>Excepciones:</b>	Error en conexión USB – <i>ALDL</i> .

**Tabla 10** – Descripción del caso de uso guardar mensajes generados.

<b>Nombre del caso de uso:</b>	Guardar mensajes generados.	
<b>Actores:</b>	Ingeniero de validación.	
<b>Descripción:</b>	Los mensajes generados por el sistema son almacenados en un archivo.	
<b>Pre-condiciones:</b>	Por lo menos un mensaje creado por el ingeniero de validación.	
<b>Flujo normal:</b>		<b>Flujo alternativo:</b>
1-	El ingeniero de validación da clic en el botón de <i>File</i> ubicado en la barra de herramientas del sistema.	
2-	El sistema despliega las opciones: <i>Logon, New, Open..., Import, Save Platform Changes, Save, Save as, Explore Data Directory, Review Buffer... y Exit.</i>	
3-	El ingeniero de validación da clic en la opción <i>Save as</i> .	
4-	El sistema abre una nueva ventana con la dirección por defecto.	
5-	El ingeniero de validación selecciona el directorio y nombre donde desee guardar el archivo.	
6-	El ingeniero de validación da clic en botón <i>Save</i> .	6. Si el ingeniero de validación no da clic en el botón <i>Save</i> , el sistema no guardará la información.
7-	El sistema genera el archivo con formato <i>VS3</i> .	
<b>Post-condiciones:</b>	Un archivo con extensión <i>VS3</i> que contiene toda la configuración de los mensajes creados.	
<b>Excepciones:</b>	Error al generar el archivo.	

**Tabla 11** – Descripción del caso de uso leer mensajes previamente creados.

<b>Nombre del caso de uso:</b>	Leer mensajes previamente creados.	
<b>Actores:</b>	Ingeniero de validación.	
<b>Descripción:</b>	Lectura, a partir de un archivo existente con formato <i>VS3</i> , de los mensajes creados.	
<b>Pre-condiciones:</b>	Existencia de un archivo con formato <i>VS3</i> .	
<b>Flujo normal:</b>		<b>Flujo alternativo:</b>
1-	El ingeniero de validación da clic en el botón de <i>File</i> ubicado en la barra de herramientas del sistema.	1. El ingeniero de validación ubica el archivo con formato <i>VS3</i> mediante el explorador de sistema operativo. Continúa el paso seis.
2-	El sistema despliega las opciones: <i>Logon, New, Open..., Import, Save Platform Changes, Save, Save as, Explore Data Directory, Review Buffer... y Exit.</i>	
3-	El ingeniero de validación da clic en la opción <i>Open....</i>	
4-	El sistema abre una nueva ventana con la dirección por defecto.	
5-	El ingeniero de validación selecciona el directorio donde se encuentre el archivo.	
6-	El ingeniero de validación da doble clic en el archivo que desee abrir.	
7-	El sistema despliega los mensajes previamente creados en el apartado <i>Tx Panel</i> .	
<b>Post-condiciones:</b>	Mensajes previamente creados ahora desplegados en la ventana del sistema ( <i>Tx Panel y Messages Editor</i> ).	
<b>Excepciones:</b>	Error al abrir el archivo. Volver al paso 1.	

## 2.2 Requerimientos

### 2.2.1 Definición de requerimientos

A través de los años se ha podido constatar que los requerimientos son la pieza fundamental en un proyecto de desarrollo de software, ya que marcan el punto de partida para actividades como la planeación, (...) la definición de recursos necesarios (...) y mecanismos de control con los que se contará durante la etapa de desarrollo. Además, la especificación de requerimientos es la base que permite verificar si se alcanzaron o no los objetivos establecidos en el proyecto (...) (Arias, 2005).

El correcto entendimiento de los procesos, herramientas e información sobre los cuales se encuentra involucrado el (los) cliente(s) es crucial para que un proyecto de sistemas de la información pueda ser culminado con éxito. Desafortunadamente,

en ocasiones, ni siquiera el cliente mismo conoce lo que realmente necesita, generando gran incertidumbre para el ingeniero de requerimientos.

Con el afán de conocer las características, vocabulario, entorno, proceso y herramientas para la creación, transcripción y ejecución de *mensajes UDS*, fue necesario realizar un entrenamiento detallado de todos los aspectos que involucran la comunicación de estos servicios; dicho entrenamiento fue basado en cursos provistos por la empresa donde se realizó el proyecto. En adición al entrenamiento, se contó con el apoyo de un mentor, quien es el líder ingeniero de validación de comunicación de diagnóstico.

Para la captura y extracción de información del cliente se utilizó la entrevista, además de la aplicación de una encuesta al equipó de validación de diagnóstico (conformado por tres ingenieros). A partir de los resultados obtenidos, se establecieron los criterios funcionales y no funcionales, tomando en cuenta el objetivo principal el cual es la implementación de una herramienta de software de comunicación de datos que permita validar los casos de prueba generados a partir del estándar *ISO 14229*.

Por lo tanto, la herramienta de software a implementar deberá ser capaz de:

- Generar los mensajes que se transmitirán a partir de un archivo que fue previamente creado (archivo de *test procedure* con formato *XSLX*) el cual contiene los mensajes que serán transmitidos y las respuestas esperadas (evitando transcripciones).
- Transmitir mensajes con *headers* de 29 bits sobre un bus *CAN* (formato de tramas extendido) basado en direcciones propias de las *ECUs* de la empresa automotriz.
- Realizar la transmisión de *single* y *multi-frames* (conforme al estándar *ISO 15765-2:2011*, mensajes *single frame*, *first frame* y *consecutive frame*).
- Manejar mensajes *flow control* (conforme al estándar *ISO 15765-2:2011*) para la transmisión y recepción de mensajes *multi-frame*.
- Mantener sesiones de servicios activas dentro de la ejecución de *test steps* mediante mensajes de *tester present* (sesiones activas con duración de tres segundos por mensaje de *tester present*).

- Enviar mensajes a un módulo en específico (*ECM* o *TCM*) o a todos los módulos conectados a la misma red de comunicación de un vehículo a través de transmisiones *físicas* o *funcionales* (*physical* y *functional request*).
- Comparar las respuestas recibidas contra las respuestas esperadas, las cuales se encuentran establecidas por cada *test step* dentro del *test procedure*, colocando como resultado de la comparación un mensaje de despliegue (*PASS* o *FAIL*).
- Ejecutar los *test procedures* en un lapso de tiempo menor con respecto al tiempo actual, permitiendo la ejecución de los 16 *test procedures* disponibles dentro del marco de las 48 horas *bench* mensuales.
- Ejecutar todos los *test steps* de cada *test case* con un solo clic.
- Desplegar en pantalla el *buffer de datos* de la transmisión y recepción de mensajes.
- Ser ejecutada en el sistema operativo usado actualmente por la empresa automotriz.
- Utilizar *NeoVi Fire* como interfaz de comunicación para la transmisión y recepción de tramas *CAN*.
- Desplegar mensajes de ayuda para la interfaz gráfica en idioma inglés.

Los requerimientos antes mencionados fueron ratificados por el ingeniero líder de validación (Anexo A). Estos requerimientos fueron establecidos como marco de inicio para el desarrollo del proyecto, pese a esto, a lo largo de las iteraciones posteriores y la validación de cliente, surgieron nuevos requerimientos para mejorar la automatización. Estos requerimientos son enunciados posterior a las pruebas de validación dentro del capítulo III del presente reporte.

Dentro de las solicitudes para el desarrollo del software de comunicación no fue solicitado ningún requerimiento relacionado al diseño de la interfaz gráfica, exceptuando los mensajes desplegados en idioma inglés.

Con respecto a los términos de seguridad, no se realizó ninguna observación (requerimiento) debido a que el sistema operativo con el que cuenta la compañía

automotriz contiene niveles de seguridad altos y únicos para cada ingeniero de validación (credenciales), así como para usuarios no registrados dentro del ambiente operativo de la empresa.

Por el motivo anterior tampoco fue necesario ningún requerimiento que abarcara configuración de tipos de usuarios o jerarquías de los mismos debido a que los tres ingenieros de validación cuentan con el mismo perfil de acceso.

En términos de lenguaje de programación para un posible desarrollo de aplicación no existió ninguna restricción por parte del ingeniero de validación, sin embargo, debido a los términos y políticas de la empresa automotriz, no es posible implementar un nuevo *software* en alguno de los servidores existentes de la empresa ni el uso de *software libre* existente.

### 2.2.2 Comparación de software actual contra requerimientos

En comunicación con el líder ingeniero de validación surgieron comentarios como:

- *Vehicle Spy* no permite ejecutar todos los *test steps* de un *test case*, o toda una hoja (*sheet*) del *test procedure*.
- El proceso de transcribir los mensajes del *test procedure* al *Vehicle Spy* es muy tardado pues hay que colocar byte por byte cada uno de los *test steps*.
- La verificación de las respuestas recibidas es difícil porque se deben comparar los *test steps*, uno por uno, contra el *test procedure*. *Vehicle Spy* no dice si lo que estoy recibiendo es correcto o no.

Por tal motivo, se decidió hacer una tabla comparativa (Tabla 12) que muestre las deficiencias del *software* involucrado en el proceso actual (*Vehicle Spy*) en balance con los requerimientos establecidos y validados por el ingeniero líder de validación. Asimismo, en la Tabla 12 se puede apreciar que cuatro de los trece requerimientos no son cumplidos, es decir, un 30.77% del proyecto.

Este análisis permite comprender que el *software Vehicle Spy* es una herramienta poderosa para los objetivos que se pretenden alcanzar y por ese motivo es empleada para la ejecución de los *test procedures*; sin embargo, su eficiencia, dentro del proceso de validación del software embebido en los controladores que la empresa automotriz realiza en la ciudad de Toluca, no es la óptima para poder alcanzar



los métricos que la misma empresa establece, por lo que es necesario buscar una solución alterna a este *software*.

**Tabla 12** – Tabla comparativa entre *Vehicle Spy* y los requerimientos del cliente.

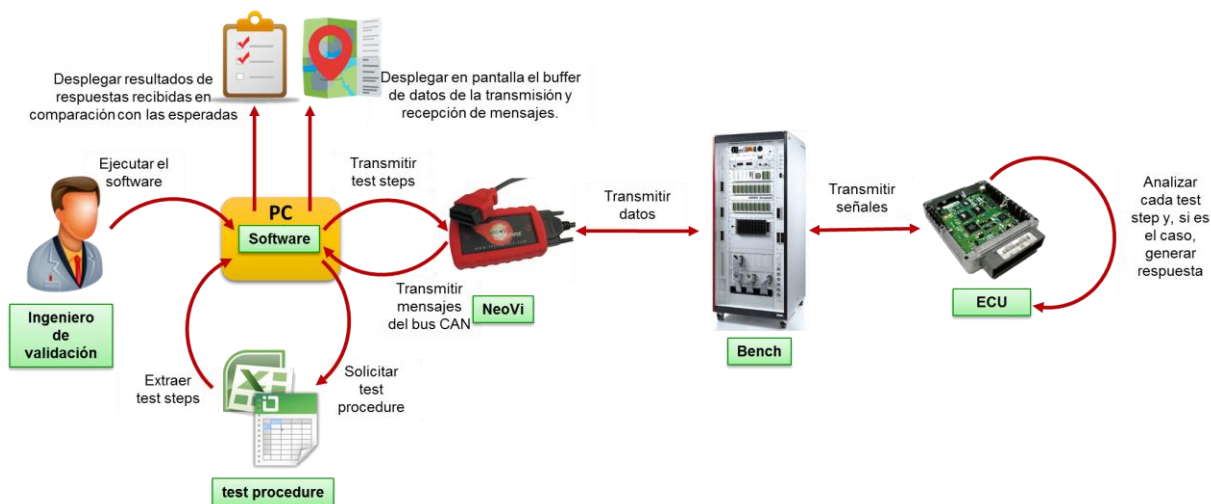
Expectativas del cliente	Medida	Objetivo	Vehicle Spy
Generar los mensajes que se transmitirán a partir de un archivo que fue previamente creado.	Binario	Si	No
Transmitir mensajes con <i>headers</i> de 29 bits sobre un bus CAN.	Binario	Si	Si
Realizar la transmisión de <i>single</i> y <i>multi-frames</i> .	Binario	Si	Si
Manejar mensajes <i>flow control</i> .	Binario	Si	Si
Mantener sesiones de servicios activas dentro de la ejecución de <i>test steps</i> mediante mensajes de <i>tester present</i> .	Binario	Si	Si
Enviar mensajes a un módulo en específico o a todos los módulos conectados a la misma red de comunicación de un vehículo a través de transmisiones físicas o funcionales.	Binario	Si	Si
Comparar las respuestas recibidas contra las respuestas esperadas colocando como resultado de la comparación un mensaje de despliegue ( <i>PASS</i> o <i>FAIL</i> ).	Binario	Si	No
Ejecutar los <i>test procedures</i> en un lapso de tiempo menor con respecto al tiempo actual.	Tiempo	48 horas <i>bench</i>	152 horas <i>bench</i>
Ejecutar todos los <i>test steps</i> de cada <i>test case</i> con un solo clic.	Binario	Si	No
Desplegar en pantalla el buffer de datos de la transmisión y recepción de mensajes.	Binario	Si	Si
Ser ejecutada en el sistema operativo usado actualmente por la empresa automotriz.	Binario	Si	Si
Utilizar <i>NeoVi Fire</i> como interfaz de comunicación para la transmisión y recepción de tramas CAN.	Binario	Si	Si
Desplegar mensajes de ayuda para la interfaz gráfica en idioma inglés.	Binario	Si	Si

### 2.2.3 Análisis y modelado de requerimientos

Se construyen modelos, diagramas y esquemas para que resalten o enfatizen ciertas características críticas de un sistema, al tiempo que ignoran otros aspectos del mismo (Pressman, 2010), pues únicamente rescatan aquello relacionado con las problemáticas o soluciones aportadas. Por lo tanto, tomando en consideración los requerimientos del proyecto y conociendo la estructura funcional de *Vehicle Spy*, se desarrollaron diferentes diagramas que permitieron modelar un *software* que cumpla con las necesidades del cliente.

El modelo de requisitos tiene como objetivo delimitar el sistema y capturar la funcionalidad que ofrecerá (Weitzenfeld, 2005). Así pues, identificando como principal objetivo la transmisión de mensajes, se muestra un diagrama de contexto general que indica los componentes necesarios para poder realizar el proceso de validación (

Figura 33) mediante los requerimientos del cliente; es decir, la relación entre el sistema y las entidades externas.



**Figura 33** – Diagrama de bloques del proceso de validación cumpliendo los requerimientos establecidos.

Una vez identificados los componentes externos y la relación que cumplirán los requerimientos del sistema, se desarrolló un diagrama de actividades desempeñadas por cada uno de los elementos externos (Figura 34).

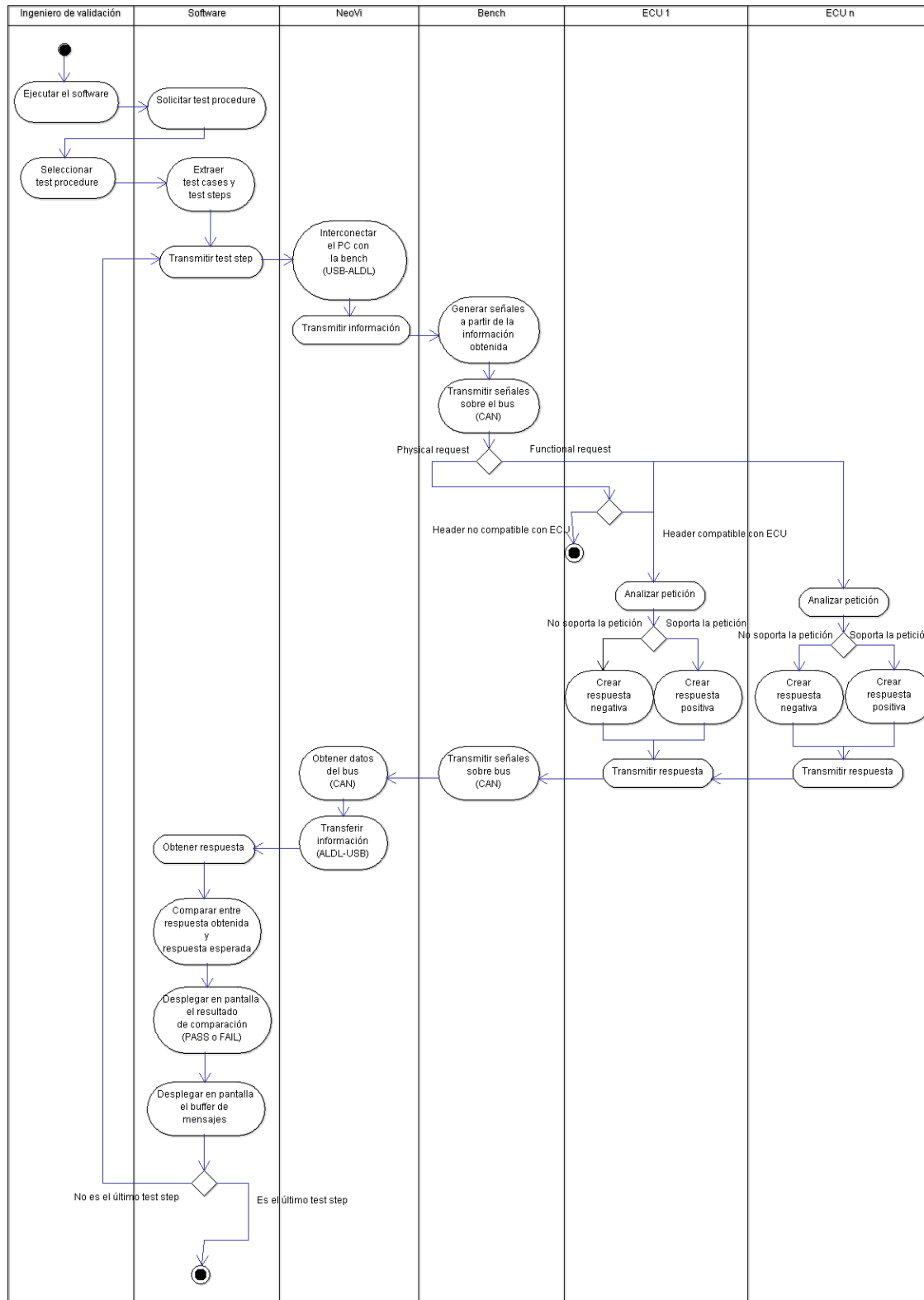


Figura 34 – Diagrama de actividades para el modelado de requerimientos.

Mostrando un enfoque relacionado al comportamiento y/o actividades que los componentes externos y el sistema deben desempeñar, se desarrolló un diagrama de secuencia general (Figura 35). Este diagrama contiene todas las actividades que, en conjunto, debe realizar el sistema para poder cumplir con los requerimientos solicitados y ejecutar la validación de un *test procedure* de manera completa.

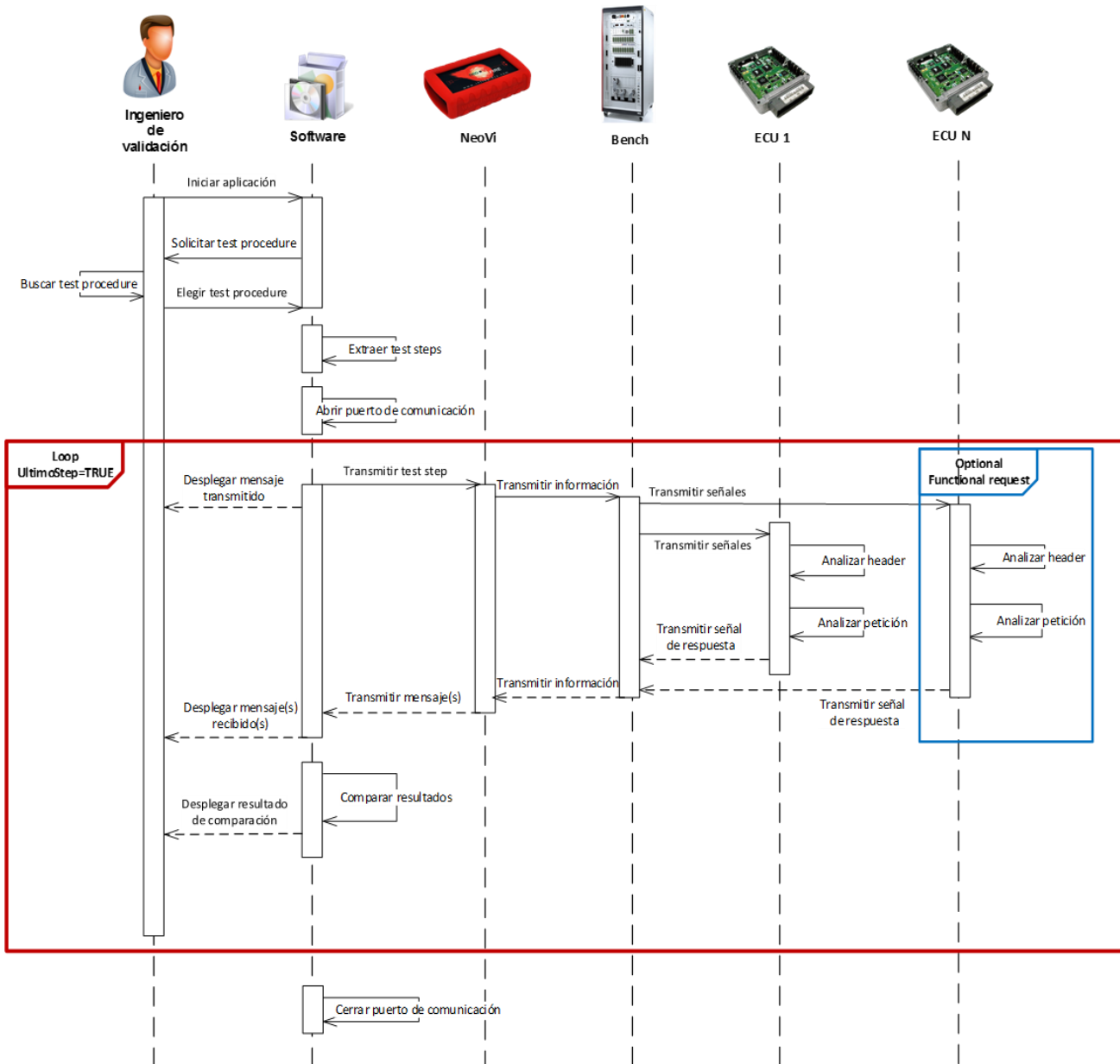


Figura 35– Diagrama de secuencia general para el modelado de requerimientos.

El siguiente diagrama de caso de uso muestra la funcionalidad a partir de los requerimientos mostrados anteriormente y su relación con el ingeniero de validación, semejante al diagrama de caso de uso de *Vehicle Spy*.



**Figura 36** – Diagrama de caso de uso para el modelado de requerimientos.

A continuación, se expone una breve descripción por cada caso de uso que ha sido identificado para el sistema que cumple con los requerimientos del cliente (Tabla 13 a 16).

**Tabla 13** – Descripción del caso de uso extraer *test steps*.

<b>Nombre del caso de uso:</b>	Extraer <i>test steps</i> .	
<b>Actores:</b>	Ingeniero de validación.	
<b>Descripción:</b>	Se deben extraer todos los <i>test steps</i> que se encuentran contenidos en el documento con formato <i>XLSX (test procedure)</i> .	
<b>Pre-condiciones:</b>	Debe existir por lo menos un <i>test step</i> dentro del documento.	
	Cada <i>test step</i> debe estar identificado por el número de <i>test case</i> al cual corresponde.	
	Cada <i>test step</i> debe estar identificado por un número que lo identifique dentro del <i>test case</i> .	
	Los <i>test steps</i> deben contener mensajes ( <i>Tx &amp; Rx</i> ) con estructura de: un <i>header</i> , un <i>DLC</i> , un <i>SID</i> , un sub-parámetro (si el <i>servicio UDS</i> lo requiere) y el contenido del mensaje.	
	El <i>header</i> de todos los <i>test steps</i> debe ser de 29 bits (sin importar que la dirección sea física o funcional) y siendo éstos los usados por la empresa automotriz.	
	Los mensajes contenidos en los <i>test steps</i> deben estar divididos con un espacio entre bytes.	
<b>Flujo normal:</b>		<b>Flujo alternativo:</b>
1-	El ingeniero de validación ejecuta el <i>software</i> .	
2-	El sistema solicita el documento <i>XSLX</i> del <i>test procedure</i> .	
3-	El ingeniero de validación ubica y selecciona el <i>test procedure</i> .	
4-	El sistema analiza el archivo ( <i>sheet</i> por <i>sheet</i> ) identificando el inicio de cada <i>test case</i> (Identificar <i>test case</i> ).	
5-	El sistema detecta el renglón con el número de <i>test case</i> y de <i>test step</i> .	
6-	El sistema identifica y extrae la descripción del mensaje a transmitir (Identificar descripción del mensaje <i>Tx &amp; Rx</i> ).	6 Si el campo de descripción del mensaje <i>Tx</i> está en blanco, el sistema extraerá una descripción vacía.
7-	El sistema identifica el inicio del mensaje a transmitir y extrae el <i>header</i> (Identificar tipo de request).	
8-	El sistema analiza el mensaje <i>Tx</i> y extrae el <i>DLC</i> (Identificar <i>DLC</i> del mensaje <i>Tx &amp; Rx</i> ).	8 Si el campo destinado para el <i>DLC</i> de <i>Tx</i> cuenta con un byte <b>1Y<sub>h</sub></b> (Y tomando cualquier valor hexadecimal), el <i>DLC</i> deberá de ser tomado desde el valor <b>Y</b> así como el siguiente byte ( <b>Obtener DLC para multiframe Tx</b> ).
9-	El sistema analiza los bytes después del <i>DLC</i> y extrae el contenido del mensaje <i>Tx</i> (Identificar contenido del mensaje <i>Tx &amp; Rx</i> ).	
10-	El sistema identifica y extrae la descripción del mensaje esperado a recibir (Identificar descripción del mensaje <i>Tx &amp; Rx</i> )	10 Si el campo de descripción del mensaje <i>Rx</i> está en blanco, el sistema extraerá una descripción vacía.

11- El sistema identifica el inicio del mensaje esperado a recibir y extrae el <i>header</i> ( <b>Identificar tipo de request</b> ).	
12- El sistema analiza el mensaje <i>Rx</i> y extrae el <i>DLC</i> ( <b>Identificar <i>DLC</i> del mensaje <i>Tx</i> &amp; <i>Rx</i></b> ).	12 Si el campo destinado para el <i>DLC</i> de <i>Rx</i> cuenta con un byte <b>1Y<sub>h</sub></b> (Y tomando cualquier valor hexadecimal), el <i>DLC</i> deberá de ser tomado desde el valor <b>Y</b> así como el siguiente byte ( <b>Obtener <i>DLC</i> para multiframe <i>Rx</i></b> ).
13- El sistema analiza los bytes después del <i>DLC</i> y extrae el contenido del mensaje <i>Rx</i> ( <b>Identificar contenido del mensaje <i>Tx</i> &amp; <i>Rx</i></b> ).	
<b>Post-condiciones:</b>	Todos los <i>test steps</i> identificados por el sistema y divididos por campos.
<b>Excepciones:</b>	Error al leer el archivo del <i>test procedure</i> . Volver a ejecutar el caso de uso.
<b>Nota:</b>	Este caso de uso se realiza para todos los <i>test steps</i> a partir del paso cinco, una vez ingresado el <i>test procedure</i> al sistema.

**Tabla 14** – Descripción del caso de uso transmitir *test steps*.

<b>Nombre del caso de uso:</b>	Transmitir <i>test steps</i> .
<b>Actores:</b>	Ingeniero de validación.
<b>Descripción:</b>	Una vez extraídos los <i>test steps</i> contenidos en el documento de <i>test procedure</i> se deben transmitir cada uno de estos.
<b>Pre-condiciones:</b>	<p>Debe existir por lo menos un <i>test step</i> previamente identificado y extraído por el sistema.</p> <p>La interfaz de comunicación <i>NeoVi</i> debe estar conectado al <i>CPU</i> donde se está ejecutando el sistema.</p> <p>La interfaz de comunicación <i>NeoVi</i> debe estar conectado en su otro extremo a una terminal <i>ALDL (bench)</i>.</p>
<b>Flujo normal:</b>	
1- El ingeniero de validación da clic en un botón de transmitir.	<b>Flujo alternativo:</b>
2- El sistema abre el puerto de comunicación al que se encuentra conectado el <i>NeoVi</i> .	
3- El sistema comienza a transmitir los <i>test steps</i> .	
a- Si el <i>test step</i> a transmitir está compuesto por más de siete bytes de contenido, el sistema divide el mensaje extraído.	a. El mensaje a transmitir es un <i>single frame</i> , solo se transmite el mensaje extraído ( <b>transmitir mensaje simple</b> ).
b- El sistema transmite el <i>first frame</i> y espera el <i>flow control</i> del <i>ECU</i> para poder transmitir los <i>consecutive frames</i> .	
4- El sistema finaliza la transmisión de los <i>test steps</i> y cierra el puerto de comunicación.	
<b>Post-condiciones:</b>	Todos los <i>test steps</i> identificados por el sistema son transmitidos hacia el bus con protocolo <i>CAN</i> vía <i>NeoVi</i> .

<b>Excepciones:</b>	Error al transmitir algún <i>test step</i> del <i>test procedure</i> . Posible fallo en la conexión. Volver a ejecutar el caso de uso.
<b>Nota:</b>	La ejecución de los <i>test steps</i> de un <i>test case</i> no debe rebasar el umbral de los tres segundos desde que un mensaje <i>tester present</i> es transmitido para poder realizar el <b>Manejo de tiempos de ejecución del <i>tester present</i></b> .

**Tabla 15** – Descripción del caso de uso Desplegar mensajes del bus CAN.

Nombre del caso de uso:	Desplegar mensajes del bus CAN.	
Actores:	Ingeniero de validación.	
Descripción:	Los mensajes que son transmitidos y recibidos por parte del ingeniero de validación hacia la(s) ECU(s) y que se encuentran en el bus con protocolo CAN, son desplegados.	
Pre-condiciones:	La interfaz de comunicación NeoVi debe estar conectado al CPU donde se está ejecutando el sistema.	
	La interfaz de comunicación NeoVi debe estar conectado en su otro extremo a una terminal ALDL (bench).	
	Haber sido transmitido o recibido algún mensaje sobre el bus con protocolo CAN.	
Flujo normal:		Flujo alternativo:
1-	El sistema, a partir del NeoVi, captura los mensajes trasmitidos y recibidos que se encuentran en el bus con protocolo CAN de la red intra-vehicular (Leer buffer de datos).	
2-	El sistema despliega, en pantalla, el (los) mensaje(s) transmitido(s) y/o recibido(s).	
3-	El ingeniero de validación visualiza los mensajes desplegados.	
Post-condiciones:	Mensaje(s) desplegados en pantalla.	
Excepciones:	No se despliega mensaje recibido debido al tipo de request (referencia ISO-14229).	



**Tabla 16** – Descripción del caso de uso comparar resultados obtenidos.

<b>Nombre del caso de uso:</b>	Comparar resultados obtenidos.
<b>Actores:</b>	Ingeniero de validación.
<b>Descripción:</b>	Un mensaje de <i>PASS</i> o <i>FAIL</i> es desplegado una vez obtenido una respuesta por parte de la <i>ECU</i> ante un <i>request</i> del ingeniero de validación.
<b>Pre-condiciones:</b>	Mensaje enviado por parte del ingeniero de validación ( <i>Tx</i> ).
	Posible existencia de mensaje de respuesta ( <i>Rx</i> ) por parte del <i>ECU</i> .
<b>Flujo normal:</b>	<b>Flujo alternativo:</b>
1- El sistema lee el buffer de datos ( <b>Leer buffer de datos</b> ) y obtiene la respuesta que emitió la <i>ECU</i> ante algún <i>request</i> que haya creado el ingeniero de validación.	
2- El sistema compara el mensaje de respuesta ( <i>Rx</i> ) contra la respuesta esperada que previamente fue identificada y extraída del <i>test procedure</i> .	2.1. Si el sistema no recibió ninguna respuesta, el mensaje recibido será contemplado como vacío.  2.2. La respuesta esperada extraída del <i>test procedure</i> puede estar vacía.
3- El sistema despliega un mensaje <i>PASS</i> si el mensaje recibido es igual al mensaje esperado, en caso contrario desplegará el mensaje <i>FAIL</i> .	3.1. Si la respuesta esperada está vacía y el mensaje recibido contiene alguna respuesta, el mensaje desplegado será <i>FAIL</i> .  3.2. Para el caso en el que se tenga una respuesta esperada pero la <i>ECU</i> no emita ningún mensaje se desplegará el mensaje <i>FAIL</i> .
4- El ingeniero de validación visualiza la respuesta desplegada determinada de la comparación de las respuestas.	
<b>Post-condiciones:</b>	<i>Test steps</i> con resultado de comparación entre el mensaje esperado y el recibido ( <i>PASS</i> o <i>FAIL</i> ).
<b>Excepciones:</b>	Existencia de algún error en la transmisión o recepción de mensajes. Despliegue de mensaje <i>ERROR</i> .

### 2.3 Análisis de alternativas de solución

A continuación se hace una descripción de las herramientas de *software* existentes que podrían ser útiles para el cumplimiento del objetivo de proyecto desarrollado. Asimismo, se comparan las ventajas y desventajas que tienen cada una de estas herramientas de *software* en contraste con los requerimientos establecidos y validados por el líder ingeniero de comunicación de diagnóstico.

### 2.3.1 Vehicle Spy – Function blocks

La herramienta que actualmente utiliza el área de validación de diagnóstico de la empresa automotriz donde se desarrolló el presente reporte, cuenta con una función que no había sido probada. Su nombre, *function blocks*.

Los *functions blocks* brindan la oportunidad de ejecutar de manera secuencial, y con un solo clic, los mensajes que han sido previamente creados en la suite de *Vehicle Spy*. Descritos por *Vehicle Spy* como *Triggers* que permiten la captura de datos ya que mediante estructuras de control (IF, ELSE, WHILE, FOR, entre otras) permiten realizar la transmisión de mensajes CAN y a partir de una respuesta esperada pueden generar comportamientos.

Step	Description	Value	Comment
1	Set Value	{Transmission Counter :sig0-index(0)} = {Transmission Counter :sig0-index(0)} + 1	// Increment transmission counter.
2	If	{Transmission Counter :sig0-index(0)} mod 3 = 0	// If counter divides evenly by 3, send Message Alpha.
3	Transmit	Message Alpha	
4	Pause	Successfully transmitted Message Alpha	// Inform user.
5	Else If	{Transmission Counter :sig0-index(0)} mod 3 = 1	// If the counter divided by 3 leaves a remainder of 1, send Message Beta.
6	Transmit	Message Beta	
7	Pause	Successfully transmitted Message Beta	// Inform user.
8	Else		// Otherwise, the counter divided by 3 leaves a remainder of 2, so send Message Gamma.
9	Transmit	Message Gamma	
10	Pause	Successfully transmitted Message Gamma	// Inform user.
11	End If		
12	Wait For	5.000000 sec	// Wait before repeating.

**Figura 37** – Ejemplo de un *function block*.

La principal ventaja de esta funcionalidad es que se pueden ejecutar todos los mensajes creados en la suite de *Vehicle Spy* con un solo comando lo que permitiría reducir el tiempo de ejecución más no el tiempo de transcripción debido a que los mensajes a transmitir aún deben de ser transcritos del *test procedure* a esta herramienta.

Otro punto importante a tomar en cuenta es que los *functions block* permiten reconocer una respuesta esperada y así desplegar un mensaje, sin embargo, este mensaje no es desplegado en el *buffer de datos* en la visualización en tiempo real ni en el reporte generado a partir de este mismo *buffer*. La empresa automotriz, y en específico el área donde se desarrolló el reporte, ya cuenta con la licencia de *Vehicle Spy* para poder utilizar el software, por lo que no se requiere un pago por compra de licencia ni actualizaciones por parte del área de diagnóstico.

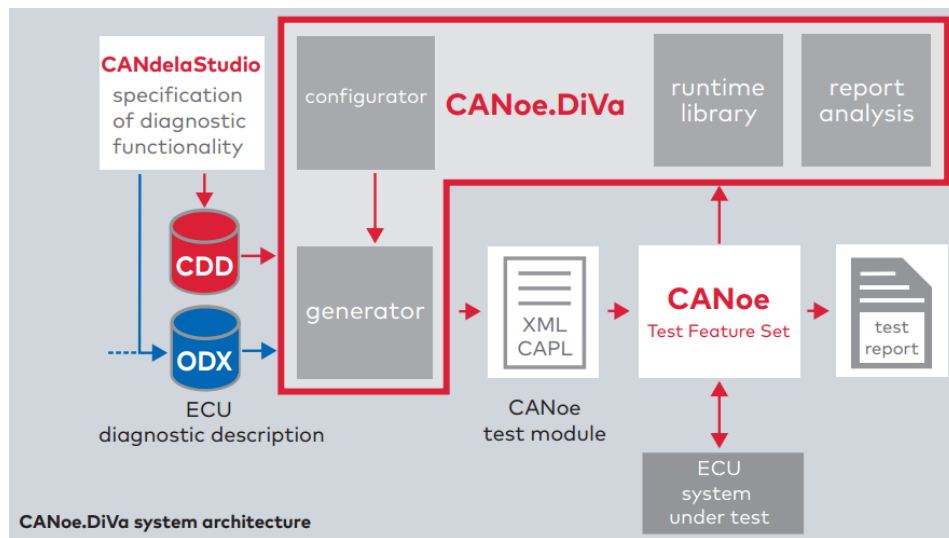
### 2.3.2 Vector Cantech - CANoe DiVa

*DiVa* es una extensión de *CANoe* para el *testing* automatizado de implementaciones de software de diagnóstico en *ECUs*. Test cases son generados con base en una descripción de diagnóstico de la *ECU* que se puede encontrar en formato *CANdela* u *ODX* (Open Diagnostic data Exchange, por sus siglas en inglés) (Vector, 2016).

*DiVa* permite la generación y ejecución de test cases a partir de archivos compatibles (diagnostic specifications) con la suite de *CANoe*, la cual generará un archivo que contendrá todos los mensajes *CAN* que serán transmitidos (contenidos en servicios *UDS*).

La interfaz comunicación utilizada por *diva* puede ser cualquier interfaz de *CANoe* que soporte el protocolo *CAN* (por ejemplo *VN1600* que soporta *CAN*, *CAN FD*, *LIN*, *K-Line*, etc.), y en este caso, que tenga una conexión *USB-ALDL*. No obstante, el uso de *NeoVi* no es compatible con esta *suite*.

*DiVa* permite realizar la ejecución de los servicios contenidos en las especificaciones de diagnóstico mediante un solo clic, emitiendo respuestas del resultado de dicha ejecución de manera simbólica (palomas para TRUE o taches para FALSE). La arquitectura de *DiVa* se ilustra en la Figura 38.



**Figura 38** – Arquitectura del sistema *CANoe.DiVa* (Vector, 2016).

El área de diagnóstico de la empresa no cuenta con ningún elemento (software & hardware) que permita realizar la validación de los *test procedures* mediante esta herramienta por lo que a continuación se despliegan los costos brindados por parte del

proveedor (*Vector Cantech*) para poder efectuar dicha actividad (Tabla 17). Debido a la actual variación del peso mexicano con respecto al dólar, los costos se indican en moneda Estadounidense (23 de Mayo, 2016).

**Tabla 17** – Costo de implementación de *Diva*.

Herramienta	Descripción	Precio (USD)
<b>CANoe Standard v9</b>	Herramienta de software para la creación y ejecución de simulación, comunicación, análisis y <i>testing</i> de <i>ECUs</i> en sistemas distribuidos.	\$9,172.50
<b>CANoe DiVa v4.0</b>	Extensión de <i>CANoe</i> que brinda una generación y ejecución automatizada de <i>test cases</i> basados en una descripción de diagnóstico in formato <i>CDD</i> u <i>ODX</i> .	\$6,535.00
<b>Mantenimiento de CANoe</b>	Actualizaciones de software. 1 vez al año. Condicionado a tener la licencia de software actual.	\$2,201.00
<b>Mantenimiento de DiVa</b>	Actualizaciones de software. 1 vez al año.	\$1,176.00
<b>CANdelaStudio Standard v8.5</b>	Herramienta para editar descripciones de diagnóstico de <i>ECUs</i> . Primer año de mantenimiento gratis.	\$7,495.00
<b>Mantenimiento de CANdelaStudio Standard</b>	Actualizaciones de software dentro del periodo de mantenimiento. 1 vez al año.	\$321.00
<b>VN1630A CAN/LIN Network interface</b>	Interface <i>USB</i> para protocolos <i>CAN</i> , <i>LIN</i> , <i>J1708</i> , <i>K-Line</i> e <i>I/O</i> (4+1 canales).	\$931.00
<b>CANpiggy 1051cap</b>	Módulo <i>Tx (transceiver)</i> con capacitiva de <i>HS CAN</i> .	\$332.50
	<b>Total:</b>	<b>\$28,164.00</b>

Dentro de los requerimientos para poder ejecutar *DiVa* se encuentran (Vector, 2016):

- Contar con cualquier variante de la suite *CANoe* que esté licenciada con la versión 9.0.
- El sistema operativo puede ser Windows en sus versiones 8/7/Vista/XP SP3.
- Un procesador Intel Core 2 Duo de 2.6 GHz (mínimo).
- Memoria RAM de 1GB (mínimo).
- Capacidad en HD de 1GB (mínimo).
- Resolución de pantalla de 1024x768 (mínimo).

### 2.3.3 Herramienta de ejecución automática de pruebas (In-house software)

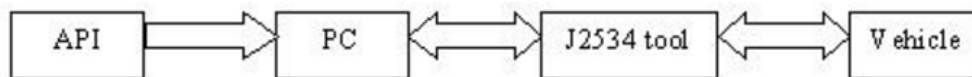
Por motivos de confidencialidad el nombre del software no puede ser expuesto, por lo que a partir de este momento se le referirá como *In-house software*, sin embargo, se da una descripción técnica acerca de su funcionamiento.

*In-house software* fue desarrollado dentro de la compañía por un equipo especializado ubicado en Europa con la finalidad de poder realizar el *testing* de software y hardware relacionado a *ECUs*.

*In-house software* se encuentra compuesto por dos partes: la primera es una interfaz gráfica que permite configurar comandos para la transmisión de mensajes, pruebas I/O, cálculos de respuestas obtenidas, entre otras actividades. Estos comandos están compuestos por características como una descripción, un mensaje a transmitir (*Tx*), una respuesta recibida (*Rx*), una respuesta esperada (*expected Rx*), información acerca del resultado (*Passed* o *Failed*), así como el tiempo de ejecución del comando (teniendo en cuenta el tiempo de respuesta).

La segunda parte es un motor de comunicación (cerebro de la aplicación), el cual permite realizar la comunicación entre la aplicación con el controlador mediante canales de comunicación como *CAN*. Este motor se encuentra creado en *Python* y se intercomunica con la interfaz gráfica mediante un protocolo *TCP-IP*.

Por su parte, *Python* realiza la comunicación serial mediante el estándar *J2534* el cual fue diseñado por *SAE* con el propósito de reprogramar (*flash programming*) las *ECUs* de los vehículos (Figura 39).



**Figura 39** – Configuración de *J2534* (Kvaser, 2016).

El motivo de usar el estándar *J2534* para realizar la comunicación entre un *tester* y la *ECU* es debido a que este estándar soporta los protocolos *ISO 9141*, *ISO 14230*, *J1850*, *ISO 11898*, *ISO 15765*, *SAE J2610* y *J1939*, lo que permite realizar el diagnóstico de vehículos.

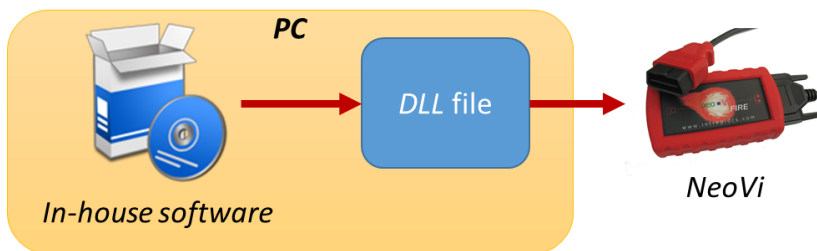
El estándar *J2534* define la conexión entre un dispositivo de hardware y la red intra-vehicular mediante una aplicación de software (*API*) que controle el dispositivo (Drew Technologies, Inc., 2003).

Debido a que esta herramienta fue desarrollada y se encuentra en constante actualización no es necesario el pago de ninguna licencia y/o actualización de *suites*. Sin embargo, es importante mencionar que *In-house software* contenía el diagnóstico del protocolo *CAN* solamente para *headers de 11 bits*.

En cuanto a los requerimientos, *In-house software* necesita:

- Sistema operativo superior o igual a Windows XP SP3.
- Python 2.6v para 64 bits.
- Especificaciones del estándar *J2534* (ejemplo, icsneo40.dll).
- Procesador de 500 MHz (mínimo).
- Memoria RAM de 3 GB (mínimo).
- Capacidad en HD de 3 GB (mínimo).
- Resolución de pantalla de 1024x576 (mínimo).

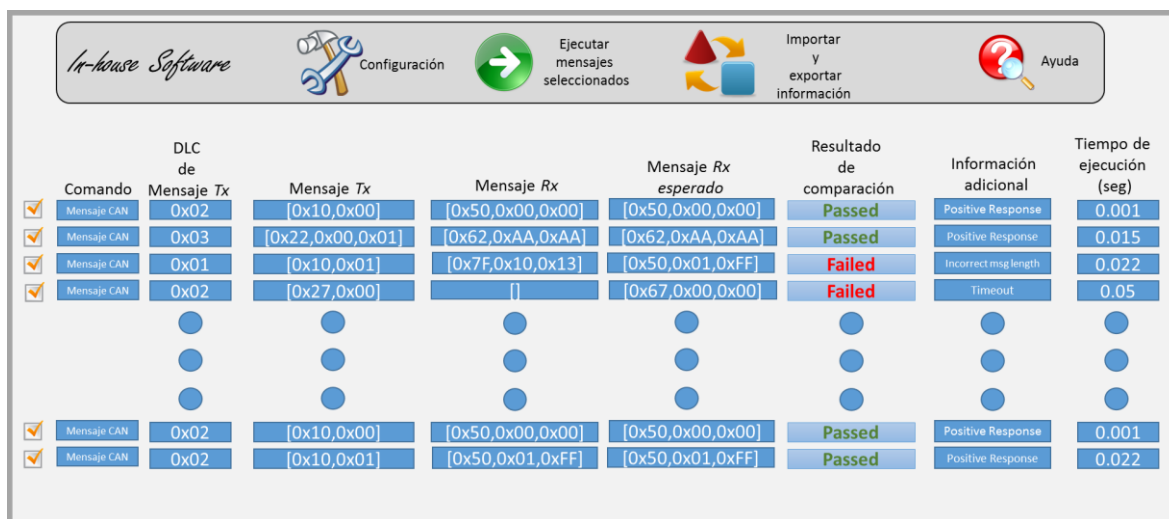
Además, para poder realizar la transmisión de mensajes, es necesario establecer la configuración de la interfaz que se usará (por ejemplo *NeoVi Fire*), detectando el archivo *DLL* que contiene la especificación del estándar *J2534* para establecer un canal de comunicación (Figura 40).



**Figura 40** – Archivo *DLL* para comunicación con interfaz *NeoVi*.

Como ya se mencionó anteriormente, *In-house software* realiza la comunicación de mensajes *CAN* a través de su interfaz gráfica y *Python*. Los mensajes que se transmiten y las respuestas esperadas así como las que son recibidas (*Tx*, *Rx* y *Rx expected*) son expresados en formato hexadecimal, además, *In-house software* ejecuta una comparación automática entre la respuesta recibida y la respuesta esperada desplegando un mensaje de dicha comparación.

*In-house software* permite realizar la ejecución de los mensajes *CAN* que se deseen transmitir, seleccionando el compendio de mensajes y transmitiéndolos en conjunto con un solo comando (clic). La respuesta de cada mensaje es recibida automáticamente y desplegada en el apartado de mensaje recibido (Figura 41).



Comando	DLC de Mensaje Tx	Mensaje Tx	Mensaje Rx	Mensaje Rx esperado	Resultado de comparación	Información adicional	Tiempo de ejecución (seg)
<input checked="" type="checkbox"/> Mensaje CAN	0x02	[0x10,0x00]	[0x50,0x00,0x00]	[0x50,0x00,0x00]	Passed	Positive Response	0.001
<input checked="" type="checkbox"/> Mensaje CAN	0x03	[0x22,0x00,0x01]	[0x62,0xAA,0xAA]	[0x62,0xAA,0xAA]	Passed	Positive Response	0.015
<input checked="" type="checkbox"/> Mensaje CAN	0x01	[0x10,0x01]	[0x7F,0x10,0x13]	[0x50,0x01,0xFF]	Failed	Incorrect msg length	0.022
<input checked="" type="checkbox"/> Mensaje CAN	0x02	[0x27,0x00]	[ ]	[0x67,0x00,0x00]	Failed	Timeout	0.05
<input checked="" type="checkbox"/> Mensaje CAN	0x02	[0x10,0x00]	[0x50,0x00,0x00]	[0x50,0x00,0x00]	Passed	Positive Response	0.001
<input checked="" type="checkbox"/> Mensaje CAN	0x02	[0x10,0x01]	[0x50,0x01,0xFF]	[0x50,0x01,0xFF]	Passed	Positive Response	0.022

**Figura 41** – Ejemplo de resultado de ejecución de *In-house software* (interfaz gráfica).

Como ejemplo se muestra en las figuras de la aplicación *In-house* solamente funcionalidad relevante a la transmisión de mensajes *CAN*, sin embargo, esta aplicación contiene más características para el *testing* de *I/O*, cálculos, estructuras de control, y controles de comportamiento del mismo *testing*.

*In-house software* cuenta con un comando de configuración para poder establecer las direcciones de los *ECUs* a los cuales se les enviarán el compendio de mensajes que se encuentren por debajo de esas direcciones (Figura 42).

*In-house software* permite agrupar a todos aquellos *test steps* que formen parte del comportamiento de una prueba (*test case*) y así poder obtener datos estadísticos como el número de *test steps* que “pasaron o fallaron” y desplegando el resultado total del *test case*, es decir, si algún *test step* falla entonces el resultado del conjunto de mensajes se desplegará como *failed*, pero si todos *test steps* pasan, un mensaje de *passed* será desplegado. De la misma forma tiempo total de ejecución del *test case* se mostrará (Figura 42).

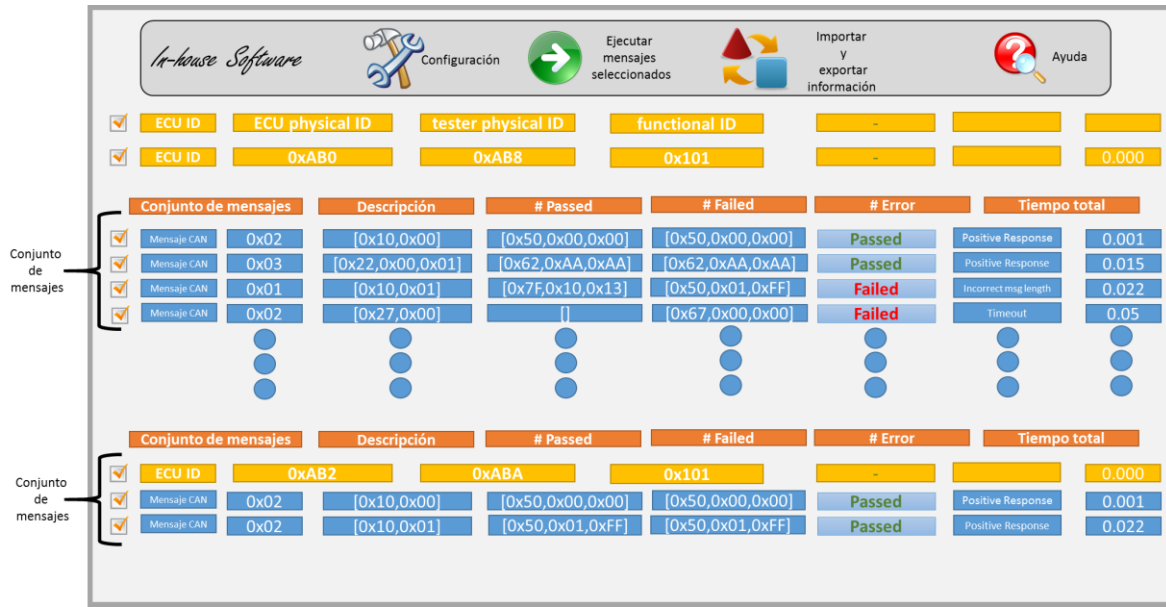


Figura 42- Configuración de headers y conjunto de mensajes de In-house software.

También, *In-house software* cuenta con la función de exportar e importar la información contenida en los *test steps* y los resultados obtenidos una vez ejecutados, a través de archivos con formato *TXT*. La estructura de este archivo es mostrado en la Figura 43 como una referencia sin embargo, por motivos de confidencialidad, esta estructura no es completamente la utilizada por la herramienta.

```

1  </>
2  </>
3  ECUID → 0xAB0 → 0xAB8 → 0x101 → 0.0 </>
4  </>
5  Conjunto de mensajes Descripción → 2 → 2 → 0 → 0.5 </>
6  Mensaje CAN Descripción 1 → 0x02 → [0x10,0x00] → [0x50,0x00,0x00] → [0x50,0x00,0x00] → Passed → Positive Response → 0.001 </>
7  Mensaje CAN Descripción 2 → 0x03 → [0x22,0x00,0x01] → [0x62,0xAA,0xAA] → [0x62,0xAA,0xAA] → Passed → Positive Response → 0.015 </>
8  Mensaje CAN Descripción 3 → 0x01 → [0x10,0x01] → [0x7F,0x10,0x13] → [0x50,0x01,0xFF] → Failed → Incorrect msg length → 0.022 </>
9  Mensaje CAN Descripción 4 → 0x03 → [0x27,0x00] → [ ] → [0x67,0x00,0x00] → Failed → Timeout → 0.05 </>
10 </>
11 </>
12 </>
13 </>
14 ECUID → 0xAB2 → 0xABA → 0x101 → 0.0 </>
15 </>
16 Conjunto de mensajes Descripción → 2 → 2 → 0 → 0.5 </>
17 Mensaje CAN Descripción n-1 → 0x02 → [0x10,0x00] → [0x50,0x00,0x00] → [0x50,0x00,0x00] → Passed → Positive Response → 0.001 </>
18 Mensaje CAN Descripción n → 0x02 → [0x10,0x01] → [0x50,0x01,0xFF] → [0x50,0x01,0xFF] → Passed → Positive Response → 0.001 </>

```

Figura 43 – Ejemplo de archivo importado o exportado con contenido similar al de la Figura 42.

*In-house software* brinda la oportunidad de transmitir mensajes *CAN* con contenido de *tester present* para poder mantener sesiones activas, no obstante, la



duración de este mensaje es solamente de tres segundos por lo que todo mensaje transmitido después de este umbral no estará contenido en la sesión activa que aporta el *tester present*. De esta forma, no existe algún comando que permita detener la sesión que activa el *tester present*, por lo que cualquier mensaje dentro de los tres segundos tendrá un comportamiento formado por la sesión.

### 2.3.4 Comparación del software existente

La Tabla 18 muestra la comparación de las herramientas existentes previamente mostradas contra los requerimientos del ingeniero de validación. Una de las columnas de esta tabla muestra una solución desarrollada desde cero sin hacer mención al lenguaje de programación que puede ser usado.

El criterio de evaluación fue tomando con base en la unidad de medición binaria con la que cuentan algunos requerimientos. Para aquellos requerimientos que no cuentan con esta medida (tiempo) se tomó en cuenta el tipo de ejecución. Además de los requerimientos establecidos por el ingeniero de validación se establecieron puntos críticos que deben tenerse en cuenta para el diseño, desarrollo e implementación de una solución, como lo son el tiempo y costo además del soporte brindado.

Los criterios relacionados con el costo se establecieron de la siguiente forma: si no existe un costo de implementación (“ninguno”) el valor será de 0, en contraparte para la existencia de implementación su valor será determinado por el grado de trabajo (tiempo, esfuerzo y dinero) que requiere dicho costo teniendo criterios de bajo (-1), medio (-2) y alto (-3).

Analizando la comparación entre el software que permiten realizar la comunicación de mensajes *CAN*, se pudo determinar que existe un empate entre tres herramientas que cuentan con un mayor número de requerimientos que se pueden cumplir.

Una empresa que puede desarrollar un software el cual satisfaga las necesidades del cliente de forma predecible y puntual con el uso eficiente y efectivo de recursos tanto humanos como materiales, tiene un negocio sostenible (Booch, Rumbaugh y Jacobson 2006), por esta razón el costo de implementación de la herramienta de *Vector – DiVa* es alto en comparación con las otras herramientas de *software* analizadas, lo que la descartaría de ser una posible ganadora.

Tomando en cuenta el resultado que arrojó la opción de *Vehicle Spy- function blocks* se puede identificar que esta herramienta no permite realizar la extracción de

información a partir de un archivo creado, por lo que el tiempo de transcripción de los *test procedures* seguiría siendo de la misma forma que el del software que actualmente se utiliza. Además, aunque los *functions blocks* permitan realizar una ejecución automática de los *test steps*, la comprobación no se realiza de forma automática por lo que se tendría que identificar cada respuesta que el *ECU* transmite para verificar el correcto funcionamiento del mismo.

Por su parte *In-house software* no cumple con uno de los requerimientos críticos que es la transmisión de mensajes con *headers* de 29 bits. Específicamente para este requerimiento se realizó una junta (vía remota) con el grupo de desarrolladores de esta herramienta (quienes se encuentran en Europa) para determinar la posibilidad de implementación del diagnóstico *UDS* en la herramienta *In-house software*. El resultado de esta reunión fue el establecimiento de un nuevo requerimiento que permitiera realizar la transmisión de *headers* de 29 bits con respecto a la *ISO 14229 (UDS)*. La respuesta de los desarrolladores de *In-house software* fue positiva, determinando tiempos de entrega de 15 a 21 días.

Una vez que los ingenieros de *In-house software* compartieron la primer versión del software, fueron realizadas pruebas de *servicios UDS* en las *benches* de la ciudad de Toluca con la finalidad de verificar la correcta funcionalidad del software y detectar posibles áreas de oportunidad del mismo. Este requerimiento fue el primero que se realizó a nivel global para la implementación de mensajes con *headers* de 29 bits con la herramienta de *software In-house*.

Por último, se planteó el desarrollo desde cero de una nueva herramienta de software que permitiera cumplir con todos los requerimientos establecidos por el ingeniero de validación. Esta herramienta contaría con un módulo que permitiera la extracción de información a partir del *test procedure* elegido y la transmisión de los *servicios UDS* de forma simplificada mediante el protocolo *J2534* (archivo *DLL*). Una vez recibida una respuesta por parte del *ECU*, ésta se compararía con la respuesta esperada, también obtenida del *test procedure* y desplegando un mensaje de dicha comparación, así como la respuesta obtenida en tiempo real. Al ser una herramienta de desarrollo propio tendría la ventaja de apegarse completamente al estándar de los *test procedures* (ventaja que no se ve en las otras herramientas analizadas) y mencionando que no existiría un costo monetario en el desarrollo e implementación de esta herramienta, no obstante, el tiempo de desarrollo sería mayor en comparación al software ya existente.

**Tabla 18** – Benchmark de software que implementa la misma funcionalidad.

Expectativas del cliente	Software actual	Function blocks	DiVa	In-house software	Desarrollo de software
Generar los mensajes que se transmitirán a partir de un archivo que fue previamente creado.	No	No	No	No	Si
Transmitir mensajes con <i>headers</i> de 29 bits sobre un bus <i>CAN</i> .	Si	Si	Si	Si	Si
Realizar la transmisión de <i>single</i> y <i>multi-frames</i> .	Si	Si	Si	Si	Si
Manejar mensajes <i>flow control</i> .	Si	Si	Si	Si	Si
Mantener sesiones de servicios activas dentro de la ejecución de <i>test steps</i> mediante mensajes de <i>tester present</i> .	Si	Si	No	No	Si
Enviar mensajes a un módulo en específico o a todos los módulos conectados a la misma red de comunicación de un vehículo a través de transmisiones físicas o funcionales.	Si	Si	Si	Si	Si
Comparar las respuestas recibidas contra las respuestas esperadas colocando como resultado de la comparación un mensaje de despliegue ( <i>PASS</i> o <i>FAIL</i> ).	No	No	Si	Si	Si
Ejecutar los <i>test procedures</i> en un lapso de tiempo menor con respecto al tiempo actual.	152 horas <i>bench</i>	Ejecución automática	Ejecución automática	Ejecución automática	Ejecución automática
Ejecutar todos los <i>test steps</i> de cada <i>test case</i> con un solo clic.	No	Si	Si	Si	Si
Desplegar en pantalla el buffer de datos de la transmisión y recepción de mensajes.	Si	Si	Si	Si	Si
Ser ejecutada en el sistema operativo usado actualmente por la empresa automotriz.	Si	Si	Si	Si	Si
Utilizar <i>NeoVi Fire</i> como interfaz de comunicación para la transmisión y recepción de tramas <i>CAN</i> .	Si	Si	No	Si	Si
Desplegar mensajes de ayuda para la interfaz gráfica en idioma inglés.	Si	Si	Si	Si	Si
<b>SUBTOTAL</b>	<b>9</b>	<b>11</b>	<b>10</b>	<b>11</b>	<b>13</b>
Costo de implementación	Ninguno	Ninguno	Alto (-3)	Ninguno	Ninguno
Tiempo de desarrollo de la herramienta	Ninguno	Ninguno	Ninguno	Ninguno	Alto (-3)
<b>TOTAL</b>	<b>9</b>	<b>11</b>	<b>7</b>	<b>11</b>	<b>10</b>

Observando los resultados obtenidos de la comparación de las herramientas existentes y una posible solución desarrollada desde cero, se pueden distinguir dos herramientas (*Function blocks* e *In-house software*), que si bien, no cumplen con todos los requerimientos, obtuvieron un total mayor que las otras herramientas, en contraste, el desarrollo de una herramienta de software completamente nueva permitiría cumplir con todos los requerimientos pero el costo en cuanto al tiempo de diseño e implementación tendría un impacto en el desarrollo del proyecto.

Para qué reinventar la rueda si podemos mejorarla (Anónimo). Tomando en consideración la investigación que muestra la existencia de software que permite cumplir con la funcionalidad establecida por los requerimientos del cliente se puede identificar un paradigma de desarrollo o a lo que Sommerville (2005) ha identificado como ingeniería de software basado en componentes ya existentes, la cual se describe como la reutilización informal de software o sistemas que proporcionan una funcionalidad específica.

Como segundo análisis, nuevamente se compararon las herramientas ganadoras pero esta vez utilizando el modelo de reutilización de componentes, el cual permite realizar híbridos entre sistemas por lo que se puede lograr cumplir los requerimientos funcionales a un costo menor.

La propuesta de híbridos está compuesta de dos posibles soluciones:

- La primera propuesta comprende el empleo de la funcionalidad que brinda *Vehicle Spy* con los *function blocks*, agregando un script (nuevo software) que permita extraer y convertir la información del *test procedure* en mensajes dentro de la suite de *Vehicle Spy*. Una vez contando con la información de los *test steps* se realizaría la transmisión de mensajes *UDS* mediante los *function blocks*, después se debe extraer la información del buffer de comunicación de la suite para comparar las respuestas recibidas contra las esperadas mediante el mismo script, por lo que, el resultado de la comparación sería desplegado por el *software* creado (Figura 44).
- La segunda propuesta hace referencia al uso de *In-house software*, el cual permite, desde un inicio, realizar la comparación de los resultados obtenidos de la transmisión de mensajes desplegando un mensaje positivo o negativo de dicha comparación. La creación de un script tendría el objetivo similar al que se utilizaría en la propuesta anterior, la cual extraería información de *test procedure* transformando todos los *test steps* en un formato admisible para *In-house software*. Contando con la información

dentro de *In-house software*, ésta realizaría la transmisión de mensajes como comúnmente lo hace (Figura 45). Para el manejo de sesiones mediante un *tester present* la solución estaría en la transmisión de este mensaje y verificando que los mensajes siguientes se encuentren dentro del umbral de los tres segundos. Para el caso de detener una sesión activa, este *step* se encuentra explícito en el documento del *test case* como un paso más, sin transmisión de mensaje, por lo que, mediante el script que se pretende crear se podría añadir una pausa mayor a tres segundos, y después continuar con el siguiente *step* a transmitir.

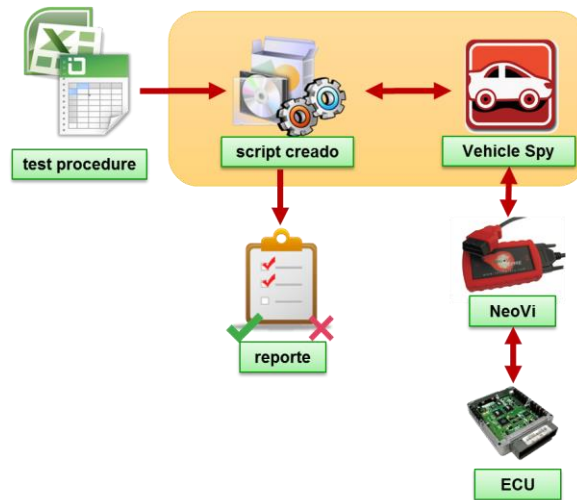


Figura 44 – Diagrama de contexto de híbrido 1.

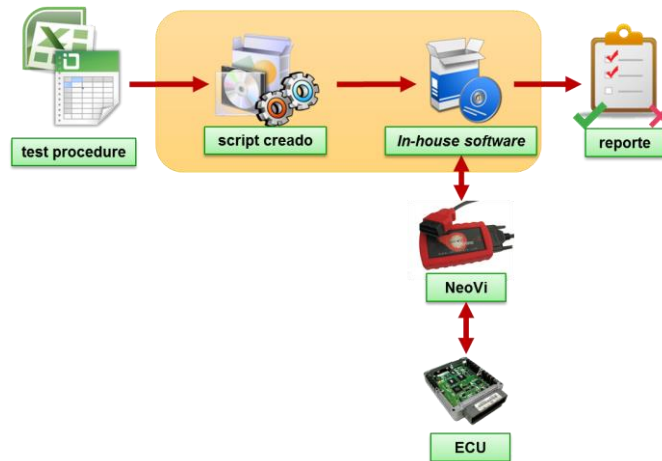


Figura 45 – Diagrama de contexto de híbrido 2.

La siguiente Tabla 19 realiza una comparación entre las posibles soluciones. En este caso se realiza la comparación de los híbridos propuestos.

**Tabla 19** – Comparación entre propuestas híbridas.

Expectativas del cliente	Software actual	Híbrido 1	Híbrido 2
Generar los mensajes que se transmitirán a partir de un archivo que fue previamente creado.	<b>No</b>	<b>Si</b>	<b>Si</b>
Transmitir mensajes con <i>headers</i> de 29 bits sobre un bus <i>CAN</i> .	Si	Si	<b>Si</b>
Realizar la transmisión de <i>single</i> y <i>multi-frames</i> .	Si	Si	Si
Manejar mensajes <i>flow control</i> .	Si	Si	Si
Mantener sesiones de servicios activas dentro de la ejecución de <i>test steps</i> mediante mensajes de <i>tester present</i> .	Si	Si	<b>Si</b>
Enviar mensajes a un módulo en específico o a todos los módulos conectados a la misma red de comunicación de un vehículo a través de transmisiones físicas o funcionales.	Si	Si	Si
Comparar las respuestas recibidas contra las respuestas esperadas colocando como resultado de la comparación un mensaje de despliegue ( <i>PASS</i> o <i>FAIL</i> ).	<b>No</b>	<b>Si</b>	Si
Ejecutar los <i>test procedures</i> en un lapso de tiempo menor con respecto al tiempo actual.	<b>152 horas bench</b>	Ejecución automática	Ejecución automática
Ejecutar todos los <i>test steps</i> de cada <i>test case</i> con un solo clic.	<b>No</b>	Si	Si
Desplegar en pantalla el buffer de datos de la transmisión y recepción de mensajes.	Si	Si	Si
Ser ejecutada en el sistema operativo usado actualmente por la empresa automotriz.	Si	Si	Si
Utilizar <i>NeoVi Fire</i> como interfaz de comunicación para la transmisión y recepción de tramas <i>CAN</i> .	Si	Si	Si
Desplegar mensajes de ayuda para la interfaz gráfica en idioma inglés.	Si	Si	Si
<b>SUBTOTAL</b>	<b>9</b>	<b>13</b>	<b>13</b>
Costo de implementación	Ninguno	Ninguno	Ninguno
Tiempo de desarrollo de la herramienta	Ninguno	<b>Medio (-2)</b>	<b>Bajo (-1)</b>
<b>TOTAL</b>	<b>9</b>	<b>11</b>	<b>12</b>

Como se puede apreciar tanto en las Figuras 44 y 45 como en la Tabla 19, ambas propuestas de híbridos cumplen con los requerimientos que el ingeniero de validación solicitó, pero en el caso del híbrido compuesto por *In-house software* y un *script* el tiempo de desarrollo es menor puesto que *In-house software* realiza de forma automática la comparación de respuestas mientras que con el híbrido 1 (*Vehicle Spy* –

*function blocks* y un *script*) el número de fases para el cumplimiento de los requerimientos tendería a ser mayor.

El *benchmark* anterior fue presentado al mentor del proyecto con la finalidad de tomar una decisión acerca de la mejor solución para completar de manera satisfactoria (tiempo y forma) los requerimientos.

## 2.4 Desarrollo del concepto

### 2.4.1 Diseño

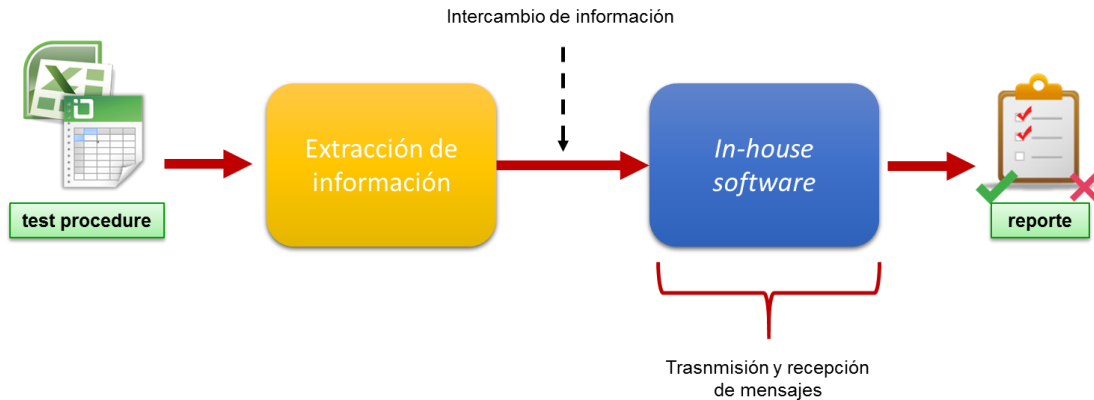
Parafraseando a Pressman (2010), el diseño de software es una representación que, a diferencia del modelado de requerimientos, proporciona detalles sobre la arquitectura, estructura de datos, interfaces y componentes que son necesarios para la implementación de un sistema.

La propuesta de solución mostrada en el capítulo anterior fue aprobada por el mentor del proyecto, la herramienta *In-house software* contribuye ampliamente a la resolución del problema, no obstante, se debe implementar un *software* que funja como interfaz entre el documento que contiene la información (*test procedure*) y la herramienta de transmisión de mensajes.

Para la resolución completa del presente proyecto no se impuso un lenguaje de programación como tal para la codificación del nuevo *software*, sin embargo, si se puntualizó el uso de la solución en un sistema operativo específico, el cual puede variar en sus versiones por lo que se decidió utilizar un lenguaje que permitiera la variación de dichas versiones, siendo éste uno soportado en múltiples plataformas así como un paradigma de programación adecuado a dicho lenguaje. De la misma forma se propone el uso de una arquitectura por capas donde:

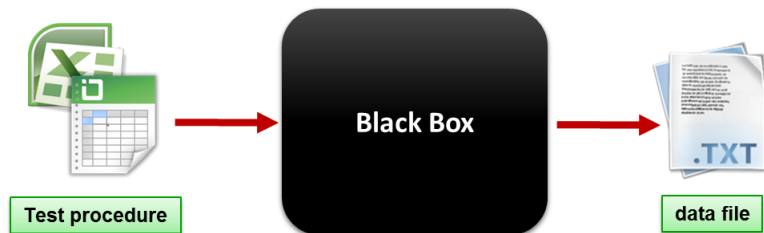
- La capa de presentación permitirá la interacción del usuario con el sistema tanto para cargar el archivo del *test procedure*, como para guardar el nuevo formato compatible con *In-house software*.
- La capa de negocio fungirá como administrador de la información que sea captada una vez leído el archivo, identificando los elementos que componen al mismo, así como la extracción de información y re-estructuración para la generación de un nuevo archivo.
- La capa de datos permitirá gestionar la extracción y almacenamiento de datos (persistencia de información a través de archivos).

La Figura 46 permite identificar el alcance que tendrá la herramienta de *software* que se pretende implementar, así como la interacción que tendrá con *In-house software*. El principal objetivo de esta herramienta es la extracción de información a partir del documento *test procedure*, así como su análisis para el intercambio de información con *In-house software*.



**Figura 46** – Alcance de la nueva herramienta de *software*.

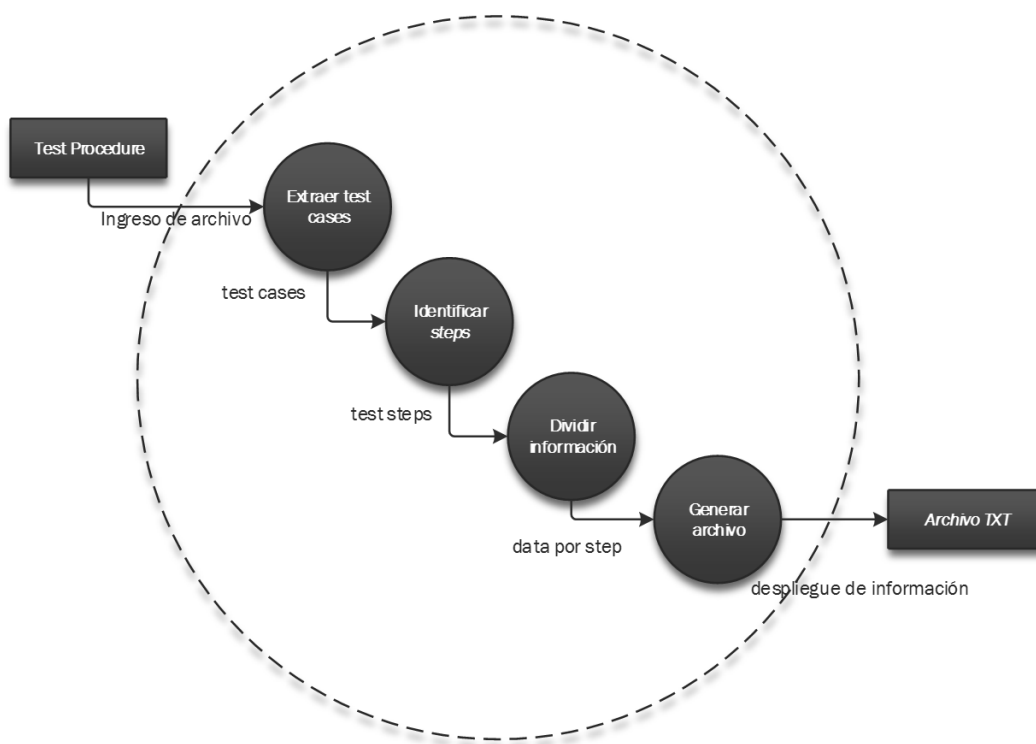
Como se mostró en el capítulo anterior, *In-house software* permite la importación de información a partir de un documento en formato *TXT* lo cual puede traducirse en una posible salida del *software* a implementar, es decir, tomando en cuenta un modelo *Black Box* (Figura 47), la entrada del *software* debe ser el *test procedure* y la salida que generaría, para poder comunicar ambas herramientas de *software*, sería un archivo con formato *TXT* compatible con *In-house software*.



**Figura 47** – Diagrama de black box del nuevo *software*.

La Figura 47 modela un diagrama de contexto nivel 0. Con la finalidad de ir refinando la información y actividades que se podrían realizar (capa de datos), se despliega el diagrama de contexto nivel 1 (Figura 48).





**Figura 48** – Diagrama de contexto nivel 1 del nuevo *software*.

#### 2.4.1.1 Análisis del *test procedure*

Dentro del diseño de la herramienta de *software* se realizó el análisis de la fuente de información (*test procedure*). Al inicio del capítulo II se dio una pequeña descripción acerca de la composición de un *test procedure*, sin embargo, es necesario ahondar más acerca de este documento puesto que la herramienta a diseñar extrae toda la información del mismo.

Como ya se ha mencionado anteriormente un *test procedure* es un documento en formato *XLSX* (Excel) que por definición se conforma por hojas o *sheets*. Las *sheets* a su vez contienen múltiples *test cases* que prueban una funcionalidad en específico, pero además de estas *sheets* existen otras que contienen información diferente a *test cases* y que sirven de referencia al ingeniero de validación. Por motivos de confidencialidad sólo se mencionará su existencia dentro del *test procedure*.

Los *test cases* se componen de información con un determinado número de *test case* y su descripción. Después, se enlistan los *test steps* iniciando con el número del *test procedure* al que pertenecen seguido del número de *test step*, una descripción del mensaje a transmitir, el mensaje a transmitir en formato hexadecimal y dividido en bytes por un espacio (ejemplo, FF FF FF), una descripción del mensaje que se espera recibir por parte del controlador, y el mensaje esperado con el mismo formato

que el que se va a transmitir. Para dividir la información entre *test cases* se deja un espacio (renglón) en blanco.

Tomando en consideración la información anterior y el actual formato del *test procedure*, se puede bosquejar un diagrama de contexto de nivel 2 que muestre con más detalle el proceso de extracción de información (Figura 49).

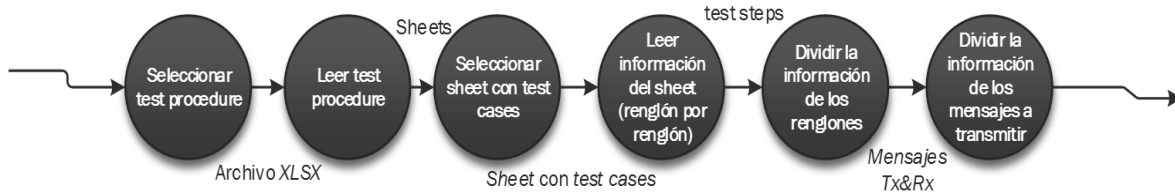


Figura 49 – Diagrama de contexto nivel 2 del nuevo *software*.

#### 2.4.1.2 Diseño arquitectónico

Aludiendo a uno de los cuatro principios básicos de modelado de Booch (2006), un único modelo no es suficiente. Cualquier sistema no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes con múltiples puntos de vista

Apegados al requerimiento de implementación de la solución dentro de un Sistema Operativo determinado, se estableció la orientación a objetos como paradigma a utilizar. Este paradigma permite el uso de un lenguaje unificado de modelado (UML por sus siglas en inglés) que permite visualizar, especificar, construir y documentar todos los componentes involucrados en el diseño del *software*.

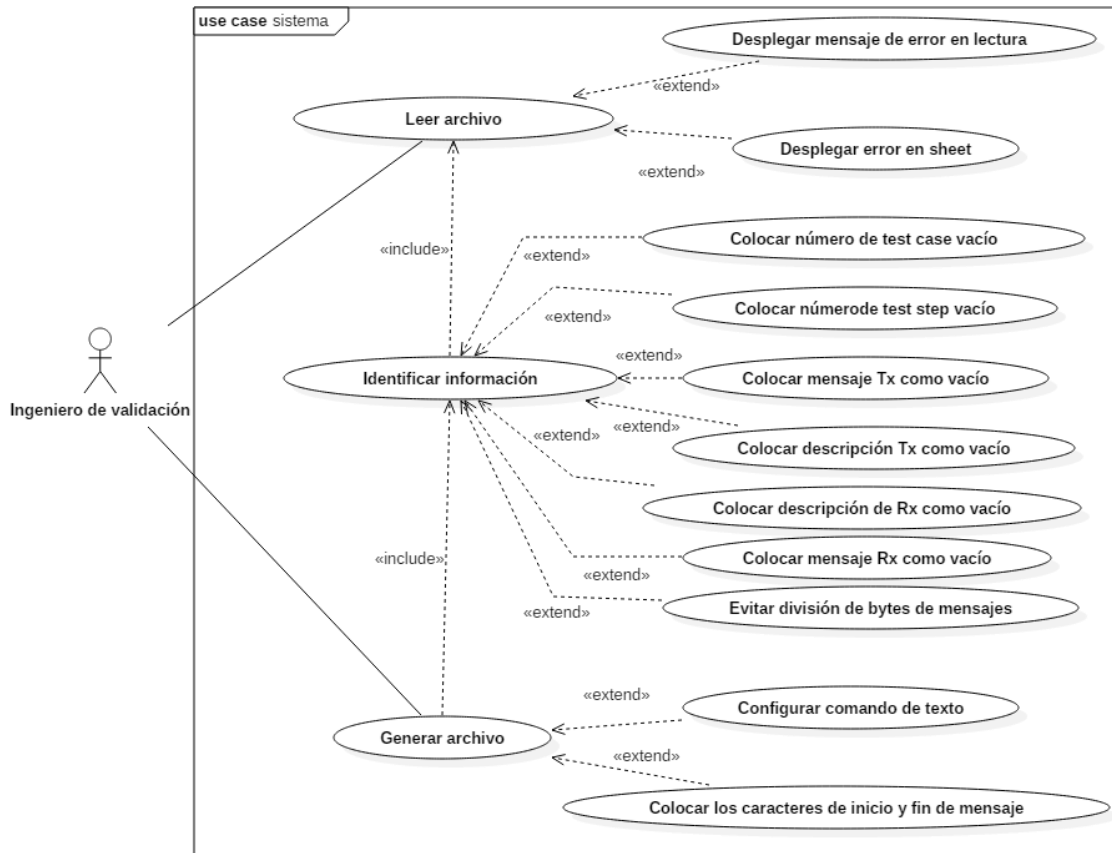
Booch (2006) brinda un esquema de diferentes vistas UML que pueden ser utilizadas para el diseño y construcción de un sistema:

Tabla 20 – Vistas y diagramas de UML.

Área	Vista	Diagramas
<b>Estructural</b>	Vista estática	Diagrama de clases
	Vista de casos de uso	Diagrama de casos de uso
	Vista de implementación	Diagrama de componentes
	Vista de despliegue	Diagrama de despliegue
<b>Dinámica</b>	Vista de máquina de estados	Diagrama de estados
	Vista de actividad	Diagrama de actividad
	Vista de interacción	Diagrama de secuencia
		Diagrama de colaboración

0 0 0

La Figura 51 representa el caso de uso general del nuevo software que será implementado. Dentro de las actividades que el sistema realizará son identificadas la lectura de un archivo que contenga información relacionada a un *test procedure*, y, a su vez, se determinan los elementos que conforman a un *test step*. Dicha información es ordenada de tal forma que permita la creación de un archivo compatible con la estructura de *In-house software*.



**Figura 51** – Diagrama de caso de uso del nuevo *software*.

A partir del diagrama anterior se crearon las descripciones de los casos de uso (representadas en las siguientes Tablas 21 a 23) en los que se explica el comportamiento, las actividades y la interacción sistema – usuario del nuevo *software*.

**Tabla 21** – Descripción del caso de uso *leer archivo*.

Nombre del caso de uso:	Leer archivo.	
Actores:	Ingeniero de validación.	
Descripción:	Se debe realizar la lectura del archivo el cual contiene información referente a un <i>test procedure</i> .	
Pre-condiciones:	Existencia de un documento en formato <i>XLSX</i> .	
Flujo normal:		Flujo alternativo:
1- El ingeniero de validación ejecuta el <i>software</i> .		
2- El sistema despliega un mensaje de bienvenida.		
3- El sistema despliega una ventana de explorador solicitando la selección del documento <i>XSLX</i> del <i>test procedure</i> .		
4- El ingeniero de validación ubica y selecciona el <i>test procedure</i> .		
5- El sistema verifica la existencia del archivo seleccionado.	5 Si el archivo seleccionado no es encontrado, el sistema desplegará un mensaje de error.	
6- El sistema identifica los <i>sheets</i> contenidos en el archivo.		
7- El sistema despliega los <i>sheets</i> contenidos en <i>test procedure</i> .		
8- El usuario selecciona un <i>sheet</i> ( <b>Seleccionar un <i>sheet</i></b> ).		
9- El sistema obtiene la <i>sheet</i> seleccionada.		
10- El sistema guarda todos los renglones de la <i>sheet</i> seleccionada.		
11- El sistema elimina los renglones vacíos.		
12- El sistema filtra la información de la <i>sheet</i> .	12 Si la <i>sheet</i> seleccionada no contiene información referente a <i>test steps</i> , el sistema desplegará un mensaje de error en el <i>sheet</i> seleccionado.	
Post-condiciones:	Toda la información de la <i>sheet</i> seleccionada referente a los <i>test cases</i> almacenada en memoria volátil ( <i>RAM</i> ).	
Excepciones:	Error al leer el archivo del <i>test procedure</i> . Volver a ejecutar el caso de uso desde el step 3.	
Nota:	Una vez desplegado algún mensaje de error el caso de uso se ejecutará nuevamente desde el step 3.	

**Tabla 22** – Descripción del caso de uso *identificar información*.

<b>Nombre del caso de uso:</b>	Identificar información.	
<b>Actores:</b>		
<b>Descripción:</b>	Se realiza la identificación y análisis de información.	
<b>Pre-condiciones:</b>	Información referente a <i>test cases</i> almacenada en memoria volátil.	
	Los <i>test steps</i> deberán de estar estructurados de acuerdo al formato del protocolo <i>UDS</i> descrito en el capítulo 1 del presente reporte.	
Flujo normal:		Flujo alternativo:
1- El sistema divide el primer campo de los renglones almacenados y lo identifica como el número de <i>test case</i> ( <b>Identificar número test case</b> ).	1 Si el primer campo se encuentra vacío o sin contenido, el sistema identificará dicho campo como <i>null</i> .	
2- El sistema divide el segundo campo de los renglones almacenados y lo identifica como el número de <i>test step</i> ( <b>Identificar número de step</b> ).	2 Si el segundo campo se encuentra vacío o sin contenido, el sistema identificará dicho campo como <i>null</i> .	
3- El sistema divide el tercer campo de los renglones almacenados y lo identifica como la descripción del mensaje a enviar ( <b>Identificar descripción Tx</b> ).	3 Si el tercer campo se encuentra vacío o sin contenido, el sistema identificará dicho campo como <i>null</i> .	
4- El sistema divide el cuarto campo de los renglones almacenados y lo identifica como el mensaje a enviar ( <b>Identificar mensaje Tx</b> ).	4 Si el cuarto campo se encuentra vacío o sin contenido, el sistema identificará dicho campo como <i>null</i> .	
5- El sistema divide el quinto campo de los renglones almacenados y lo identifica como la descripción del mensaje que se espera recibir ( <b>Identificar descripción Rx</b> ).	5 Si el quinto campo se encuentra vacío o sin contenido, el sistema identificará dicho campo como <i>null</i> .	
6- El sistema selecciona el mensaje identificado <i>Tx</i> y <i>Rx</i> del <i>test step</i> y lo divide en bytes.	6 Si los mensajes <i>Tx</i> y <i>Rx</i> se encuentran identificados como <i>null</i> , el sistema no realizará la división en bytes.	
7- El sistema identifica los campos de <i>header</i> , <i>DLC</i> , <i>tipo de mensaje</i> y contenido del mismo y los almacena en memoria volátil.		
<b>Post-condiciones:</b>	Información referente a los <i>test steps</i> identificada y dividida.	
<b>Excepciones:</b>	Error al identificar los campos de los mensajes <i>Tx</i> y <i>Rx</i> , traslapando información.	

**Tabla 23** – Descripción del caso de uso *generar archivo*.

<b>Nombre del caso de uso:</b>	Generar archivo.
<b>Actores:</b>	Ingeniero de validación.
<b>Descripción:</b>	Un nuevo archivo es creado con toda la información contenida en la <i>sheet</i> previamente seleccionada y referente a los <i>test steps</i> . Este archivo contendrá información con estructura compatible a la de <i>In-house software</i> .
<b>Pre-condiciones:</b>	<i>Test steps</i> previamente analizados e identificada su composición.
<b>Flujo normal:</b>	<b>Flujo alternativo:</b>
1- El sistema obtiene la información de cada uno de los <i>test steps</i> leídos e identificados.	
2- El sistema comienza la construcción de los mensajes compatibles con <i>In-house software</i> identificando el <i>header</i> para poder establecer el comando apropiado con respecto al tipo de <i>request</i> solicitado ( <i>functional o physical</i> ).	2 Si el <i>test step</i> identificado no cuenta con un mensaje a transmitir, el sistema fijará un comando de texto (ejemplo de descripción de inicio de <i>test case</i> ).
3- El sistema coloca el comando que permite la transmisión de mensajes en <i>In-house software</i> .	
4- El sistema coloca el <i>DLC</i> con la información obtenida acerca del tipo de mensaje <i>Tx</i> .	
5- El sistema coloca el mensaje <i>Tx</i> que será transmitido con formato de <i>In-house software</i> .	
6- El sistema coloca el mensaje <i>Rx</i> que se espera obtener como respuesta con formato de <i>In-house software</i> .	6 Si el campo de mensaje <i>Rx</i> se encuentra vacío ( <i>null</i> ), el sistema colocará los caracteres de inicio y fin del formato de mensajes de <i>In-house software</i> .
7- El sistema desplegará una ventana solicitando el nombre y dirección del archivo que se va a crear.	
8- El ingeniero de validación ingresa el nombre del archivo y la ruta destino donde será almacenado el archivo que contiene toda la información del <i>sheet</i> seleccionado.	8 Si el ingeniero de validación selecciona algún archivo con formato <i>TXT</i> ya existente, el sistema lo utilizará como base.
9- El sistema escribe la información en un archivo con formato <i>TXT</i> .	9 El sistema sobre escribe el archivo que se eligió con la nueva información.
10- El sistema despliega un mensaje de creación de archivo satisfactorio.	10 Si el sistema no puede escribir el archivo, será desplegada una pantalla con un mensaje de error.
<b>Post-condiciones:</b>	Un archivo con formato <i>TXT</i> el cual contiene información de los <i>test steps</i> en un formato compatible con <i>In-house software</i> .
<b>Excepciones:</b>	Error al seleccionar o generar el archivo a crear.

En la Figura 52 se muestra un diagrama de estados realizado con el objetivo de modelar el aspecto dinámico del nuevo *software*. Se tiene en cuenta que una máquina de estados representa la vida de un objeto en particular, pero para este caso

fue utilizado con la finalidad de simbolizar, de manera completa, la vida del sistema a desarrollar (estados en cada una de las etapas de ejecución del sistema).

Para resaltar el orden temporal de los mensajes así como de la información recabada, se empleó un diagrama de secuencia (Figura 53) que muestra la interacción entre los objetos que componen al sistema y los mensajes que son transmitidos entre estos.

Inicialmente se puede apreciar la interacción entre un objeto ventana y un objeto generador el cual funge como control de las actividades base a ejecutar. Entre dicha interacción surge la creación de nuevos objetos como lo son un *FileChooser* y *File* hacen posible la selección y carga al sistema de un archivo, siendo éste el *test procedure*. A su vez, un objeto de lectura es creado con el objetivo de extraer todo elemento que conforma un *test procedure* (*workbook, sheet, test case, test step*).

El manejo de la información se encuentra basado en renglones que contienen toda la información de un *test step*, para ser analizada por un objeto que segmente la información en diferentes campos y divida aquellos mensajes que se van a transmitir (tanto transmisibles como esperados) en un conjunto de bytes (*nibbles*).

Una vez segmentada la información e identificadas cada una de las partes de los *test steps* se procede a realizar la generación de *steps* compatibles con el formato de *In-house software* para posteriormente mandarlos a un nuevo objeto que permita la escritura de un archivo con toda la información. El sistema le solicitará al ingeniero de validación un nombre y una ruta específica para poder guardar el archivo que se creará. Una vez realizado lo anterior, el sistema generará un archivo con formato *TXT* con todos los *test steps* que conforman la *sheet* que fue previamente seleccionada.



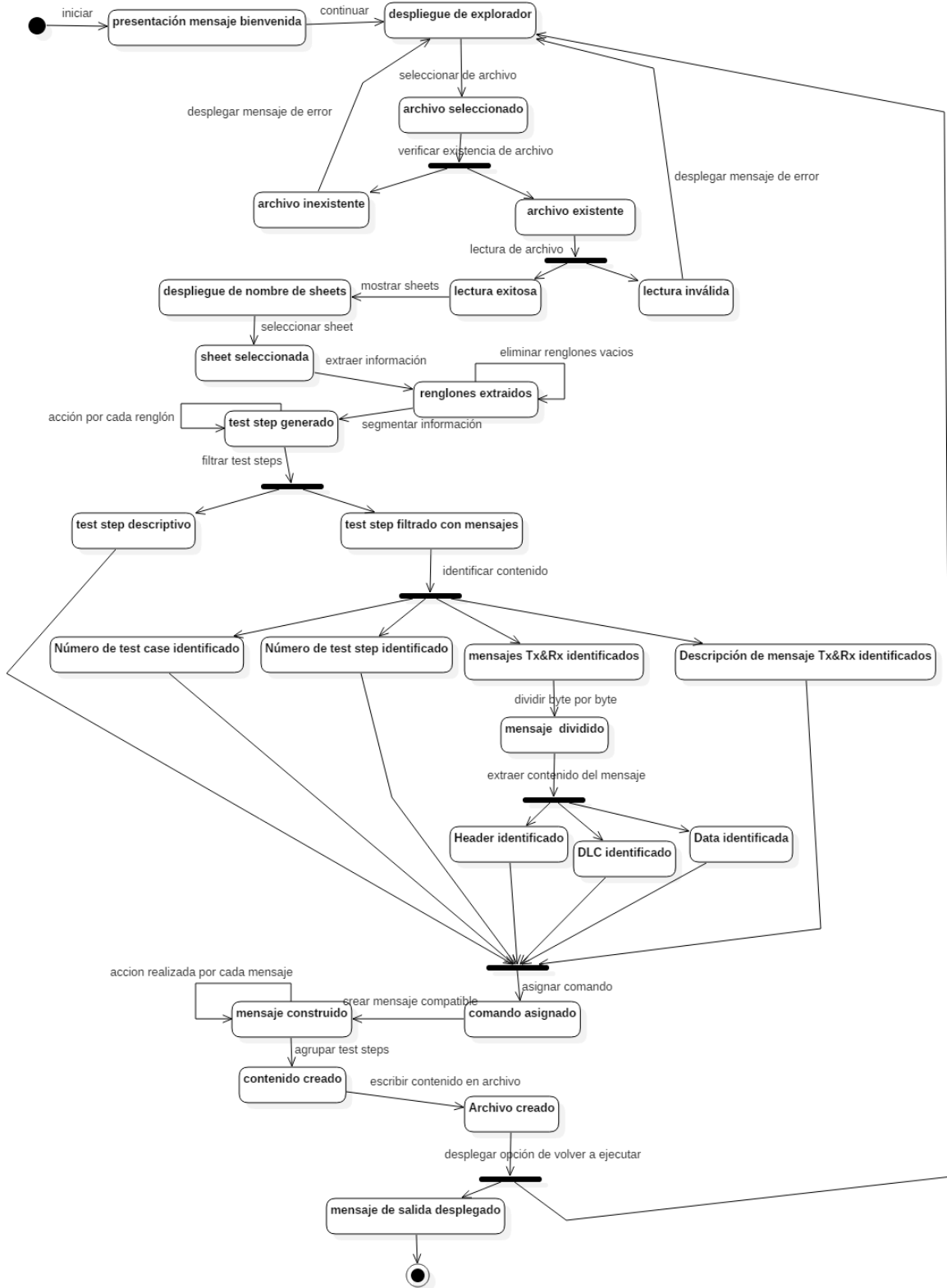


Figura 52 – Diagrama de estados del nuevo *software*.

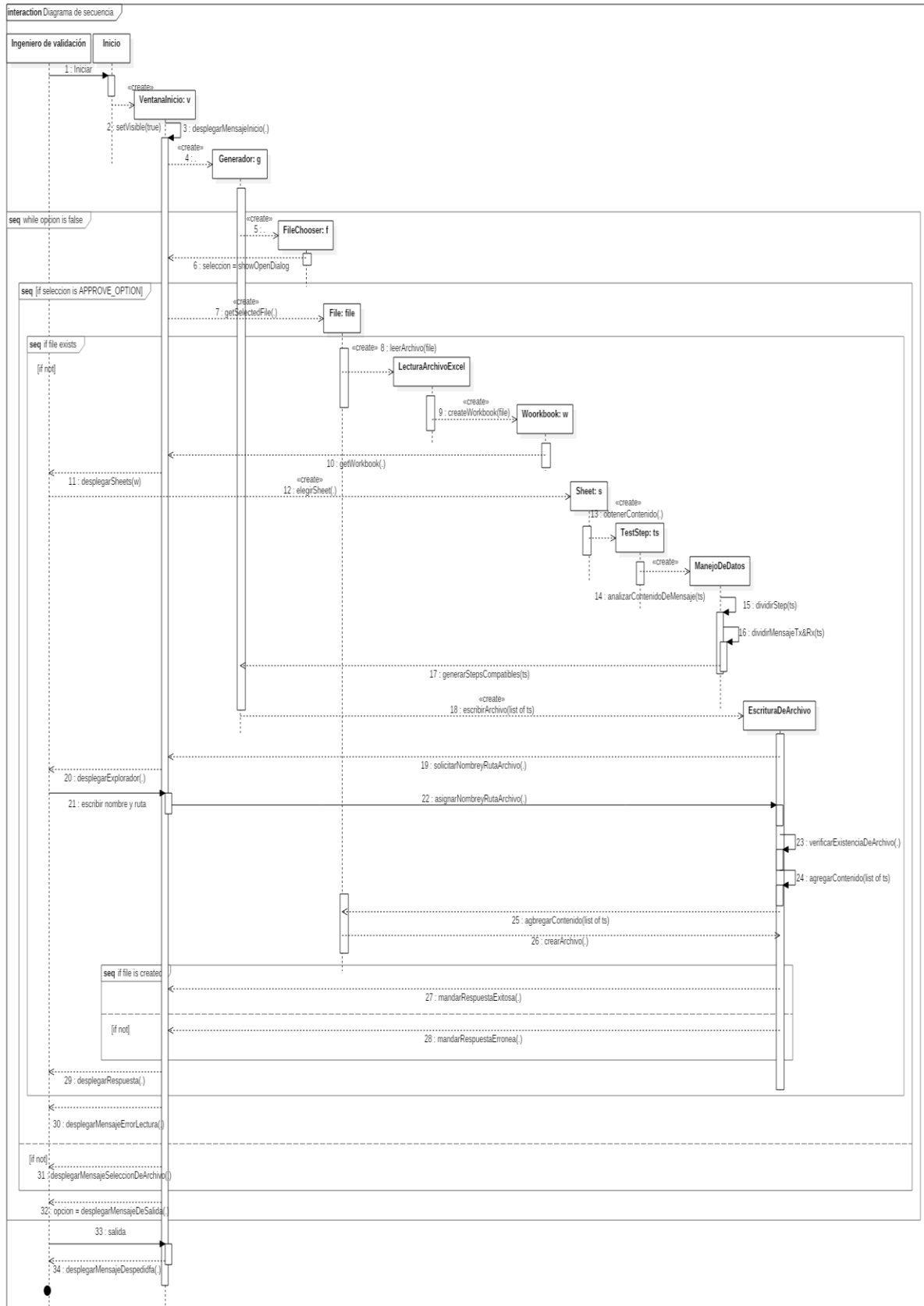


Figura 53 – Diagrama de secuencia del nuevo *software*.

Entre las conclusiones que se pudieron obtener tras el diseño del sistema se hace referencia al uso de un objeto que permita la confección de información, así como un nuevo objeto compatible con los campos que *In-house software* utiliza, apegándose a su estructura.

De esta forma se identificaron casos en donde existen excepciones o flujos alternos y su posible acción ante tal situación.

## 2.4.2 Implementación

La Real Academia Española (RAE) define a la implementación como una acción que permite poner en funcionamiento o aplicar métodos, medidas, etc., para llevar algo a cabo; por lo que una vez comprendida la problemática, realizado el diseño del sistema y siguiendo una metodología en cascada, se continua con la implementación del nuevo sistema.

Algunos autores enuncian esta fase como implementación y otros la identifican como codificación. Sommerville (2005) describe a esta etapa como el proceso de convertir una especificación del sistema previamente analizada (análisis y diseño) en un sistema ejecutable.

El lenguaje de programación, utilizado para el paradigma que fue propuesto en el apartado de diseño del presente capítulo, es *Java* con el *IDE* (Integrated Development Environment) NetBeans versión 8.1 y el *JDK* 1.7.0\_79 (Java Development Kit).

Para la lectura de los archivos con formato *XLSX* se utilizó la librería *Apache POI* que funge como *API* para manipular documentos específicos de Microsoft (archivos de *Excel*).

Como punto de partida se tomó el diagrama de clases modelado en el apartado de diseño en el cual especifica los atributos y comportamientos de los objetos. Para la lectura del archivo del *test procedure* se generó una clase que utiliza la librería *POI* y que es descrita a continuación por el siguiente pseudocódigo (Tablas 24 a 31).

Para la programación (codificación) se hizo uso de un patrón por capas similar al MVC (Modelo-Vista-Controlador) apegándose estrictamente al uso de capas que permitieran el despliegue, análisis y extracción de información, así como fue mostrado en los diagramas anteriores, permitiendo la asignación de responsabilidades. Este

patrón no se encuentra codificado de manera distribuida puesto que la integración del *software* resultante se encontrará instalado de formal local (Figura 54).

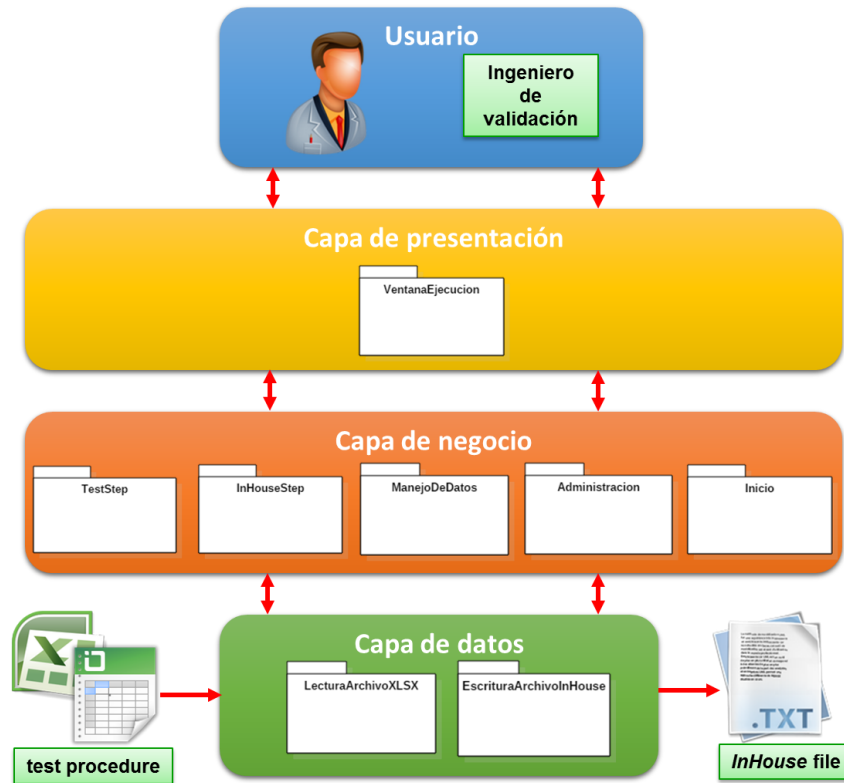


Figura 54 – Diagrama del modelo de capas.

#### 2.4.2.1 Capa de presentación

Como ya se mencionó anteriormente, la capa de presentación permite la interacción entre el usuario y el sistema. Para el desarrollo de esta capa fue codificada una clase (Tabla 24) que permite dicha interacción.

**Tabla 24** – Tabla pseudocódigo de la clase VentanaEjecucion.

Nombre de la clase: VentanaEjecucion – hereda de JFrame			
Inicio			
	Variables		
	Descripción	Tipo	Alcance
	Chooser	JFileChooser	Privado
	Valor retornado	int	Privado
	Label de despliegue de información	JLabel	Privado
	Métodos		
	Nombre	Parámetros	Valor de retorno
	Constructor	Ninguno	Ninguno
	El constructor de la clase llamará a la función que inicializa los componentes de la ventana.		
	Set & get de todos los atributos de la clase (Chooser y valor retornado)		
	Crear tanto los métodos <i>accesores</i> como <i>mutadores</i> de todos los atributos de la clase.		
	inicializarComponentes	Ninguno	Ninguno
	Inicio		
		Configurar el título de la ventana.	
		Establecer la imagen que tendrá la ventana como ícono.	
		Detallar el mensaje de bienvenida desplegado en inglés.	
		Determinar el tamaño por defecto de la ventana.	
		Establecer los comportamientos de los botones.	
	Fin		
	desplegarEleccionArchivo	Ninguno	Ninguno
	Inicio		
		Crear un filtro de archivos para formatos <i>XLSX</i> .	
		Asignar dicho filtro al <i>chooser</i> creado.	
		Desplegar una ventana que permita elegir un archivo.	
		Asignar el valor del resultado de la elección a la variable de retorno.	
	Fin		
	desplegarMensajeBienvenida	Ninguno	Ninguno
	Inicio		
		Desplegar un mensaje de bienvenida mediante un diálogo (pop up).	
	Fin		
	desplegarSheetsContenidas	Ninguno	Ninguno
	Inicio		
		Desplegar las <i>sheets</i> que se encontraron en el archivo leído.	
		Obtener la selección del usuario ( <i>sheet</i> seleccionada).	

	<b>Fin</b>			
	desplegarMensajeErrorLectura	Ninguno	Ninguno	
	<b>Inicio</b>			
		Desplegar un mensaje de error al leer un archivo.		
	<b>Fin</b>			
	desplegarMensajeErrorSheet	Ninguno	Ninguno	
	<b>Inicio</b>			
		Desplegar un mensaje de error en la lectura de una <i>sheet</i> debido a incompatibilidad de estándares de los <i>test steps</i> .		
	<b>Fin</b>			
	desplegarMensajeDeSalida	Ninguno	Ninguno	
	<b>Inicio</b>			
		Desplegar mensaje de finalización de proceso y pregunta de re-ejecución.		
		Asignar el resultado de la pregunta en la variable de salida.		
		Regresar la opción de salida.		
	<b>Fin</b>			
	desplegarMensajeGuardarArchivo	Ninguno	Ninguno	
	<b>Inicio</b>			
		Crear un objeto de tipo <i>JFileChooser</i> y un filtro para asignarle al <i>chooser</i> .		
		Regresar el <i>chooser</i> .		
	<b>Fin</b>			
	desplegarMensajeDeSalida	Ninguno	Ninguno	
	<b>Inicio</b>			
		Desplegar mensaje de que el archivo se ha escrito y guardado correctamente.		
	<b>Fin</b>			
	<b>Fin</b>			

#### 2.4.2.2 Capa de negocio

La capa de negocio permite realizar procesos como el análisis de datos, cálculos que permiten la identificación de información mediante patrones diseñados. Las clases codificadas para esta capa son representadas por las Tablas 25, 26, 27, 28 y 29.

**Tabla 25** - Tabla pseudocódigo de la clase TestStep.

Nombre de la clase: TestStep			
Inicio			
	Variables		
	Descripción	Tipo	Alcance
	Número de <i>test case</i>	<i>String</i>	Privado
	Número de <i>test step</i>	<i>String</i>	Privado
	Descripción de mensaje <i>Tx</i>	<i>String</i>	Privado
	Mensaje <i>Tx</i>	<i>String</i>	Privado
	Descripción de mensaje <i>Rx</i>	<i>String</i>	Privado
	Mensaje <i>Rx</i> esperado	<i>String</i>	Privado
	Lista de bytes <i>Tx</i>	<i>List&lt;String&gt;</i>	Privado
	Lista de bytes <i>Rx</i>	<i>List&lt;String&gt;</i>	Privado
	Métodos		
	Nombre	Parámetros	Valor de retorno
	<i>Set &amp; get</i> de todos los atributos de la clase		
	Crear tanto los métodos <i>accesores</i> como <i>mutadores</i> de todos los atributos de la clase.		
Fin			

**Tabla 26** - Tabla pseudocódigo de la clase InHouseStep.

Nombre de la clase: InHouseStep			
Inicio			
	Variables		
	Descripción	Tipo	Alcance
	Comando	String	Privado
	Descripción	String	Privado
	Data length code	String	Privado
	Mensaje Tx	String	Privado
	Mensaje Rx esperado	String	Privado
	Métodos		
	Nombre	Parámetros	Valor de retorno
	Set & get de todos los atributos de la clase		
	Crear tanto los métodos <i>accesores</i> como <i>mutadores</i> de todos los atributos de la clase.		
Fin			

**Tabla 27** - Tabla pseudocódigo de la clase ManejoDeDatos.

Nombre de la clase: ManejoDeDatos			
Inicio			
	Variables		
	Descripción	Tipo	Alcance
	Expresión regular de inicio de <i>test step</i> que tenga uno o más dígitos seguidos de un guion medio	<i>String</i>	Privado y final
	Expresión regular de inicio de comentarios que tenga uno o más dígitos seguidos de un punto y coma	<i>String</i>	Privado y final
	Expresión regular de inicio de <i>test case</i> que contenga la cadena de caracteres “ <i>test case</i> ”	<i>String</i>	Privado y final
	Expresión regular de inicio de comentario de <i>stop tester present</i> que contenga la cadena de caracteres “ <i>stop TP</i> ”	<i>String</i>	Privado y final
	Patrón de compilación para la expresión regular de inicio de <i>test step</i> que tenga uno o más dígitos seguidos de un guion medio	<i>Pattern</i>	Privado y final
	Patrón de compilación para la expresión regular de comentarios que tenga uno o más dígitos seguidos de un punto y coma	<i>Pattern</i>	Privado y final
	Patrón de compilación para la expresión regular de inicio de test case que contenga la cadena de caracteres “ <i>test case</i> ”	<i>Pattern</i>	Privado y final
	Patrón de compilación para la expresión regular de comentario de <i>stop tester present</i> que contenga la cadena de caracteres “ <i>stop TP</i> ”	<i>Pattern</i>	Privado y final
	Constante de elementos necesarios de un <i>test step</i>	<i>int</i>	Privado y final
	Identificador del tipo de <i>request</i>	<i>String</i>	Público
Nodos	Métodos		
	Nombre	Parámetros	Valor de retorno



**CAPÍTULO II DISEÑO E IMPLEMENTACIÓN**

	identificarInformacion	String		String
	<b>Inicio</b>			
1		Crear un objeto <i>Matcher</i> para comparar el renglón leído contra el patrón de inicio de <i>test step</i> .		
2		Crear un objeto <i>Matcher</i> para comparar el renglón leído contra el patrón de inicio de comentarios.		
3		<b>Si</b>	La expresión regular del patrón de inicio de <i>test step</i> o inicio de comentarios es encontrada <b>entonces</b> ,	
4			Regresar el renglón leído.	
		<b>De otro modo</b>		
5			Regresar “”.	
		<b>Fin si</b>		
6	<b>Fin</b>			
	dividirInformacion	String		TestStep
	<b>Inicio</b>			
1		Crear un objeto <i>TestStep</i>		
2		Crear un <i>array</i> de tipo <i>String</i> que almacena los elementos del renglón, divididos por el caracter punto y coma.		
3		<b>Si</b>	El tamaño del <i>array</i> es mayor o igual a la constante de elementos de un <i>test step</i> <b>entonces</b> ,	
4			<b>Para</b>	Un valor desde cero y hasta el tamaño del <i>array</i> menos uno.
5			<b>Switch</b>	De cada uno de los valores de la iteración.
6				<b>Caso 0:</b> Se agrega el valor del índice leído, en el atributo <i>test case</i> del objeto <i>TestStep</i> y se configura un <b>break</b> .
7				<b>Caso 1:</b> Se agrega el valor del índice leído, en el atributo <i>test step</i> del objeto <i>TestStep</i> y se configura un <b>break</b> .
8				<b>Caso 2:</b> Se agrega el valor del índice leído, en el atributo descripción del mensaje <i>Tx</i> del objeto <i>TestStep</i> y se configura un <b>break</b> .
9				<b>Caso 3:</b> Se agrega el valor del índice leído, en el atributo mensaje <i>Tx</i> del objeto <i>TestStep</i> y se configura un <b>break</b> .
10				<b>Caso 4:</b> Se agrega el valor del índice leído, en el atributo descripción del mensaje <i>Rx</i> del objeto <i>TestStep</i> y se configura un <b>break</b> .
11				<b>Caso 5:</b> Se agrega el valor del índice leído, en el atributo mensaje <i>Rx</i> del objeto <i>TestStep</i> y se configura un <b>break</b> .
				<b>Default:</b> <b>Break</b> .

			<b>Fin switch</b>	
		<b>Fin para</b>		
		<b>De otro modo</b>		
12			El <i>test step</i> será asignado como nulo.	
		<b>Fin si</b>		
13		Regresar el objeto <i>TestStep</i> .		
	<b>Fin</b>			
	dividirBytesDeMensaje		<i>String</i>	<i>List</i>
	<b>Inicio</b>			
1		Crear una lista de <i>Strings</i> .		
2		Crear un objeto de tipo <i>StringTokenizer</i> que divida la información del mensaje.		
3		<b>Mientras</b>	El objeto <i>StringTokenizer</i> tenga más elementos <b>entonces</b> ,	
4			Los mensajes serán agregados a la lista de <i>Strings</i> .	
		<b>Fin mientras</b>		
5		Regresar lista de <i>Strings</i> .		
	<b>Fin</b>			
	identificarDLC		<i>TestStep, InHouseStep</i>	Ninguno
	<b>Inicio</b>			
1		<b>Si</b>	Dentro de la lista de bytes de <i>Tx</i> del objeto <i>TestStep</i> de entrada se encuentra en una posición específica el valor de “1” <b>entonces</b> ,	
2			Será identificado un mensaje <i>multiframe</i> y el <i>DLC</i> será calculado y fijado en el <i>InHouseStep</i> en el campo de <i>Data Length Code</i> conforme al estándar <i>ISO 15765-2</i> .	
3			Se mandará a llamar al método que construya el mensaje <i>Tx</i> a partir de byte donde termina el <i>DLC</i> .	
		<b>De otro modo,</b>		
4			Será identificado un mensaje <i>single frame</i> y será fijado el byte dentro del atributo <i>DLC</i> del objeto <i>InHouseStep</i> .	
5			Se mandará a llamar al método que construya el mensaje <i>Tx</i> a partir de byte donde termina el <i>DLC</i> .	
		<b>Fin si</b>		
6	<b>Fin</b>			
	identificarTipoDeRequest		<i>List</i>	Ninguno
	<b>Inicio</b>			
1		Crear un objeto de tipo <i>InHouseStep</i>		
2		<b>Si</b>	La variable de clase de <b>ManejoDeDatos</b> la cual permite la identificación del tipo de <i>request</i> es diferente a un contenido vacío <b>entonces</b> ,	
3			Se le asignará el contenido del tipo de <i>request</i> identificado como comando del <i>InHouseStep</i> creado y dicho objeto será agregado a la lista de <i>InHouseSteps</i> entrante.	
		<b>Fin si</b>		
4	<b>Fin</b>			

	construirMensajeTxInHouse		TestStep, InHouseStep, int	Ninguno
	Inicio			
1		Crear una lista de Strings que contendrá los mensajes en hexadecimal.		
2		Para	Un valor inicial de cero hasta el valor del tamaño de la lista de mensajes Tx del objeto TestStep.	
3			Extraer los elementos de la lista de mensajes Tx del TestStep entrante y se agregan a la nueva lista de mensajes con formato hexadecimal con formato inicial de In-House software.	
		Fin para		
4		Crear una variable de tipo String inicializada con el caracter especial de inicio de los mensajes de In-house software.		
5		Para	Todos los elementos de la lista de mensajes Tx con formato hexadecimal.	
6			Concatenar la variable con caracter de inicio con el contenido de la lista y un caracter de una coma.	
		Fin para		
7		Agregar un caracter de cierre de mensaje a la variable que contiene todos los mensajes en formato hexadecimal.		
8		Eliminar el último caracter de coma.		
9		Asignar el String resultante al objeto InHouseStep en el atributo mensaje Tx.		
	Fin			
	construirMensajeRxInHouse		TestStep, InHouseStep	Ninguno
	Inicio			
1		Crear una lista de mensajes Rx con formato hexadecimal compatible con el formato de In-house software.		
2		Crear una variable String con el caracter de inicio de los mensajes de In-house software.		
3		Si	La lista de mensajes Rx del objeto TestStep no contiene ningún elemento entonces,	
4			Agregar al String de inicio de mensaje el caracter de fin de mensaje.	
5		En otro caso si	El primer caracter del cuarto conjunto de bytes del mensaje Rx es uno entonces,	
6			Colocar un límite indicando un caso de mensaje multi frame.	
		En otro caso		
7			Colocar un límite indicando un caso de mensaje single frame.	
		Fin si		
8		Para	Un valor inicial de cero hasta el tamaño de la lista que contiene a los bytes del mensaje Rx.	
9			Concatenar cada uno de esos bytes con el formato hexadecimal de In-house software y se agregará a la lista de mensajes con dicho formato.	
		Fin para		
10		Para	Todos los mensajes de la lista de mensajes con formato hexadecimal.	
11			Concatenar un caracter de una coma al final de dicho mensaje.	
		Fin para		
12		Elimina la el caracter de la última coma.		
13		Agregar el caracter de cierre de mensaje.		
14		Asignar el String resultante al objeto InHouseStep en el atributo mensaje Rx.		

	<b>Fin</b>		
	construirInHouseStep	TestStep, InHouseStep	InHouseStep
	<b>Inicio</b>		
<b>1</b>		<b>Si</b>	Dentro de la lista de bytes de <i>Tx</i> del objeto <i>TestStep</i> de entrada contiene la dirección física del <i>ECU entonces,</i>
<b>2</b>			La variable de clase de <b>ManejoDeDatos</b> almacenará el texto que identifica a un <i>physical request</i> .
<b>3</b>			Se fijará el comando para transmisión de mensajes <i>UDS</i> en el atributo de comando del <i>InHouseStep</i> .
<b>4</b>			Se llamará al método que identifica el <i>DLC</i> .
<b>5</b>		<b>En otro caso si</b>	Dentro de la lista de bytes <i>Tx</i> del objeto <i>TestStep</i> de entrada contiene un dirección funcional del <i>ECU entonces,</i>
<b>6</b>			La variable de clase de <b>ManejoDeDatos</b> almacenará el texto que identifica a un <i>functional request</i> .
<b>7</b>			Se fijará el comando para transmisión de mensajes <i>UDS</i> en el atributo de comando del <i>InHouseStep</i> .
<b>8</b>			Se llamará al método que identifica el <i>DLC</i> .
		<b>Para todos los demás casos</b>	
<b>9</b>			Configurar la variable de clase de <b>ManejoDeDatos</b> como vacía.
<b>10</b>			Crear un objeto <i>Matcher</i> que identificará la existencia de un patrón de inicio de <i>test case</i> en de la descripción del mensaje <i>Tx</i> .
<b>11</b>			Crear un objeto <i>Matcher</i> que identificará la existencia de un patrón de paro de <i>tester present</i> en la descripción del mensaje <i>Tx</i> .
<b>12</b>			<b>Si</b> Se encuentra un patrón de inicio de <i>test case entonces,</i>
<b>13</b>			Colocar un comando especial para indicar el inicio de un nuevo <i>test case</i> que permitirá agrupar a los siguientes <i>test steps</i> en <i>In-House software</i> . Este valor se fijará en el atributo de comando del <i>InHouseStep</i> .
<b>14</b>			El atributo de mensaje <i>Tx</i> se colocará vacío.
<b>15</b>		<b>De otra forma si</b>	Se encuentra un patrón de paro de <i>tester present entonces,</i>
<b>16</b>			Colocar un comando especial que permite realizar una pausa de un determinado tiempo, permitiendo que transcurra la vida del <i>tester present</i> .
<b>17</b>			Colocar como mensaje <i>Tx</i> el tiempo (en segundos) de la pausa que se realizará. Este tiempo es calculado conforme a la <i>ISO 14229-1</i> tomando en cuenta el comportamiento de un <i>tester present</i> .
		<b>Para todos los demás casos</b>	
<b>18</b>			Identificar como texto informativo y se coloca un comando que permita asemejar a la información como descripción (dentro del atributo comando del objeto <i>InHouseStep</i> ).
<b>19</b>			Fijar como vacío el atributo de mensaje <i>Tx</i> del objeto <i>InHouseStep</i> .
		<b>Fin si</b>	
<b>20</b>			Fijar como vacío el atributo de mensaje <i>Rx</i> del objeto <i>InHouseStep</i> .
		<b>Fin si</b>	

<b>21</b>		Integrar la descripción del mensaje eliminando saltos de línea o valores nulos. La descripción se compone por número de <i>test case</i> , número de <i>test step</i> y la descripción del mensaje. A su vez, agregar todo lo integrado al atributo <i>descripción</i> del objeto <i>InHouseStep</i> .
<b>22</b>		Regresar el <i>InHouseStep</i> .
	<b>Fin</b>	
<b>Fin</b>		

**Tabla 28** - Tabla pseudocódigo de la clase Administración.

<b>Nombre de la clase:</b> Administracion			
<b>Inicio</b>			
	<b>Variables</b>		
	<b>Descripción</b>	<b>Tipo</b>	<b>Alcance</b>
<b>Nodos</b>	<b>Métodos</b>		
	<b>Nombre</b>	<b>Parámetros</b>	<b>Valor de retorno</b>
	leerArchivo	<i>File</i>	<i>List</i>
	<b>Inicio</b>		
	Crear una lista de <i>Strings</i> .		
	Instanciar un objeto de tipo <b>LecturaArchivoXLSX</b> con el archivo de entrada del método.		
	Llamar al método leerArchivo del objeto instanciado.		
	Obtener la lista de renglones del objeto instanciados y asignarle la misma dirección a la lista creada dentro de este método.		
	Regresar dicha lista.		
	<b>Fin</b>		
	escribirArchivo	<i>List</i>	ninguno
	<b>Inicio</b>		
	Instanciar un objeto de tipo <b>EscrituraArchivoInHouse</b> mandándole la lista de <i>InHouseSteps</i> .		
	Mandar a llamar al método escribirArchivo del objeto instanciado.		
	<b>Fin</b>		
	analizarInformacion	<i>List</i>	<i>List</i>
	<b>Inicio</b>		
<b>1</b>	Crear una lista de <i>Strings</i> que sea direccionada a la lista que entra como parámetro y que contenga a todos los renglones leídos.		
<b>2</b>	Crear una lista de <i>Strings</i> que permita almacenar los renglones una vez filtrados.		
<b>3</b>	Crear una lista de <i>TestSteps</i> .		
<b>4</b>	Crear una lista de <i>InHouseSteps</i> .		
<b>5</b>	Instanciar un objeto de la clase <b>ManejoDeDatos</b> .		
<b>6</b>	<b>Para</b>	Todos los elementos de la lista de renglones leídos.	
<b>7</b>		Llamar al método identificarInformacion de la clase <b>ManejoDeDatos</b> .	

8			Si	El resultado obtenido de la ejecución del método identificarInformacion no se encuentra vacío <b>entonces</b> ,	
9				Agregar a la lista de renglones filtrados dicho renglón.	
			De otro modo		
10				No se agrega nada.	
			Fin si		
		Fin para			
11		Se instancia un objeto de la clase <b>TestStep</b> .			
12		Para	Todos los elementos de la lista de renglones reducida.		
13			Llamar al método dividirInformacion de la clase <b>ManejoDeDatos</b> y el resultado se asignará al objeto <i>TestStep</i> creado.		
14			Si	Dicho resultado no se encuentra vacío ( <i>null</i> ) <b>entonces</b> ,	
15				Ese <i>TestStep</i> será agregado a la lista de <i>TestSteps</i> .	
			Fin si		
		Fin para			
16		Para	Todos los elementos que componen la lista de <i>TestSteps</i> .		
17			Llamar al método dividirBytesDeMensaje de la clase <b>ManejoDeDatos</b> tanto para los mensajes <i>Tx</i> como <i>Rx</i> y serán asignados a la lista de bytes <i>Tx</i> y <i>Rx</i> del objeto <i>TestStep</i> respectivamente.		
18			Instanciar un nuevo objeto <i>InHouseStep</i> y asignarle el resultado de la llamada del método construirInHouseStep de la clase <b>ManejoDeDatos</b> .		
19			Llamar al método identificarTipoDeRequest de la clase <b>ManejoDeDatos</b> .		
20			Agregar el <i>InHouseStep</i> construido a la lista de <i>InHouseSteps</i> .		
		Fin para			
21		Regresar la lista de <i>InHouseSteps</i> .			
	Fin				
	administrarActividades		Ninguno	Ninguno	
	Inicio				
1		Instanciar un objeto de tipo <b>VentanaEjecucion</b> y desplegar mensaje de bienvenida.			
2		Hacer			
3			Crear una lista de <i>Strings</i> que almacene los renglones.		
4			Crear una lista de <i>InHouseSteps</i> .		
5			Llamar al método que despliega el mensaje de elección de archivo.		
6			Si	Un archivo fue elegido <b>entonces</b> ,	
7				Se crea un nuevo objeto de tipo <i>File</i> que almacene el archivo elegido.	
8				Si	Se verifica que el archivo existe <b>entonces</b> ,
9					Llamar al método leerArchivo de esta misma clase y el resultado se almacena en la lista de renglones creada.
10					Llamar al método analizarInformacion con la lista de reglones anterior como parámetro y el resultado se asignará en la lista de <i>InHouseSteps</i> creada.
11					Llama al método escribirArchivo de esta misma clase con la lista de <i>InHouseSteps</i> como parámetro.

				De otro modo	
12					Llamar al método que despliegue el mensaje de error de lectura de archivo.
				Fin si	
13				Desplegar la ventana de salida capturando la opción elegida por el usuario.	
			Fin si		
14		Mientras	La opción de salida elegida por el usuario no sea positiva.		
15		Cerrar la ventana de ejecución.			
	Fin				
Fin					

Tabla 29 - Tabla pseudocódigo de la clase Inicio.

Nombre de la clase: Inicio			
Inicio			
	Variables		
	Descripción	Tipo	Alcance
	Métodos		
	Nombre	Parámetros	Valor de retorno
	main	String[] args	Ninguno
	Inicio		
		Esta clase es la primera en ser ejecutada por el <i>software</i> y contiene la instancia de la clase <b>Administracion</b> así como la llamada de su método administrarActividades.	
	Fin		
Fin			

### 2.4.2.3 Capa de datos

La capa de datos se encuentra principalmente definida para el almacenamiento y extracción de los datos (persistencia). La codificación de esta capa no hace conexión alguna con bases de datos. Para este caso se realizó la extracción de información a partir de un archivo con formato *XLSX* que contiene la información de todos los mensajes que se transmitirán hacia la *ECU* (Tabla 30). La persistencia de los datos construidos por la capa de negocio se realiza mediante un archivo con formato *TXT* que tiene una estructura compatible de *In-house software* (Tabla 31).

**Tabla 30** - Tabla pseudocódigo de la clase `LecturaArchivoXLSX`.

Nombre de la clase: LecturaArchivoXLSX				
Inicio				
	Variables			
	Descripción	Tipo	Alcance	
	Variable	File	Privado	
	Lista de Strings	List<String>	Privado	
Nodo	Métodos			
	Nombre	Parámetros	Valor de retorno	
	Constructor	File	Ninguno	
	Crear el constructor de la clase con un parámetro de entrada de tipo file; el constructor asignará el valor del parámetro de entrada a la variable de tipo File.			
	Set & get de la lista de Strings	String	List<String>	
	leerArchivo	Ninguno	Ninguno	
	Inicio			
1		Declarar una variable de tipo String que almacene el valor capturado de las celdas.		
2, 17		Crear un objeto de tipo InputStream asignándole el objeto de tipo file de la clase, si el objeto file tiene error, se llamará al método desplegarErrorLectura.		
3		Crear un objeto de tipo XSSFWorkbook asignándole el objeto InputStream como parámetro del constructor.		
4		Declarar e inicializar una variable de tipo entero con el valor obtenido del objeto XSSFWorkbook y su función getNumberOfSheets.		
		Declarar un array de Strings del tamaño del entero obtenido en el paso anterior.		
5		Extraer el nombre de todas las sheets contenidas en el archivo leído.		
6		Mandar llamar el método desplegarSheetsContenidas para el despliegue de información y selección de sheet.		
7		Obtener el resultado de selección en una variable de tipo String.		
8		Crear un objeto de tipo XSSFSheet y asignarle el valor resultante del método getSheetAt (con el resultado de la selección) del objeto XSSFWorkbook.		
		Crear los objetos XSSFRow y XSSFCell.		
		Crear un objeto de tipo Iterator para los renglones y asignarle el valor de la extracción del contenido del objeto XSSFSheet seleccionado.		
9		Mientras	El objeto Iterator de renglones tenga otro elemento entonces,	
10			El objeto XSSFRow tomará el contenido del renglón leído.	
11			Crear un objeto de tipo Iterator específico para para dividir el contenido en columnas del renglón leído en el objeto XSSFRow.	
			Crear una variable de tipo String que almacene el contenido leído tanto de renglones como de columnas.	
12			Mientras	El objeto Iterator de las columnas tenga otro elemento entonces,
13				El valor de la celda sería inicializado como un String sin caracteres (“”).
14				El objeto XSSFCell tomará el contenido de la columna leída.



15				La variable que almacena el contenido tanto de columnas como de renglones será concatenado con el valor del objeto <i>XSSFCell</i> y un caracter punto y coma.
			<b>Fin mientras</b>	
16				Mandar llamar al <i>set</i> de la lista de <i>Strings</i> y agregar el <i>String</i> que almacena el contenido completo.
		<b>Fin mientras</b>		
18	<b>Fin</b>			
<b>Fin</b>				

Tabla 31 – Tabla pseudocódigo de la clase EscrituraArchivoInHouse.

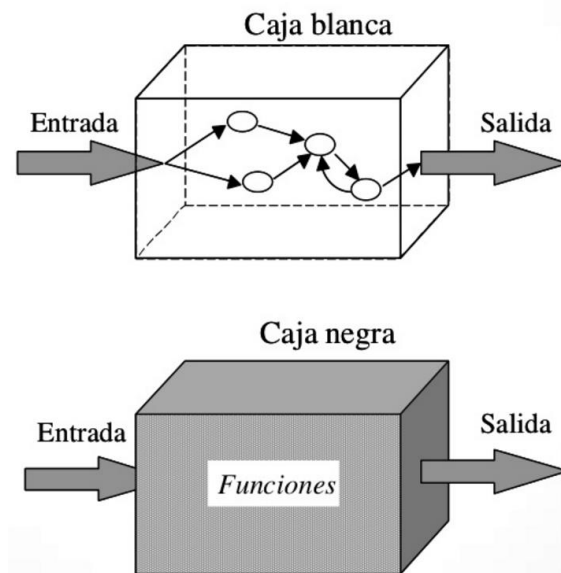
Nombre de la clase: EscrituraArchivoInHouse				
Inicio				
	Variables			
	Descripción	Tipo		Alcance
	Lista de <i>InHouse steps</i>	<i>List&lt;InHouseStep&gt;</i>		Privado
Nodo	Métodos			
	Nombre		Parámetros	Valor de retorno
	Constructor		<i>List</i>	Ninguno
	Crear el constructor de la clase con un parámetro de entrada de tipo <i>List</i> ; el constructor asignará la dirección del parámetro de entrada a la lista de esta misma clase.			
	escribirArchivo		Ninguno	Ninguno
	Inicio			
1		Crear objetos para la escritura de archivo y de contenido ( <i>FileWriter</i> y <i>PrintWriter</i> ).		
2		Solicitar a la ventana de ejecución el despliegue de explorador para capturar la ruta y el nombre del archivo.		
3		Obtener dirección y nombre del archivo, pasarle la información a los objetos específicos para la escritura.		
4		Colocar en el buffer de escritura la configuración inicial de direcciones ( <i>headers</i> ) del <i>ECU</i> .		
5		Extraer la información de la lista de <i>InHouseSteps</i> así como los atributos de todos los <i>Steps</i> adecuándolos a la estructura del <i>software</i> , es decir, colocando los caracteres específicos de dicha herramienta.		
6		Si	La escritura del archivo se realizó satisfactoriamente <b>entonces</b> ,	
7			Solicitar a la ventana de ejecución desplegar el mensaje de ejecución satisfactoria.	
		De otro modo		
8			Solicitar a la ventana de ejecución desplegar el mensaje de error en escritura y reintentar.	
		Fin sí		
9		Para cualquiera de los dos casos se debe cerrar el archivo generado.		
	Fin			

Fin	

## 2.5 Pruebas

Con la finalidad de poner a prueba el *software* codificado en la fase de implementación, se realizaron pruebas que permitieron verificar el correcto funcionamiento del mismo. Una vez generado el código fuente, el *software* debe probarse para descubrir y corregir tantos errores como sea posible antes de entregarlo al cliente (Pressman, 2010).

Entre las pruebas generadas, se encuentran las pruebas de caja blanca y las pruebas de caja negra (Figura 55). Ambas serán explicadas más adelante.



**Figura 55** – Diseño de casos de prueba (Benitez A., 2012).

### 2.5.1 Pruebas de caja blanca

Las pruebas de caja blanca o caja de vidrio permiten verificar todos los caminos o casos posibles con los que cuenta el *software*. Estas pruebas tienen un enfoque hacia rutas básicas con la finalidad de identificar los trayectos de ejecución existentes por cada una de las funciones (métodos) codificadas.

El número de métodos por clase codificados es mostrado a continuación:

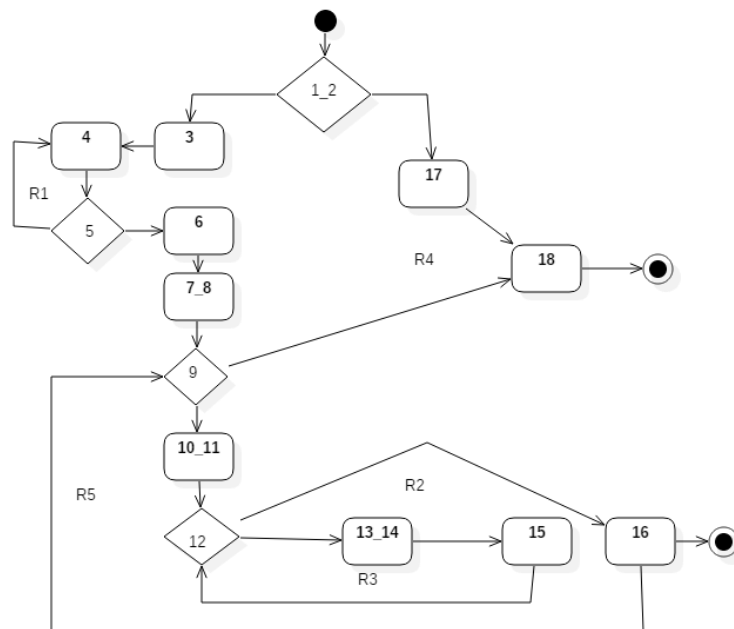
- Administración – 4 métodos.
- InHouseStep – 10 métodos.

- Inicio – 1 método.
- LecturaArchivoXLSX – 3 métodos.
- ManejoDeDatos – 9 métodos.
- TestStep – 16 métodos.
- VentanaEjecucion – 10 métodos.
- EscrituraArchivoInHouse – 1 método

Debido al tamaño de las pruebas y al conjunto de métodos codificados, sólo serán mostrados aquellos casos que cuenten con una lógica amplia, es decir, las clases InHouseStep, Inicio, TestStep y VentanaEjecucion cuentan con una estructura de ejecución única, además InHouseStep y TestStep cuentan con el número de métodos mostrado anteriormente debido a que se crearon los *accesores* y *mutadores* de cada uno de sus atributos por lo que el nivel de verificación de esos métodos es mínimo.

Se hizo uso de diagramas de flujo para representar el método de rutas básicas, por consecuencia, fueron identificados los nodos de cada una de las líneas de código descritas en las tablas de pseudocódigo anteriores (2.4.2 Implementación). Para cada uno de los diagramas de flujo se calculó la complejidad lógica (ciclomática) del código a partir de las regiones generadas, permitiendo encontrar las rutas o caminos por los que el *software* podría pasar. A partir de estos resultados son determinados escenarios (caminos) que permiten obtener posibles errores.

El primer diagrama de flujo expuesto en la Figura 56 representa las posibles rutas en la ejecución del método **leerArchivo** de la clase **LecturaArchivoXLSX**.



**Figura 56** – Diagrama de flujo del método leerArchivo.

La complejidad ciclomática de la Figura 56 es de cinco regiones, generando las siguientes rutas de prueba:

- Ruta 1: 1, 2, 17, 18.
- Ruta 2: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 9, 18.
- Ruta 3: 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 9, 18.
- Ruta 4: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 12, 16, 9, 18.
- Ruta 5: 1, 2, 3, 4, 5, 6, 7, 8, 9, 18.

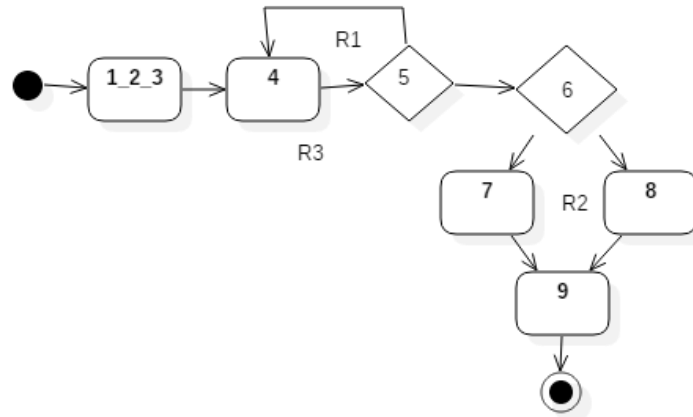
A partir de estas rutas se generaron los casos de prueba que se muestran a continuación (Tabla 32):

**Tabla 32** – Rutas del método leerArchivo.

Ruta 1	
<b>Descripción:</b>	El archivo de <i>Excel</i> que fue elegido contiene algún error lo que no permite realizar la lectura.
<b>Resultado:</b>	El mensaje de error de lectura de archivo es desplegado.
Ruta 2	
<b>Descripción:</b>	El archivo de <i>Excel</i> solamente está compuesto por una <i>sheet</i> que es la seleccionada para lectura y contiene información en la primera columna.
<b>Resultado:</b>	El valor de la primer columna es leído y almacenado (lectura renglón por renglón).
Ruta 3	
<b>Descripción:</b>	El archivo de <i>Excel</i> está compuesto de por lo menos dos <i>sheets</i> y aquella <i>sheet</i> seleccionada solamente tiene información en la primera columna.
<b>Resultado:</b>	El valor de la primer columna es leído y almacenado (lectura renglón por renglón).
Ruta 4	
<b>Descripción:</b>	El archivo de <i>Excel</i> está compuesto por una <i>sheet</i> que es seleccionada y que contiene información tanto en columnas como en renglones.
<b>Resultado:</b>	El contenido de la <i>sheet</i> es almacenado renglón por renglón (lectura de celda por celda mediante iteración de renglones y columnas).
Ruta 5	
<b>Descripción:</b>	El archivo de <i>Excel</i> está compuesto por una <i>sheet</i> que es seleccionada pero que se encuentra vacía (ningún valor en renglones ni columnas).
<b>Resultado:</b>	No almacena ningún dato por lo que la lista que almacena esta información queda vacía.

Para el método **escribirArchivo** de la clase **EscrituraArchivoInHouse** fue desarrollado el diagrama de flujo de la Figura 57 el cual tiene una complejidad de tres regiones desplegadas en la Tabla 33:

- Ruta 1: 1, 2, 3, 4, 5, 6, 7, 9.
- Ruta 2: 1, 2, 3, 4, 5, 4, 5, 6, 7, 9.
- Ruta 3: 1, 2, 3, 4, 5, 6, 8, 9.



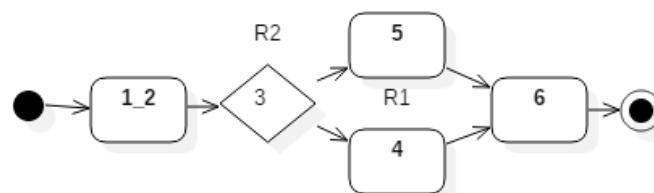
**Figura 57-** Diagrama de flujo del método EscribirArchivoInHouse.

**Tabla 33** - Rutas del método escribirArchivo.

Ruta 1	
<b>Descripción:</b>	La entrada está especificada por el nombre y ruta del archivo. Solo se tiene registrado un elemento en la lista de <i>InHouseSteps</i> .
<b>Resultado:</b>	Archivo con el nombre de entrada y almacenado en la ruta especificada. Despliegue de una ventana con el mensaje de actividad completada satisfactoriamente.
Ruta 2	
<b>Descripción:</b>	La entrada está especificada por el nombre y ruta del archivo. La lista de <i>InHouseSteps</i> cuenta con más de un elemento que serán escritos.
<b>Resultado:</b>	Archivo almacenado en la ruta especificada y con el nombre dado. Despliegue de una ventana con el mensaje de actividad completada satisfactoriamente.
Ruta 3	
<b>Descripción:</b>	La entrada está especificada por el nombre y ruta del archivo. La lista de <i>InHouseSteps</i> cuenta con más de un elemento que serán escritos.
<b>Resultado:</b>	No se puede generar el archivo con el contenido deseado debido a una dirección inexistente. Se despliega en pantalla un mensaje de error de escritura de archivo.

La Figura 58 muestra el diagrama de flujo del método **identificarInformacion** que cuenta con dos regiones descritas por las siguientes rutas y cada una es detallada en la Tabla 34:

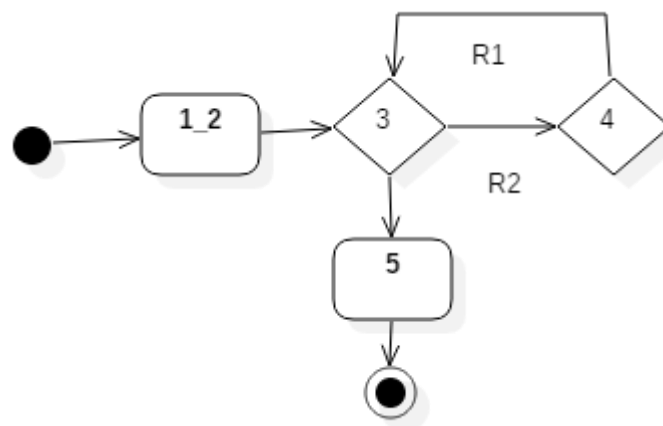
- Ruta 1: 1, 2, 3, 4, 6.
- Ruta 2: 1, 2, 3, 5, 6.



**Figura 58** - Diagrama de flujo del método identificarInformacion.

**Tabla 34** - Rutas del método identificarInformacion.

Ruta 1	
<b>Descripción:</b>	El renglón previamente leído y almacenado es analizado, este renglón contiene el formato específico de un <i>test step</i> .
<b>Resultado:</b>	Se regresa ese renglón leído.
Ruta 2	
<b>Descripción:</b>	El renglón previamente leído y almacenado es analizado, este renglón no contiene el formato específico de un <i>test step</i> .
<b>Resultado:</b>	Se regresa una cadena de caracteres vacía.



**Figura 59** - Diagrama de flujo del método dividirBytesDeMensaje.

La complejidad ciclomática del método **dividirBytesDeMensaje** es presentada en la Figura 59, la cual contiene dos regiones que permiten determinar las siguientes rutas (Tabla 35):

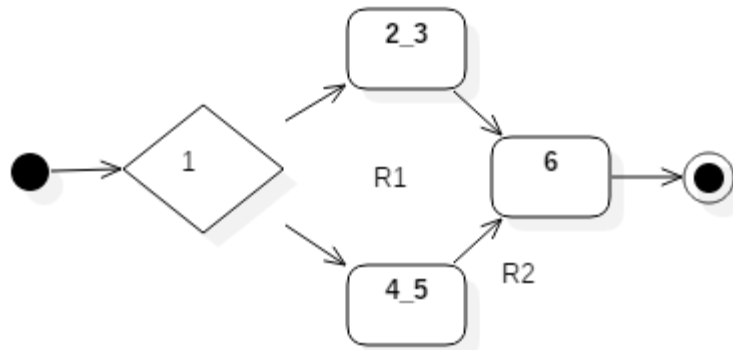
- Ruta 1: 1, 2, 3, 4, 3, 4, 5.
- Ruta 2: 1, 2, 3, 5.

**Tabla 35** - Rutas del método dividirBytesDeMensaje.

Ruta 1	
<b>Descripción:</b>	El mensaje que se encuentra en el <i>test step</i> (ya sea <i>Tx</i> y/o <i>Rx</i> ) es leído. Este mensaje cuenta con por lo menos dos bytes separados por un espacio.
<b>Resultado:</b>	Lista de mensajes divididos cada 8 bits (cada elemento de la lista es 1 byte).
Ruta 2	
<b>Descripción:</b>	El mensaje que se encuentra en el <i>test step</i> (ya sea <i>Tx</i> y/o <i>Rx</i> ) es leído. Este mensaje está vacío (No tiene bytes) debido a que se leyó un renglón que hace referencia al inicio de un <i>test case</i> .
<b>Resultado:</b>	Lista de mensajes sin elementos (vacía).

Para el método **identificarDLC** se muestra el siguiente diagrama de flujo (Figura 60) con las rutas generadas y descritas en la Tabla 36:

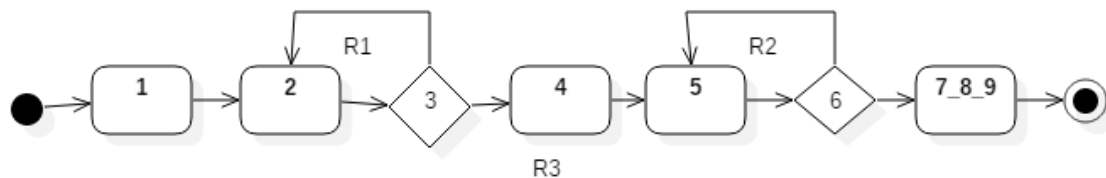
- Ruta 1: 1, 2, 3, 6.
- Ruta 2: 1, 4, 5, 6.



**Figura 60** - Diagrama de flujo del método identificarDLC.

**Tabla 36** - Rutas del método identificarDLC.

Ruta 1	
<b>Descripción:</b>	La lista de mensajes <i>Tx</i> del <i>test step test step</i> cuenta, en el cuarto byte, con un 1 en su primer <i>nibble</i> .
<b>Resultado:</b>	Identificación de un mensaje <i>multiframe</i> . Se establece el <i>DLC</i> del mensaje y se comienza a llenar la lista de mensajes <i>InHouseStep</i> .
Ruta 2	
<b>Descripción:</b>	La lista de mensajes <i>Tx</i> del <i>test step test step</i> no cuenta, en el cuarto byte, con un 1 en su primer <i>nibble</i> .
<b>Resultado:</b>	Identificación de un mensaje <i>single frame</i> . Se establece el <i>DLC</i> del mensaje y se comienza a llenar la lista de mensajes <i>InHouseStep</i> a partir del byte siguiente al <i>DLC</i> .



**Figura 61**- Diagrama de flujo del método construirMensajeTxInHouse.

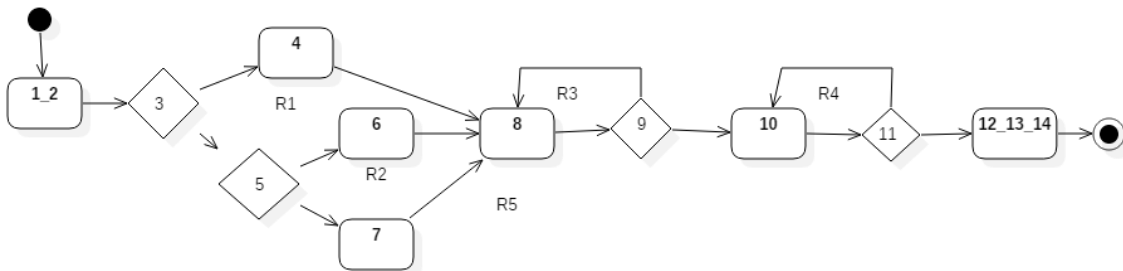
La complejidad ciclomática del método **construirMensajeTxInHouse** (Figura 61) es de tres regiones con las siguientes rutas (Tabla 37):

- Ruta 1: 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Ruta 2: 1, 2, 3, 2, 3, 4, 5, 6, 7, 8, 9.
- Ruta 3: 1, 2, 3, 4, 5, 6, 5, 6, 7, 8, 9.

**Tabla 37** - Rutas del método construirMensajeTxInHouse.

Ruta 1	
<b>Descripción:</b>	La lista de bytes del mensaje <i>Tx</i> , a partir del byte del <i>DLC</i> , solo cuenta con un mensaje, se coloca separador de mensajes.
<b>Resultado:</b>	Se agrega la información del mensaje <i>Tx</i> en el atributo de mensaje <i>Tx</i> del <i>InHouseStep</i> .
Ruta 2	
<b>Descripción:</b>	La lista de bytes del mensaje <i>Tx</i> , a partir del byte del <i>DLC</i> , cuenta con más de un mensaje, se coloca separador de mensajes.
<b>Resultado:</b>	Se agrega la información de los mensajes <i>Tx</i> en el atributo de mensaje <i>Tx</i> del <i>InHouseStep</i> .
Ruta 3	
<b>Descripción:</b>	La lista de bytes del mensaje <i>Tx</i> , a partir del byte del <i>DLC</i> , cuenta con más de un mensaje, se coloca separador de mensajes.
<b>Resultado:</b>	Se agrega la información de los mensajes <i>Tx</i> en el atributo de mensaje <i>Tx</i> del <i>InHouseStep</i> .

Como se puede identificar en la descripción de las rutas dos y tres, ambos tienen la misma entrada y generan la misma salida, esto es debido a que el segundo ciclo o flujo depende del primero, por lo que existe una mejora en cuanto al rendimiento del *software* aunque no tenga efecto en la funcionalidad del mismo.



**Figura 62** - Diagrama de flujo del método construirMensajeRxInHouse.

La complejidad ciclomática del método **construirMensajeRxInHouse** (Figura 62) es de cinco regiones sin embargo, como se explicó en la descripción del método anterior (**construirMensajeTxInHouse**), las estructuras de control representadas por los nodos ocho, nueve, diez y once están ligadas con la estructura de control (IF) del nodo tres, es decir, existen nodos dependientes por lo que las rutas a probar son las siguientes (Tabla 38):



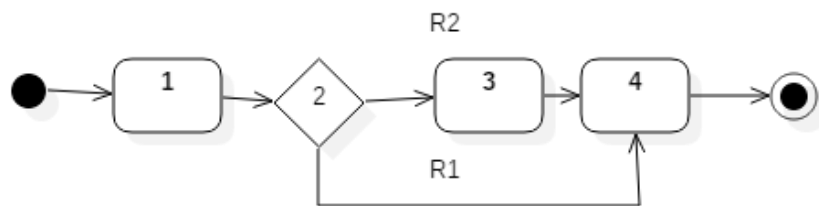
- Ruta 1: 1, 2, 3, 4, 8, 9, 10, 11, 12, 13, 14.
- Ruta 2: 1, 2, 3, 5, 6, 8, 9, 8, 9, 10, 11, 10, 11, 12, 13, 14.
- Ruta 3: 1, 2, 3, 5, 7, 8, 9, 8, 9, 10, 11, 10, 11, 12, 13, 14.

**Tabla 38** - Rutas del método construirMensajeRxInHouse.

Ruta 1	
<b>Descripción:</b>	Como entrada se tiene el <i>TestStep</i> ya construido con una lista de mensajes <i>Rx</i> . Este es el caso cuando esta lista no contiene ningún elemento a esperar como respuesta. Con esto no habría ninguna iteración de los nodos 8, 9, 10 y 11.
<b>Resultado:</b>	El atributo mensaje <i>Rx</i> del <i>InHouseStep</i> cuenta solamente con los caracteres de apertura y cierre.
Ruta 2	
<b>Descripción:</b>	Como entrada se tiene el <i>TestStep</i> ya construido con una lista de mensajes <i>Rx</i> (AA BB CC DD 10 0A 67 01 00 00 00 00 00 00 00).
<b>Resultado:</b>	El atributo mensaje <i>Rx</i> del <i>InHouseStep</i> fue fijado como un mensaje <i>multiframe</i> .
Ruta 3	
<b>Descripción:</b>	Como entrada se tiene el <i>TestStep</i> ya construido con una lista de mensajes <i>Rx</i> (AA BB CC DD 03 7F 22 33).
<b>Resultado:</b>	El atributo mensaje <i>Rx</i> del <i>InHouseStep</i> fue fijado como un mensaje <i>single frame</i> .

El método **identificarTipoDeRequest** cuenta con dos regiones de complejidad ciclomática (Figura 63) lo que genera las siguientes rutas (Tabla 39):

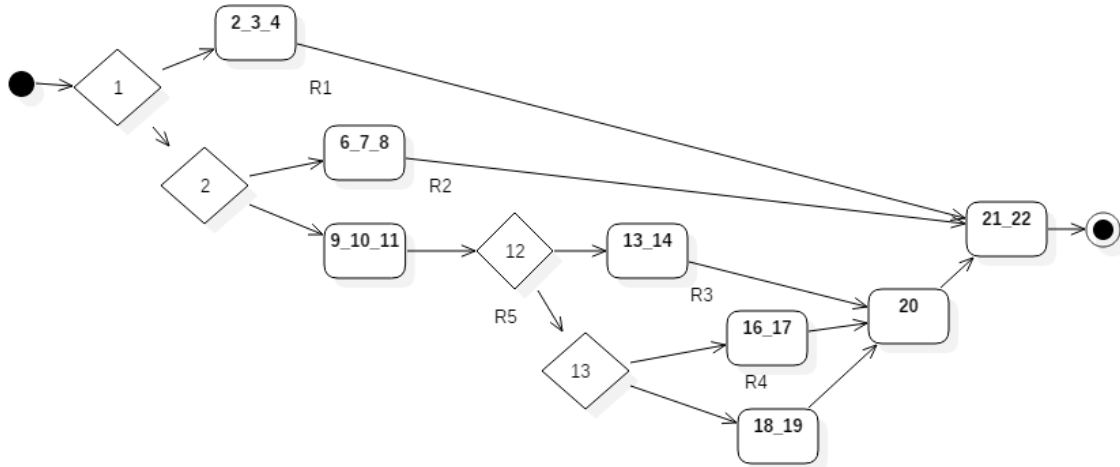
- Ruta 1: 1, 2, 3, 4.
- Ruta 2: 1, 2, 4.



**Figura 63** - Diagrama de flujo del método identificarTipoDeRequest.

**Tabla 39** – Rutas del método identificarTipoDeRequest.

Ruta 1	
<b>Descripción:</b>	La entrada es un mensaje con un <i>header</i> (ya sea <i>functional</i> o <i>physical request</i> ).
<b>Resultado:</b>	El comando identificador del tipo de <i>request</i> es agregado a la lista de <i>InHouseSteps</i> .
Ruta 2	
<b>Descripción:</b>	La entrada es un <i>step</i> que inicia un <i>test case</i> por lo que no contiene ningún mensaje.
<b>Resultado:</b>	No se agrega ningún comando a la lista de <i>InHouseSteps</i> .



**Figura 64** - Diagrama de flujo del método construirInHouseStep.

El método **construirInHouseStep** es aquel que reúne todos los métodos de la clase **ManejoDeDatos** este método genera cinco regiones desplegadas en el diagrama de flujo de la Figura 64 originando las siguientes rutas (Tabla 40):

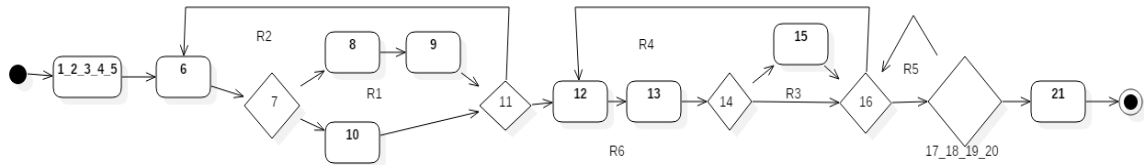
- Ruta 1: 1, 2, 3, 4, 21, 22.
- Ruta 2: 1, 5, 6, 7, 8, 21, 22.
- Ruta 3: 1, 5, 9, 10, 11, 12, 13, 14, 20, 21, 22.
- Ruta 4: 1, 5, 9, 10, 11, 12, 15, 16, 17, 20, 21, 22.
- Ruta 5: 1, 5, 9, 10, 11, 12, 15, 18, 19, 20, 21, 22.

**Tabla 40** – Rutas del método construirInHouseStep.

Ruta 1	
<b>Descripción:</b>	La entrada es el <i>TestStep</i> con todos sus atributos, el mensaje tiene un <i>header</i> con <i>physical request</i> .
<b>Resultado:</b>	Se construye un <i>InHouseStep</i> con el comando de transmisión de mensajes <i>physical</i> , con el <i>DLC</i> y construcción de mensajes <i>Tx</i> , <i>Rx</i> y descripción.
Ruta 2	
<b>Descripción:</b>	La entrada es el <i>TestStep</i> con todos sus atributos, el mensaje tiene un <i>header</i> con <i>functional request</i> .
<b>Resultado:</b>	Se construye un <i>InHouseStep</i> con el comando de transmisión de mensajes <i>functional</i> , con el <i>DLC</i> y construcción de mensajes <i>Tx</i> , <i>Rx</i> y descripción.
Ruta 3	
<b>Descripción:</b>	La entrada es el <i>TestStep</i> que no empata con ningún <i>header</i> debido a que es el inicio de un <i>test case</i> .
<b>Resultado:</b>	Se construye un <i>InHouseStep</i> descriptivo con el comando de inicio de conjunto de mensajes ( <i>test case</i> ).
Ruta 4	

<b>Descripción:</b>	La entrada es el <i>TestStep</i> que no empata con ningún <i>header</i> ni con el inicio de un <i>test case</i> debido a que es un mensaje descriptivo de paro de <i>tester present</i> (“ <i>Stop sending tester present</i> ”).
<b>Resultado:</b>	Se construye un <i>InHouseStep</i> descriptivo con el comando de pausa de transmisión de mensajes asignándole un tiempo de espera definido.
Ruta 5	
<b>Descripción:</b>	La entrada es el <i>TestStep</i> que no empata con ningún <i>header</i> ni con el inicio de un <i>test case</i> y tampoco con la descripción de paro de <i>tester present</i> .
<b>Resultado:</b>	Se construye un <i>InHouseStep</i> descriptivo con el comando de texto informativo asignándole la descripción del <i>step</i> .

La complejidad ciclomática del método **analizarInformacion** (Figura 65) es de seis regiones con las siguientes rutas y descripciones (Tabla 41):



**Figura 65** - Diagrama de flujo del método *analizarInformacion*.

- Ruta 1: 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21.
- Ruta 2: 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21.
- Ruta 3: 1, 2, 3, 4, 5, 6, 7, 8, 9, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 12, 13, 14, 15, 16, 17, 18, 19, 20, 16, 17, 18, 19, 20, 21.
- Ruta 4: 1, 2, 3, 4, 5, 6, 7, 10, 11, 6, 7, 10, 11, 12, 13, 14, 16, 12, 13, 14, 16, 17, 18, 19, 20, 17, 18, 19, 20, 21.
- Ruta 5: 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 6, 7, 10, 11, 12, 13, 14, 15, 16, 12, 13, 14, 16, 17, 18, 19, 20, 16, 17, 18, 19, 20, 21.
- Ruta 6: 1, 2, 3, 4, 5, 6, 7, 10, 11, 6, 7, 8, 9, 11, 12, 13, 14, 16, 12, 13, 14, 15, 16, 17, 18, 19, 20, 16, 17, 18, 19, 20, 21.

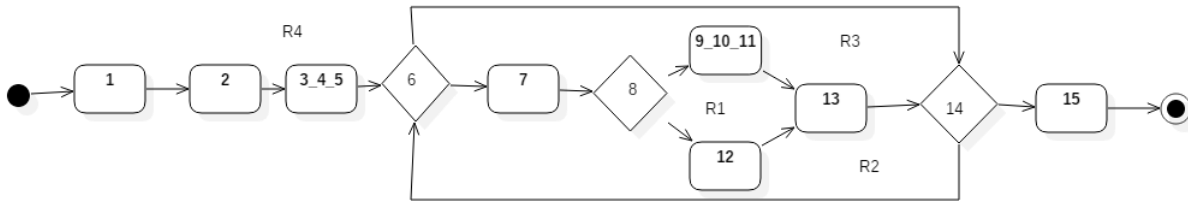
**Tabla 41** – Rutas del método *analizarInformacion*.

Ruta 1	
<b>Descripción:</b>	La <i>sheet</i> leída solo contiene un <i>step</i> .
<b>Resultado:</b>	La lista de <i>InHouseSteps</i> solo contiene un solo <i>step</i> .
Ruta 2	
<b>Descripción:</b>	La <i>sheet</i> leída solo contiene un <i>step</i> pero éste no tiene el formato establecido para ser leído.
<b>Resultado:</b>	La lista de <i>InHouseSteps</i> se encuentra vacía.
Ruta 3	
<b>Descripción:</b>	La <i>sheet</i> leída contiene dos (o más) <i>steps</i> con el formato específico para su lectura.
<b>Resultado:</b>	La lista de <i>InHouseSteps</i> almacena a los dos <i>steps</i> .

Ruta 4	
<b>Descripción:</b>	La <i>sheet</i> leída contiene dos (o más) <i>steps</i> que no tienen el formato especificado para que el sistema los almacene.
<b>Resultado:</b>	La lista de <i>InHouseSteps</i> se encuentra vacía.
Ruta 5 y 6	
<b>Descripción:</b>	La <i>sheet</i> leída contiene uno (o más) <i>steps</i> tiene el formato que permite su lectura y uno (o más) <i>steps</i> que no tienen dicho formato.
<b>Resultado:</b>	La lista de <i>InHouseSteps</i> almacena aquellos mensajes que tienen el formato específico de lectura.

El método **administrarActividades** tiene una complejidad ciclomática de cuatro regiones ilustradas en la Figura 66. Las rutas que surgen de este método son (Tabla 42):

- Ruta 1: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15
- Ruta 2: 1, 2, 3, 4, 5, 6, 14, 15.
- Ruta 3: 1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15.
- Ruta 4: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15.



**Figura 66** - Diagrama de flujo del método administrarActividades.

**Tabla 42** – Rutas del método administrarActividades.

Ruta 1	
<b>Descripción:</b>	La ejecución del <i>software</i> se realiza eligiendo el archivo a analizar.
<b>Resultado:</b>	Archivo elegido leído, analizado y a partir de éste, es generado uno nuevo.
Ruta 2	
<b>Descripción:</b>	Se despliega la ventana de elección de archivo pero no se elige ninguno.
<b>Resultado:</b>	La ventana de salida es desplegada para culminar con la ejecución del sistema.
Ruta 3	
<b>Descripción:</b>	Un archivo <i>XLSX</i> es elegido
<b>Resultado:</b>	Existe un error en el archivo elegido debido a la ruta o contenido del mismo. Se despliega el mensaje de salida para que el usuario escoja volver al explorador o terminar la ejecución del programa.
Ruta 4	
<b>Descripción:</b>	Dos archivos son elegidos para su análisis.

---

<b>Resultado:</b>	Un primer archivo es elegido y se genera su archivo compatible. El sistema pregunta al usuario la re-ejecución y se selecciona otro archivo del cual se genera un segundo archivo compatible.
-------------------	---

---

Entre las conclusiones que se obtuvieron al realizar la prueba de caja blanca se encontró una mejora en los métodos **construirMensajeTxInHouse** y **construirMensajeTxInHouse**. Esta mejora se debe a la dependencia entre regiones, lo que determina caminos similares. Los puntos detectados fueron corregidos conforme fueron descubiertos.

## 2.5.2 Pruebas de integración

Aludiendo a Sommerville (2005), las pruebas de integración son aquellas en las que se tiene acceso al código fuente del sistema para identificar problemas y componentes que tienen que ser depurados. Para determinar el cumplimiento total de los requerimientos fue necesario acoplar el sistema codificado a un entorno de trabajo de In-house software. Lo anterior es posible generando pruebas que abarquen el proceso de automatización por completo, es decir, determinar entradas, o casos de prueba, que permitan verificar el correcto funcionamiento de ambas partes (software) obteniendo como resultado mensajes recibidos y comparados.

Como se mencionó al inicio del capítulo II, el área de diagnóstico cuenta con 16 *test procedures*. Se tiene un promedio de 11 *sheets*, 34 *test cases* y 454 *test steps* por documento. De esos 454 *test steps*, 50 no son especificados como mensajes con *headers* de 29 bits a transmitir (comentarios, paro de *tester present*, entre otros) por lo que el promedio total de mensajes que se transmiten es de 404 *steps*. Tomando el dato anterior como referencia se hizo uso de un par de *test procedures* que, entre otras cosas, contarán con el número de *test steps* promedio calculado, así como la configuración que permitiera identificar el cumplimiento de los requerimientos.

Entre los errores y posibles mejoras se encontró que el método **administrarActividades** permite determinar si una *sheet* es compatible o no, por lo que el *software* despliega un mensaje de error regresando a la ventana del explorador para elegir nuevamente el archivo, lo que se traduce en un re-trabajo para volver a ubicar el archivo previamente elegido. Además, se pudo detectar un mal manejo de información al elegir una *sheet* que no contaba con mensajes a transmitir. Estos errores fueron corregidos durante el proceso de pruebas.

Por otra parte, una vez transformadas las *sheets* que contenían mensajes a transmitir, éstas se incorporaron al ambiente de trabajo de *In-house software* mediante la importación de los archivos generados.

Tomando la cantidad promedio de *test steps* por *test procedure*, así como el número de *sheets* que se traduce en el número de archivos generados y el tiempo de importación, el tiempo de ejecución calculado fue de 10 minutos, resaltando que la prueba fue realizada por el sustentante del presente reporte quien conoce la estructura del *software*.

Teniendo los mensajes de cada una de las *sheets* dentro de *In-house software*, se requiere seleccionar aquellos mensajes que componen a un *test procedure* y ejecutar la transmisión de mensajes. El promedio de *test steps* por *test case* es de 13 *steps* por lo que el tiempo promedio de ejecución de un *test case* es de 1 minuto, sin embargo, no se está considerando el tiempo de selección del siguiente *test case* por lo que el tiempo total de ejecución de un *test procedure* es de:

$$1 \text{ minuto por } test \text{ step} * 34 \text{ test cases} = 34 \text{ mins}$$

$$\text{Tiempo promedio de ejecución por } test \text{ procedure} = 34 \text{ mins} + \text{selección de } test \text{ case} + \text{cambio entre sheets}$$

Con la finalidad de obtener un resultado que se acerque a la realidad esperada se estimó que el tiempo de la selección de *test cases* y el cambio de *sheet* es de un minuto por cada actividad (en promedio 33 minutos de selección completa del *test case*), por lo que, si se tienen 34 *test cases* y de las 11 *sheets* que lo componen, seis son usadas para la transmisión de mensajes (debido a que las demás se utilizan como referencia), entonces:

$$\text{Tiempo promedio de ejecución por } test \text{ procedure} = 34 \text{ mins} + 33 \text{ mins} + 6 \text{ mins}$$

o

$$1 \text{ hr } 13 \text{ mins}$$

Adicionando al resultado anterior el tiempo de generación de los archivos (actividad realizada por el *software* codificado), se tiene un gran total de una hora con 23 minutos para la ejecución completa de un *test procedure*.

Una vez ejecutados los *test procedures* para la prueba de integración, se pudo encontrar una discrepancia en la transmisión de mensajes *multi-frame* con un tipo de *request functional*, puesto que después de ser enviada la primer trama (*first frame*) y haber recibido la trama de control de flujo (*flow control*) por parte del *ECU*, las tramas consecutivas (*consecutive frames*) eran enviados con un direccionamiento físico (*physical*) lo que invalidaba la prueba.

Para la corrección de este problema fue necesario comunicarse con el equipo de desarrollo *In-house software* comentando la inconsistencia y mostrando las pruebas

que validaban la existencia del error. El equipo de desarrollo del *software* corroboró la falla y creó una actualización para la transmisión de ese tipo de mensajes. Este cambio afectó al *software* que se codificó para la conversión de formatos. El cambio se vio reflejado en el agregado de líneas de código (no más de 10).

La primera versión del proceso que involucra la implementación del *software* que permite unificar dos formatos de información diferentes y del *software* de comunicación de datos, que en conjunto permiten la validación de casos de prueba de las *ECUs* que soportan comunicación de diagnóstico *ISO 14229*, fue presentada al líder ingeniero de validación para la verificación del cumplimiento de los requerimientos descrita en el capítulo III del presente reporte.

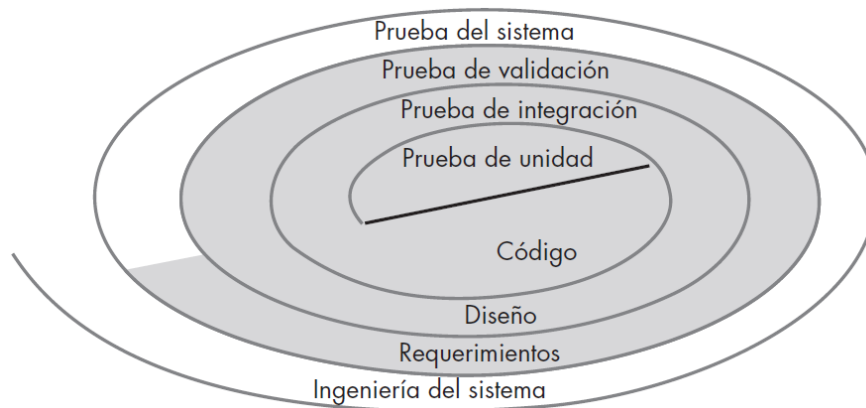
## Capítulo III. Pruebas de aceptación de la solución

---

*En este capítulo se documentan las pruebas realizadas por los ingenieros del grupo de comunicación de diagnóstico, así como los resultados obtenidos y la puesta a punto del sistema integrado.*

### 3.1 Pruebas de validación con el usuario final y optimización

Pressman (2010) muestra un diagrama (Figura 67) donde resalta el proceso de pruebas de software. El objetivo principal de las pruebas de validación con el usuario, una vez realizada la verificación del software integrado, es garantizar que todos los requerimientos hayan sido cubiertos. Lo anterior fue posible solicitando apoyo al grupo de comunicación de diagnóstico (tres ingenieros de validación) para realizar un conjunto de actividades que permitieran la obtención de resultados desde su punto de vista y consolidando el cumplimiento de los requerimientos.



**Figura 67** – Estrategia de prueba del software (Pressman, 2010).

Dentro del ejercicio de validación de requerimientos con el grupo de diagnóstico, se les proporcionó una guía de uso de los sistemas integrados para realizar las actividades consideradas para la prueba. Como principal tarea, se solicitó a cada miembro del grupo realizar la transmisión de mensajes de tres *test procedures* diferentes, sin ayuda alguna por parte del desarrollador a menos que el sustentante de la prueba tuviera un momento de incertidumbre y no supiera los siguientes pasos a realizar. Los *test procedures* tomados para esta prueba fueron diferentes a los usados en la prueba de integración, abarcando un mayor umbral en la prueba de los *test procedures* con los que cuenta el grupo de diagnóstico.



Los resultados obtenidos en las pruebas de verificación realizados por los ingenieros del grupo de comunicación de diagnóstico en cuanto al uso del *software* para la transmisión de mensajes (ver Tabla 43), fueron analizados y comparados contra los requerimientos establecidos, cumpliendo con el cien por ciento de éstos. Sin embargo, el líder ingeniero de validación y mentor del proyecto, hizo énfasis en los tiempos medidos de las pruebas pues, además de la ejecución, se debe hacer una verificación exhaustiva de las respuestas recibidas de cada *test step*, debido a que pueden existir casos en que alguno de los mensajes de los *test steps* se encuentren escritos de manera incorrecta y esto interfiera en los resultados obtenidos, re-trabajando en las correcciones del problema desde el *test procedure* y volviendo a ejecutar todo el proceso.

Otro comentario fehaciente por parte del líder ingeniero de validación, fue el hecho de seleccionar *test case* por *test case* para la ejecución de los *test steps*, no obstante, se le hizo notar que se estaba cumpliendo con uno de los requerimientos validados por él mismo. Entonces, se llegó a un acuerdo con el ingeniero de validación para reducir el trabajo de seleccionar los *test cases* simplificando la tarea en una sola ejecución por *sheet*, para esto fue necesario analizar nuevamente el comportamiento de los *test cases* en virtud de que cada uno de estos puede tener mensajes que involucren *requests* al *ECU* donde existan tiempos límites o *time outs* de dichos comportamientos; tal es el caso de los mensajes *tester present* que, si no es detenido mediante una descripción dentro del *test procedure*, éste continua hasta un tiempo determinado como lo especifica el estándar *ISO 14229*.

Analizado el nuevo requerimiento se estableció una solución que abarcara el uso de un comando que permitiera ejecutar una pausa por un lapso de tiempo establecido (similar al comando de detención del *tester present*) permitiendo al *ECU* recobrar un estado por defecto, al no ser transmitido ningún mensaje de diagnóstico. Este comando fue implementado al inicio de cada *test case* permitiendo seleccionar toda la *sheet* para su ejecución y no interfiriendo en el completo comportamiento generado por cada *test case*.

**Tabla 43** – Resultados de pruebas de verificación con el usuario, se recomienda tener presente los datos desplegados en la Tabla 12 donde se especifican los requerimientos.

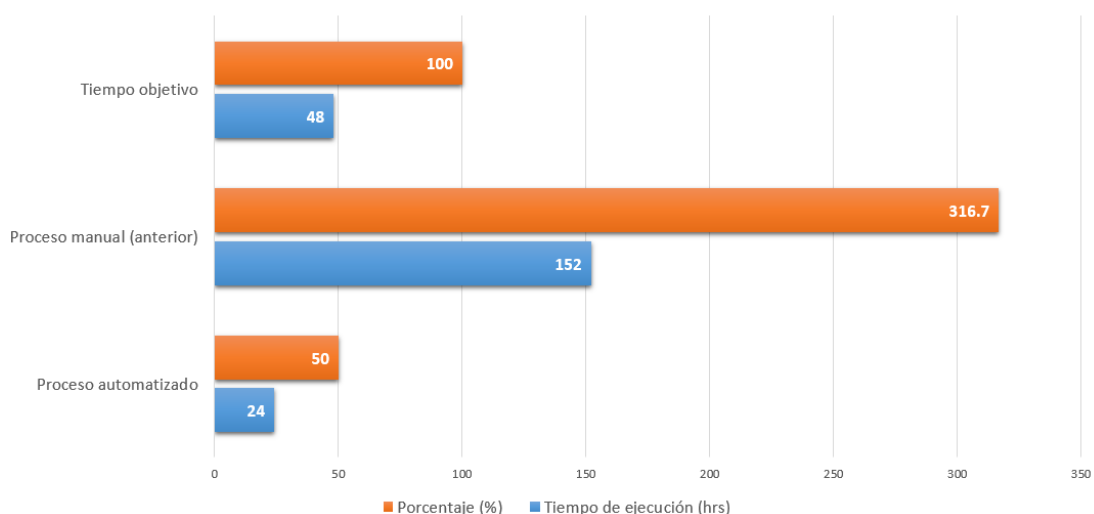
Pregunta	Target	Ingeniero 1	Ingeniero 2	Ingeniero 3
¿El software usado te permitió crear mensajes a partir de un test procedure?	Si	Si	Si	Si
¿El software usado te permitió transmitir mensajes con <i>headers</i> de 29 bits?	Si	Si	Si	Si
¿El software usado te permitió transmitir mensajes <i>single</i> y <i>multi-frames</i> ?	Si	Si	Si	Si
¿El software usado brindó la transmisión de mensajes <i>flow control</i> ?	Si	Si	Si	Si
¿El software usado permitió el manejo de sesiones mediante mensajes de <i>tester present</i> ?	Si	Si	Si	Si
¿El software usado permitió transmitir mensajes a un <i>ECU</i> en específico o a todos los conectados a la misma red de comunicación?	Si	Si	Si	Si
¿El software usado desplegó el mensaje que recibió como respuesta por parte del (los) <i>ECU(s)</i> (en caso de existir una respuesta esperada)?	Si	Si	Si	Si
¿El software usado desplegó una comparación entre el mensaje esperado y el recibido?	Si	Si	Si	Si
En una escala del 1 al 3, donde uno es difícil y 3 es fácil, indica el nivel de dificultad percibido al utilizar el <i>software</i> que permite la generación de archivos a partir del <i>test procedure</i> .	3	2	2	1
En una escala del 1 al 3, donde uno es difícil y 3 es fácil, indica el nivel de dificultad percibido al utilizar el <i>software</i> que permite la transmisión y recepción de mensajes.	3	1	2	2
¿La guía entregada al principio de la prueba fue de utilidad para la ejecución de las actividades solicitadas?	Si	Si	Si	Si
Tiempo promedio de tarea (ejecución de <i>test procedure</i> )	3h	2h 20m	2h 24m	2h 27m

Dentro de las acciones efectuadas para esta mejora se encuentran: la actualización de las clases **Administración** y **ManejoDeDatos**, configurando banderas que permitieran detectar el inicio de un nuevo *test case*; asimismo iniciar la construcción del comando establecido con un tiempo definido.

Una vez implementado el cambio se ejecutaron las pruebas unitarias, de integración y de verificación por parte del usuario. El único cambio que sufrieron los resultados obtenidos en las pruebas de verificación fue la reducción de tiempo (sin ninguna afectación en el cumplimiento de los requerimientos funcionales).

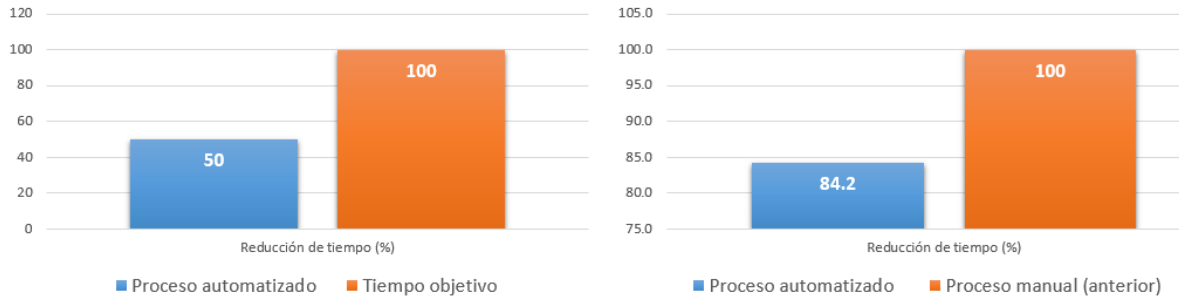
Dependiendo del contenido del *test procedure* (número de *test steps*) puede existir una variación en el tiempo de ejecución. En promedio, combinando la ejecución del *software* el cual extrae la información, la importación de los archivos generados, la selección única de toda la *sheet* y la transmisión de mensajes, se requiere una hora con 30 minutos por *test procedure*.

Considerando el tiempo promedio obtenido de las pruebas de verificación con el usuario, se puede estimar que la sola ejecución de los 16 *test procedures* con los que cuenta el grupo de comunicación de diagnóstico puede ser efectuada en un lapso de 24 horas (una hora 30 minutos por 16 *test procedures*). Si tomamos en cuenta el tiempo disponible de los bancos de prueba para esta actividad (12 horas a la semana) y el tiempo límite para obtener los resultados (48 horas al mes) además de comparar el tiempo que les tomaba a los ingenieros de comunicación de diagnóstico realizar las pruebas (proceso manual con *Vehicle Spy*), se pueden obtener los resultados desplegados en la Figura 68.



**Figura 68** - Comparación del tiempo destinado al proceso de transmisión de mensajes.

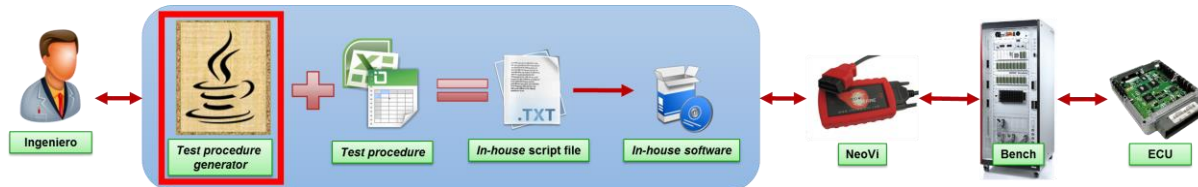
Como se puede apreciar en la Figura 68, el proceso usado anteriormente por el grupo de comunicación de diagnóstico excedía más de un 300% el tiempo objetivo destinado para dicha actividad. Una vez realizadas las pruebas de verificación del usuario, se pudo determinar el tiempo que tomaría ejecutar el mismo número de pruebas pero con un proceso y *software* diferentes. Con este nuevo proceso se obtiene una reducción de tiempo de un 50% en comparación con el tiempo objetivo y un 84.21% en comparación con el tiempo del proceso manual (Figura 69).



**Figura 69** – Comparación de reducción de tiempos (objetivo y proceso anterior).

En comparación con el número de actividades manuales dedicadas al ingreso de información, transmisión de mensajes y verificación de respuestas recibidas de forma manual, las actividades correspondientes para el nuevo proceso automatizado son (Figura 70):

- Ejecutar el *software* que permite la extracción de la información
- Importar los archivos generados al *software* de transmisión de mensajes.
- Seleccionar toda la *sheet*.
- Ejecutar la transmisión de mensajes.



**Figura 70** – Actividades para el nuevo proceso de transmisión de mensajes.

### 3.2 Implantación del nuevo proceso

Como parte del desarrollo del proyecto redactado en el presente reporte de aplicación de conocimientos, fue necesario realizar la documentación correspondiente del *software* implementado (*test procedure generator*), para esto se llevó a cabo una lectura completa de la codificación con la finalidad de enunciar, mediante comentarios dentro de las clases, la funcionalidad de una o varias líneas de código. Además, se elaboró un documento que contiene diagramas y pseudocódigo referentes al diseño del *test procedure generator*.

Fue creada una guía que permitiera a usuarios que no hayan tenido interacción alguna con las nuevas herramientas de *software* conocer su funcionalidad básica para

poder efectuar la transmisión de mensajes. Del mismo modo, se impartió un entrenamiento que consistió en una plática acerca del uso de las herramientas de *software* así como la ejemplificación de la ejecución de un *test procedure* realizada por los mismos usuarios.

Dentro de los requerimientos de *hardware* y *software* necesarios para la puesta a punto del nuevo proceso automatizado, no se requirió ningún elemento extra en las computadoras destinadas para efectuar tal operación; sin embargo, éstas deben contar con una versión de *Java JRE* (Java SE Runtime Environment) igual o superior a la versión siete (en cualquiera de sus sub-versiones), una versión de Python especial para bancos de prueba y el *patch* generado por el equipo de Europa para la corrección de la transmisión de *functional requests*. Todos estos puntos fueron marcados dentro de los documentos ya mencionados.

Debido a que el nuevo proceso implantado involucra dos herramientas de *software*, es necesario tener contacto con el equipo de desarrollo de Europa para el mantenimiento de *In-house software*, brindando una retroalimentación que ayude en la actualización y manejo de errores de dicha herramienta para las nuevas versiones. Por otra parte, se generó un control de versiones básico para el *test procedure generator*, identificando a la primer versión liberada (versión 1.0.0) como base para nuevas actualizaciones y una vez liberada dicha actualización del *software*, generar un número de versión dependiendo el tipo de cambio (Tabla 44).

**Tabla 44** – Control de versiones dependiendo del tipo de cambio.

Identificador de versiones	X.0.0	0.X.0	0.0.X
<b>Descripción</b>	Cambios de interfaz y/o funcionalidad	Solución de errores	Mejoras en el <i>software</i> que no involucran errores o cambios en interfaz.

Una vez liberada la primera versión del *test procedure generator*, el equipo de comunicación de diagnóstico del Centro Regional de Ingeniería de Toluca ha podido realizar la validación correspondiente a los *test procedures* con los que cuenta, brindando una retroalimentación al grupo de expertos en desarrollo de *software* de comunicación de diagnóstico en los tiempos establecidos. Del mismo modo, el *Test*

*procedure generator* ha estado en constante actualización gracias a la puntual comunicación de aquellos usuarios quienes han interactuado con el sistema.

Como resultado de la implantación de este nuevo proceso automatizado, mediante un *software* que permita realizar la validación de los casos de prueba de las *Unidades de Control Electrónicas*, que soportan comunicación de diagnóstico ISO 14229 se puede concluir que, tomando en cuenta la funcionalidad y eficiencia, en conjunto, *In-house software* y *Test procedure generator* cumplen cabalmente los requerimientos establecidos, superando los métricos de eficiencia respecto al tiempo de ejecución.

De esta forma, la empresa automotriz se ha beneficiado con la reducción de los tiempos de pruebas que repercuten en la validación del código embebido en los controladores *ECM* y *TCM* de los 16 *test procedures* con los que cuenta actualmente, incrementando la eficiencia en el uso de herramientas, tiempo y recursos que se encuentran disponible por el grupo de validación del Centro Regional de Ingeniería de Toluca. Lo anterior se debe al uso compartido de los bancos de prueba con otros equipos de validación, lo que permite la coordinación del tiempo dedicado por los ingenieros a la ejecución de *test procedures*.

También, el ingeniero de validación puede otorgar una retroalimentación robusta y a tiempo de las pruebas realizadas a los expertos en desarrollo de *software* de comunicación de diagnóstico permitiendo reducir el riesgo de incumplimiento de estándares, logrando detectar en etapas tempranas de desarrollo posibles errores de funcionalidad del código programado.

## Conclusiones y trabajo futuro

---

### Conclusiones

El proyecto desarrollado y descrito en el presente reporte, cumplió con el objetivo general de forma positiva en virtud de que los requerimientos establecidos por el líder ingeniero de validación fueron cubiertos en su totalidad, superando los resultados esperados.

El uso de una herramienta de *software* existente como solución de la problemática dada, no debe minimizar el trabajo efectuado, pues para encontrar esa posible alternativa fue necesario realizar una investigación profunda sobre aquellas herramientas ya desarrolladas e implementadas que cumplieran con los requerimientos solicitados y al menor costo de implementación posible.

Gracias a este análisis, fue posible tener interacción con grupos y proveedores de otras naciones propiciando relaciones laborales con la intención de obtener los mejores resultados posibles para la ejecución de las pruebas de validación del software de comunicación pues el equipo de desarrollo de Europa, así como en otras naciones donde la empresa automotriz tiene centros de ingeniería, cuentan con procedimientos de validación del software embebido en controladores similares al utilizado en Toluca.

Así que, a través del uso de una metodología de *Design For Six Sigma* en conjunto con una metodología clásica o de cascada, fue posible el desarrollo de un *software* que permitiera la unificación de dos formatos de información diferentes, permitiendo la fabricación de un puente de comunicación entre documentos que incluyeran datos acerca de las pruebas de validación, previamente desarrollados (*test procedures*) y la herramienta de ejecución automática de pruebas. Para el cumplimiento del objetivo, con el uso de estas dos herramientas se creó una solución híbrida, tomando de cada metodología, aspectos que se complementaron.

En palabras del líder de validación y cliente directo del desarrollo del proyecto mencionó que tanto los avances como la liberación del *software* fueron completados en tiempo y forma. Además, hizo mención acerca de que la herramienta de *Test procedure generator* así como el uso de *In-house software* en conjunto cumplen con los requerimientos que se plantearon al inicio del proyecto resaltando la ayuda proporcionada por éstos en cuanto a la reducción de tiempos, el cual puede servir para desempeñar de manera satisfactoria sus actividades.

También, el líder de validación mencionó lo siguiente: “El nuevo software ha sido de mucha ayuda porque nos permite concentrar en la tarea más importante de nuestro proceso que es convertir los requerimientos funcionales en los procedimientos de pruebas y no en la conversión del procedimiento a los mensajes usados por la herramienta de diagnóstico, tarea a la que teníamos que dedicarle un tiempo considerable. Durante el uso que le hemos dado, la herramienta *Test procedure generator* ha demostrado ser muy consistente y confiable obteniendo una correcta conversión del procedimiento a los mensajes de diagnóstico”.

### **Trabajo futuro**

Como se documentó en el desarrollo del proyecto, un *test procedure* está compuesto de entre un 10% a 15% por *test steps* que no son considerados como mensajes CAN con *headers* de 29 bits, pues algunos de estos cuentan con *headers* de 11 bits o establecen un patrón de comportamiento como la generación de errores o configuración de valores del simulador de la *bench* (cambio de *engine speed*, diferente *power mode*, activación de *DTCs*, entre otros).

Debido a lo anterior, es conveniente indagar en aquella funcionalidad que *In-house software* brinda para el manejo de variables dentro del modelo (*software*) utilizado por las *benches*, dado que, desde un inicio, cuenta con características destacadas como modificación de los valores del simulador (*software de la bench*) en tiempo real mediante la configuración de archivos y comandos ya preestablecidos. Tomando esto en consideración, el impacto que esta mejora involucraría sería exclusivamente para el *test procedure generator*.

Otra de las mejoras sobre las que ya se está trabajando en conjunto con un integrante del área de comunicación de diagnóstico es la creación de un *patch* que permita, mediante el uso de algoritmos de cifrado (*AES-CMAC*), desbloquear los controladores en sus diferentes niveles de seguridad con la finalidad de abarcar un umbral más amplio en la validación de uno de los *servicios UDS*.

Además, “se tiene un área de oportunidad en el uso de la interfaz gráfica, al seleccionar los *test procedures* pues actualmente el *Test procedure generator* no almacena la *path* del último archivo seleccionado, lo que hace que trabajo de búsqueda de información sea laborioso”, contribuyó el líder de validación.

Por último, se pretende, mediante una estandarización de los *test procedures*, la extracción de información identificando aquellas *sheets* que cuenten con *test cases* e ignorando aquellas que solamente tengan información extra o de configuración de



forma automática reduciendo el trabajo de selección que puede generar la actual versión del *Test procedure generator*. Buscando que las pruebas de validación del software embebido en los controladores puedan realizarse en otros bancos de prueba situados en diferentes lugares como Estados Unidos trascendiendo fronteras y alcances, creando un proceso global.

### ***Competencias y aprendizajes adquiridos***

El desarrollo de este proyecto me permitió poner en práctica los conocimientos adquiridos durante la carrera profesional dentro de un ámbito completamente laboral en una empresa automotriz de clase mundial. Por tanto, al estar en contacto con ingenieros expertos en el área de comunicación de diagnóstico me brindó un panorama diferente al marcado por las materias cursadas en la Universidad, es decir, pude reconocer el uso de los estándares y protocolos de comunicación, más allá de las redes de computadoras usuales que se pueden observar como redes *LAN* o *WAN* adecuándome a la ambigüedad de las temáticas que conlleva la adaptación inicial y aprendiendo rápidamente aquellos conceptos clave para el desarrollo de este proyecto.

El uso de redes intra-vehiculares a partir de controladores con código embebido hace posible contar con autos inteligentes y en muy poco tiempo (ya visible por marca automotriz *Tesla*) la conducción vehicular autónoma. En consecuencia, gracias a los entrenamientos brindados por el mentor de este proyecto, tuve la oportunidad de adquirir conocimiento relacionado a los estándares y arquitectura utilizados por la empresa automotriz para la construcción y validación del software embebido en los controladores.

Gracias al desarrollo de este proyecto y como se hizo mención en este reporte, interactué con proveedores de otras compañías ubicadas fuera de México y con el equipo de desarrolladores de *In-house software* en Europa, lo que me ayudó a mejorar mis habilidades en el idioma inglés, conjuntamente desarrollando astucia en las relaciones interpersonales.

Por otra parte, fue necesario involucrar habilidades para la planeación de actividades para el desarrollo de proyectos, pues la entrega se tuvo que ajustar al tiempo solicitado por parte del mentor y la empresa automotriz. Asimismo, dentro del aprendizaje sobre la marcha de los tópicos antes señalados, mejoré mis habilidades de literatura especializada del estado del arte, así como mis habilidades de comprensión de información escrita en el idioma inglés.

Respecto a implantación del nuevo proceso automatizado puedo resaltar el hecho de que se podría esperar un desinterés o rechazo por parte de los usuarios finales en el uso de un *software* distinto o actividades diferentes a las acostumbradas; sin embargo, el grado de aceptación, así como la adaptación a las nuevas herramientas del *software*, fueron favorables para su pronta puesta en marcha.

Desde otro punto de vista, la creación de un nuevo proceso que permita validar automáticamente los casos de prueba de los módulos que soportan comunicación de diagnóstico vehicular *ISO 14229* con base en la implantación de un *software* de comunicación de datos y de un *software* unificador de formatos, me permitió establecer un esquema innovador para dicha validación dentro de la empresa automotriz.

Gracias a lo anterior, tuve la oportunidad de competir en un concurso interno de la compañía en el área de *tren motriz* a nivel global, el cual reconoce aquellos proyectos que demuestran un trabajo innovador además de reducciones de tiempo y costo así como el impacto que tiene para el cliente final, logrando que este proyecto fuera presentado mediante *slides* en una expo que se dio lugar en Michigan, Estados Unidos, en Febrero de 2016.

De la misma forma, el desarrollo del presente proyecto me permitió obtener la certificación de *Green Belt*, ofrecida por la compañía automotriz, que califica el correcto uso, implementación y conocimiento de la metodología *DFSS*.

El haber trabajado en una organización de la iniciativa privada, me permitió adquirir conocimientos incalculables y una grata experiencia de trabajo en equipo. Aplicar los conocimientos adquiridos durante la carrera, sin duda ha sido una gran base para la culminación de este proyecto, sin embargo, solamente son un parteaguas para el inicio de una trayectoria profesional.

# Anexo A – Formato de validación de requerimientos

---

## TEST PROCEDURES AUTOMATION FOR TREC DIAGNOSTIC TRB REQUIREMENT VALIDATION DOCUMENT

### 1 Introduction

This document is a software requirement specification that will help in the project development for the current validation of *diagnostic communication test procedures* for controllers that support the *ISO 14229 standard*.

#### 1.1 Purpose

The purpose of this specification is to generate a complete view of the problem resolution requirements of the *test procedure execution process*. Below are listed the requirements requested by the Technical Lead of BSW Verification.

#### 1.2 Acronyms and abbreviations

ISO: International Standardization Organization.

BSW: Basic Software.

Tx: Transmit message.

Rx: Received message.

ECU: Electronic control Unit.

CAN: Controller Area Network.

OS: Operative System.

TRB: Technical Review Board.

ECM: Engine Control Module.

TCM: Transmission Control Module.

GM: General Motors.

### 2 Description

The present project must meet all the requirements indicated in this document.

#### 2.1 User interfaces

The present project must provide an user-friendly interface for any validation engineer from the Diagnostics TRB by following next requirements:

1. Display the transmitted message description, transmitted messages and the expected response. Similar to the *test procedure* structure.
2. Display the comparison between the expected message and the received message with a FAIL or PASS note.
3. Run all *test steps* from every *test case* with just one button.
4. All alert, help or inform messages should be in English language.

## 2.2 Hardware interfaces

According to the current resources and constrains of the Engineering Center, the project must meet:

1. Usage of *NeoVi Fire* as the communication interface (tester tool).
2. Usage of *ECM* and *TCM Test Benches* as the testing environment.

## 2.3 Software interfaces

The *software tool* should extract information from specific documents (*test procedures*), for this reason, it will be necessary the usage of libraries or interfaces which help to performed this extraction.

## 2.4 Design constrains

Design and implementation of this project are not limited. Nevertheless, *software tools* used for this activities has to be certified by the automotive company GM. In case of a proposed solution involving the purchase of equipment (*hardware* or *software*), it will be analyzed by the Technical Lead BSW Verification and the Propulsion Systems Supervisor.

## 2.5 Metadata

Validation engineers must be responsible of the *test procedure* data quality included in the execution process (*test steps standardization*), with the purpose of guaranteeing data integrity since information extraction until transmission of information (*CAN messages*) and received information comparison.

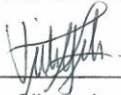
### 3 Software Requirements

The *software tool* should be able to:

1. Generate transmit messages from *test procedure* files automatically avoiding manual transcripts of *Tx* and *Rx* messages.
2. Transmit 29 bit header messages through the *CAN* bus (*extended format*) base on *GM ECU IDs*.
3. Handle single & multi-frame messages according to the *ISO 15765-2:2011* standard (*single frames, first frames and consecutive frames*).
4. Handle *flow control* messages to transmit and receive multi-frame messages according to the *ISO 15765-2:2011* standard.
5. Keep the service session active within the execution of test steps by sending a *tester present message* (active sessions by every 3 seconds *tester present message*).
6. Send messages to a single module and all modules by transmitting *physical* and *functional* requests.
7. Compare between received and expected data and display a message with the result of comparison (*PASS* or *FAIL*).
8. Be able to execute the 16 available *test procedures* within 48 *bench hours* per month reducing the current time spent with the actual manual process.
9. Run all *test steps* from every *test case* with a single click.
10. Display the data buffer with the transmitted and received messages.
11. Able to be used in the current OS used by the automotive company.
12. Use *NeoVi Fire* as the communication interface.
13. Display help data information in English.


Technical Lead BSW Verification

Victor M. Giles López



Client signature

Daniel Mejía Hernández



Software Analyst signature

## Glosario

---

<b>ACK</b>	Acknowledgment o reconocimiento, bit usado dentro de una trama de protocolos de comunicación para identificar la consistencia del mensaje recibido.
<b>ALDL</b>	Assembly Line Data Link, es un conector usado para diagnóstico y mantenimiento de los controladores que componen una red intra-vehicular.
<b>API</b>	Application Programming Interface, es un conjunto de reglas, comandos, protocolos y clases que permiten crear una interacción entre el <i>software</i> creado y objetos del exterior del sistema como documentos, sensores, actuadores, entre otros.
<b>BCM</b>	Body Control Module, es un controlador intra-vehicular que permite ejecutar funciones de confort del automóvil como ventanas eléctricas, luces, así como el despliegue de información en el cluster del automóvil.
<b>Bench</b>	Banco de prueba utilizado en ámbito automotriz que permite la simulación de sistemas embebidos para la verificación de funciones específicas.
<b>BS</b>	Block Size, es un conjunto de ocho bits dentro de una trama de tipo <i>flow control</i> usado para identificar el número de <i>consecutive frames</i> soportados por el receptor, previo a la transmisión del siguiente <i>flow control</i> (si es necesario).
<b>CAN</b>	Controller Area Network, es un protocolo de comunicación desarrollado por Bosch para uso automotriz.
<b>CAN Xtd</b>	Controller Area Network Extended, desarrollado para disponer de un número mayor de identificadores en relación al tamaño del <i>header</i> de una trama <i>CAN</i> .
<b>CRC</b>	Cyclic Redundancy Check, conjunto de 15 bits que contienen el <i>checksum</i> contra el que el receptor del mensaje comparará los resultados del mensaje recibido verificando la integridad del mismo.
<b>CSV</b>	Comma Separated Values, es un tipo de formato de archivo de Excel que permite la delimitación de columnas de información con base en comas.

---



---

<b>DFSS</b>	Design For Six Sigma, es una metodología desarrollo de nuevos productos, procesos o servicios mediante actividades como identificación del problema, definición de requerimientos, desarrollo del concepto, optimización y verificación.
<b>DLC</b>	Data Length Code, es el conjunto de cuatro a 12 bits (dependiendo el tipo de mensaje) que expresan el tamaño en bytes del total del contenido de la información a transmitir.
<b>DLL</b>	Dynamic Link Library, tipo de formato de archivo que contiene código y procedimientos para la ejecución de servicios o tareas del sistema. Para este caso, permite la interacción entre la interfaz de comunicación <i>NeoVi</i> y la computadora.
<b>DoCAN</b>	Diagnostic Communication over CAN, está relacionado al estándar <i>ISO 15765</i> que define las reglas del protocolo de transporte para la transmisión de mensajes <i>CAN</i> .
<b>DTC</b>	Diagnostic Trouble Code, es un código identificador almacenado por el controlador generado ante una falla detectada por sistema.
<b>ECM</b>	Engine Control Module, es un controlador con <i>software</i> embebido que permite efectuar funciones referentes al motor del vehículo como son la generación de la chispa de bujías, apertura y cierre del cuerpo del <i>throttle</i> , cantidad de combustible propagado por los inyectores, entre otros.
<b>ECU</b>	Electronic Control Unit, es un dispositivo electrónico que controla uno o más sistemas eléctricos en un vehículo.
<b>EEPROM</b>	Electrically Erasable Programmable Read Only Memory, tipo de almacenamiento programable de solo lectura útil para el almacenamiento de datos que requieren no ser borrados después de un ciclo de manejo.
<b>FlexRay</b>	Protocolo de comunicación intra-vehicular similar a <i>CAN</i> y <i>MOST</i> .
<b>FS</b>	Flow Status, cuatro bits que componen a una trama de tipo <i>flow control</i> que permiten identificar si el receptor puede recibir los datos que el emisor está por transmitir.
<b>Header</b>	Encabezado de una trama o mensaje con un identificador compuesto por el tipo de <i>request</i> ( <i>functional</i> o <i>physical</i> ), el nodo emisor y el nodo receptor.

---



---

<b>IDE</b>	Identifier Extension, es un bit que compone una trama <i>CAN</i> el cual indica si el <i>header</i> de esa trama es de formato extendido o no (11 o 29 bits).
<b>ISO</b>	International Standardization Organization, es una organización no gubernamental que desarrolla estándares que involucran a la industria, la tecnología, calidad y salud.
<b>JRE</b>	Java Runtime Enviroment, plataforma que permite la ejecución de aplicaciones <i>Java</i> con el mínimo de requerimientos como <i>Java Virtual Machine</i> (JVM), clases base y archivos de soporte.
<b>LIN</b>	Local Interconnect Network, es un estándar comunicación de bajo costo para la conexión embebida de dispositivos inteligentes dentro de una red intra-vehicular.
<b>LSB</b>	Less Significant Bit, es el bit menos significativo de un conjunto de bits. Leído de derecha a izquierda es el bit que está más a la derecha.
<b>MOST</b>	Media Oriented Systems Transport, es un protocolo de comunicación de datos intra-vehicular con soporte en aplicaciones de info-entretenimiento.
<b>MSB</b>	Most Sgnificant Bit, es el bit más significativo de un conjunto de bits. Leído de derecha a izquierda es el bit que está más a la izquierda.
<b>Multi-frames</b>	Conjunto de dos o más tramas necesarias para la transmisión completa de un mensaje, el cual cuenta con más de ocho bytes de información.
<b>Nibble</b>	Conjunto de cuatro bits.
<b>OBD</b>	On Board Diagnostics, es un sistema basado en computadora diseñado para la reducción y monitoreo de emisiones con base en los componentes del tren motriz del vehículo.
<b>OSI</b>	Open System Interconnection, modelo arquitectónico que permite representar la transmisión de información mediante capas.
<b>PCI</b>	Protocol Control Information, conjunto de ocho bits donde los primeros cuatro bits (del <i>MSB</i> al <i>LSB</i> ) representan el tipo de trama ( <i>single frame</i> , <i>first frame</i> , <i>etc.</i> ) y los siguientes cuatro bits representan el <i>DLC</i> .
<b>PDU</b>	Protocol Data Unit, término usado por el modelo OSI para referirse al conjunto de información construida por cada capa para su transmisión.
<b>POI</b>	Problem-Oriented Interfaces, librerías desarrolladas con la finalidad de interconectar el sistema en desarrollo con objetos exteriores a éste.



---



---

	Desarrolladas por Apache con la finalidad de extraer y generar texto relacionado a Office Open XML Standars ( <i>OOXML</i> ).
<b>RTR</b>	Remote Transmission Request, es un bit que especifica si el mensaje es transmitido como petición por otro nodo de la red.
<b>Rx</b>	Received message, es el mensaje recibido a partir de un mensaje transmitido, solicitando información.
<b>SAE</b>	Society of Automotive Engineers, definida como una asociación global con más de 128,000 ingenieros y técnicos expertos en temas aeroespaciales, automotrices e industrias de vehículos comerciales.
<b>SID</b>	Service Identifier, es un número identificador del tipo servicio <i>UDS</i> incorporado en un mensaje <i>CAN</i> con el objetivo de obtener información del controlador.
<b>Sheet</b>	Conocida comúnmente como <i>worksheet</i> u hoja la cual contiene celdas organizadas por renglones y columnas que pueden almacenar datos como números, textos o fórmulas.
<b>SN</b>	Sequence Number, número dentro de una trama de tipo <i>consecutive frame</i> que identifica el número de trama que es transmitida, de todo el conjunto de tramas.
<b>SRR</b>	Substitute Remote Request, bit dentro de una trama <i>CAN</i> que permite marcar un formato estándar o extendido de la trama.
<b>ST</b>	Separation Time, es el tiempo de separación que puede soportar el receptor para la transmisión de <i>consecutive frames</i> por parte del emisor.
<b>TCM</b>	Transmission Control Module, es el controlador que permite realizar los cambios automáticos de la transmisión, controlando el fluido del aceite, el movimiento de los engranes, entre otros componentes.
<b>Test case</b>	Conjunto de mensajes, configuraciones o actividades que permiten modelar un comportamiento evaluable del código embebido en el controlador de un vehículo automotor.
<b>Test procedure</b>	Conjunto de <i>test cases</i> que permiten la validación de un servicio <i>UDS</i> específico así como todas sus configuraciones ( <i>positive</i> y <i>negative responses</i> ).
<b>Test step</b>	Mensaje, configuración o acción que puede ser transmitida hacia el <i>ECU</i> para verificar su correcto funcionamiento. Contiene un <i>Tx</i> y un <i>Rx</i> esperado con su respectiva descripción.

## GLOSARIO

---

<b>Tx</b>	Broadcast message, es el mensaje a transmitir por parte del <i>tester</i> .
<b>TXT</b>	Tipo de formato que identifica a los archivos de textos simples o comúnmente llamados de texto plano.
<b>UDS</b>	Unified Diagnostic Service, es un protocolo de comunicación usado para el diagnóstico de vehículos automotores (especificado en el estándar ISO 14229).
<b>UML</b>	Unified Modeling Language, lenguaje con base en diseños, modelos y estructuras que permite visualizar y documentar los elementos que componen a los sistemas de software.
<b>Workbook</b>	Archivo de la suite de Microsoft Office Excel que contiene una o más <i>sheets</i> .
<b>XLSX</b>	Extensión de un archivo de la suite de Microsoft Office Excel que permite almacenar valores de tablas, así como su estilo y diseño.

## Referencias

---

Assawinjaipetch P., Heeg M., Gross D, Kowalewski S. (2014) *Unified Diagnostic Services Protocol Implementation in an Engine Control Unit*, King Mongkut's University of Technology North Bangkok magazine. (6). Bangkok, Thailand.

Arias Chaves, Michael. (2005). *La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software*. InterSedes: Revista de las Sedes Regionales, Sin mes, 1-13.

AXIOMATIC. (2006). *What is CAN?*. CAN bus Controls, Software. []. Canadá. Recuperado el 26 de Enero de 2016 de <http://www.axiomatic.com/canbus/>.

Booch G., Rumbaugh J., y Jacobson I. (2006). *El lenguaje unificado de modelado*. Madrid, España: Pearson Education.

Bosch. (1991). *CAN Specification Version 2.0*. Data Frame. Stuttgart, Alemania: Robert Bosch GmbH.

ChipsAway. (2015). *ECUs have been modified, or 'chipped', for many years, typically to improve performance or fuel efficiency*. Automakers: We Know Our Cars Better. Inglaterra. Recuperado el 12 de Enero de 2016 de <https://www.chipsaway.co.uk/our-world/blog/2015/4/24/automakers-we-know-our-cars-better/>.

Drew Technologies, Inc. (2003). *SAE J2534 API REFERENCE*. Estados Unidos. Recuperado el 10 de Septiembre de 2016 de [https://tunertools.com/prodimages/DrewTech/Manuals/PassThru\\_API-1.pdf](https://tunertools.com/prodimages/DrewTech/Manuals/PassThru_API-1.pdf)

HongKe. (2014). *NeoVI Fire*. China. Recuperado el 11 de Febrero de 2016 de <http://www.hongkeqiche.com/>

Intrepid Control Systems, Inc. (2016a). *Automatically Respond to a Message*. Vehicle Spy Help Documentation. Estados Unidos. Recuperado el 27 de Febrero de 2016 de <http://www.intrepidcs.com/support/ICSDocumentation/VehicleSpy/icsVehicleSpy.pdf>

Intrepid Control Systems, Inc. (2016b). *Vehicle Spy Overview*. Vehicle Spy Help Documentation. Estados Unidos. Recuperado el 25 de Febrero de 2016 de <http://www.intrepidcs.com/support/ICSDocumentation/VehicleSpy/icsVehicleSpy.pdf>

Intrepid Control Systems, Inc. (2016c). *Enter a Transmit Message*. Vehicle Spy Help Documentation. Estados Unidos. Recuperado el 27 de Febrero de 2016 de <http://www.intrepidcs.com/support/ICSDocumentation/VehicleSpy/icsVehicleSpy.pdf>

Intrepid Control Systems, Inc. (2016d). *OBD-II To Db-25 Adapter*. OBDII / to neoVI Pin-out. Estados Unidos. Recuperado el 9 de Febrero de 2016 de <http://store.intrepidcs.com/neoVI-OBD-1-p/neoVI-obd-1.htm>.

ISO. (2003a). *Data link layer and physical signalling*. En ISO 11898 Road vehicles – Controller area network (CAN) (Part 1, 45) Suiza: ISO.

ISO. (2011a). *Transport protocol and network layer services*. ISO 15765 Road vehicles – Diagnostics over Controller Area Network (DoCAN) (Part 2, 36) Suiza: ISO.

ISO. (2013a). *Introduction*. ISO 14229-1 Road vehicles - Unified diagnostic services (UDS) (Part 1, 393) Suiza: ISO.

ISO. (2013b). *Specification and requirements*. ISO 14229-1 Road vehicles - Unified diagnostic services (UDS) (Part 1, 393) Suiza: ISO.

ISO. (2013c). *Application protocol control information*. ISO 14229-1 Road vehicles - Unified diagnostic services (UDS) (Part 1, 393) Suiza: ISO.

ISO. (2013d). *Negative response codes*. ISO 14229-1 Road vehicles - Unified diagnostic services (UDS) (Part 1, 393) Suiza: ISO.

Kvaser. (2016). *Introduction to SAE J2534*. Estados Unidos. Recuperado el 10 de Septiembre de 2016 de <https://www.kvaser.com/about-can/can-standards/j2534/>.

Konrad Reif. (2015a). *Electronic control unit*. Automotive Mechatronics. Friedrichshafen, Germany: Springer Vieweg.

Konrad Reif. (2015b). *Basic principles of networking*. Automotive Mechatronics. Friedrichshafen, Germany: Springer Vieweg.

Konrad Reif. (2015c). *Automotive networking*. Automotive Mechatronics. Friedrichshafen, Germany: Springer Vieweg.

Larman, C. (2003). UML y patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Madrid, España: Pearson Education.

OBDTester. (2015). *OBD2 connector*. OBDTester: OBD-2 Diagnostic software and interfaces. Republica Checa. Recuperado el 9 de Febrero de 2016 de [http://www.obdtester.com/obd2\\_connector](http://www.obdtester.com/obd2_connector).

Pressman S. Roger (2010). Ingeniería del software - un enfoque práctico. Nueva York, Estados Unidos: Mc Graw Hill.

Reuss H. Christian. (1993). *Extended Frame Format – A New Option of CAN Protocol* (Product Concept & Application Laboratory Rep. No. HAI/AN 92 002). Hamburg, F. R. Alemania. Phillips Export B.V.

Schach Stephen R. (2006). Ingeniería de software clásica y orientada a objetos. Tennessee, Estados Unidos: Mc. Graw Hill.

Sommerville Ian. (2005). Ingeniería de software. Madrid, España: Pearson Educación.

Vector. (2016). CANoe.DiVa. Automated Testing of Diagnostic Software in ECUs. Alemania. Recuperado el 11 de Julio de 2016 de [http://vector.com/portal/medien/cmc/factsheets/CANoe.DiVa\\_FactSheet\\_EN.pdf](http://vector.com/portal/medien/cmc/factsheets/CANoe.DiVa_FactSheet_EN.pdf)

Weitzenfeld Alfredo. (2005). Ingeniería de software orientada a objetos con UML, Java e Internet. México, México: Thomson.

Whitten J., Bentley L., Barlow V. (2003). Análisis y diseño de sistemas de información. México, México: Mc Graw Hill.

## Índice de figuras

---

<b>Figura 1</b> - Black box de una Unidad de Control Electrónica.	- 15 -
<b>Figura 2</b> - <i>ECUs</i> que componen a un vehículo (ChipsAway, 2016).	- 16 -
<b>Figura 3</b> – Arquitectura típica del microcontrolador de un vehículo (Automotive Mechatronics, 2015a).	- 17 -
<b>Figura 4</b> – Interacción de las capas del modelo OSI.	- 21 -
<b>Figura 5</b> – Estados dominante y recesivo del protocolo <i>CAN</i> (AXIOMATIC, 2006).	- 22 -
<b>Figura 6</b> – Topología de red de un vehículo de lujo (Automotive Mechatronics, 2015c).	- 23 -
<b>Figura 7</b> – Protocolo <i>CAN</i> basado en mensajes.	- 24 -
<b>Figura 8</b> – Estructura de un mensaje <i>CAN</i> con <i>header</i> de 29 bits.	- 26 -
<b>Figura 9</b> – Representación del contenido de un mensaje <i>CAN 2.0B</i> .	- 28 -
<b>Figura 10</b> – Trama simple.	- 29 -
<b>Figura 11</b> – Primer trama.	- 29 -
<b>Figura 12</b> – Trama consecutiva.	- 30 -
<b>Figura 13</b> – Trama de control de flujo.	- 31 -
<b>Figura 14</b> – Transmisión simple.	- 33 -
<b>Figura 15</b> – Transmisión compuesta receptora.	- 34 -
<b>Figura 16</b> – Transmisión compuesta emisora.	- 35 -
<b>Figura 17</b> – Positive service response.	- 39 -
<b>Figura 18</b> – Estructura de los mensajes UDS.	- 40 -
<b>Figura 19</b> – Estructura de una respuesta negativa.	- 40 -
<b>Figura 20</b> – Pines del conector ALDL (OBDTester, 2015).	- 42 -
<b>Figura 21</b> – NeoVi Fire (HongKe, 2014).	- 42 -
<b>Figura 22</b> – Conexión del NeoVi Fire.	- 43 -
<b>Figura 23</b> – Estructura de un <i>test procedure</i> .	- 45 -
<b>Figura 24</b> – Horas necesarias por Test Procedure.	- 46 -
<b>Figura 25</b> – Comparación entre tiempo objetivo esperado y real de ejecución.	- 47 -
<b>Figura 26</b> – Construcción de tramas en <i>Vehicle Spy</i> (Intrepid Control Systems, Inc., 2016c).	- 48 -
<b>Figura 27</b> – Mensajes en tiempo real en <i>Vehicle Spy</i> (Intrepid Control Systems, Inc., 2016a).	- 49 -
<b>Figura 28</b> – Diagrama de bloques del proceso de validación actual.	- 50 -
<b>Figura 29</b> – Diagrama de contexto del proceso de validación actual.	- 51 -
<b>Figura 30</b> – Diagrama de flujo del proceso de validación actual.	- 52 -
<b>Figura 31</b> – Diagrama de actividades del proceso de validación actual.	- 53 -
<b>Figura 32</b> – Diagrama de caso de uso del proceso de validación actual.	- 54 -
<b>Figura 33</b> – Diagrama de bloques del proceso de validación cumpliendo los requerimientos establecidos.	- 66 -
<b>Figura 34</b> – Diagrama de actividades para el modelado de requerimientos.	- 67 -
<b>Figura 35</b> – Diagrama de secuencia general para el modelado de requerimientos.	- 68 -
<b>Figura 36</b> – Diagrama de caso de uso para el modelado de requerimientos.	- 69 -
<b>Figura 37</b> – Ejemplo de un <i>function block</i> .	- 74 -

<b>Figura 38</b> – Arquitectura del sistema <i>CANoe.DiVa</i> (Vector, 2016).	- 75 -
<b>Figura 39</b> – Configuración de <i>J2534</i> (Kvaser, 2016).	- 77 -
<b>Figura 40</b> – Archivo <i>DLL</i> para comunicación con interfaz <i>NeoVi</i> .	- 78 -
<b>Figura 41</b> – Ejemplo de resultado de ejecución de <i>In-house software</i> (interfaz gráfica).	- 79 -
<b>Figura 42</b> – Configuración de <i>headers</i> y conjunto de mensajes de <i>In-house software</i> .	- 80 -
<b>Figura 43</b> – Ejemplo de archivo importado o exportado con contenido similar al de la Figura 42.	- 80 -
<b>Figura 44</b> – Diagrama de contexto de híbrido 1.	- 85 -
<b>Figura 45</b> – Diagrama de contexto de híbrido 2.	- 85 -
<b>Figura 46</b> – Alcance de la nueva herramienta de <i>software</i> .	- 88 -
<b>Figura 47</b> – Diagrama de black box del nuevo <i>software</i> .	- 88 -
<b>Figura 48</b> – Diagrama de contexto nivel 1 del nuevo <i>software</i> .	- 89 -
<b>Figura 49</b> – Diagrama de contexto nivel 2 del nuevo <i>software</i> .	- 90 -
<b>Figura 50</b> – Diagrama de clases del nuevo <i>software</i> .	- 91 -
<b>Figura 51</b> – Diagrama de caso de uso del nuevo <i>software</i> .	- 92 -
<b>Figura 52</b> – Diagrama de estados del nuevo <i>software</i> .	- 97 -
<b>Figura 53</b> – Diagrama de secuencia del nuevo <i>software</i> .	- 98 -
<b>Figura 54</b> – Diagrama del modelo de capas.	- 100 -
<b>Figura 55</b> – Diseño de casos de prueba (Benitez A., 2012).	- 114 -
<b>Figura 56</b> – Diagrama de flujo del método leerArchivo.	- 115 -
<b>Figura 57</b> – Diagrama de flujo del método EscrituraArchivoInHouse.	- 117 -
<b>Figura 58</b> – Diagrama de flujo del método identificarInformacion.	- 117 -
<b>Figura 59</b> – Diagrama de flujo del método dividirBytesDeMensaje.	- 118 -
<b>Figura 60</b> – Diagrama de flujo del método identificarDLC.	- 119 -
<b>Figura 61</b> – Diagrama de flujo del método construirMensajeTxInHouse.	- 119 -
<b>Figura 62</b> – Diagrama de flujo del método construirMensajeRxInHouse.	- 120 -
<b>Figura 63</b> – Diagrama de flujo del método identificarTipoDeRequest.	- 121 -
<b>Figura 64</b> – Diagrama de flujo del método construirInHouseStep.	- 122 -
<b>Figura 65</b> – Diagrama de flujo del método analizarInformacion.	- 123 -
<b>Figura 66</b> – Diagrama de flujo del método administrarActividades.	- 124 -
<b>Figura 67</b> – Estrategia de prueba del software (Pressman, 2010).	- 128 -
<b>Figura 68</b> – Comparación del tiempo destinado al proceso de transmisión de mensajes.	- 131 -
<b>Figura 69</b> – Comparación de reducción de tiempos (objetivo y proceso anterior).	- 132 -
<b>Figura 70</b> – Actividades para el nuevo proceso de transmisión de mensajes.	- 132 -

## Índice de tablas

---

<b>Tabla 1</b> – Representación de estándares aplicados a las capas del modelo OSI (ISO, 2013a).	- 19 -
<b>Tabla 2</b> - Versiones de CAN (Reuss H. Christian, 1993).	- 24 -
<b>Tabla 3</b> – Clasificación de servicios UDS por tipo de funcionalidad.	- 38 -
<b>Tabla 4</b> – Rangos del identificador de servicios (ISO, 2013c).	- 39 -
<b>Tabla 5</b> – Respuestas negativas más comunes de los servicios de <i>UDS</i> .	- 41 -
<b>Tabla 6</b> – Descripción del caso de uso administrar mensajes.	- 55 -
<b>Tabla 7</b> – Descripción del caso de uso transmitir mensajes.	- 56 -
<b>Tabla 8</b> – Descripción del caso de uso generar archivo de datos.	- 58 -
<b>Tabla 9</b> – Descripción del caso de uso visualizar mensajes.	- 59 -
<b>Tabla 10</b> – Descripción del caso de uso guardar mensajes generados.	- 60 -
<b>Tabla 11</b> – Descripción del caso de uso leer mensajes previamente creados.	- 61 -
<b>Tabla 12</b> – Tabla comparativa entre <i>Vehicle Spy</i> y los requerimientos del cliente.	- 65 -
<b>Tabla 13</b> – Descripción del caso de uso extraer <i>test steps</i> .	- 70 -
<b>Tabla 14</b> – Descripción del caso de uso transmitir <i>test steps</i> .	- 71 -
<b>Tabla 15</b> – Descripción del caso de uso Desplegar mensajes del bus <i>CAN</i> .	- 72 -
<b>Tabla 16</b> – Descripción del caso de uso comparar resultados obtenidos.	- 73 -
<b>Tabla 17</b> – Costo de implementación de <i>Diva</i> .	- 76 -
<b>Tabla 18</b> – Benchmark de software que implementa la misma funcionalidad.	- 83 -
<b>Tabla 19</b> – Comparación entre propuestas híbridas.	- 86 -
<b>Tabla 20</b> – Vistas y diagramas de UML.	- 90 -
<b>Tabla 21</b> – Descripción del caso de uso <i>leer archivo</i> .	- 93 -
<b>Tabla 22</b> – Descripción del caso de uso <i>identificar información</i> .	- 94 -
<b>Tabla 23</b> – Descripción del caso de uso <i>generar archivo</i> .	- 95 -
<b>Tabla 24</b> – Tabla pseudocódigo de la clase VentanaEjecucion.	- 101 -
<b>Tabla 25</b> - Tabla pseudocódigo de la clase TestStep.	- 103 -
<b>Tabla 26</b> - Tabla pseudocódigo de la clase InHouseStep.	- 103 -
<b>Tabla 27</b> - Tabla pseudocódigo de la clase ManejoDeDatos.	- 104 -
<b>Tabla 28</b> - Tabla pseudocódigo de la clase Administración.	- 109 -
<b>Tabla 29</b> - Tabla pseudocódigo de la clase Inicio.	- 111 -
<b>Tabla 30</b> - Tabla pseudocódigo de la clase LecturaArchivoXLSX.	- 112 -
<b>Tabla 31</b> – Tabla pseudocódigo de la clase EscrituraArchivoInHouse.	- 113 -
<b>Tabla 32</b> – Rutas del método leerArchivo.	- 116 -
<b>Tabla 33</b> - Rutas del método escribirArchivo.	- 117 -
<b>Tabla 34</b> - Rutas del método identificarInformacion.	- 118 -
<b>Tabla 35</b> - Rutas del método dividirBytesDeMensaje.	- 118 -
<b>Tabla 36</b> - Rutas del método identificarDLC.	- 119 -
<b>Tabla 37</b> - Rutas del método construirMensajeTxInHouse.	- 120 -
<b>Tabla 38</b> - Rutas del método construirMensajeRxInHouse.	- 121 -
<b>Tabla 39</b> – Rutas del método identificarTipoDeRequest.	- 121 -
<b>Tabla 40</b> – Rutas del método construirInHouseStep.	- 122 -
<b>Tabla 41</b> – Rutas del método analizarInformacion.	- 123 -



**Tabla 42** – Rutas del método administrarActividades. - 124 -

**Tabla 43** – Resultados de pruebas de verificación con el usuario, se recomienda tener presente los datos desplegados en la Tabla 12 donde se especifican los requerimientos. - 130 -

**Tabla 44** – Control de versiones dependiendo del tipo de cambio. - 133 -