Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

ESTUDIO PARA MEJORAR LA IDENTIFICACIÓN DE PROBLEMAS DE SOFTWARE

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **MIGUEL DE JESÚS ZAMORA CORTÉS**

Director **DR. LUIS RIZO DOMÍNGUEZ**

Tlaquepaque, Jalisco. 31 de mayo de 2021.

En primer lugar, quisiera agradecer a mi asesor el Doctor Luis Rizo Domínguez, por su constante orientación, valiosos comentarios y sugerencias que enriquecieron el desarrollo de este trabajo.

A mis padres, Miguel y Pilar, por enseñarme que la perseverancia y la disciplina son los caminos para lograr el éxito. Les reitero mi gratitud por el apoyo incondicional que he recibido de ellos desde el día de mi nacimiento. A mis hermanas, Alondra y Andrea que comprendieron mis limitantes de tiempo y siempre me soportaron.

Tampoco sería justo olvidar a mis amigos, quienes aceptaron mis nuevas responsabilidades y a pesar de mis limitantes de tiempo siempre buscaron el espacio para verme y hacerme mejorar como persona, Nadia Ramírez, Luis Machuca, Agustín Díaz, Carlos Sandoval, Miguel Luna y Samantha Vázquez.

Resumen

El estudio de este documento se centra en la identificación y análisis de un problema de software en el sector automotriz, lo primero que este documento aborda es la descripción de una falla, en esta sección se describe el problema con el que vamos a trabajar, posteriormente se define el problema y su posible causa utilizando tres herramientas muy comunes en manufactura, que son: la definición del problema, el diagrama Ishikawa y diagrama de procesos.

La definición del problema se aborda mediante la técnica de 5W + 2H, mientras que los diagramas de proceso e Ishikawa se desarrollaran de manera normal. Al trabajar con estas herramientas obtendremos numerosas ideas de donde podría estar el problema; en este trabajo se documentan todas y se explica por qué no todas son válidas.

Después de analizar las ideas, se procede a trabajar con las predominantes y se convierten en hipótesis. En este trabajo se abordan tres hipótesis, dichas serán experimentadas y validadas. Se obtendrá una teoría con la cual se implementará una solución al problema,

Finalmente, en la validación de pruebas, se hablará sobre las pruebas unitarias, de análisis estático y funcionales que se realizaron, con la finalidad de garantizar que el problema se solucionó y la implementación es la correcta.

Contenido

Introducción

El desarrollo de sistemas embebidos para resolver necesidades cotidianas resulta en la transformación de la época en la que vivimos. Hoy en día casi todo con lo que interactuamos es un sistema embebido. En el ramo automotriz, los sistemas embebidos son vitales para dar funcionalidades que necesitan los vehículos, como pueden ser; el estéreo, las luces, los frenos ABS, controles de estabilidad, etc. Prácticamente un vehículo tiene alrededor de 10 a 40 computadoras dependiendo de la gama, sin embargo, el desarrollo de dichas computadoras cada día se vuelve más complejo y exigente, un vehículo de hace 10 años no tiene ni la mitad de las funcionalidades de un vehículo actual. Por ende, las empresas se ven en la necesidad de tener personal capacitado, que sea capaz de resolver las necesidades que se presenten en el desarrollo de estos sistemas embebidos.

Este trabajo tiene como propósito describir cómo un problema de un sistema embebido se identifica, se analiza, se estudia, se soluciona y cómo se implementan estrategias para que no vuelva a suceder. Cabe mencionar que los problemas en sistemas embebidos son normales en fase de desarrollo, pero cuando los problemas se presentan en fase de producción y venta esto puede significar grandes pérdidas monetarias y peor aún en vidas humanas.

1. Descripción de la falla

El problema de software se detectó después de producción y validación. El problema fue encontrado en pruebas vehiculares del cliente, dicho problema consistía en que el sistema embebido, computadora central, se reiniciaba al poner el vehículo en encendido por unos 5 minutos, o cuando se estresaba el encendido por 1 minuto. Este problema se presentaba en 3 de 10 camionetas con las que se probaban las nuevas piezas. Dicho problema era grave ya que, si llegaban 100 unidades, 30 presentarían fallas. Cabe mencionar que en algunas unidades el problema se presentaba cuando en la camioneta se realizaban pruebas de manejo, provocando que los vehículos perdieran el control y se expusiera la integridad de los operadores.

2. Metodología del desarrollo de la hipótesis del problema

El diagnóstico se abordó por un grupo multidisciplinario, conformado por el grupo de pruebas y desarrollo de software. Como parte del proceso, la empresa utilizó la metodología de definir, analizar, mejorar y controlar. Para poder comprender e identificar la causa raíz del problema se utilizó las siguientes herramientas.

- **Definición del problema**

Se empleó la técnica de 5W + 2H como herramienta para el análisis del problema, ésta tiene como objetivo definir cuál es el problema e identificar sus causas y proponer posibles soluciones, sin embargo, no se enfoca en encontrar una solución [Progressa-Lean-15].

- **Diagrama Ishikawa**

Es una herramienta gráfica que representa la relación entre un efecto y las posibles causas que lo ocasionan.

- **Diagrama de procesos**

Es la representación gráfica de las secuencias por las que paso el producto y cuyo fin es identificar los puntos más críticos que pudieron provocar fallas e identificar los controles que se tienen para detectar amenazas.

2.1. Definición del problema

Para poder definir el problema se utilizó la técnica de las 5W + 2H de la que se habló anteriormente, esta herramienta es utilizada para el análisis de problemas y determinar la raíz del problema. La Fig. 2-1 muestra cómo se contestó.

5 W + 2H	Respuestas
¿Qué?	El módulo deja de comunicar en CAN después de 5 minutos y entra en modo bootloader, provocando que todas las funcionalidades se desactiven y la aplicación se corrompa.
Donde	El problema se detectó en la planta de pruebas del cliente
¿Quién?	El cliente lo detectó
¿Cuándo?	Del 12/18/18 hasta 01/18/19
¿Por qué?	Bajo investigación, se cree que este problema se ingresó con los últimos cambios
¿Cómo?	Se detectó el problema al realizar pruebas en el vehículo, las pruebas fueron realizadas con normalidad
¿Cuántos?	El problema se detectó en 3 de 10 vehículos

Fig. 2-1 Tabla de las 7 preguntas con sus respuestas para el problema de comunicaciones.

A partir del análisis con esta técnica encontramos los siguientes hechos:

- El problema es crítico porque estamos hablando que 30% de las unidades van a presentar este problema si no se corrige.
- El problema se reprodujo en vehículos que se encontraban en preventa y entrega para clientes.
- Otras líneas de vehículos no presentaban la falla, sólo camionetas de año 2019 hasta 2020.

2.2. Diagrama Ishikawa

El diagrama Ishikawa ayuda a comprender la causa-efecto, ésta es una herramienta que se utiliza para estructurar y clasificar las causas potenciales [Rodriguez-Johanna-19]. En el diagrama clasificamos que el problema puede estar relacionado con 8 posibles causas las cuales son:

- Hardware
- Cambios en las versiones de SW[i]
- En el proceso de programación
- En la verificación de llaves
- Cambios en el vehículo a nivel arquitectura
- Microcontrolador incorrecto
- Bootloader
- Memoria sobrescrita

Se desea identificar la causa por las cual el BCM[ii] modelo 2019 variante DT[iii] pierde comunicaciones con otros ECUs[iv], por medio de una lluvia de ideas el equipo de software y pruebas identificó las posibles causas que se muestran en la Fig. 2-2.

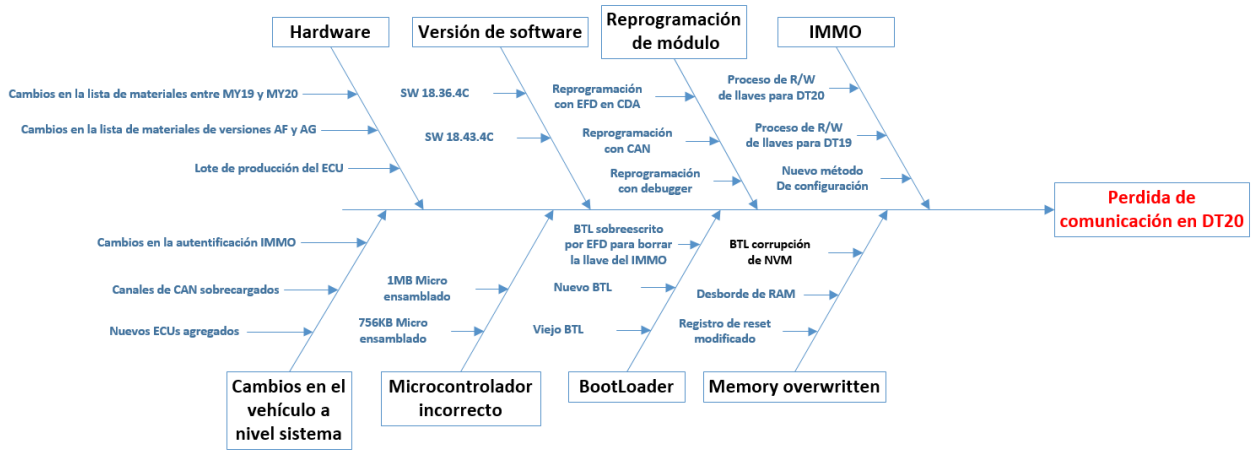

Fig. 2-2 Diagrama Ishikawa para la perdida de comunicación en DT20

Del diagrama podemos conseguir varios temas de investigación. Dichos temas se investigaron y se menciona el resultado preliminar de cada uno, los análisis de mayor peso se convertirán en hipótesis y se profundizará en cómo se realizó la investigación de estos.

- Cambios en el BOM[v] de MY[vi]19 y MY20
    - Las pruebas en tarjetas se hicieron en la versión VP-B con hardware del MY19, el cliente aún no prueba con hardware de MY20
- Cambios en el BOM para versiones AF y AG
    - Los desarrolladores no creen que sea problema por cambio de hardware
- Fecha de producción
    - Se cree que el lote de producción está comprometido, sin embargo, el problema fue reproducido con otro lote.
- Versión de software SW 18.36.4C y 18.43.4C
    - El problema fue reproducido en las dos versiones
- Programación at través de herramienta de diagnóstico (EFD[vii] por CDA[viii])
    - El problema no se pudo reproducir con este método
- Programación a través de debugger
    - El problema no se pudo reproducir por este método
- Escritura de llaves en la NVM[ix] del módulo MY20
    - Se cree que el cliente puede estar haciendo mal la escritura de las llaves, sin embargo, se validó y se continuo con el problema
- Escritura de llaves en la NVM del código de MY19 en MY20 ECU
    - El procedimiento de prueba se llevó correctamente como se hizo en MY20 y la falla sigue presente
- Cambios relacionados con la ignición y autentificación
    - El cliente y el equipo de desarrollo confirman que no hay cambios en el código
- Se cree que los canales de CAN[x] tienen sobre carga
    - El cliente confirma que no se agregaron mensajes y el equipo de software confirma que la carga máxima es igual que en el modelo anterior.
- Nuevos ECU agregados en MY20
    - No se agregaron nuevos ECUs en la arquitectura del vehículo

- Cambio de microcontrolador de 1 MB a 756 KB
    - Se reportan fallas en ambos microcontroladores
- Problemas relacionados con el Bootloader
    - Se descartó la idea de que el BTL[xi] borrara las llaves de autentificación
    - Se descartó la idea de que la nueva versión de BTL tenga el problema, al poder reproducir el problema en ambas versiones
    - Problema en que el BTL corrompa algunas secciones de memoria, se descartó al realizar un chequeo a los estados de la memoria.
- Desbordamiento de la RAM
    - Se realizó un análisis y no se comprobó desbordamientos
- Revisar la corrupción de registros
    - Se pudo conseguir los registros del evento del reset, se puso un breakpoint y los registros fueron capturados indicando que el reset se había originado mediante una petición del watchdog. Esto se reprodujo en un módulo que presentó la falla en el vehículo.

2.3. Diagrama de proceso

El diagrama de proceso tiene como objetivo identificar, las etapas por las que pasó el producto e identificar las situaciones que pudieron agregar un riesgo [Martins-Rosemary-18]. Las etapas más importantes de la fabricación del producto son detalladas en este diagrama, además se listan los riesgos que existen en cada una de ellas. La Fig. 2-3 ilustra el proceso por el que pasaron las unidades con falla.

Con la información recabada de este diagrama podemos identificar los riesgos de cada etapa. Al analizar estos riesgos se obtienen posibles causas del problema, las cuales se describen a continuación.

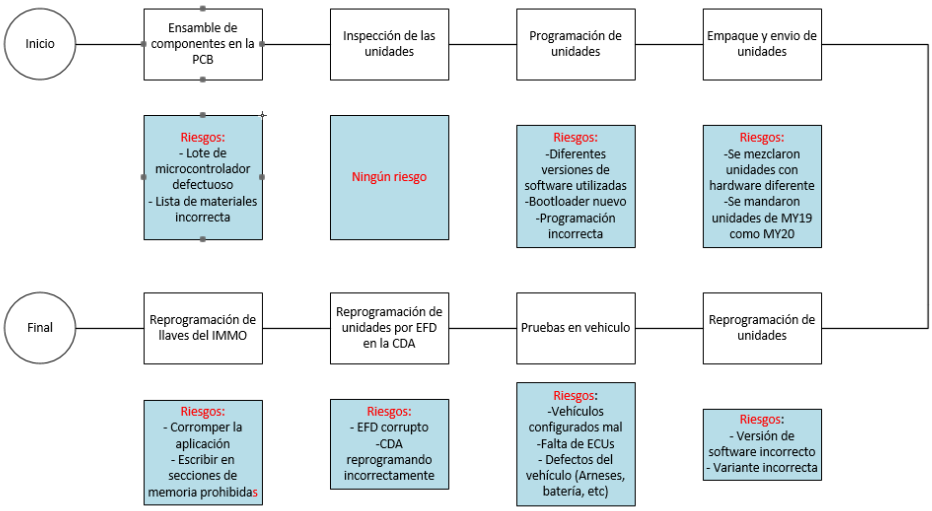

Fig. 2-3 Diagrama de proceso de las unidades con pérdida de comunicaciones

- El proceso de montaje de hardware: ya sea materiales incorrectos con la actualización de la lista de materiales, o que el lote del microcontrolador este defectuoso, o el cambio de microcontrolador
- El proceso de programación: la unidad pasa por cuatro programaciones, en las cuales se le pudo cargar imágenes erróneas, sobrescribir el bootloader, programar las tarjetas con software para otro microcontrolador, corromper la memoria al tratar de borrar las llaves del IMMO[xii]
- El proceso de pruebas: que los vehículos de pruebas tengan componentes dañados, que los arneses no estén en buenas condiciones, que el vehículo tenga nuevos ECUs y haya algún problema de compatibilidad, que el estado de la batería no sea el óptimo.

3. Análisis y pruebas de hipótesis

Esta sección abordará y evaluará todas las posibles causas del problema, estas causas fueron previamente recabadas a través del diagrama Ishikawa, diagrama de procesos y de la tabla 5W + 2H. Dado que las ideas sobre el problema raíz son demasiadas, se ha decidido delimitar las hipótesis mediante disciplinas, por ejemplo; software, pruebas y hardware. La evaluación de dichas hipótesis se realizará en las instalaciones de la empresa, ya que ésta tiene todos los elementos necesarios para realizar la investigación. Cabe mencionar que la evaluación de las hipótesis inicia

en paralelo y se suspenderán hasta que el problema sea corregido y aceptado por todas las áreas involucradas.

3.1. Hipótesis 1 Error en la aplicación de software.

Software fue la primera disciplina en ser notificada que se presentaba un problema de comunicaciones en el BCM, al analizar la situación se creyó que podría ser debido a unos cambios realizados en el software recientemente, por ende, lo primero que se intentó hacer fue tratar de reproducir el problema en uno de los bancos de pruebas, de principio no se tuvo éxito.

El banco de pruebas es un simulador de todas las cargas electrónicas que tiene el vehículo algunos ejemplos serían: los relevadores de seguros, las luces interiores, las luces exteriores, el claxon, etc. Debido a que los bancos de pruebas no tienen todos los ECUs que se encuentran en las redes de comunicación, para simular eso se utiliza la herramienta de CANoe[xiii] en donde mediante las bases de datos se simulan los mensajes de los ECUs faltantes. El banco se constituye por una fuente de poder, cargas de luces, tarjeta del BCM, computadora con CANoe y hardware de vector. En la Fig. 3-1 se aprecia un simulador de cargas parte elemental de un banco de pruebas.

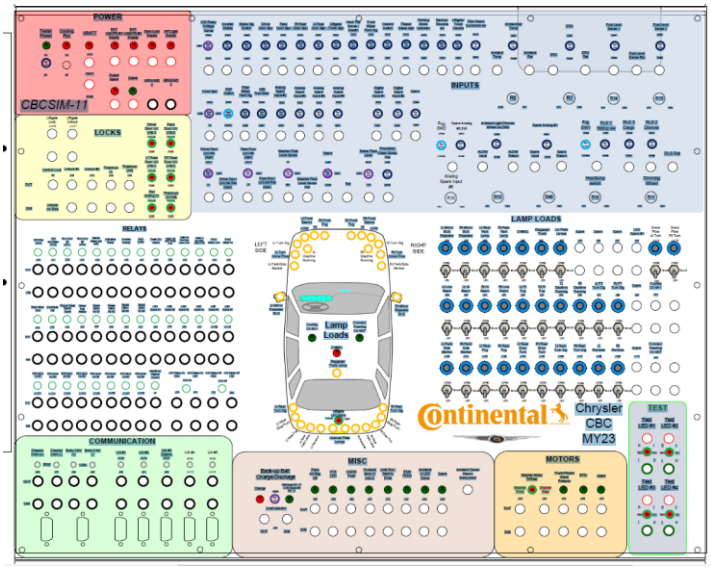

Fig. 3-1 Simulador de cargas genérico para simular un vehículo

Durante cada entrega de software se realizan pruebas de integración donde se evalúan que los últimos cambios funcionen correctamente y no se haya comprometido el buen funcionamiento del BCM. Por ende, todas las versiones de software no tenían el problema de comunicación cuando se realizaron pruebas internas. La Fig. 3-2 muestra la plantilla donde se documenta la prueba, la persona que ejecutó la prueba y el estado de la prueba.

			Integration Test Results for "name_of_s19_file.s19"		Active Diagnostic Variant:	NN	Active Diagnostic Version	NN	Model in test:	BFNNNNNN
Test Category #	**Test Category**	**Test #**	**Test Name (TID Name)**	**Tested by**	PASS or **FAIL**	**Source of Problem (SW/ HW/ Script/ Spec)**	**Issue Resolution**	**Details of Issue(s) with date found**	**Spec Refs (VF's, SRS, Others)**	**Comments**
3.1	Model Application	3.1.1	Hardware IO							
		3.1.2	CAN IO							
		3.1.3	Hardware Ignition							
3.2	Network Messaging	3.2.1	Cyclic Message							
		3.2.2	CAN Sleep							
		3.2.3	CAN-I Wakeup							
		3.2.4	CAN-C Wakeup							
		3.2.5	LIN-1 Wake LIN-1							
		3.2.6	LIN-1 Wake Up BUS							
		3.2.7	Hardware IO CAN Wakeup							
3.3	Microcontroller Sleep	3.3.1	Sleep							
		3.3.2	Hardware ECU Wakeup							
		3.3.3	CAN-I ECU Wakeup							
		3.3.4	CAN-C ECU Wakeup							
3.4	Diagnostics	3.4.1	SID 22							
	Diagnostics	3.4.2	SID 2F							
	Diagnostics	3.4.3	SID 2E							
	Diagnostics	3.4.4	Exit Diagnostics							
	Diagnostics	3.4.5	IM_VER							
	Diagnostics	3.4.6	Active Diagnostic DDT							
3.5	DTC	3.5.1	DTC Active							
		3.5.2	DTC Stored							
		3.5.3	DTC Ignition Cycles							
		3.5.4	DTC Removed							
		3.5.5	DTC chronostack is restored after a POR							
3.6	Back-Up Strategy	3.6.1	CBC Not Programmed							
		3.6.2	CBC Request Configuration Data							
		3.6.3	CBC Programmed by Backup Configuration messages							
		3.6.4	CBC Configuration retained after POR							
3.7	Lamp Test	3.7.1	Incandescent Lamps test							
		3.7.2	LED lamps test							
		3.7.3	Dedicated Diagnostic Line test							
3.8	Immobilizer Interface	3.8.1	Immobilizer							
		3.8.2	Configurations for IMMO							
		3.8.3	Previous RUN the IMMO							
		3.8.4	Running the IMMO ACC State							
		3.8.5	Running the IMMO RUN State							
		3.8.6	Sustain Ignition Run state after POR							
		3.8.7	Ignition OFF							
3.9	Gateway test	3.9.1	Gateway CAN to CAN							
		3.9.1	Gateway CAN to LIN							

Fig. 3-2 Plantilla de las pruebas de integración

Dado que de manera inicial no se pudo reproducir el problema de comunicaciones, se contactó con el equipo de pruebas que logró detectar este problema. Ellos proporcionaron las configuraciones y los pasos necesarios para reproducir la falla, desafortunadamente después de largos intentos la falla no se pudo reproducir. Los archivos de configuración se cargan a través de una herramienta de diagnóstico (Fig. 3-3) y los pasos son condiciones que se deben cumplir, por ejemplo, poner el módulo con ignición encendida.

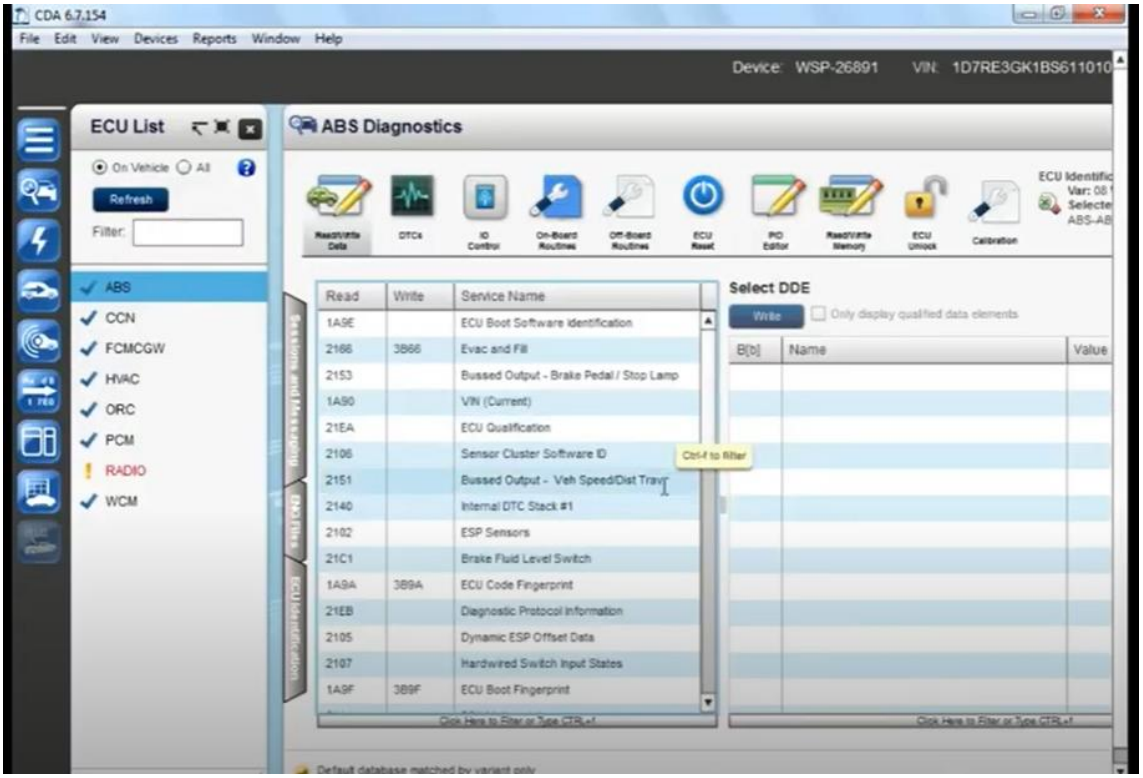

Fig. 3-3 Herramienta de diagnóstico para configurar un BCM

Los cambios de software son registrados en una herramienta que se conoce como control de versiones. Ésta almacena cada cambio que se ha realizado a las líneas de código, lo cual permite poder rastrear dónde se pudo integrar el problema y regresar a una versión de software estable. El problema fue reportado por el cliente en los entregables de la semana 43, 36, 21. Por consecuente se procedió a realizar una evaluación de las diferencias agregadas en los códigos. La Fig. 3-4 muestra cómo funciona un control de versiones.

Las versiones son trabajadas en computadoras locales de los desarrolladores y estos cambios se suben a los servidores de la empresa, parte del proceso para subir los cambios es verificar que estos funcionen correctamente, lo cual se logra a través de pruebas de unidad, pruebas de integración y una revisión de las diferencias con integrantes de proyecto en busca de posibles errores. El contar con esta herramienta permite poder regresar a versiones anteriores y buscar errores que antes no existían. Al comienzo de este problema, el cliente reportó que en el software de la semana 36, el problema no se reproducía, por lo que se procedió a revisar las diferencias, pero ninguna causa fue encontrada.

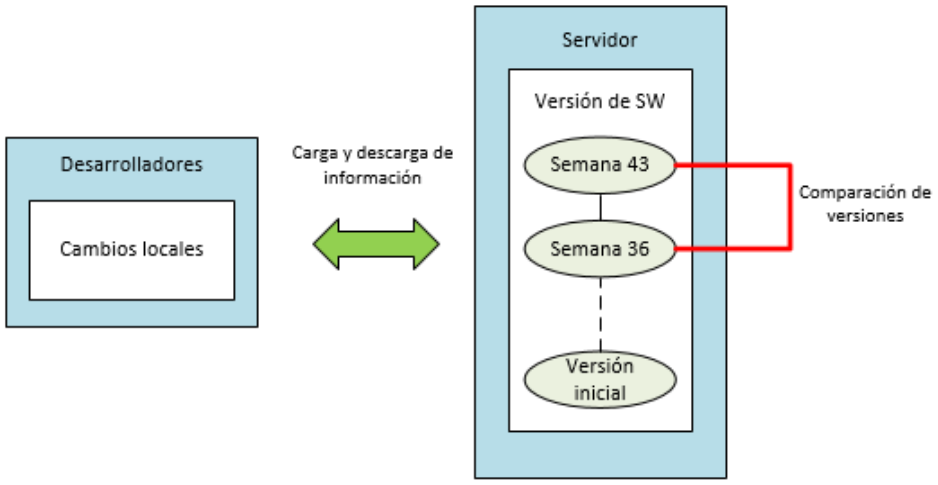

Fig. 3-4 Ilustración del funcionamiento de un control de versiones

Durante las primeras semanas, la empresa no pudo reproducir el problema en ninguno de sus laboratorios de software. Por ende, se contactó al cliente para pedir una unidad que tuviera el problema, cuando esta unidad se probó en el laboratorio de otra localidad la falla fue reproducible, se procedió a investigar que tenía de diferente esa unidad, dado que ambas unidades tenían el mismo software cargado deberían funcionar igual. Se decidió a hacer un memory-dump para buscar secciones de memorias corruptas.

La traducción de un memory dump es volcado de memoria, este proceso consiste en leer todos los sectores de memoria del microcontrolador y guardarlos en una unidad de almacenamiento [TechTarget-16], los sectores de memoria pueden ser todos a los que el debugger tenga acceso, en nuestro caso se enfocó en leer y guardar las secciones de RAM[xiv] y flash. Este método de diagnóstico sirve para poder reconstruir la aplicación que se tiene descargada en el módulo con la falla y poder comparar ese archivo con el archivo s19 que se liberó por parte de software, con la intención de buscar si la aplicación se corrompió.

La Fig. 3-5 muestra como se ve la memoria a través de un debugger, también se aprecia la dirección y el valor que se tiene almacenado en ella. Para hacer el memory dump lo único que se tiene que saber es: la dirección de memoria que se quiere guardar y la longitud de la sección que

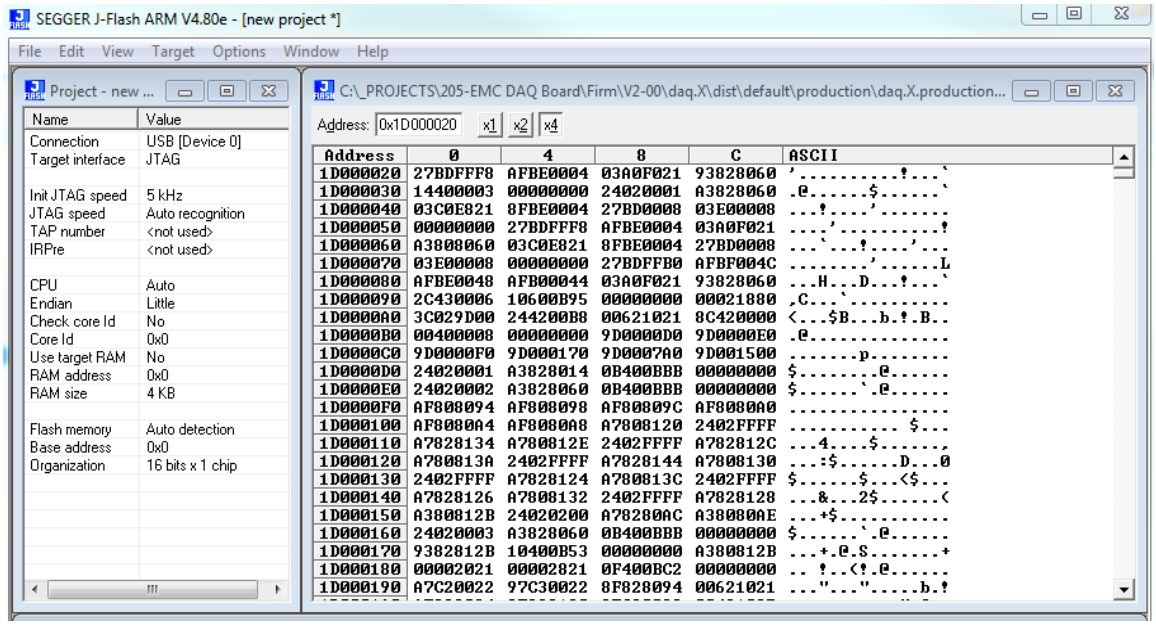

Fig. 3-5 Memoria de un microcontrolador monitoreada por JTAG

se desea almacenar, para este caso se revisaron 3 secciones, la sección de la aplicación, la sección del bootloader y la sección de RAM.

Al realizar las comparaciones con los archivos generados no existió diferencias en aplicación, ni en el bootloader pero sí en la RAM; sin embargo es normal que la RAM sea diferente entre los módulos ya que es aleatoria. La comparación se hizo mediante hexview y beyond compare.

La unidad con el problema reportado fue enviada a la localidad más cercana de la empresa, dicha localidad realizó parte para el memory dump y proporcionaron el s19 generado con la finalidad de tratar de replicar una unidad con la falla en la otra localidad. Además de realizar las reprogramaciones que habían pasado las unidades con la falla, dichas programaciones se hicieron por tres principales razones, la falta de unidades, añadir los cambios más recientes y poder hacer pruebas en vehículo más rápido.

Las programaciones por las que se sometieron las unidades con las fallas fueron las siguientes: La primera de ellas fue al salir de la empresa, en que las unidades fueron reprogramadas con el software del modelo 2020, ya que no existía diferencias en el hardware que impidiera correr el programa. La segunda reprogramación fue en la planta de ensamble con otro software de 2020

más nuevo, de la semana 43 mientras que el anterior era de la semana 36. Posteriormente se procedió a borrar las llaves que se encuentran en una sección de memoria del bootloader, dicha sección es inaccesible por aplicación y diagnóstico, por ende, se tuvo que realizar mediante la reprogramación.

Las llaves son importantes porque es la forma en que se autentifican el módulo del RF, con la computadora central y a su vez con la computadora del motor, si estas llaves no se guardaran y no fueran únicas, significaría que cualquier persona no autorizada podría encender el vehículo. El detalle de la situación es que el cliente contaba con un programa que abría una puerta de programación, dicha puerta consistía en borrar las llaves de la sección de bootloader y poder grabar una nueva llave, esto le permitía al cliente cambiar de computadoras en diferentes vehículos. Esto al ser un programa no verificado se creía que podía haber corrompido la memoria o incluso el bootloader, ya que la aplicación con la que se realizaba el borrado de llaves era para otro vehículo. Sin embargo, al terminar con todas las reprogramaciones no fue posible reproducir la falla. La siguiente imagen ejemplifica como esta seccionada la memoria de este ECU.

Los colores de la Fig. 3-6 limitan accesos, color verde puede ser escrita y leída por el bootloader, sin embargo, estas secciones verdes no pueden acceder a amarillo y mucho menos a

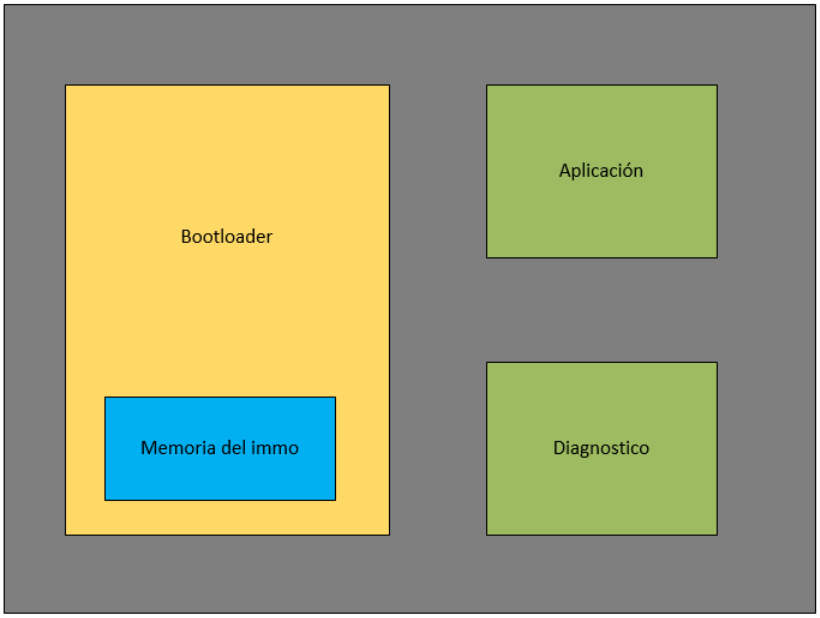

Fig. 3-6 Secciones de memoria del microcontrolador.

azul, la sección de color azul está protegida para ser escrita una vez y ya no podrá ser borrada por ninguna otra sección de memoria y solamente podrá borrarse mediante una herramienta externa.

3.2. Hipótesis 2 Error en el banco de pruebas / vehículos.

El problema fue encontrado y reproducido por el equipo de pruebas del cliente. La fase en la que se encontró fue en las pruebas funcionales del vehículo a meses de lanzar la línea a ventas. Este problema era muy impactante, ya que unidades con la misma configuración ya habían pasado las pruebas.

Para el cliente era muy fácil reproducir la falla, sin embargo, para nuestro equipo de pruebas era muy difícil. El cliente compartió las configuraciones, la versión de software y las condiciones de prueba con las que el módulo fallaba. Con estas condiciones iniciales el equipo fue incapaz de reproducir la falla, estuvieron intentado con estos escenarios cerca de una semana, sin éxito alguno. Afortunadamente existió buena comunicación y después de varias pruebas lograron replicar el error.

Mediante juntas y la buena comunicación lograron identificar ambos equipos algunas discrepancias en el entorno de pruebas, las cuales fueron las siguientes;

- El problema no era reproducible en todos los vehículos.
- Si un vehículo tenía el error, este no funcionaba, aunque se cambiara de módulo
- Si un módulo había estado en un vehículo con el error, este iba a presentar problemas en cualquier otro vehículo.

Esta información fue vital para encontrar uno de los puntos más importantes que ayudaron a resolver el problema: la identificación de que no todas las unidades presentaban la falla y que el problema se iniciaba en algunos vehículos identificados. Es decir, una unidad virgen podría ser programada e instalada en un vehículo que antes no había presentado falla y funcionaría

correctamente, pero si era instalada en un vehículo que tuviera la falla, esta unidad comenzaría a fallar en cualquier vehículo.

El análisis anterior fue clave para entender que el problema se encontraba en los vehículos que estaban en cuarentena, sin embargo, se debía demostrar que la prueba en estos vehículos era lo que corrompía los módulos. Por ende, se realizó un log file con las comunicaciones que existía en los vehículos en cuarentena al realizar la prueba. El log file es un archivo de texto, usualmente en formato .txt o .asc, el cual guarda en forma de texto plano los mensajes y señales que se transmitieron, la herramienta que se utilizó para ello fue CANoe.

El log file es un archivo que permite analizar las transferencias de datos en los canales de CAN y LIN[xv]. Además, con la herramienta de CANoe es posible simular mensajes y crear un entorno de comunicaciones igual a la del vehículo (Fig. 3-7). Cuando el equipo de pruebas recibió el log file se enfocó en tratar de reproducir el entorno de pruebas del cliente. Para lograr dicho objetivo: conectaron una tarjeta con la versión de software igual a la del vehículo, cargaron las configuraciones que se proporcionaron y se conectó el módulo con la simulación de CANoe que transmitía las señales del log file. Después de un día realizando pruebas, el equipo logró reproducir la falla en un módulo que anteriormente funcionaba bien. Sin embargo, no se detectó que fue lo que logró activar la falla, no se observó un comportamiento inusual.

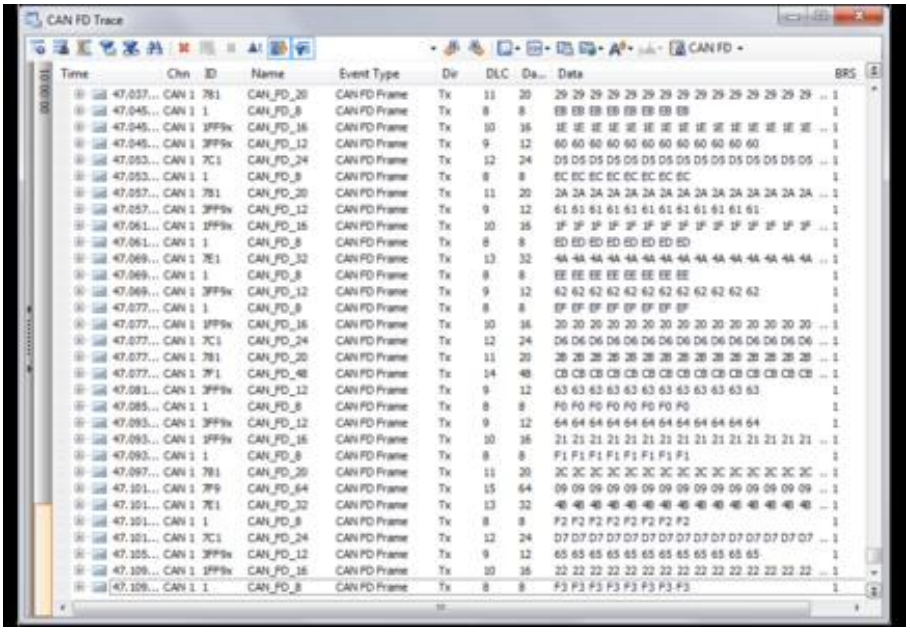

Fig. 3-7 Reproducción de un log file con CANoe

3.3. Hipótesis 3 Error en el hardware de las tarjetas.

Al inspeccionar las unidades se encontraron varios modelos y discrepancias, sin embargo, todas estas diferencias estaban justificadas en las listas de materiales, ya que muchos cambios eran por piezas similares. Por ejemplo: se cambiaba el proveedor de alguna resistencia, capacitor, etc. Pero al hacer estos cambios se consideran sus valores y tolerancias correctamente, este trabajo fue realizado por los ingenieros de hardware y no impactaba de manera critica. Sin embargo, se logró identificar que algunas unidades tenían microcontroladores diferentes.

Los microcontroladores se diferenciaban por dos características: capacidad de memoria y lote de producción. Inicialmente se evaluó que el problema podía radicar en el cambio de microcontrolador, ya que se pasó de 750 kilobytes a 1 megabyte. En aplicación este cambio no impactaba ya que la aplicación no media más de 600 kilobytes, se propuso ampliar la memoria para desarrollo futuro y porque existe un requerimiento que estipula; la aplicación no debe exceder más del 70 por ciento de memoria.

En un inicio, el cliente afirmo que los problemas se presentaban solamente en los microcontroladores de un megabyte, esto dio la idea de que el problema podía relacionarse con el bootloader, ya que se habían modificado los rangos de memoria y quizá se había desbordado al definirla. Sin embargo, mientras se estudiaba esta idea, el equipo de pruebas del cliente logró reproducir el problema en un módulo con un microcontrolador de 750 kilobytes. Por ende, se descartó esta idea como raíz del problema.

Se mencionó anteriormente que los módulos tenían cambios de hardware cada semana, esto se debe a cambios en resistencias, capacitores y componentes pasivos en general. Estos cambios son evaluados por el equipo de hardware y son recopilados en el BOM o lista de materiales. La lista de materiales es un archivo en el que se menciona todos los componentes junto con sus valores, tolerancias, número de parte, costo y cantidades. Este documento es vital para poder producir los módulos en la planta de manufactura, en la Fig. 3-8 se puede apreciar como luce una lista de materiales.

No	PartNumber	Quantity	Description
1	GRM155R71C822KA01D	1	Cap = 8.2 nF, VDC = 16 V
2	RC0201FR-0710KL	2	Resistance = 10 kΩ Tolerance = 1.0% Power = 50 mW
3	HM4443-ND	1	XFRMR LAMINATED 84VA CHAS MOUNT
4	539-CG252U050U3C	2	Aluminum Electrolytic Capacitors - Screw Terminal 2500uF 50V
5	25SVPF47M	2	Cap = 47 µF, VDC = 25 V
6	XAL4020-122MEB	1	L = 1.2 µH, IDC = 7.9 A
7	C0805C104M5RACTU	1	Cap = 100 nF VDC = 50 V
8	C3216X6S1V106K160AC	1	Cap = 10 µF, TVDC = 35 V
9	LMR14050SDDAR	1	IC REG BUCK ADJ 5A
10	B540C-13-F	1	Type = Schottky VRRM = 40 V Io = 5 A
11	CRCW0402150KFKED	1	Resistance = 150 kΩ Tolerance = 1.0% Power = 63 mW
12	GRM033R71E471KA01D	1	Cap = 470 pF VDC = 25 V

Fig. 3-8 Fragmento de una lista de materiales

4. Experimentación

Con el análisis de las hipótesis se logró consolidar los siguientes resultados; el equipo de pruebas logró reproducir la falla en una unidad, el equipo de software se enfocó en realizar un volcado de memoria y compararla con la versión del entregable, se procedió a conectar un simulador al módulo, grabar como interactuaban las ejecuciones de funciones y descubrir la razón del reset.

Un simulador es una herramienta que se conecta sobre el JTAG[xvi], permite ver las ejecuciones de las tareas y también los valores de las variables en tiempo real. También ofrece herramientas gráficas que facilitan el análisis de datos. La Fig. 4-1 es un ejemplo de las ejecuciones de tareas y funciones del módulo.

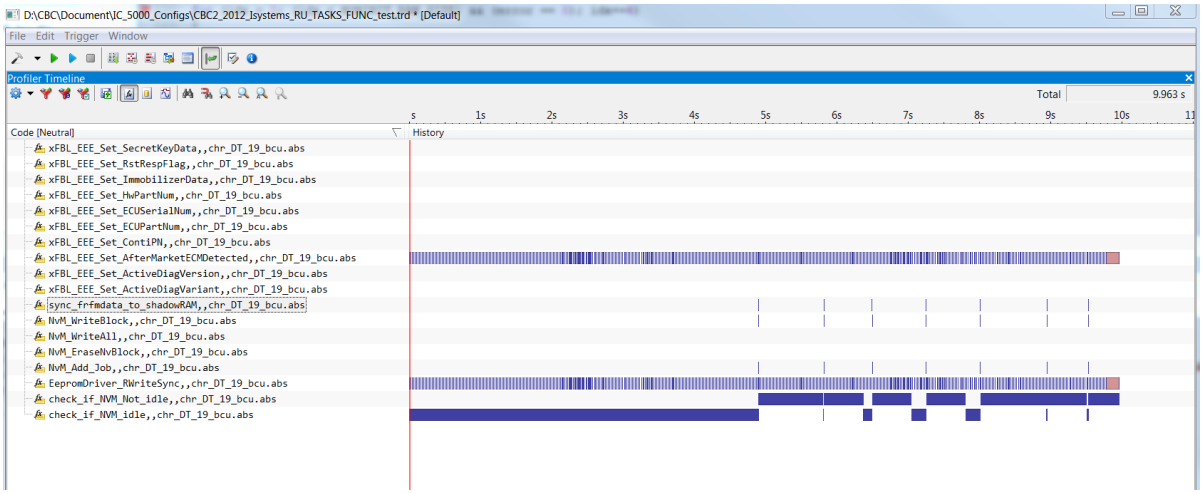

Fig. 4-1 Imagen de una simulación y ejecución de tareas de un microcontrolador

Después de validar y estudiar las ejecuciones de las tareas, se encontró que existía una carga de procesamiento el controlador de NVM, ya que estaba escribiendo constantemente. Este comportamiento no era correcto y se daba porque no se diseñó con semáforos ni mutex. Esto provoca que exista la posibilidad de sobrescribir datos y que al verificar la escritura sea incorrecta. Al tener una corrupción de datos, el controlador de NVM inicia de manera segura el bootloader para buscar una aplicación valida o una nueva para reprogramar.

Esto fue precisamente lo que sucedió, la escritura excesiva del controlador de NVM provocaba que las secciones de memoria se corrompieran y estas a la hora de ser verificadas, se marcaran como corruptas, provocando un reset en el módulo. Al reset, el bootloader evalúa si la aplicación esta corrupta o no; si esta corrupta busca una nueva imagen para reprogramar, si no encuentra se queda en un bucle infinito. Si encuentra que la aplicación es correcta salta a la aplicación y esta es ejecutada con normalidad. En este caso particular como la NVM corrompió información y no se reprogramaba la unidad se quedaba en el bucle infinito en espera de una aplicación válida. La Fig. 4-2 explica de manera gráfica estas transiciones.

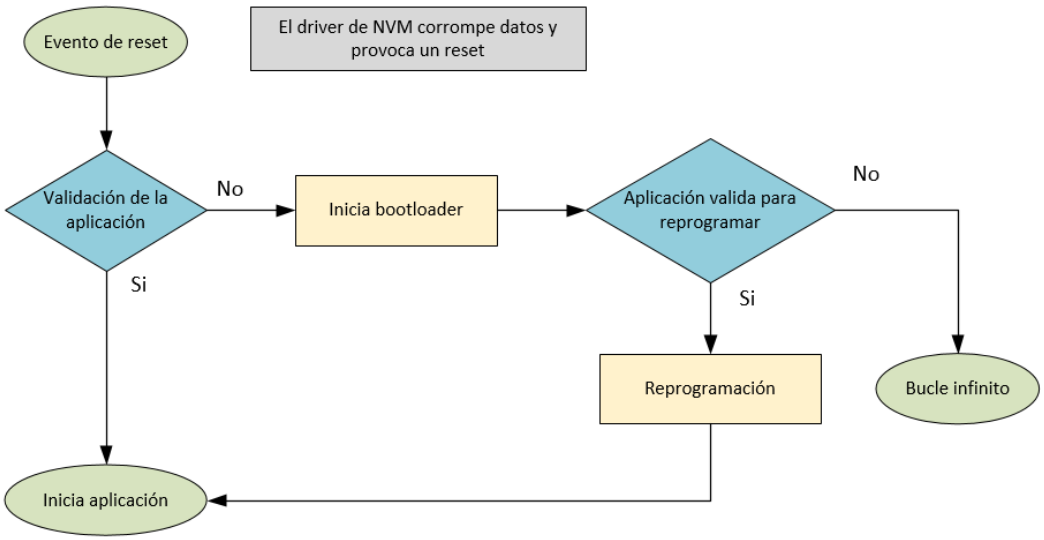

Fig. 4-2 Ilustración de los saltos entre la aplicación y booloader

El reset fue ocasionado por la corrupción de datos que originó el controlador de NVM, esto se debía por la gran carga de trabajo que demandaba la aplicación. Al investigar la función que necesitaba del servicio de escritura a NVM se encontró que tenía un error de codificación, este error de codificación consistía en que las evaluaciones estaban siendo ejecutadas en un orden incorrecto. Sin embargo, no se había detectado antes porque esta funcionalidad es inusual. La funcionalidad se llama aftermarket, y consiste en guardar en una sección de memoria no-volatile, una bandera que se activa al detectar modificaciones en las computadoras, esto con la finalidad de proteger la integridad del vehículo.

Esta funcionalidad no es probada usualmente, ya que el módulo guarda la bandera del aftermarket en una sección protegida: esta no se puede borrar, ni restaurar. Los vehículos de prueba que se tenían en el parque industrial, tres de ellos tenían modificaciones en la computadora del motor, esta modificación activaba la funcionalidad de aftermarket y provocaba que el módulo fallara por las escrituras constantes a NVM, si este mismo módulo se pasaba a otro vehículo que no tuviera estas modificaciones, el módulo continuaba con la falla ya que el aftermarket se recuperaba de la sección de memoria y volvía a sobrescribir el controlador de NVM.

En la Fig. 4-3 se puede apreciar la diferencia que provoca el problema, esta diferencia es que el primer IF tiene dos condicionales; el primero tiene a la bandera del aftermarket y el segundo todas las líneas de vehículo para los que esta funcionalidad aplica, estas condiciones tienen un AND entre ellas. En el lado derecho de la imagen se tiene una modificación en el orden y la lógica, ya que el AND aplica a la primera línea de vehículos y los otros quedan en un OR. En resumen, la

```
/* CBC shall write the state of AM_ECM_Prsnt signal into EEPROM only if
if((TRUE == byAMCountsExpired) || (TRUE == byAfterMarketECMStored ))
{
    /* Save to EEPROM only for the required vehicle lines */
    if((TRUE != ee_DID_A053.after_market_ECM_detected) &&
       ((uint8)kVC_VEH_LINE_0_VEH_WK == ee_DID_0122.veh_line ||
        (uint8)kVC_VEH_LINE_0_VEH_WD == ee_DID_0122.veh_line ||
        (uint8)kVC_VEH_LINE_0_VEH_PF == ee_DID_0122.veh_line ||
        //VEH_JK == ee_DID_0122.veh_line ||  macro not available
        (uint8)kVC_VEH_LINE_0_VEH_LD == ee_DID_0122.veh_line ||
        (uint8)kVC_VEH_LINE_0_VEH_LA == ee_DID_0122.veh_line ||
        (uint8)kVC_VEH_LINE_0_VEH_LX == ee_DID_0122.veh_line)))))/* A
    {
        byWriteRequest = TRUE;
        cpy_ee_DID_A053.after_market_ECM_detected = TRUE;
    }
}
```

```
2217    /* CBC shall write the state of AM_ECM_Prsnt signal into EEPROM onl
2218    if((TRUE == byAMCountsExpired) || (TRUE == byAfterMarketECMStored )
2219    {
2220        /* Save to EEPROM only for the required vehicle lines */
2221        if (((((((TRUE != ee_DID_A053.after_market_ECM_detected) &&
2222            ((uint8)kVC_VEH_LINE_0_VEH_WK == ee_DID_0122.veh_line))||
2223            ((uint8)kVC_VEH_LINE_0_VEH_WD == ee_DID_0122.veh_line))||
2224            ((uint8)kVC_VEH_LINE_0_VEH_PF == ee_DID_0122.veh_line))||
2225            ((uint8)kVC_VEH_LINE_0_VEH_DT == ee_DID_0122.veh_line))||
2226            ((uint8)kVC_VEH_LINE_0_VEH_LD == ee_DID_0122.veh_line))||
2227            ((uint8)kVC_VEH_LINE_0_VEH_LA == ee_DID_0122.veh_line))||
2228            ((uint8)kVC_VEH_LINE_0_VEH_LX == ee_DID_0122.veh_line))/* A
2229        {
2230            byWriteRequest = TRUE;
2231            cpy_ee_DID_A053.after_market_ECM_detected = TRUE;
2232        }
2233    }
```

Fig. 4-3 Comparación entre dos versiones donde cambia la lógica por el orden de los paréntesis

primera imagen hace la evaluación correctamente y la segunda imagen no evalúa contra aftermarket, esto desencadena la constante escritura y la corrupción de datos en la NVM. La Fig. 4-4 explica de manera detallada la transición de la funcionalidad.

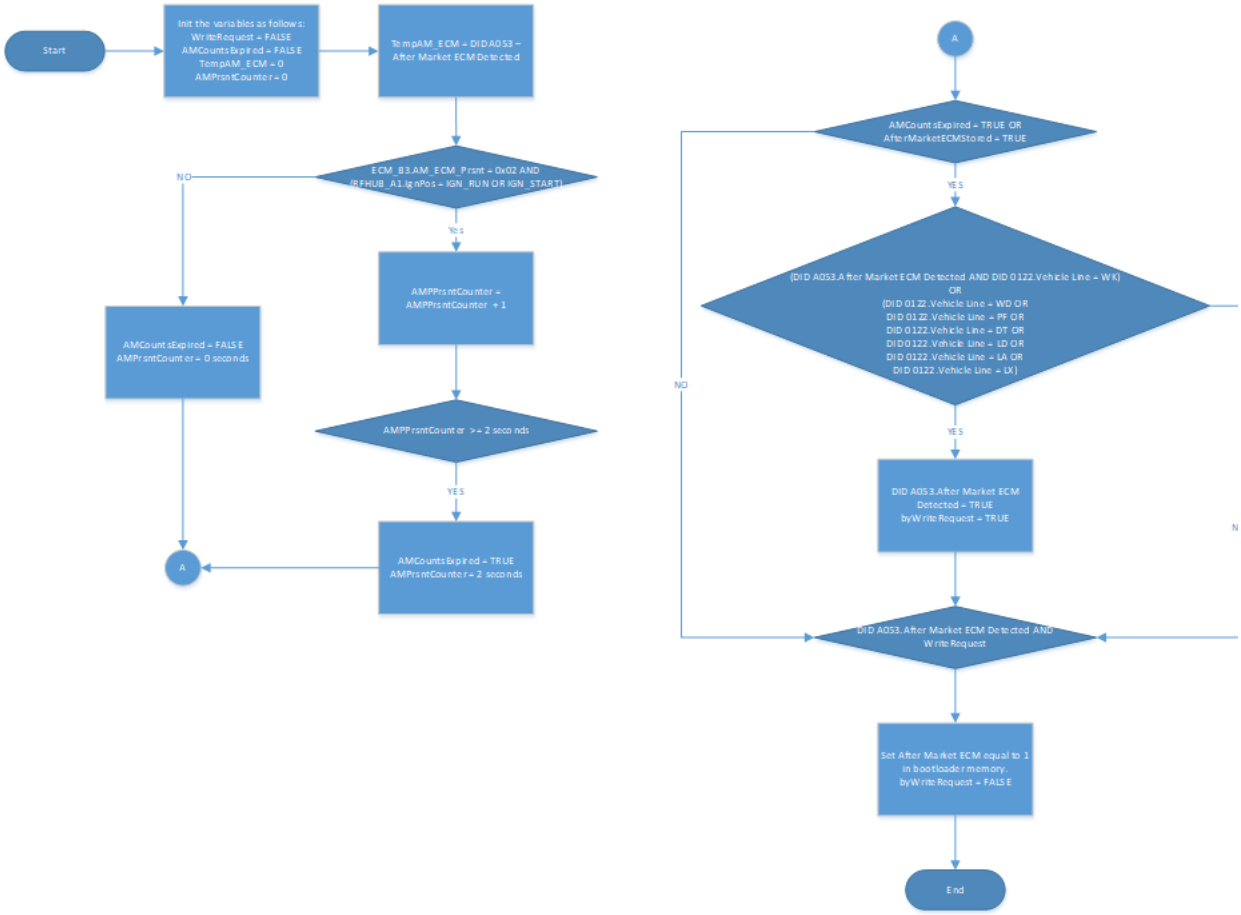

Fig. 4-4 Diagrama de flujo de la funcionalidad aftermarket con el problema

Ya con la raíz del problema identificada se procedió a elaborar la solución, la solución inicial era regresar la implementación como estaba. Sin embargo, se descartó por el hecho de que no cumplía con los estándares de codificación (MISRA). Se procedió a rehacer la funcionalidad de aftermarket con la retroalimentación del cliente y la experiencia adquirida al evaluar todas las pruebas. El rediseño de la funcionalidad quedó con mayor robustez y más segura. Además, se le agregaron semáforos y mutex al controlador de NVM para mejorar su desempeño.

5. Validación de pruebas

Para validar el correcto comportamiento se procedió a generar varias fases de prueba, las cuales son:

- Pruebas de unidad y verificación de código mediante herramientas, SWATT y MISRA.
- Pruebas funcionales en los bancos realizadas por el equipo de software y testing.
- Pruebas en vehículos realizadas en el parque vehicular del cliente.

Las pruebas de unidad se utilizan para comprobar que un código concreto funciona correctamente, esto se logra evaluando asignaciones y condiciones que el código debe cumplir, una buena prueba de unidad debe cumplir las siguientes características; ser repetible, ser automatizada, cubrir casi la totalidad del código, ser portable y ejecutarse en cualquier entorno, no afectar a otras pruebas, y conocer claramente el objetivo del código [Universidad-de-Alicante-19].

La herramienta que se usó para verificar las pruebas de unidad fue SWATT, esta herramienta fue creada dentro de la empresa y tiene el objetivo de poder probar componentes de software a los siguientes niveles; pruebas de unidad, pruebas a nivel funcional y pruebas a hardware ciclado. Estas técnicas se usan para reducir los riesgos de implementar errores en código. Para dar seguridad y robustez se estudió la nueva implementación y se desarrolló todos los niveles de prueba. Después de revisar las pruebas y los resultados se concluyó que la nueva implementación estaba bien elaborada. La Fig. 5-1 muestra de color rojo las modificaciones que se hicieron a la funcionalidad de aftermarket.

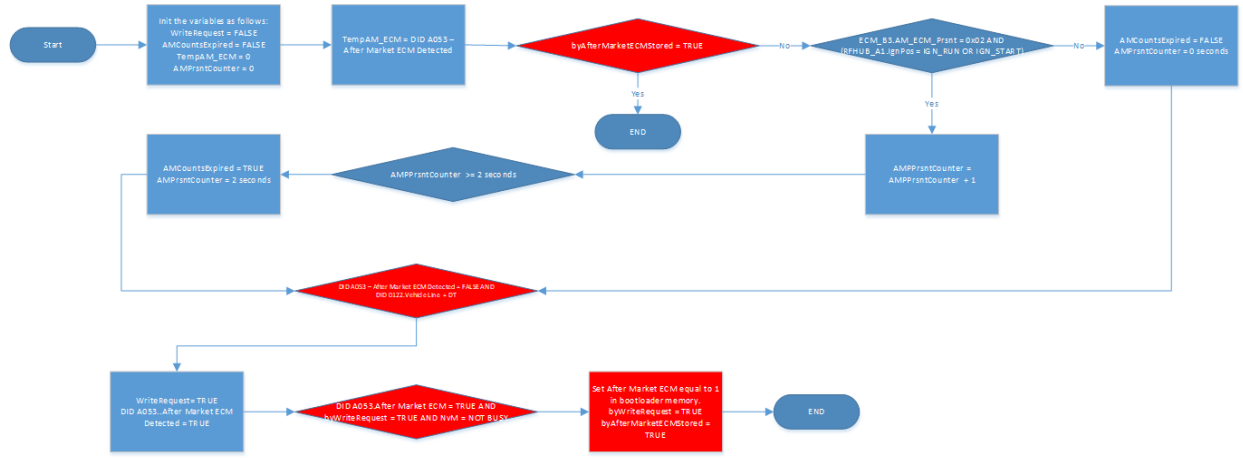

Fig. 5-1 Diagrama de flujo de la funcionalidad aftermarket con la solución implementada

Las pruebas de análisis estático tienen como fin mejorar la calidad de código, esto se hace en base a reglas o buenas prácticas que se han estandarizado a lo largo de los años. Este análisis busca que el código tenga confiabilidad, rendimiento, seguridad y mantenibilidad [MISRA-12]. La empresa usa MISRA para realizar este análisis, ya que tiene más de 25 años dentro de la industria automotriz en sistemas embebidos.

La implementación de la funcionalidad de aftermarket, se analizó con MISRA 2012, al realizar el análisis la primera vez se encontraron varias violaciones como el orden de ejecución de las condicionales dentro de un IF y asignaciones incorrectas a variables por diferencias en tipos de datos. Después de realizar las correcciones, la implementación de aftermarket quedo limpia de errores y precauciones, la revisión de la implementación fue realizada con 4 personas, 2 de ellas conocían la funcionalidad y las otras 2 eran externas al proyecto.

Ya terminada la implementación se procedió a validarla en los bancos de pruebas, se decidió que probaran dos equipos de software, un grupo de la localidad de Troy y otro grupo en la localidad de Guadalajara. Ambos equipos diseñaron sus casos de pruebas y abarcaron los mismos objetivos de la funcionalidad. Después de realizar las pruebas los dos equipos compartieron sus resultados y se revisaron a detalle. Ambas pruebas salieron satisfactorias y el problema ya no era reproducible. En la Fig. 5-2 se aprecia que las comunicaciones se recuperan después de un reset provocado.

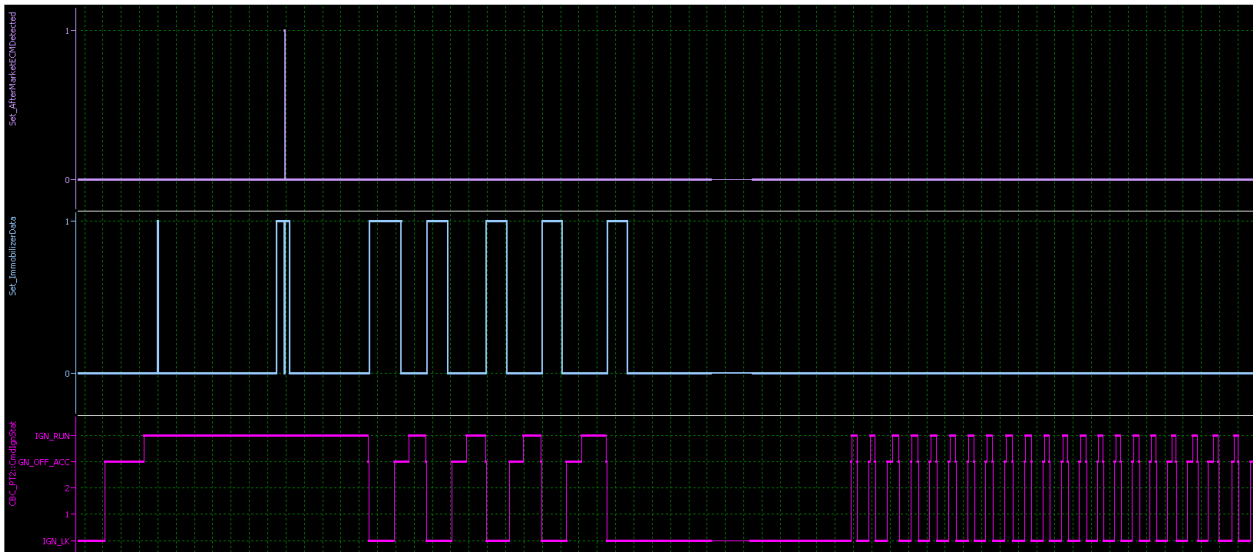

Fig. 5-2 Simulación de CANoe, donde se aprecia las comunicaciones a pesar de un reset provocado

Por último, se realizaron pruebas funcionales en vehículos, estas pruebas fueron creadas y evaluadas con ayuda del cliente, el primer vehículo a evaluar se encontraba en Troy y los demás vehículos en el parque del cliente. Al no poder reproducirse el problema en el primer vehículo, se procedió a probar en los vehículos restantes. El problema ya no pudo replicarse.

Con ayuda del cliente, se realizó un análisis de que tanto se usa la funcionalidad de aftermarket en los vehículos y dado que es una medida para detectar que se ha invalido la garantía, su reproducción en cliente potencial debería ser baja o nula, ya que no deberían realizarse modificaciones en el vehículo fuera de la agencia, sin embargo, si existen empresas que modifican la computadora del motor para obtener más potencia, estas personas serían vulnerables si no se incorpora el parche a todas las unidades.

Conclusión

En este trabajo se presentó como se aborda la identificación y análisis de un problema de software en particular, la intención de dicho documento es explicar y demostrar que los diagramas existentes en manufactura también pueden funcionar en desarrollo de software, ya que estos proveen varias hipótesis de la causa raíz del problema. Las herramientas que se usaron fueron la definición del problema, un diagrama Ishikawa y el diagrama del proceso, con ellas pudieron crearse tres hipótesis que se sometieron a investigación.

El problema al ser complejo se abordó con el método científico, cuyas características son ser reproducible y refutable, parte de la dificultad de problema era que el equipo no podía encontrar las condiciones de reproducción, este fue uno de los mayores retos, pero gracias a las herramientas antes mencionadas y con las hipótesis obtenidas, se procedió a observar, analizar, verificar y experimentar cada una de ellas hasta encontrar el entorno de reproducción.

Dos de los resultados de las hipótesis fueron refutados porque se demostró que la hipótesis era errónea y que la causa raíz no recaía en ese factor de investigación, sin embargo, se obtuvo una teoría a partir de una hipótesis, fue esta teoría la que ayudo a conocer el entorno de reproducción, identificar la causa raíz y posteriormente la solución del problema.

Los aprendizajes obtenidos del trabajo son; reafirmar los conceptos del método científico, aprender a utilizar herramientas para la identificación de problemas, adaptar las herramientas al entorno y con las características que se necesita.

Bibliografía

[Martins-Rosemary-18] Martins, Rosemary. (2018). Diagrama de Flujo (Flujograma) de Proceso, 18/11/2020, Qualiex, https://blogdelacalidad.com/diagrama-de-flujo-flujograma-de-proceso/

[MISRA-12] MISRA. (2012). A GUIDE TO CODING STANDARDS MISRA C and MISRA C++.12/11/2020, Performance, https://www.perforce.com/resources/qac/misra-c-cpp

[Progressa-Lean-15] Progressa Lean. (2014). 5W+2H Técnica de análisis de problemas, 18/11/2020, Progressa lean, https://www.progressalean.com/5w2h-tecnica-de-analisis-de-problemas/

[Rodriguez-Johanna-19] Rodriguez, Johanna. (2019). Qué es el diagrama de Ishikawa y cómo aplicarlo en tus procesos, 18/11/2020, Lecturas de 10 Min, https://blog.hubspot.es/sales/diagrama-ishikawa

[TechTarget-16] TechTarget. (2016). ¿What is memory dump?. 12/09/2020. WhatIs.com, https://whatis.techtarget.com/definition/memory-dump

[Universidad-de-Alicante-19] Universidad de Alicante. (2019). Pruebas unitarias. 12/11/2020. Curso .NET con C#, https://si.ua.es/es/documentacion/c-sharp/documentos/pruebas/07pruebasunitarias.pdf

Índice

Acrónimos

[i] **SW** Software

[ii] **BCM** Body control module, es el módulo que controla la mayor cantidad de funciones del vehículo

[iii] **DT** es una variante vehicular, en este caso son plataformas para camionetas de carga

[iv] **ECU** Unidad de control electrónico, son módulos que desempeñan actividades especificas en el vehículo

[v] **BOM** Bill of materials / Lista de materiales

[vi] **MY** Model year

[vii] **EFD** Formato de archivo para programar mediante diagnostico

[viii] **CDA** Herramienta de diagnostico de Chrysler

[ix] **NVM** Non volatile memory

[x] **CAN** Protocolo de comunicación / controller area network

[xi] **BTL** Bootloader

[xii] **IMMO** Immobilizer

[xiii] **CANoe** Programa de software de la compañía Vector

[xiv] **RAM** Random access memory

[xv] **LIN** Local interconnected network

[xvi] **JTAG** Joint test action group