UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

TRABAJO FIN DE MÁSTER

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

2015/2016

Verificación y encapsulamiento de test para codificación en PLC en la empresa Stadler

AUTOR: Álvaro Sanz Garrigues

TUTOR: Santiago Escobar Román

COMPAÑÍA: Stadler Rail Valencia

FECHA: lunes, 19 de septiembre de 2016

Resumen

Este trabajo fin de máster se ha realizado bajo el marco de una relación de prácticas con la empresa ferroviaria Stadler Rail Valencia. El objetivo principal del proyecto es analizar la normativa europea EN 50128, que define los requisitos que se han de cumplir tanto para el desarrollo, implantación y mantenimiento de cualquier software relacionado con la seguridad y control de los vehículos ferroviarios, para adecuar la herramienta de verificación y testeo para PLC de la empresa a lo especificado en la normativa.

Es necesario analizar esta herramienta usada y desarrollada por Stadler Rail para sus tareas de ensayos y verificación con el objetivo de proponer mejoras y funcionalidades extra que debería incluir para adecuarse más a la normativa. Para definir el estado del arte en este campo y conocer mejor este tipo de herramientas, se hará una búsqueda de herramientas existentes en el ámbito de verificación y testeo para software PLC.

Agradecimientos

Me gustaría expresar mis más sinceros agradecimientos a varias personas sin las cuales este Trabajo Final de Máster no hubiera sido posible. Por todos los momentos durante este tiempo en los que gracias a ellos he aprendido, he mejorado y he conseguido llegar a este punto.

He de dar gracias a todos mis profesores/as, principalmente a los de este máster por haber compartido conmigo y mis compañeros sus conocimientos, descubrirnos cosas nuevas e interesantes y en muchos casos hacernos sentir la pasión que sienten por aquello que imparten, haciéndonos ver lo mucho que nos queda por aprender y descubriéndonos nuevas formas de seguir formándonos.

También quiero agradecer a mis compañeros/as de clase por todos los momentos compartidos. Las interminables horas de estudio en la biblioteca, las tardes enteras haciendo trabajos, así como otros ratos menos duros y un poco más festivos, pero igualmente enriquecedores. En especial a Adrián y Raducu los cuales han sido algo más que compañeros de clase, han sido buenos amigos.

A mis padres y familia por el apoyo incondicional que siempre me han prestado tanto en los buenos y malos momentos.

A Sandra, que ha sido mi pilar, que siempre me entiende y tiene las palabras perfectas para cada momento hacerme ver que puedo con todo y más.

A la empresa Stadler Rail Valencia por brindarme la oportunidad de trabajar con ellos y por el buen trato e interés que han mostrado durante el desarrollo de este trabajo. En particular a Javier y Ricardo que han sido una buena fuente de información disponibles siempre para cualquier duda y que han hecho un buen seguimiento semanal del trabajo y nos daban indicaciones de como continuar.

Y por último a mi tutor Santiago, que ha estado muy involucrado en nuestra relación con la empresa siempre velando por nuestro bienestar y que me echaba una mano siempre que la solicitaba o cuando encontraba algo interesante que podía ayudarme. Además de aclarar las ideas cuando más difícil era la situación y no estaba claro el camino a seguir.

Tabla de contenido

1. INTRODUCCIÓN

Este trabajo final de máster ha sido desarrollado como parte de una beca en la Cátedra Stadler dentro de la Universidad Politécnica de Valencia. La empresa Stadler Rail, es una empresa líder en el sector ferroviario mundial y que se ha consolidado como un referente en el mercado español en este sector gracias a la compra de su nueva filial Stadler Rail valencia.

En la actualidad desarrollan, verifican y validan sistemas de control y monitorización para sus vehículos ferroviarios. EL software PLC tiene una gran importancia en aplicaciones de control y seguridad del vehículo ferroviario. Los dispositivos lógicos programables (PLC) permiten la automatización de procesos industriales y el control del funcionamiento de máquinas a las que están asociados. Para la fase de desarrollo y verificación de este tipo de software, que será el tema principal en el que se centrará este proyecto, la llevan a cabo con una herramienta desarrollada por ellos, SoftPro.

Esta herramienta permite realizar y almacenar test para el software PLC. Permitiendo su simulación sobre dispositivos PLC reales siendo capaz de acceder a sus posiciones de memoria tanto para variables de entrada como de salida permitiendo definir los valores del caso de ensayo como se quiera. La lectura de las posiciones de memoria de las variables de salida le permite comprobar los resultados del caso de ensayo.

El problema que se plantea es que su herramienta y sus procesos de verificación y casos de prueba no se ajusta a la normativa europea EN 50128 con título; Aplicaciones ferroviarias; Sistemas de comunicación, señalización y procesamiento; Software para sistemas de control y protección del ferrocarril. Este estándar europeo define los requisitos que se han de cumplir tanto para el desarrollo, implantación y mantenimiento de cualquier software utilizado por las aplicaciones de control y seguridad de los ferrocarriles. Estos requisitos son distintos según el nivel de integridad del software, que se especifican también en la normativa y van ligados a la criticidad del software

Para cubrir las necesidades requeridas por la norma será necesario ampliar las funcionalidades que ofrece SoftPro con ciertas técnicas de análisis estático, que es bastante importante dentro de la normativa y también medidas de prestaciones de los programas realizados.

A pesar de que para Stadler Rail es importante crear sus propias soluciones, es interesante realizar una búsqueda de herramientas de verificación y casos de prueba para PLC. Este es un campo en el que pese a su criticidad no se ha invertido y no existen muchas soluciones en este ámbito. A pesar de ello existen alguna serie de herramientas que se centran en esta temática. Las herramientas analizadas se pueden dividir en dos bloques a groso modo, “model checkers” o analizadores estáticos. PLC Verif es un model checker desarrollado por el CERN adaptado a lenguajes PLC que permite su uso a usuarios que no son expertos en la verificación formal. ARCADE PLC, otro “model checker”, pero menos funcional que el anterior que también verifica lenguajes PLC de

forma nativa. En el otro grupo de herramientas encontramos la solución desarrollada por ITRIS Automation, que se centra en el análisis del código para que cumpla ciertas normas predefinidas de codificación o de funcionalidad. Por último, la herramienta o conjunto de herramientas REFLEX Studio que aborda todas las fases del desarrollo de Software PLC, diseñar, desarrollar, simular, testear y validar aplicaciones de control.

2. STADLER RAIIL

Stadler Rail AG es una empresa multinacional del sector ferroviario que produce trenes de alta fiabilidad y comodidad para adaptarse a las necesidades de sus clientes y pasajeros, que esperan lo mejor.

Sus más de 7000 empleados repartidos por todo el mundo, están constantemente trabajando para mejorar el rendimiento y eficiencia de sus vehículos ferroviarios. Haciendo uso de tecnologías de estado del arte, el objetivo es prestar un servicio preciso y de máxima calidad.

Recientemente, Stadler ha adquirido la filial de "Vossloh España". Con esta adquisición, Stadler con la nueva filial Stadler Rail Valencia, pretende convertirse en el mayor proveedor de material ferroviario en España y otros mercados diseñando y produciendo una gran variedad de locomotoras y vehículos de pasajeros. Ayudando así, a Stadler Rail AG a mantener su liderazgo mundial ofreciendo productos innovadores y competitivos siguiendo una estrategia de alta calidad.

Stadler Rail Valencia cuenta con más de 150 investigadores, ingenieros y técnicos que tienen a su alcance las más avanzadas tecnologías y herramientas. Conscientes de la importancia de la investigación y desarrollo, han firmado acuerdos de colaboración con universidades y centros de investigación con el objetivo de encontrar nuevas soluciones y diseños basados en las últimas tecnologías, así como para mejorar las propiedades de los productos ya existentes. Para producir productos innovadores con tecnologías punteras y de calidad óptima.

La filial Valenciana tiene más de un siglo de historia desde sus orígenes en 1897 con el nombre de Talleres Devis. Pero no fue hasta 1929, bajo el nombre de Construcciones Devis, cuando los fabricantes valencianos se colocaron en primera línea del mercado de productos ferroviarios destacando en la reparación de vagones y expandiéndose en el mercado nacional. En 1947, con la fusión con una compañía catalana se fundó MACOSA (Material y Construcciones SA) donde se consolidó como proveedor de material rodante.

En 1989 fue comprada por la multinacional conocida como Alstom y se desplazaron en 1997 a las actuales y más modernas instalaciones en Albuixech. En 2005 Vossloh, uno de los más importantes competidores en el sector de las tecnologías ferroviarias adquirió la compañía estableciéndola como Vossloh España y la consolidaron, gracias a sus altos estándares de tecnología y la experiencia de sus trabajadores, se colocaron como líder del sector de locomotoras eléctricas y diésel. Fue finalmente el 1 de enero de 2016 cuando Stadler Rail AG tomó el control del negocio ferroviario de Vossloh en Valencia.

La compañía Stadler Rail tiene las ideas claras respecto a los valores que han de representar sus vehículos.

Su objetivo es trabajar con **tecnologías de estado del arte** y siempre mirar al futuro para desarrollar los vehículos perfectos para sus clientes.

Focalizándose en la **eficiencia**, gran disponibilidad, baja demanda de mantenimiento y consumo energético de sus productos. Lo que les permite, gracias también a sus bajos costes de ciclos de vida, ofrecer a sus clientes las soluciones más económicas.

El **buen trato** que se le da al cliente, se demuestra, desde la primera interacción hasta después de la entrega. Tratándolos como lo más importante. Lo que se demuestra en sus largas relaciones comerciales, que son productivas, positivas y beneficiosas para todos.

Estas buenas relaciones se forjan gracias a la **fiabilidad** que proporciona un socio tan experimentado como Stadler Rail AG, que produce a tiempo productos de una calidad puntera siempre bajo los términos acordados.

La **flexibilidad** de sus productos es característica ya que sus ingenieros y diseñadores prestan atención a todas las necesidades de sus clientes y hacen realidad todas sus peticiones.

3. NORMATIVA EN50128

Existe una normativa Europea EN 50128 con título: Aplicaciones ferroviarias; Sistemas de comunicación, señalización y procesamiento; Software para sistemas de control y protección del ferrocarril. Esta normativa se centra en los métodos que han de ser usados para que el software desarrollado cumpla los requisitos determinados por los niveles de integridad de la seguridad.

Provee un conjunto de requisitos que se han de cumplir tanto para el desarrollo, implantación y mantenimiento de cualquier software utilizado por aplicaciones de control, protección o seguridad de las aplicaciones de ferrocarriles. Estos requisitos son relativos a la estructura organizativa, las relaciones entre las organizaciones y la división de responsabilidades involucradas en las actividades de desarrollo, implantación y mantenimiento. También se proponen unos criterios de las calificaciones y experiencia del personal encargado de estas actividades.

La clave de este estándar europeo son los niveles de integridad de la seguridad del software (Software Integrity Level- SIL). Se definen cinco niveles de integridad de la seguridad del software siendo 0 el más bajo y 4 el más alto. El nivel viene definido según cuan peligrosas sean las consecuencias de un fallo del software, es decir, cuanto mayor sean las consecuencias mayor será el nivel de integridad de la seguridad el software.

Cada uno de los niveles requiere y viene especificado en la normativa, unos requisitos distintos y cada vez más acorde a la criticidad del software. Las técnicas y medidas para cada uno de los cinco niveles de integridad de la seguridad, están definidas en la normativa. En la versión analizada del estándar, los niveles 1 y 2 presentan los mismos requisitos al igual que los niveles 3 y 4. De forma contraria, no se define en esta norma ni se da indicaciones de que nivel de integridad de seguridad del software es el apropiado para un riesgo en determinado.

La especificación de funciones de seguridad que se asignan al software queda dentro del alcance de otras normativas asociadas a ésta como son, EN 50126-1 y EN 50129. Lo que sí se define en la norma EN 50128 son las medidas necesarios para cumplir los requisitos de los niveles de integridad de la seguridad del software.

Pero es cierto que con las tecnologías actuales es imposible garantizar la seguridad absoluta del sistema, ya que no se conoce ninguna forma de demostrar la ausencia de errores en un software con cierto nivel de complejidad. Para lidiar con esto, la normativa especifica las etapas funcionales que se han de aplicar para el ciclo de desarrollo de un software relacionado con la seguridad y control de sistemas ferroviarios:

1. Definir la especificación de Requisitos del Software al mismo tiempo que se ha de considerar la arquitectura de éste.
2. Diseñar, desarrollar y testear el software según lo especificado en el Plan de Garantía de la Calidad del Software, el nivel de integridad de seguridad del software y su ciclo de vida.

3. Llevar a cabo la integración del software con su hardware y verificar así su funcionalidad.
4. Aceptación e implantación del software.
5. Por último, mantenimiento del software durante su vida operativa si fuera necesario. Reactivando esta normativa siempre que sea necesaria.

Durante estas etapas se llevan a cabo distintas actividades que también está especificadas en la normativa europea. Entre éstas están los test, verificación, validación, evaluación, control de la calidad y modificación y control de las modificaciones.

Además también se abarcan la definición de requisitos para herramientas de soporte y sistemas configurados mediante datos de aplicación o algoritmos.

3.1. CAMPO DE APLICACIÓN

Esta normativa aplica sobre los requisitos técnicos y procedimientos necesarios para el desarrollo de software para sistemas electrónicos programables para su uso en aplicaciones de control y seguridad del ferrocarril. Se pretende que sea usado en cualquier área que implique riesgos en la seguridad siendo indiferente la arquitectura usada para esos sistemas, ya sea usando microprocesadores dedicados, controladores lógicos programables, sistemas multiprocesadores distribuidos u otras. Además se ha de aplicar a todo software relacionado con los sistemas de control y protección del ferrocarril como por ejemplo programación de aplicaciones, sistemas operativos, herramientas de soporte y firmware.

También se contempla su la aplicación de la normativa en el uso de software y herramientas preexistentes para los cuales se han definido ciertos requisitos. De forma similar también se tiene en cuenta y es adecuado el uso de software genérico que se configura específicamente para producir el software particular a ejecutar. Partes de esta normativa definen los requisitos de uso de software genérico.

Por el contrario, la aplicación de esta norma no es relevante para aquel software que no tenga ningún impacto en la seguridad.

Por último, no se pretende tener un carácter retroactivo, se aplicará esta normativa principalmente a los nuevos desarrollos y solo se aplicará a los ya existentes siempre y cuando estos se hayan vistos sujetos a modificaciones considerables. Para cambios menores también se especifica una serie de requisitos.

3.2. NORMATIVA BAJO ESTUDIO

El ámbito de aplicación de la normativa presentada es muy amplio, ya que cubre toda la vida del software. Pero para las necesidades de este trabajo no toda la normativa es objetivo de estudio. Aunque es interesante conocerla entera para comprenderla mejor, los puntos más importantes para este caso son los apartados de casos de ensayo y verificación.

3.2.1. ENSAYOS DEL SOFTWARE

El primer punto que se detallará en este trabajo es el de casos de ensayo. La normativa fija como objetivo de estas actividades la comprobación del comportamiento y prestaciones del software desarrollado en base a la especificación de ensayos en la manera de lo posible con la cobertura del test seleccionada. Se especifican como documentos necesarios para llevar a cabo esta fase toda la documentación del Sistema, del Hardware y del Software especificada en el Plan de Verificación del Software. A su vez, también se fijan todos los documentos de salida necesarios para cumplimentar la normativa, son los siguientes:

- Especificación de Ensayos del Software en Conjunto.
- Informe de Ensayos del Software en Conjunto.
- Especificación de Ensayos de Integración del Software.
- Informe de Ensayos de Integración del Software.
- Especificación de Ensayos de Integración del Software/ Hardware.
- Informe de Ensayos de Integración del Software/ Hardware.
- Especificación de Ensayos de los Componentes Software.
- Informe de Ensayos de los Componentes Software.

Cualquier equipo de medición como hardware, herramientas o software ha de estar apropiadamente calibrado para adaptarse a sus objetivos. Los ensayos han de documentarse como se ha mostrado anteriormente con una especificación y un informe siguiendo las siguientes normas

3.2.1.1. ESPECIFICACIONES DEL TEST

Toda especificación de test ha de incluir la siguiente información relevante según indica la normativa.

- Objetivos del test.
- Casos de prueba, datos de los test y resultados previstos.
- Tipos de test a realizar.
- Entorno de los test, herramientas, configuración y programas.
- Criterios de los ensayos que servirán para juzgar la consecución o no del test.
- Los criterios a satisfacer y los grados de cobertura a alcanzar por los ensayos.

- Las responsabilidades y roles del personal encargado en el proceso de test.
- Los requisitos que cubre la especificación.

3.2.1.2. INFORMES DEL TEST

El formato y contenido de los informes de los test ha de contener cierta información según lo indicado en la norma.

- El Informe debe mencionar el nombre de los encargados de realizar el test, los resultados de las pruebas y si se han cumplido los objetivos y seguido los criterios definidos en la Especificación del test. Además, los fallos han de ser documentados.
- Los casos de prueba y sus resultados han de ser guardados, preferiblemente en un formato entendible por una máquina para su posterior análisis.
- Los test han de ser repetibles y de ser posible llevados a cabo de manera automática.
- Los scripts de los casos de prueba para su ejecución automática, deben de ser verificados.
- Se ha de documentar todas las configuraciones de todos los elementos envueltos en los casos de prueba.
- Ha de evaluarse la cobertura y la completitud del test. También se han de anotar cualquier modificación de éstos.

3.2.2. VERIFICACIÓN DEL SOFTWARE

El objetivo que se persigue en la fase de verificación del software es examinar y llegar a una conclusión basándose en pruebas, de que los documentos u objetos de salida de cada una de las fases, cumplen con los requisitos y planes de acción respecto a su completitud, corrección y consistencia. Todas estas acciones son llevadas a cabo por el verificador.

Como documentos necesarios para llevar a cabo esta tarea se necesita toda la documentación necesaria del Sistema, del Hardware y Software. Y se pretende obtener como resultados de esta fase los siguientes documentos:

- Plan de Verificación del Software.
- Informe(s) de Verificación del Software.
- Informe de Verificación de Garantía de Calidad del Software.

Por tanto, la fase de verificación ha de documentarse con un Plan de Verificación de Software y al menos un informe de Verificación. Se especifican ciertos requisitos para los documentos de esta fase.

3.2.2.1. PLAN DE VERIFICACIÓN DEL SOFTWARE

Este documento ha de describir las actividades que se han de llevar a cabo para asegurar una verificación correcta y que se tiene en cuenta cualquier necesidad particular en materia de diseño u otras verificaciones. Dependiendo del tamaño del sistema, durante el desarrollo, el plan puede ser subdividido en documentos relacionados para así aclarar las necesidades de verificación para cada parte del sistema. Además, se han de documentar todos los criterios, técnicas y herramientas que se usarán en el proceso de verificación. En la normativa vienen descritas técnicas entre las cuales se ha de hacer una selección a implementar y justificarla para que cumpla los criterios que se expondrán más abajo.

El plan de Verificación del Software debe describir las actividades que se han de realizar para asegurar la corrección y coherencia con respecto a las entradas de cada fase. Para cada fase, se ha de demostrar que se cumplen los requisitos funcionales relativos a las prestaciones y la seguridad. Los resultados de cada verificación se han de presentar en los Informes de Verificación del Software según un formato definido en el Plan de Verificación del software.

3.2.2.2. INFORMES DE VERIFICACIÓN

Cada informe de verificación debe de documentar la identidad y configuración de aquello verificado, así como los nombres de los verificadores. También se han de mostrar los elementos que no cumplan las especificaciones. Así como los componentes, datos, estructuras y algoritmos que se adapten mal al problema y los errores y deficiencias detectados. Se ha de informar del cumplimiento o desvío del Plan de verificación del Software, en caso de desvío se ha de explicar si es crítico o no. Para concluir es necesario un resumen de los resultados de verificación.

3.3. CRITERIOS DE SELECCIÓN

Se definen unas tablas en la normativa para ayudar en la selección de técnicas y medidas apropiadas para el Nivel de Integridad del Software. En las tablas se etiquetan las técnicas con una 'M' (Mandatory. obligatoria), 'HR' (Highly Recommended, altamente recomendable), 'R' (Recommended, recomendable), '-' (sin recomendación) y 'NR' (Not Recommended, no recomendable) para definir cuan importantes son estas técnicas para cada uno de los niveles de integridad del Software (SIL- Software Integirty Level).

- '**M**' Este símbolo significa que el uso de esta técnica es obligatorio.
- '**HR**' La técnica que tenga este símbolo es altamente recomendable para su nivel de integridad del software. Si no se hace uso de esta técnica o medida, se debe especificar que técnicas o medidas alternativas se han usado en su lugar y explicar su uso en Plan de Garantía de Calidad del Software.

- '**R**' Este símbolo indica que la técnica es recomendable para el nivel de integridad del software. Es un nivel de recomendación menor que 'HR' y se puede hacer una combinación de estas técnicas para formar parte de un paquete de técnicas.
- '**-** ' Este símbolo indica que no hay recomendación a favor o en contra de su uso.
- '**NR**' Las técnicas o medidas que muestren este símbolo NO son recomendables para ese nivel de integridad del software. Si fuera usada se debe justificar su uso en el Plan de Garantía de Calidad del Software.

Existen ciertos requisitos para seleccionar las técnicas y medidas de las tablas descritas en la normativa. Si una técnica marcada como altamente recomendable (HR) no es utilizada, deberá explicarse él porque no se ha usado y las técnicas alternativas usadas. Esto ha de venir descrito en el plan de Garantía de Calidad del Software. Esto no será necesario si se utiliza una combinación de técnicas homologada indicada en las tablas de la normativa. Pero será necesario demostrar que se han aplicado las técnicas seleccionadas de forma correcta.

Si lo que se propone es la utilización de una técnica que no aparezca en las tablas, se deberá demostrar su idoneidad para cumplir con los objetivos del apartado en concreto al cual se va a aplicar, además de registrarlo en el Plan de Garantía de Calidad del software.

Para demostrar la conformidad de un apartado en particular con los requisitos y sus técnicas y medidas se deberá de verificar mediante la inspección de los documentos requeridos por la normativa. Así como tener en cuenta otras pruebas objetivas, auditorias y pruebas de ensayos cuando proceda.

3.4. TÉCNICAS Y MEDIDAS

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Demostración Formal	D1	-	R	R	HR	HR
2. Análisis Estático	Tabla 9	-	HR	HR	HR	HR
3. Análisis y Ensayos Dinámicos	Tabla 5	-	HR	HR	HR	HR
4. Métricas	D2	-	R	R	R	R
5. Trazabilidad	D3	R	HR	HR	M	M
6. Análisis Estático de los Errores de Software	D4	-	R	R	HR	HR
7. Cobertura del Ensayo para el Código	Tabla 10	R	HR	HR	HR	HR
8. Ensayos Funcionales/ Ensayos de caja negra	Tabla 6	HR	HR	HR	M	M
9. Ensayos de las Prestaciones	Tabla 8	-	HR	HR	HR	HR
10. Ensayos de la Interfaz	D5	HR	HR	HR	HR	HR

Requisitos:

1) La combinación de técnicas homologadas aplicables a los Niveles 3 y 4 de Integridad de Seguridad del Software es 3, 5, 7, 8 y una seleccionada entre la 1, 2 o 6.
2) Las combinaciones de técnicas homologadas aplicables a los Niveles 1 y 2 de Integridad de Seguridad del Software es la 5 junto con una seleccionada entre la 2, 3 u 8.

NOTA 1 Las técnicas/medidas 1, 2, 4, 5, 6, y 7 están dirigidas a actividades de verificación.
NOTA 2 Las técnicas/medidas 3, 8, 9 y 10 están dirigidas a ensayos.

Tabla 1 Verificación y Ensayos

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Ensayos Funcionales/Ensayos de Caja Negra	Tabla 6	HR	HR	HR	HR	HR
2. Ensayos de Prestaciones	Tabla 8	-	R	R	HR	HR

Tabla 2 Integración

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Ensayos de las Prestaciones	Tabla 8	-	HR	HR	M	M
2. Ensayos Funcionales/Ensayos de Caja Negra	Tabla 6	HR	HR	HR	M	M
3. Modelado	Tabla 7	-	R	R	R	R

Requisitos:

1) La combinación de técnicas homologadas aplicable a los Niveles 1 y 2 de Integridad del Software es 1 y 2.

Tabla 3 Ensayos de Software en Conjunto

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Análisis Estático de Software	Tabla 9	R	HR	HR	HR	HR
2. Análisis Dinámico de Software	Tablas 5 y 6	-	R	R	HR	HR
3. Diagramas de Causa-Efecto	D6	-	R	R	HR	HR
4. Análisis por Árbol de Eventos	D7	R	R	R	R	R
5. Análisis de los Efectos de los Errores de Software	D4	-	R	R	HR	HR
Requisitos: 1) Se deben seleccionar una o más de estas técnicas para satisfacer el Nivel de Integridad del Software utilizado.						

Tabla 4 Técnicas de Análisis del Software

3.4.1. TABLAS DETALLADAS

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Ejecución de Casos de Ensayo a partir del Análisis de Valores Límite	D8	HR	HR	HR	HR	HR
2. Ejecución de Casos de Ensayo a partir de la Suposición de Errores	D9	R	R	R	R	R
3. Ejecución de Casos de ensayo a partir de la Inserción de Errores	D10	-	R	R	R	R
4. Modelado de las Prestaciones	D11	-	R	R	HR	HR
5. Ensayos de Clases Equivalencia y Particiones de Entradas	D12	R	R	R	HR	HR
6. Ensayos Basados en la estructura	D13	-	R	R	HR	HR
Requisitos: 1) El análisis de los casos de ensayo se realiza a nivel del subsistema y se basa en la especificación y/o en la especificación y el código.						

Tabla 5 Análisis y Ensayos Dinámicos

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Ejecución de Casos de Ensayo a partir de Diagramas de Causa Efecto	D6	-	-	-	R	R
2. Prototipado/Animación	D14	-	-	-	R	R
3. Análisis de los Valores Límite	D8	R	HR	HR	HR	HR
4. Ensayos de Clases de Equivalencia y de Particiones de Entradas.	D12	R	HR	HR	HR	HR
5. Simulación de Procesos	D15	R	R	R	HR	HR
Requisitos: 1) La compleción de la simulación dependerá del alcance del nivel de integridad de seguridad del software, de la complejidad y de la aplicación.						

Tabla 6 Ensayos Funcionales/Ensayos de Caja Negra

TÉCNICA/MEDIDAD	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Modelado de datos	R	R	HR	HR	HR
2. Diagramas de Flujo de Datos	-	R	R	HR	HR
3. Diagramas de Flujo de Control	R	R	R	HR	HR
4. Máquinas de Estados Finitos o Diagramas de Transición de Estados	-	HR	HR	HR	HR
5. Redes de Petri Temporizadas	-	R	R	HR	HR
6. Tablas de Decisión / Tablas de Verdad	R	R	R	HR	HR
7. Métodos Formales	-	R	R	HR	HR
8. Modelado de las Prestaciones	-	R	R	HR	HR
9. Prototipado /Animación	-	R	R	R	R
10. Diagramas de Estructura	-	R	R	HR	HR
11. Diagramas de Secuencia	R	HR	HR	HR	HR
Requisitos: 1) Se debe definir y utilizar directrices de modelado. 2) Se debe seleccionar al menos una de las técnicas HR					

Tabla 7 Modelado

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Ensayos de Avalancha / Ensayos de Sobrecarga	D16	-	R	R	HR	HR
2. Tiempo de Respuesta y Limitaciones de Memoria	D17	-	HR	HR	HR	HR
3. Requisitos de las prestaciones	D18	-	HR	HR	HR	HR

Tabla 8 Ensayos de las prestaciones

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Análisis de los Valores Límite	D8	-	R	R	HR	HR
2. Listas de Comprobación	D19	-	R	R	R	R
3. Análisis de Flujos de Control	D20	-	HR	HR	HR	HR
4. Análisis de Flujos de Datos	D21	-	HR	HR	HR	HR
5. Suposición de Errores	D9	-	R	R	R	R
6. Revisión del Proyecto Estructurado/Revisión del Diseño	D22	-	HR	HR	HR	HR

Tabla 9 Análisis Estático

TÉCNICA/MEDIDAD	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Instrucción	D13	-	HR	HR	HR	HR
2. Rama	D13	-	R	R	HR	HR
3. Condición Compuesta	D13	-	R	R	HR	HR
4. Flujos de Datos	D13	-	R	R	HR	HR
5. Ruta	D13	-	R	R	HR	HR

Requisitos:

1) Para cada SIL, se debe desarrollar una medida cuantificada de cobertura para los ensayos realizados. Esto puede servir para apoyar la opinión sobre la confianza adquirida en los ensayos y la necesidad de técnicas adicionales.
2) Para los SIL 3 o 4, la cobertura de los ensayos a nivel componente debería medirse según las técnicas:
   - 2 y 3; o
   - 2 y 4; o
   - 5
   - O se debería medir la cobertura de ensayo a nivel de integración de acuerdo con una o más de las técnicas 2, 3,4 o 5.
3) Se pueden usar otros criterios de cobertura de ensayo, siempre que puedan justificarse. Dichos criterios dependerán de la arquitectura del software y del lenguaje de programación.
4) Se debe demostrar la corrección de los códigos que no se puedan someter a ensayo utilizando una técnica adecuada, por ejemplo, el análisis estadístico.

Tabla 10 Cobertura de los Ensayos para los Códigos

3.4.2. DEFINICIONES

D1. Demostración Formal. Su objetivo es probar la corrección de un programa o modelo sin ejecutarlo haciendo uso de reglas y modelos teóricos y matemáticos. Se establecen un número de aserciones en distintos puntos del programa que son usadas como pre y post condiciones de varias rutas del programa. La prueba consiste en demostrar que el programa transfiere las precondiciones a las postcondiciones de acuerdo a un conjunto de normas lógicas y que el programa termina.

D2. Métricas. Su objetivo es predecir atributos de los programas basados en propiedades propias del software en vez de calculadas en base a su desarrollo o historial de ensayos. Evalúan propiedades estructurales del software y las relacionan con el atributo deseado como por ejemplo la complejidad. Se necesitan herramientas software para evaluar la gran mayoría de medidas.

D3. Trazabilidad. Su objetivo es garantizar que se puede demostrar que todos los requisitos se han cumplido de forma adecuada y que no se ha introducido material no trazable. La trazabilidad de los requisitos se debe tener en cuenta para la validación del sistema y se deben de proporcionar medios para demostrarla durante todas las fases del ciclo de vida.

D4. Análisis de los Efectos de los Errores del Software. El objetivo es identificar la criticidad de los componentes software, demostrar la robustez del software y proponer medios para detectar errores de software.

D5. Ensayos de la Interfaz. Demostrar que los interfaces de los subprogramas no contienen errores que conlleven a fallos en una aplicación particular del software, así como detectar todos los errores que son relevantes.

D6. Diagramas de Causa Efecto. Pretende modelar, en forma de diagrama, la secuencia de eventos que pueden llevarse a cabo como consecuencia que una combinación de eventos básicos. En estos diagramas, empezando en un evento crítico, se dibuja un gráfico en dirección hacia atrás y otro hacia delante. En la dirección hacia atrás, es equivalente a un árbol de error con el evento crítico causante en la parte más alta. En la dirección hacia abajo, se identifican las posibles consecuencias que podrían producirse a partir del evento crítico.

D7. Análisis por Árbol de Eventos. Modelar en forma de diagrama, la secuencia de eventos que puede desencadenarse en un sistema después de un evento inicial, y como resultado, indicar las consecuencias graves que puede tener.

D8. Análisis de los Valores Límite. Es común que los errores del software se encuentren en los valores límite o frontera de los parámetros. Para ello las entradas se dividen en varias categorías de entradas y los ensayos deben cubrir los extremos de cada una de las categorías. Normalmente, los valores límites de las entradas tienen una correspondencia directa con los extremos de los rangos de los valores de salida. Por lo que también se debería escribir ensayos que forzaran los valores límites de salida o que los exceda siempre que esto fuera posible.

D9. Suposición de Errores. Eliminar errores comunes de programación gracias a la experiencia realizando ensayos, el conocimiento y curiosidad sobre el sistema bajo ensayo, añadiendo nuevos casos de prueba no calificados al conjunto de casos de prueba que se había diseñado.

D10. Inserción de Errores. Se pretende saber si el conjunto de casos de prueba es adecuado. Para ello se introducen algunos tipos de error conocidos en el programa y este es ejecutado con el caso de prueba con las condiciones necesarias. Si sólo se detectan algunos de los errores introducidos en el programa, el caso de prueba no es adecuado. Esta técnica da una aproximación del ratio de errores reales que existen en el código y los que son descubiertos basándose en los errores insertados que se han detectado y la cantidad total de errores introducidos.

D11. Modelado de las Prestaciones. Garantiza que la capacidad de trabajo del sistema es suficiente para cumplir los requisitos especificados.

D12. Clases de Equivalencia y Particiones de Entrada. Siempre se quiere testear al software de manera adecuada pero haciendo uso del mínimo de datos de prueba. Para conseguir esto, los datos de los ensayos, son obtenidos seleccionando las particiones del dominio de entrada necesarias para someter a ensayo el software. Este método se basa en la relación de equivalencia de las entradas, que determinan particiones del dominio de entrada. Siempre se pretende cubrir todos los subconjuntos de entradas y se toma como mínimo un test para cada clase de

equivalencia. Las particiones de las entradas se hacen principalmente con las dos siguientes técnicas.

- Clases de equivalencia definidas en la especificación. Puede establecerse en la especificación el conjunto de valores de entradas que han de ser tratados de la misma forma, u orientado a la salida, en el que se definen el conjunto de valores que conllevan a un mismo resultado funcional.
- Clases de equivalencia definidas por la estructura del programa. Las clases de equivalencia son resultado de un análisis estático, por ejemplo un conjunto de valores ejecutan la misma ruta del programa.

D13. Ensayos Basados en la Estructura. Aplicar casos de prueba que ejecuten ciertos subconjuntos de la estructura del programa. Basándose en el análisis del programa, se seleccionan los datos de entrada con el objetivo de que la mayor cantidad de elementos del código sean ejecutados. Los elementos del programa pueden ser ejecutados de forma más o menos rigurosa.

- Sentencias. Es el test menos riguroso, ya que se pueden ejecutar todas las sentencias de un programa sin someter a ensayo las distintas ramas de las sentencias condicionales.
- Ramas. Las ramas de cada una de las sentencias de condición deberían de ser comprobadas. Aunque puede ser imposible para ciertos códigos.
- Condiciones compuestas. Se ejercitan todas las condiciones de una condición compuesta para cada una de las ramas. (AND /OR).
- Flujo de datos. Las rutas de ejecución se eligen en base al uso de los datos, es decir, una ruta donde la misma variable sea escrita y leída.
- Ruta completa. Ejecutar todas las posibles rutas del código. Este tipo de test es normalmente inalcanzable.

D14. Prototipado / Animación. Comprobar la viabilidad de implementación de cierto sistema teniendo en cuenta las restricciones o requisitos existentes. Pudiendo comunicar al cliente la interpretación de quien ha especificado el sistema con el objetivo de localizar malos entendidos.

D15. Simulación de procesos. Comprobar la funcionalidad del software junto con su interfaz hacia el exterior. Lo que se pretende es crear un sistema que imite el comportamiento del sistema a controlar con propósitos de ensayo. La simulación puede ser solo software o hardware y software.

D16. Ensayos de Avalancha / Ensayos de Sobrecarga. Someter con una sobrecarga de trabajo al objeto bajo prueba para demostrar que dicho objeto podrá trabajar de manera fluida con una carga de trabajo normal.

D17. Tiempo de Respuesta y Limitaciones de Memoria. El sistema ha de cumplir con los requisitos de memoria y tiempo de respuesta.

D18. Requisitos de las prestaciones. Establecer que se han cumplido los requisitos de las prestaciones de un software. Se realiza un análisis tanto de las Especificaciones de

Requisitos del Software, como del Sistema, para identificar todos los requisitos generales, específicos, explícitos e implícitos de las prestaciones.

D19. Listas de Comprobación. Se trata de una serie de preguntas que ha de contestar la persona encargada de completar la lista para estimular la evaluación crítica de todos los aspectos del sistema en lugar de imponer requisitos específicos.

D20. Análisis de Flujos de Control. Detectar estructuras potencialmente incorrectas o débiles del programa. Identifica áreas sospechosas del código en las cuales no se siguen buenos métodos de programación.

D21. Análisis de Flujos de Datos. Este análisis combina la información obtenida por los análisis de flujos de control con información sobre que variables son escritas o leídas en diferentes partes del código.

D22. Revisión del Proyecto Estructurado/Revisión del Diseño. El objetivo de estas revisiones es detectar errores en cualquier producto durante su desarrollo lo más pronto posible y económicamente menos costoso.

4. PLC

Como principal objetivo en este trabajo, se pretende adaptar los procesos de test y verificación para software PLC, en la empresa Stadler Rail Valencia, a lo establecido en la normativa ferroviaria presentada. Por tanto, es necesario saber unos principios básicos de los Controladores Lógicos Programables (Programmable Logic Controller-PLC) y las características de los lenguajes PLC.

Antes de entrar en detalle, la normativa, como se ha mencionado en el apartado anterior, es consciente de la importancia de PLC en los vehículos ferroviarios y por tanto todo lo en ella descrito es aplicable independientemente de la arquitectura de los sistemas, especificando los controladores lógicos programables.

4.1. PRINCIPIOS BÁSICOS

Según la definición de la Asociación Nacional de Fabricantes Eléctricos de los Estados Unidos, un PLC es un dispositivo digital electrónico con una memoria programable para el almacenamiento de instrucciones, permitiendo la implementación de funciones específicas como ser: lógicas, secuenciales, temporizadas, de conteo y aritméticas; con el objeto de controlar máquinas y procesos

En otras palabras PLC es un dispositivo sólido o equipo electrónico diseñado para automatizar y controlar procesos industriales secuenciales. Capaces de controlar el funcionamiento de máquinas, a las que pueden estar asociados, que llevan a cabo procesos de producción.

Debido a su gran utilización en el ámbito industrial son capaces de resistir un gran rango de temperaturas, el ruido eléctrico, vibraciones e incluso golpes. Estos dispositivos pueden ser programados para controlar casi cualquier tipo de máquina y están diseñados para ello ya que cuentan con una gran variedad de entradas y salidas tanto digitales como analógicas. Los programas que controlan la funcionalidad, se almacenan en memorias no volátiles. Esta memoria forma parte de la estructura básica de los PLC que están compuestos por los siguientes elementos mostrados en la imagen.

- Procesador
- Memorias
- Alimentación
- Interfaces de salidas
- Interfaces de entradas

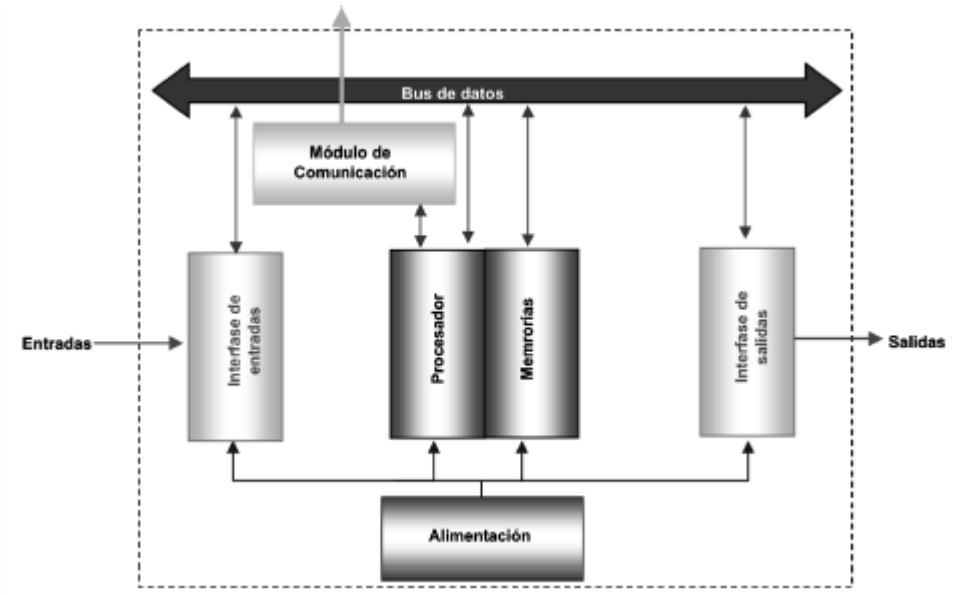

Ilustración 1 Estructura interna PLC

Procesador. Encargado de ejecutar el programa desarrollado por el usuario y administrar la comunicación entre los distintos elementos del PLC, el dispositivo de programación y la memoria o los interfaces de salida con el microprocesador. Para poder llevar a cabo todas estas funciones y otras, el procesador dispone un sistema operativo, no es accesible por los usuarios, almacenado en una memoria no volátil.

Memoria. Es esencial para el funcionamiento de los PLC que se pueda almacenar y acceder a información, con este objetivo estos dispositivos suelen contar con distintas memorias, que han de ser capaces de almacenar datos del proceso y datos de Control.

Los datos de proceso son todos aquellos relacionados con las señales de entrada y salida, las variables internas. Este tipo de datos se almacena en una memoria de datos que, en definitiva contiene la información sobre el estado del dispositivo, guardando información tanto de las interfaces como del microprocesador.

Los datos de control son las instrucciones del usuario, es decir, el programa que el usuario a cargado en el PLC, y la configuración del dispositivo, es decir su sistema operativo que está grabado en la ROM. El programa precargado por el usuario se almacena en la memoria de usuario. Estas memorias suelen ser de tipo RAM, que hacen uso de pilas ya que pierden la información al quedarse sin alimentación o de tipo EEPROM que no pierde la información en ausencia de alimentación.

Dispositivos de entrada y salida. Estos dispositivos son aquellos que intercambian señales con el PLC. Por ejemplo sensores para conocer condiciones del entorno, temperatura, presión, etc. Por otra parte otro tipo de dispositivos que responden a señales del PLC cambiando el entorno.

Los distintos dispositivos pueden trabajar con distintos interfaces de entrada o salida generalmente son digitales o analógicas.

Las interfaces digitales son también llamadas binarias y solo pueden tomar dos estados, “0” o “1”. Mientras que las interfaces analógicas admiten valores tensión o corriente intermedios dentro de un rango que puede ser 4-20mA, 0-5VDC o 0-10VDC, convirtiéndola en un número.

Alimentación. Es la fuente de alimentación que proporciona las tensiones necesarias para el correcto funcionamiento de todos los elementos del PLC.

4.2. FUNCIONAMIENTO DE UN PLC

Los PLCs tienen un funcionamiento cíclico y secuencial, las operaciones que lleva a cabo van una tras otra y se repiten continuamente mientras el autómata está conectado. El tiempo que tarda en llevar a cabo cada ciclo se conoce como tiempo de barrido o “Scan Time”. Durante este tiempo se llevan a cabo distintas acciones, una secuencia típica podría ser la siguiente.

En primer lugar, cuando el PLC se conecta a tensión, lleva a cabo un **autodiagnóstico** en el que comprueba todos sus circuitos y en caso de algún error emitiría alguna señal luminosa indicando el tipo de error.

Posteriormente el PLC hace una **lectura del registro de las entradas,** determina si están encendidas o apagadas y revisa todas las entradas. Lo que hace entonces es **crear una imagen de los estados de las entradas en memoria** para ser utilizadas más adelante.

El siguiente paso es la **lectura y ejecución del programa**. La CPU ejecuta el programa realizado por el usuario, esta ejecución se realiza instrucción a instrucción y en el orden que se determinó. Es posible, debido a que ya se leyeron las entradas y creado una imagen de ellas, tomar decisiones en base a los valores guardados. Estas decisiones en última instancia se referirán a los valores de las salidas y que serán guardados en registros para su uso futuro.

Una vez ejecutado el programa, el PLC suele **atender a las comunicaciones.** Y finalmente **actualiza el registro de las salidas** de forma simultánea en base a los valores obtenidos en la fase de ejecución del programa.

De forma general los fabricantes dan un tiempo de barrido equivalente al ejecutar 1K de instrucciones de lógica booleana. Pero como no está normalizado el tipo de instrucciones que se han de utilizar para medir el tiempo de barrido, puede suceder que este varíe respecto a cada PLC. Además, es posible que cada controlador lógico programable, ejecute un cierto tipo de instrucciones más rápido que otro o al revés.

4.3. LENGUAJES PLC

La gran variedad de fabricantes de dispositivos PLC y la creciente complejidad de los autómatas, derivó en una gran variedad de distintos lenguajes de programación para escribir programas. En 1992 se estableció como estándar la normativa IEC 1131-3 que regula la programación para PLC. Define los lenguajes de programación estandarizados y para cada uno de ellos regula su sintaxis y representación gráfica de los objetos, la estructura de programas y la declaración de variables. Los lenguajes estandarizados son los cinco siguientes:

- Ladder Diagram (LD), Diagrama de Contactos o Lógica de Escalera.

Está compuesto por elementos gráficos organizados en redes conectados por barras de alimentación. Los símbolos básicos están normalizados y empleados por todos los fabricantes (boninas, contactos, funciones…) y se pueden usar elementos de control del programa como salto, return, etc.

- Funcitional Block Diagram (FBD), Diagrama de Bloques Funcional.

Lenguaje gráfico que da la posibilidad de programar elementos en forma de bloques enlazados entre ellos. Forman una red en la que la salida de un bloque funcional puede estar conectada a la entrada de otro.

- Instruction List (IL), Lista de Instrucciones.

Está formado por un listado de instrucciones, en la que cada una debe empezar en una línea nueva. Cada instrucción está compuesta por un operador y uno o más operandos separados por comas.

- Structured Text (ST) Texto Estructurado.

Es un lenguaje de alto nivel estructurado por bloques con una sintaxis parecida al PASCAL. Permite construir sentencias complejas que manejen un amplio rango de variables de diferentes tipos de datos. Da la posibilidad de manejo de datos de tiempo muy importantes en la industria además de poseer bucles, sentencias de control y funciones matemáticas.

- Sequential Function chart (SFC), Diagrama Secuencial de Funciones.

Lenguaje gráfico que representa en forma de diagrama las secuencias del programa. Es particularmente útil para describir funciones de control secuencial. Los elementos básicos son pasos y transiciones. Debido a que los procesos industriales funcionan en forma de pasos, el SFC es una forma lógica de especificar y programar el más alto nivel de un programa para PLC.

5. TESTEO Y VERIFICACIÓN EN STADLER RAIL

Una vez analizada la normativa EN 50128, y sobre todo sus apartados relativos a las pruebas del software y verificación, el siguiente paso será definir las actividades llevadas a cabo actualmente por la empresa Stadler Rail Valencia en estos ámbitos. El objetivo de este apartado consiste en determinar la situación actual de la empresa con respecto a la normativa para tener una idea de las mejoras necesarias para ajustarse lo más fielmente a esta.

Todos los procesos de pruebas de ensayo y verificación llevados a cabo en los desarrollos de software PLC en Stadler Rail, son llevados a cabo por la herramienta desarrollada por ellos mismos SoftPro.

5.1. SOFTPRO

SoftPro juega un papel importante en el ciclo de vida del código que se ejecuta dentro de los vehículos ferroviarios desarrollados por Stadler Rail Valencia. Entre las principales características de la herramienta destacan las siguientes.

Conexión directa con DOORS un repositorio de requisitos del cual puede descargarse información, así como cargar contenido, lo que permite una **trazabilidad** de los requisitos, ya que es posible cargar casos de prueba para asociarlos a requisitos.

Tiene una conexión con una base de datos interna en la que almacena todos los objetos (test, módulos, acciones, etc.) con los que trabaja.

Compatibilidad con PLC, todos los ficheros generados desde SoftPro siguen el formato National Instruments TestStand para poder ser ejecutados y trabajar con los PLC sobre los cuales se van a ejecutar los casos de prueba que se definan.

Permite **control de versiones de software,** lo que mantiene una relación entre las versiones del software con sus librerías y módulos. Esto es gracias a la conexión con DOORS.

Como rasgo principal de la herramienta permite la **generación de casos de prueba**, lo cual incluye gestión sobre los componentes del test como funciones, pasos, acciones y variables.

Gracias a su conexión con DOORS también permite una **gestión de la documentación** relacionada con la especificación de los test, su verificación y sus informes. Toda esta documentación puede ser creada y gestionada desde la propia herramienta.

Además, incluye gestión de anomalías y notas de **cambio**, permitiendo tener localizados las anomalías que aparecen en los proyectos y las notas de cambio que pueden ser exportadas como un Informe de Mantenimiento del Software.

Con relación a la gestión de casos de prueba, permite generar test de capa baja importando toda la información necesaria para esto desde ficheros .csv,

También es capaz de **generar toda la combinatoria de los valores** de las variables de entrada de un módulo para ser comprobadas más tarde.

Por último, dispone de gestión de usuarios, definiendo diferentes niveles de usuario, cada uno de los cuales tiene permisos diferentes por lo que no todas las funcionalidades están accesibles para todos los usuarios.

6. HERRAMIENTAS EXISTENTES

La aplicación de la normativa en su totalidad es un proceso complejo y que requiere un gran estudio e implantación de nuevas herramientas o mejoras de las ya desarrolladas para cumplir con todos los requisitos. Tras el análisis de las fases de test y verificación implementadas por Stadler Rail, se detectó una falta de adaptación a la normativa EN 50128 en cuanto a la verificación y testeo de software PLC. Por lo que es necesario buscar soluciones que lleven a cabo la mayor cantidad de técnicas de verificación y testeo para cumplir con los estándares europeos. Para ello durante esta fase del trabajo, se realizará un análisis del estado del arte de las herramientas de verificación y test para software PLC.

Pese a la gran importancia y criticidad, para la seguridad de grandes procesos industriales, que puede recaer sobre software PLC, este campo no presenta grandes avances respecto a los casos de prueba y verificación. Una causa de la falta de investigación en este ámbito podría ser la ausencia de normativa que regulara estos procesos. Un ejemplo de esto es la normativa objeto de este trabajo, EN 50128, que es relativamente nueva, ya que se puso en vigor en 2011. Por tanto, gran parte de los proyectos, documentación y herramientas relacionadas con esto, encontradas en este estudio exploratorio, están recién comercializadas, o todavía en fases de desarrollo.

Como resultado de la búsqueda de herramientas o documentos relacionados con las fases de software PLC bajo estudio en este trabajo, se han encontrado algunos artículos de investigación que han desembocado en herramientas de verificación. Exactamente el resultado ha sido cuatro herramientas relacionadas con los casos de prueba y verificación para software PLC en procesos industriales.

6.1. PLC VERIF

Sitio web: https://wikis.web.cern.ch/wikis/display/EN/PLCverif/

Esta es una herramienta desarrollada por el CERN con el objetivo de aplicar “model checking” a software PLC. Plantean la necesidad de esta herramienta para cubrir las deficiencias que presenta los casos de prueba. Entre ellas, la imposibilidad de demostrar la ausencia de fallos, sólo la capacidad de encontrarlos y que los test solo pueden comprobar unos valores de salida para un conjunto de valores de entrada, no comprobar hechos generales.

“Model checking” es una técnica de verificación formal que obtiene un modelo matemático del sistema a comprobar y un conjunto de requisitos formalizados. Con estos dos objetos, el “model checker” es capaz de comprobar todas las posibles ejecuciones del programa y ver si todos los requisitos se cumplen en todas las ejecuciones. En caso de que no se cumplan, se generan contraejemplos en los que una secuencia de entradas demuestra el incumplimiento de alguna de las condiciones.

Las características principales de la herramienta PLC Verif son las siguientes.

Generación de código PLC. PLC Verif incluye un editor para programas PLC, que pretende soportar los lenguajes definidos en el estándar IEC 61131-3. El editor además cuenta con características presentes en editores de código moderno como, por ejemplo, el resaltado de algunas sintaxis, autocompletado, etc. Permite tanto escribir el código como importarlo de uno existente.

Definición de los requisitos. De manera similar a los casos de prueba, una verificación en concreta ha de ser definida por el usuario. La información necesaria para esto es la información general de la verificación, el código fuente a verificar, el requisito a comprobar, y la herramienta de "model checker" que se va a usar.

Un paso importante como se ha mencionado antes es la formalización de los requisitos para ajustarlos al "model checker". Normalmente estos requisitos se formalizan usando CTL (Computational Tree Logic, Lógica computacional en árbol) o LTL (Linear Temporal Logic). El inconveniente es que ambas técnicas son difíciles de usar incluso para expertos. Para contrarrestar esta dificultad, PLCVerif ha introducido un conjunto de requisitos predefinidos escritos en ingles con unos huecos a rellenar que se traduce a LTL o CTL usando la estructura escrita en inglés. Por tanto, la responsabilidad del usuario es seleccionar el patrón adecuado para el requisito y rellenar los huecos.

Proceso de verificación. Una vez cargado el código y formalizados los requisitos, el proceso de verificación es completamente automático. Se transforma el código en PLC a un modelo intermedio, los requisitos se formalizan basados en la información introducida por el usuario. Después el modelo intermedio se reduce usando distintas técnicas con el objetivo de simplificar el modelo sin variar su significado eliminando cualquier parte que no sea necesaria para la evaluación del requisito actual. Por lo que un único modelo se producirá para cada requisito.

Es entonces cuando el modelo intermedio simplificado y el requisito se traducen al lenguaje especifico del "model checker" utilizado. Finalmente se ejecuta el "model checker", que almacena todas sus salidas, incluidos los errores, en PLCVerif para mostrárselo al usuario.

Evaluando los resultados. Los contraejemplos producidos suelen ser complejos, muy grandes y difíciles de entender, ya que los nombres de las variables pueden ser distintos a los del código PLC original debido a la automatización de los modelos. El resultado de esta fase es un informe de verificación producido automáticamente que contiene los detalles del caso verificado, los resultados y los errores y contraejemplos simplificados para que sean entendibles.

6.2. REFLEX STUDIO

Sitio Web: http://www.en.artics.fr/#!products/c1rfn

REFLEX Studio es una plataforma innovadora de diseño de software de control. Permite diseñar, desarrollar, simular, testear y validar aplicaciones de control conformes al estándar IEC 61131-3 independientemente del hardware que usen. Genera código

fuente seguro y fiable para cualquier tipo de controlador industrial como controladores lógicos programables y controladores embebidos.

El conjunto de herramientas que conforma REFLEX Studio, abre la puerta a un nuevo sistema más eficiente y económico para la programación y validación de sistemas de control industrial ya que, se pretende desarrollar programas de control independientes del hardware seguro de forma nativa. Hacer una verificación y validación de los programas haciendo uso de casos de prueba y simulación. Además de reducir el tiempo y esfuerzo en las fases de test, validación y mantenimiento. Las siguiente cinco herramientas son las que conforman REFLEX Studio.

6.2.1. REFLEX DEVELOPMENT

REFLEX Development es un entorno de desarrollo que ofrece un editor compatible con el estándar IEC 61131-3 para diseñar y desarrollar código de control seguro por naturaleza independientemente del hardware objetivo. Entre sus características cabe destacar que su editor está basado en el lenguaje REFLEX que permite reusar software. Además, incluye editores gráficos para los lenguajes estándarizados FBD, LD y SFC totalmente compatibles con el lenguaje REFLEX. Incluye un potente compilador que detecta automáticamente típicos errores de programación como variables no inicializadas, divisiones por cero, índices fuera de vectores y verificar formalmente algunas reglas definidas por el usuario. Junto todo esto también cuenta con un control de versiones y gestor de documentación.

6.2.2. REFLEX VERIFICATION

Reflez Verification permite verificar formalmente los components y programas desarrollados con REFLEX Studio. Es una herramienta para desarrollar software seguro que puede ser utilizada durante todas las fases del desarrollo. El usuario es capaz de definir reglas de programación y que se compruebe si se cumplen o no en los componentes y programas desarrollados. Además de esto, haciendo un análisis matemático del programa, esta herramienta es capaz de detectar errores de programación, distintos a los definidos por las normas del usuario y garantizar la ausencia de errores cuando se ejecuta el programa.

6.2.3. REFLEX VALIDATION

REFLEX Validation sirve para definir y automatizar escenarios de pruebas para validar el comportamiento de los programas. Se pueden definir fácilmente los casos de prueba con el editor que tiene la herramienta, pudiendo generar desde casos de prueba muy simples hasta casos altamente complejos. Es posible encadenar dichos casos de prueba y que se ejecuten de forma automática. Cuando se han ejecutado todos los casos de prueba, se genera un informe para cada uno de los escenarios, también se puede ver la evolución de las variables durante el test.

Para poder controlar los procesos durante el testeo de los distintos escenarios de los casos de prueba, la aplicación se ejecuta en el PC (con una o varias máquinas virtuales síncronas) similar a lo que podría ocurrir en un PLC.

6.2.4. REFLEZ SIMULATION

REFLEX Simulation ofrece un entorno de simulación y depuración donde ejecutar virtualmente, testear y validar las aplicaciones realizadas con REFLEX Estudio usando controladores que se ejecutan en el PC. Permite simular sistemas completos de manera virtual ejecutando de manera paralela los programas y modelos dinámicos del sistema en el PC. Es posible también, probar y depurar un paso específico ya que se permite pausar la simulación de la aplicación, acelerarla o ralentizarla, así como restaurar el estado del sistema.

6.2.5. REFLEX GENERATION

REFLEX Generation permite, a partir de un programa de control genérico desarrollado con REFLEX Studio, generar aplicaciones específicas para distinto hardware objetivo, sobre todo controladores industriales como PLC. Es decir, genera código fuente para diferentes plataformas a partir de una aplicación REFLEX. Dispone de distintos generadores para distintos lenguajes del estándar IEC 61131-3 y otros propios de algunos fabricantes de PLC ampliamente usados.

La generación del código desde la aplicación en REFLEX, sigue esquemas formales para garantizar el cumplimiento de las condiciones de funcionamiento que se han establecido y asegura que las aplicaciones generadas a distintos lenguajes, se comportan igual que las desarrolladas en REFLEX.

6.3. ARCADE PLC

Sitio Web: https://arcade.embedded.rwth-aachen.de/

Arcade PLC es una herramienta para la verificación de programas para PLC. Empezó a desarrollarse en 2011 y su objetivo principal es ayudar a los desarrolladores a encontrar errores y verificar propiedades funcionales sin tener un amplio conocimiento sobre métodos formales. Proporciona una interfaz gráfica basada en eclipse, da soporte a diferentes lenguajes PLC y un acceso simple a funciones de análisis estático y “modle checkinng”.

Esta herramienta es un “model checker” como PLCVerif para controladores lógicos programables. Soporta como se ha dicho anteriormente, análisis estático y “model checking”, permite la especificación de requisitos en CTL y LTL. En sus resultados provee contraejemplos en caso de que no se cumplan los requisitos y la simplificación de estos. Por el momento, la herramienta es compatible con algunos lenguajes PLC estandarizados como texto estructurado y lista de instrucciones. Los programas son

traducidos a una representación intermedia para romper las dependencias con los distintos lenguajes PLC. Las ventajas que presenta esta herramienta frente a otros "model checkers" no específicos para PLC, es que soporta software PLC de forma nativa sin ningún tipo de preprocesamiento o transformación. Además, permite verificar programas compuestos por varios módulos escritos en diferentes lenguajes, algo bastante común en la realidad.

6.4. ITRIS AUTOMATION

Sitio Web: http://www.itris-automation.com/

La compañía francesa Itris Automation, ha desarrollado un conjunto de herramientas en el ámbito de desarrollo PLC. Su idea surge debido a unas necesidades de mercado de la programación en PLC que requiere una gran fiabilidad debido a las graves consecuencias que puede tener un fallo en un programa. Por ejemplo, detener una línea de montaje o incluso pérdidas humanas en caso de un fallo en un tren. Sin olvidarse del largo ciclo de vida de los programas PLC. Por lo que es necesario un conjunto de herramientas que permitan garantizar la fiabilidad y robustez de los programas con un coste razonable. El desarrollo de una forma económica, es un problema, ya que, en muchos casos, los nuevos dispositivos lógicos programables, con sus nuevos entornos de desarrollo, no son compatibles con los antiguos a pesar de que sean del mismo fabricante, por lo que los recursos existentes son difíciles de reutilizar.

La compañía, para enfrentarse a los retos existentes en el ámbito de los PLC, ha propuesto una solución innovadora para sus herramientas. Ofrecen una plataforma y un lenguaje independiente (GLIPS) de cualquier marco particular de cada PLC. Basándose en ese nuevo marco de trabajo, han desarrollado herramientas compatibles con los principales lenguajes de PLC que traducen al nuevo lenguaje desarrollado por ellos alcanzando un nuevo nivel de abstracción. De esta forma pueden traducir programas PLC en un lenguaje a otro o verificar la conformidad de estos respecto a ciertas reglas.

La solución que plantean puede ayudar en gran parte del proceso de desarrollo del software, desde la especificación hasta la validación funcional. Durante las primeras fases de desarrollo, el nuevo lenguaje de mayor abstracción, ofrece gran parte de sus beneficios. Por otra parte, en la segunda mitad de la V del desarrollo software, sus sistemas de verificación, como la interpretación abstracta, análisis estático y "model checking", proporciona grandes ventajas a los programas PLC.

Para cubrir todas estas necesidades han desarrollado tres herramientas que conforman un conjunto. PLC Checker, PLC Converter, PLC DocGen.

6.4.1. PLC CHECKER

Esta herramienta automáticamente analiza los programas PLC y verifica su conformidad con unas reglas generales o incluso si es necesario, con reglas específicas que el usuario ha definido para una industria o proceso en concreto.

6.4.2. PLC CONVERTER

Permite traducir, haciendo uso de las tecnologías explicadas anteriormente y de una forma eficiente reduciendo el tiempo y coste de conversión, programas de un lenguaje de PLC a otro.

6.4.3. PLC DocGen

Genera documentación, haciendo técnicas de ingeniería inversa, a partir de un código fuente PLC. Esto ayuda a entender y conocer la estructura de los programas gracias a los grafos de control y datos.

6.5. COMPARACIÓN DE HERRAMIENTAS.

Para tener una idea global de todas las herramientas respecto a las otras y frente a lo especificado en la normativa, en este punto se va a realizar una tabla comparativa de las herramientas con respecto a las técnicas y medidas especificadas en la norma europea. El objetivo es ver que técnicas pueden llevarse a cabo en cada una de las herramientas y cuáles no, también sirve para detectar aquellas técnicas y medidas que no se están llevando a cabo por ninguna herramienta y son requeridas por el estándar.

Para hacer esto de forma visual, en vez de crear una única tabla con todas las técnicas y medidas en las columnas, vamos a separarlo de forma que cada técnica que en la normativa estuviera especificada con una tabla detallada forme una tabla comparativa, en la que las columnas serán cada medida o técnica especifica de la técnica general. Es decir, análisis estático en la normativa está detallado en una nueva tabla con las siguientes técnicas, análisis de los valores límites, listas de comprobación, análisis de flujo de control, análisis de flujo de datos, suposición de errores y revisión del proyecto estructurado/ revisión del diseño. Por lo que en una tabla llamada análisis estático, estas técnicas serán las columnas y las herramientas serán las filas.

	Análisis Estático					
	Análisis de los Valores Límite	Listas de Comprobación	Análisis de Flujo de Control	Análisis del Flujo Datos	Suposición de errores	Revisión del Proyecto Estructurado
Itris Automation	SÍ	NO	SÍ	SÍ	SÍ	NO
REFLEX Studio	SÍ	NO	SÍ	SÍ	SÍ	NO
PLC Verif	NO	NO	NO*	NO*	NO	NO
ARCADE PLC	NO	NO	NO*	NO*	NO	NO
SoftPro	NO	NO	NO	NO	NO	NO

Tabla 11 Comparativa Análisis Estático

En la tabla anterior las herramientas PLC Verif y ARCADE PL tiene en algunas técnicas marcadas con un "NO*". Estas técnicas son el análisis de control de flujo y el análisis de datos. El significado del asterisco en este caso significa que a pesar que para aplicar el "model checking" si llevan a cabo estos análisis, no los presentan como una funcionalidad disponible para el usuario.

	Ánalisis Dinámico y Casos de Prueba					
	Ejecución de Casos de Ensayo a partir del Análisis de Valores Límite	Ejecución de Casos de Ensayo a partir de la Suposición de Errores	Ejecución de Casos de ensayo a partir de la Inserción de Errores	Modelado de las prestaciones	Ensayos de Clases de Equivalencia y Particiones de Entradas	Ensayos Basados en la Estructura
Itris Automation	NO	NO	NO	NO	NO	NO
REFLEX Studio	SÍ*	SÍ*	SÍ*	NO	SÍ*	SÍ*
PLC Verif	NO	NO	NO	NO	NO	NO
ARCADE PLC	NO	NO	NO	NO	NO	NO
SoftPro	SÍ*	SÍ*	SÍ*	NO	SÍ*	SÍ*

Tabla 12 Comparativa Análisis Dinámico

En cuanto a análisis dinámico y casos de prueba, vemos que solo dos de las herramientas implementan estas funcionalidades, REFLEX Studio y SoftPro. Aunque ambas están marcadas con un SÍ*, que en este caso significa que a pesar no presentar una funcionalidad expresamente que permita hacer esas medidas, la definición de casos de prueba se puede hacer de forma manual por lo que es posible definir ensayos para basados en cualquier tipo de análisis.

	Cobertura de los Ensayos para los Códigos				
	Instrucción	Rama	Condición Compuesta	Flujo de Datos	Ruta
Itris Automation	NO	NO	NO	NO	NO
REFLEX Studio	Dependerá del test especificado				
PLC Verif	SÍ	SÍ	SÍ	SÍ	SÍ
ARCADE PLC	SÍ	SÍ	SÍ	SÍ	SÍ
SoftPro	Dependerá del test especificado				

Tabla 13 Comparativa de cobertura

La cobertura para los "model checker" a pesar de no ejecutar casos de prueba, las verificaciones que hacen son para todas las posibles combinaciones de valores, por lo que la cobertura es total. Por otra parte, para las otras dos herramientas que permiten

definir casos de prueba, la cobertura de estos ensayos vendrá definida por la completitud del test.

	Ensayos Funcionales/Ensayos de Caja Negra				
	Ejecución de Casos de Ensayo a Partir de Diagramas de Causa Efecto	Prototipado/ Animación	Análisis de los Valores Límite	Ensayos de Clases de Equivalencia y de Particiones de Entrada	Simulación de Procesos
Itris Automation	NO	NO	NO	NO	NO
REFLEX Studio	SÍ*	NO	SÍ*	SÍ*	SÍ*
PLC Verif	NO	NO	NO	NO	NO
ARCADE PLC	NO	NO	NO	NO	NO
SoftPro	SÍ*	NO	SÍ*	SÍ*	NO

Tabla 14 Comparativa Ensayos Funcionales/Caja Negra

En este caso, para los casos de prueba funcionales o de caja negra, las herramientas que permiten definir casos de prueba, aunque no aporten la funcionalidad de definir un test en base a un diagrama o análisis, se puede hacer ya que dan la posibilidad de hacerlo de forma manual. Por otra parte, la herramienta SoftPro no permite simular procesos, pero en Stadler Rail disponen de un conjunto de PLCs que simula el comportamiento de un tren y ejecutando en ellos los casos de prueba definidos en SoftPro, consiguen simular procesos.

	Ensayos de las Prestaciones		
	Ensayos de Avalancha/ Ensayos de Sobrecarga	Tiempo de Respuesta y Limitaciones de Memoria	Requisitos de las Prestaciones
Itris Automation	NO	NO	NO
REFLEX Studio	SÍ*	NO	NO
PLC Verif	NO	NO	NO
ARCADE PLC	NO	NO	NO
SoftPro	SÍ*	NO	NO

Tabla 15 Comparativa Ensayos de las Prestaciones

En este apartado ha sido un poco más complicado determinar para la herramienta REFLEX Studio si tenía estas funcionalidades. Los ensayos de avalancha quizá se puedan conseguir concatenando casos de prueba de manera automática y aunque no presente esto como una funcionalidad, podría ser útil, al igual que pasa con SoftPro. Por otro lado, no se ha encontrado nada al respecto de las medidas de tiempo y memoria por lo que se ha supuesto que no lo implementa.

	Métodos Formales	Métricas	Trazabilidad	Análisis de los Efectos de los Errores del Software	Ensayos de la interfaz	Modelado
Itris Automation	NO	NO	SÍ	SÍ	NO	NO
REFLEX Studio	NO	NO	SÍ	SÍ	SÍ	SÍ
PLC Verif	SÍ	NO	NO	NO	NO	SI
ARCADE PLC	SÍ	NO	NO	NO	NO	SI
SoftPro	NO	NO	SÍ	NO	SÍ	NO

Tabla 16 Comparativa resto de técnicas

Finalmente, en esta tabla se muestran aquellas técnicas que no estaban definidas con otra tabla detallada. Podemos destacar que sólo, como es lógico, los "model checker" implementan métodos formales. La trazabilidad que es uno de las medidas más importantes para la normativa desde los SIL más bajos, en las herramientas viene incorporado, permitiendo generar documentación de los test o pruebas que se realicen asociadas a sus ensayos. En SoftPro además está ligado con una herramienta de toma de requisitos y esta trazabilidad es mayor aún. Ya que permite unir unos requisitos con los test y los informes que prueban que de verdad se cumplen.

7. SELECCIÓN DE TÉCNICAS Y MEDIDAS

Una vez conocido el proceso de pruebas de ensayo y verificación que se lleva a cabo en Stadler Rail Valencia, habiendo finalizado la búsqueda de herramientas existentes para PLC en este ámbito y realizado el estudio de la normativa europea. La siguiente fase tiene un objetivo claro, de todas las técnicas y medidas presentadas en la normativa, hacer la selección de técnicas más apropiadas para cumplir los requisitos del estándar europeo teniendo en cuenta la situación actual y las herramientas que ya existen. Es decir, definir que tareas o técnicas debería incorporar el proceso de test y verificación de Stadler Rail Valencia para cumplir la normativa, siempre teniendo en cuenta aquello que ya realizan y lo que podrían realizar con las herramientas ya existentes.

Para ello siempre que sea posible se seleccionará una combinación de técnicas de las propuestas como válidas en la normativa. En caso de que la normativa no especifique ninguna combinación de técnicas, se deberán llevar a cabo todas las que al menos estén marcadas como "R" (Recomendable) para el SIL que se esté aplicando.

La selección de técnicas y medidas que se muestre aquí no será algo inalterable, dependiendo el tipo de software quizá sea necesario aplicar unos métodos u otros, pero se pretende mostrar un listado de técnicas que cumpla la normativa y que sea factible de llevar a cabo con las herramientas existentes y que aproveche al máximo las tareas que se implementan en Stadler Rail Valencia para testear y verificar el software.

De las técnicas mostradas en la **tabla 1 "Verificación y Ensayos" para** SIL 1 y 2 la combinación de técnicas seleccionadas es la siguiente:

- Trazabilidad,
- Análisis Estático y
- Análisis y Ensayos Dinámicos

A pesar de que la combinación presentada en la tabla para este nivel, solo requiere trazabilidad y una más, la combinación de Análisis Estático y Análisis y Ensayos Dinámicos es muy interesante ya que se complementan muy bien además de que el Análisis Dinámico será obligatorio en otros partes de la normativa.

Continuando con la misma tabla, pero para los SIL 3 y 4, la combinación de técnicas será distinta y se muestra a continuación.

- Trazabilidad,
- Análisis y Ensayos Dinámicos,
- Cobertura del Ensayo para el código,
- Ensayos Funcionales / Ensayos de caja negra,
- Análisis Estático y
- Demostración formal.

De igual forma que antes, se ha seleccionado una técnica más de las estrictamente necesarias, esta ha sido Demostración Formal, pero debido a la gran cobertura y nivel

de verificación que aporta, es importante aplicarla y más para software que tengan un nivel de seguridad tan alto.

Continuando con la **tabla 2 "integración"**, en esta como no especifica ninguna combinación de técnicas aplicable, se seguirá el criterio especificado al principio de este apartado. Para todos los SIL por tanto la lista será la misma.

- Ensayos Funcionales/Ensayos de Caja Negra y
- Ensayos de las Prestaciones.

La siguiente tabla, la **tabla 3 "Ensayos de Software en Conjunto"** para SIL 1 y 2 la combinación de técnicas homologadas es

- Ensayos Funcionales/Ensayos de Caja Negra y
- Ensayos de las Prestaciones.

Para los SIL 3 y 4 son todas las especificadas ya que se aplica el criterio especificado arriba.

- Ensayos Funcionales/Ensayos de Caja Negra,
- Ensayos de las Prestaciones y
- Modelado.

Para la tabla 4 "Técnicas de Análisis del Software" no se expresa ningún criterio claro de combinación de técnicas apliacbles, por ello se seleccionarán todas aquellas técnicas y medidas con una "R" o más. Resultando una lista igual para todos los SIL.

- Análisis Estático de Software,
- Análisis Dinámico de Software,
- Diagramas de Causa-Efecto,
- Análisis por Árbol de Eventos y
- Análisis de los Efectos de los Errores de Software.

Las demás tablas de la normativa, que son las **tablas detalladas**, siguen el criterio anterior, por lo que todas aquellas que aparecen marcadas como "R" serán seleccionadas. Quitando la tabla de modelado, que se elegirá uno de los marcados como "HR".

8. MEJORA DE SOFTPRO

Stadler Rail Valencia desarrolla en su totalidad vehículos ferroviarios y la gran mayoría de soluciones que utilizan, están desarrolladas por ellos mismos. Esto es una política importante para la empresa, por lo que es un signo distintivo el hacer uso de sus herramientas propias. Por ello, el uso de SoftPro como principal herramienta de Verificación, habiendo sido esta desarrollada por ellos, es un símbolo más de su autosuficiencia.

En este último apartado, viendo las carencias que tiene SoftPro para cumplir la normativa EN 50128, se presentan las utilidades extras que deberían incluirse en SoftPro, para conseguir que sea una potente herramienta de Verificación y casos de prueba que reúna una gran parte de los requisitos de expresados por la normativa.

Analizando SoftPro, una de sus grandes flaquezas respecto a la normativa es la ausencia de análisis estático y la falta de técnicas para testear las prestaciones de los programas. Por ello se propondrán una serie de propuestas teóricas para mejorar la herramienta orientadas a estos aspectos.

Además, a lo largo de este apartado, se mostrará algunos prototipos de la interfaz que podría tener SoftPro para incluir estas funcionalidades y describir su posible funcionamiento de una manera visual.

8.1. PROPUESTAS TEÓRICAS

Como se ha mencionado anteriormente, es necesario incluir análisis estático en la herramienta SoftPro para mejorarla con respecto a los requisitos de los estándares europeos. Los puntos en los que se ha decidido orientar las mejoras son la parte de Análisis Estático y Ensayo de las Prestaciones.

8.1.1. ANÁLISIS ESTÁTICO

Dentro del análisis estático la normativa presenta varias medidas y técnicas, entre ellas, análisis de los valores límite, el análisis de flujo de control y de datos. Estas tres técnicas son las que nos centraremos en este punto.

En programas PLC, es relativamente fácil detectar las variables de entrada y salida, ya que vienen definidas en un bloque al principio del programa. Por lo que implementar una funcionalidad que acceda al código fuente y analizándolo detecte sus variables de entrada y su tipo no debería ser especialmente difícil. Por tanto, conociendo el tipo de variables de entrada, su rango de valores está definido por el lenguaje para cada tipo de dato, por lo que es sencillo determinar los **valores límite** de las variables de entrada. Lo que conllevaría más complicación sería determinar los valores límites de las variables intermedias o forzar las variables finales a sus valores fronterizos. Para esto serán útil las propuestas que se plantean para los flujos de control y de datos.

La solución para representar el flujo de control del programa es hacer uso del **diagrama de flujo de control**. Esto es una representación usando un diagrama de todas las posibles rutas que pueden ser atravesadas durante la ejecución de un programa. Los nodos representan una sentencia o bloque de código sin saltos. Los arcos entre los nodos representan un posible camino entre dos sentencias o bloques de código. Estos diagramas empiezan con un nodo de entrada al programa y finalizan con un nodo de salida del programa.

Para llevar a cabo un análisis de flujo de datos se plantea la utilización de otro diagrama, en este caso el "System Dependence Graph", **diagrama de dependencias del sistema**. Este diagrama muestra tanto las dependencias de control como las de datos. Está separado en nodos que representan, en este caso, cada sentencia del programa. Las dependencias de control se establecen entre dos nodos, siendo u y v dos nodos, si u es el nodo inicial, y v no está anidado en ningún bucle o condicional o cuando u representa un predicado de control y v está inmediatamente anidado a u. Por otra parte, las dependencias de datos entro dos nodos u y v se establecen si u define la variable "x" y v usa "x" y el control puede ir de v a u en un camino en el que no se define x.

Haciendo una combinación de ambas, cierto análisis del diagrama de control y definiendo ciertas invariantes que definan el valor de las variables para cada bloque del programa, se podría llegar a determinar que valores de entrada deberían definirse para que se cumpla una condición dada de una sentencia de control o bucle para que se ejecute una ruta u otra. De esta misma forma se podrían definir los valores de las variables de entrada, sabiendo a que otras variables van a afectar gracias al diagrama de dependencias del programa, para establecer los valores límite de las variables intermedias que se usen en las sentencias de control o condiciones.

8.1.2. ENSAYO DE LAS PRESTACIONES

Es necesario poder evaluar y medir las prestaciones de cada uno de los programas desarrollados para saber si cumple con los requisitos definidos para memoria y tiempo de ejecución, sobre todo para elementos o sistemas críticos. Por tanto, es importante que SoftPro contara con una forma de determinar este tipo de prestaciones. Para ello la solución más interesante podría ser incluir un "profiler". Un "**Profiler**" es un tipo de análisis dinámico del programa que mide el espacio que ocupa en memoria o el tiempo de ejecución de un programa, el uso de una función, así como su duración. Es usado comúnmente para la optimización de programas. El "profiler" lo que lleva a cabo es una medición de costes y puede llevarlo a cabo de distintas formas.

Una de ellas interrumpiendo el programa periódicamente para almacenar el valor contador de programa (CP). Así una vez finalizada la ejecución, se analizan los logs para identificar partes del programa que consumen más tiempo.

Otra manera es modificar el programa insertando código de medición de costes en lugares estratégicos. Consiguiendo así calcular el número de veces que es ejecutado y los recursos de consume.

Por último, se puede extender un compilador con capacidades de medición de costes. Así mientras se ejecuta un programa el compilador almacena el número de recursos (memoria, tiempo, etc.) que cada porción del código consume.

Para mejorar estos medidores de costes en tiempo de ejecución que son capaces de medir costes exactos de ejecución y analizar los efectos de las modificaciones del programa, se usan los medidores de costes simbólicos. Estos permiten distinguir por qué una función es costosa ya que calcula el número de operaciones básicas realizadas por una función (consultar una variable, actualizar una variable, desplegar una función, evaluar un case, almacenar un objeto, etc.).

8.2. PROTOTIPO

El objetivo de este apartado, es mostrar cómo podrían integrarse estas nuevas funcionalidades a la herramienta actual de SoftPro aprovechando las características de esta y su estructura. Adaptando estas funcionalidades al diseño actual de la aplicación y su método de trabajo.

En primer lugar, las funcionalidades que se van a integrar en este prototipo son principalmente, el análisis del flujo de control, el análisis del flujo de datos, el análisis de los valores límites, también una funcionalidad que muestre la cobertura de los casos de prueba y además una funcionalidad de comprobación de la adecuación de los casos de prueba al programa, para ello se permitirá insertar errores en el código del programa para comprobar que los casos de prueba los detectan.

El punto de partida en el que se mostrarán estas funcionalidades será en la ventana de test del SoftPro. Esta herramienta, permite generar varios test para los distintos módulos (programas PLC) generados. Cada módulo puede tener uno o más tests, aunque lo habitual es que para cada versión del módulo exista un solo test que tiene muchos casos de prueba. Una vez aclarado esto, en la ventana que aparece al seleccionar un test asociado a un módulo o un test nuevo al que le hayamos asociado un módulo, será donde se muestren estas nuevas funcionalidades.

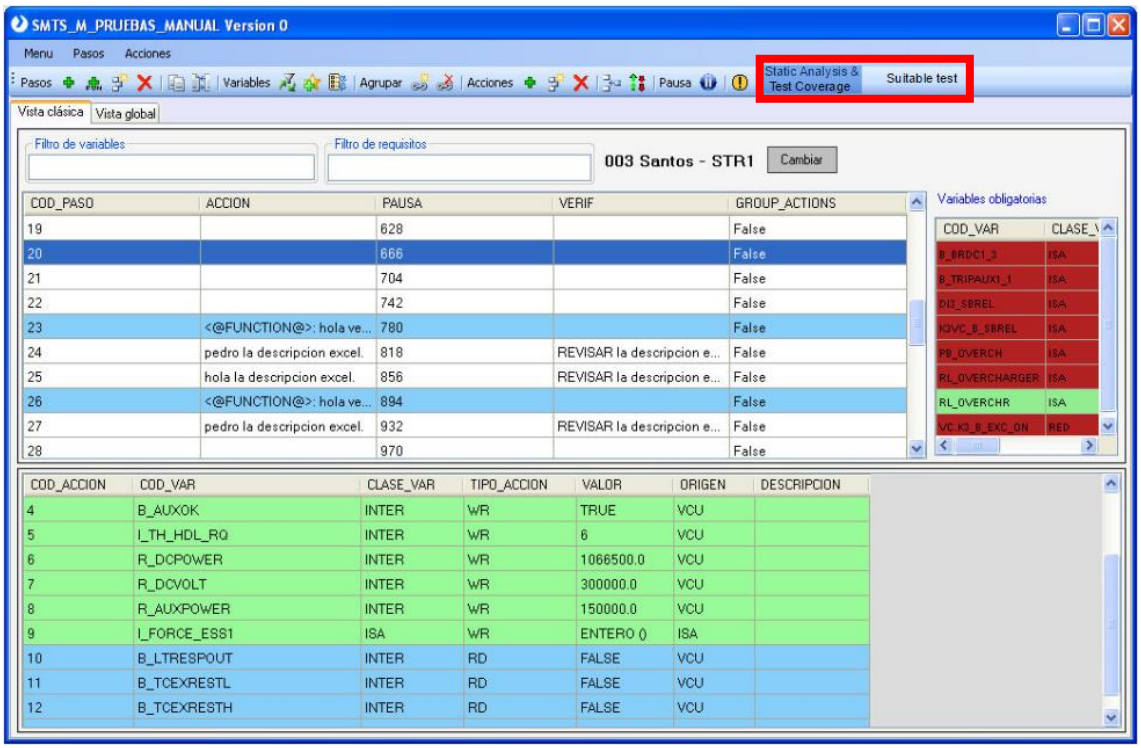

Ilustración 2 Ventana test principal

El botón “Static Analysis & Test coverage” llevará a la ventana donde se han de implementar los distintos análisis expresados arriba y también donde se mostrará la cobertura de los casos de prueba respecto al código. El segundo botón “Suitable Test” abrirá una nueva ventana en la que se podrán cargar códigos erróneos para comprobar la validez de los casos de prueba. Empezaremos por la parte de análisis estático.

8.2.1. ANÁLISIS ESTÁICO Y COBERTURA DEL TEST

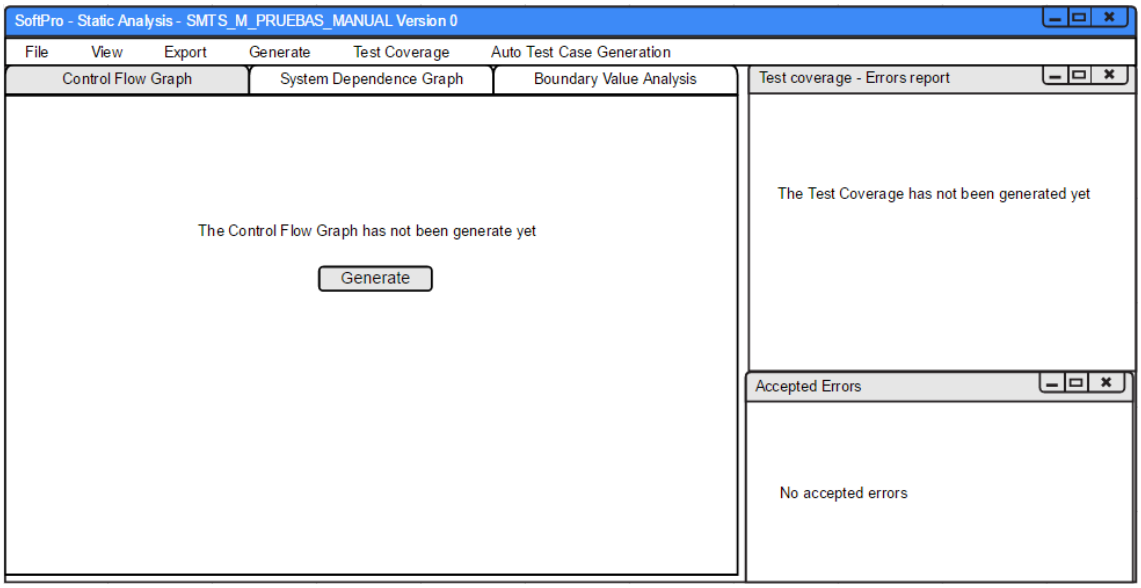

Ilustración 3 Análisis estático y cobertura del test

En esta ventana en la barra de herramientas se pueden apreciar distintos botones, los más destacables son los botones “generate” que dará opciones para generar los distintos análisis, la opción “Test Coverage” permitirá calcular la cobertura del test respecto a distintos valores y una funcionalidad extra que es “Auto Test Case Generation” que deberá generar automáticamente casos de prueba en baso al análisis estático del código.

El resto de la venta muestra en la parte principal tres pestañas que ahora no tienen nada, pero deberán mostrar los distintos resultados de los análisis. La parte derecha está separada en dos ventanas, la superior mostrará el resultado de la cobertura del test en caso de que detecte alguna deficiencia. La parte inferior mostrará los errores de cobertura que el usuario haya aceptado, es decir que por algún motivo válido ciertas cosas no se han testeado.

Como se ha dicho para generar los análisis es necesario pulsar la opción “Generate”, que tiene las siguientes opciones.

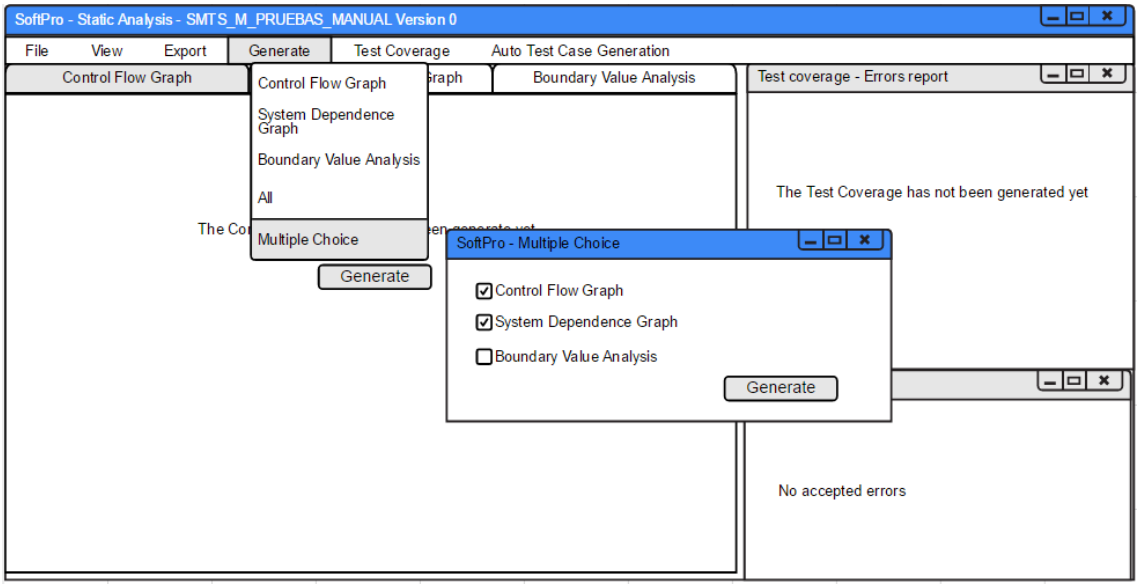

Ilustración 4 análosos estático "Generate" menú

Da la opción de generarlos de uno en uno, todos de forma consecutiva o una combinación de ellos. El resultado de los análisis se verá en la ventana principal y cada uno en su pestaña. Hay que recordar, que cada test está asociado a un único módulo al código fuente del cual es necesario acceso, por lo que este análisis estático al pertenecer a un test, está asociado a un único programa del cual se generarán los distintos diagramas. En caso de que este módulo ya tuviera un test en el cual se hubiera generado los diagramas, no debería ser necesario volverlos a generar, deberían aparecer directamente, siempre y cuando se exactamente el mismo módulo, sin variar su versión.

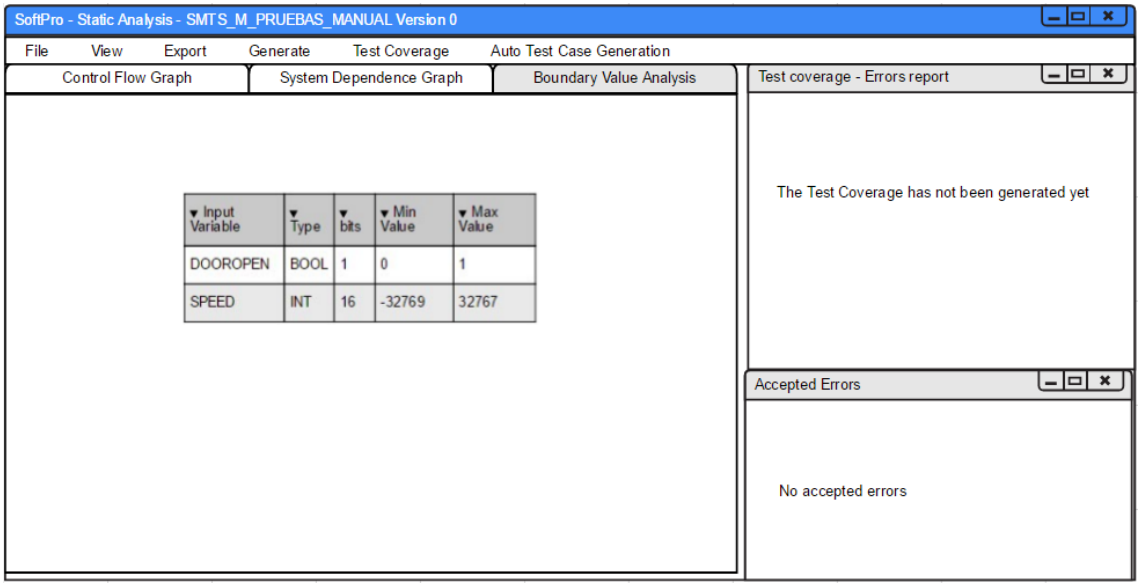

Ilustración 5 Análisis estático, análisis de los valores límite

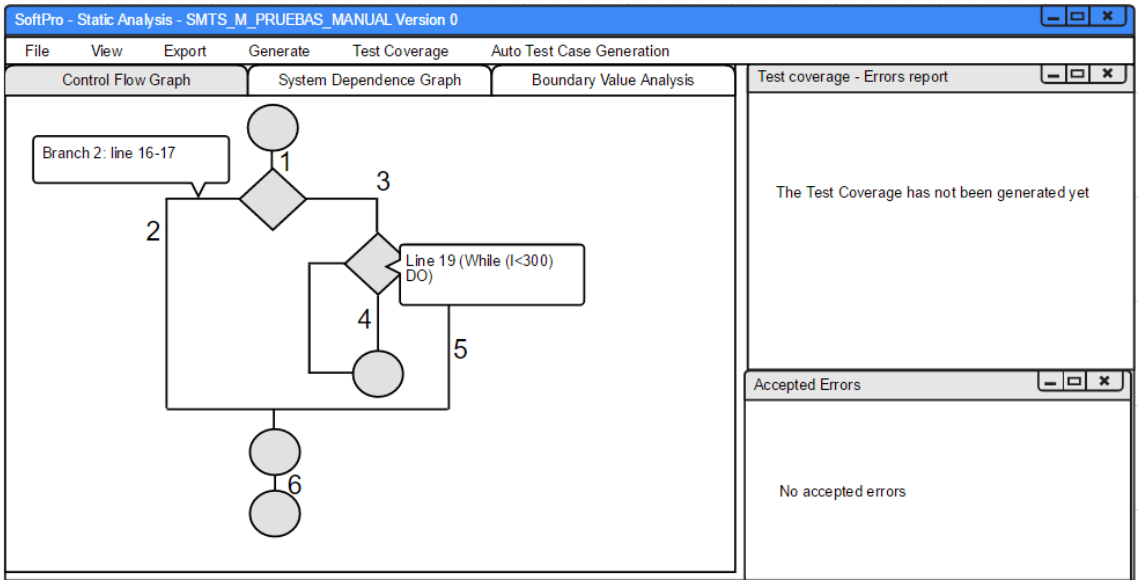

Ilustración 6 Análisis estático, Diagrama de flujo de control

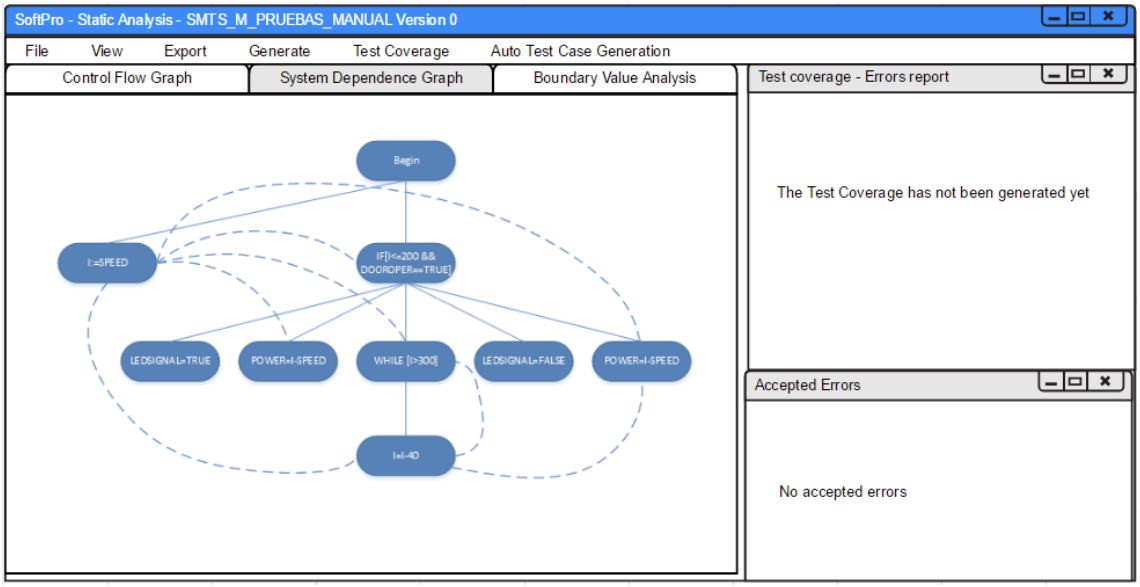

Ilustración 7 Análisis estático, Diagrama de dependencias del sistema

Una vez generados los distintos análisis vamos a pasar a la funcionalidad encargada de calcular la cobertura de los casos de prueba. Para ello hay que seleccionar la opción "Test Coverage" de la cinta de opciones.

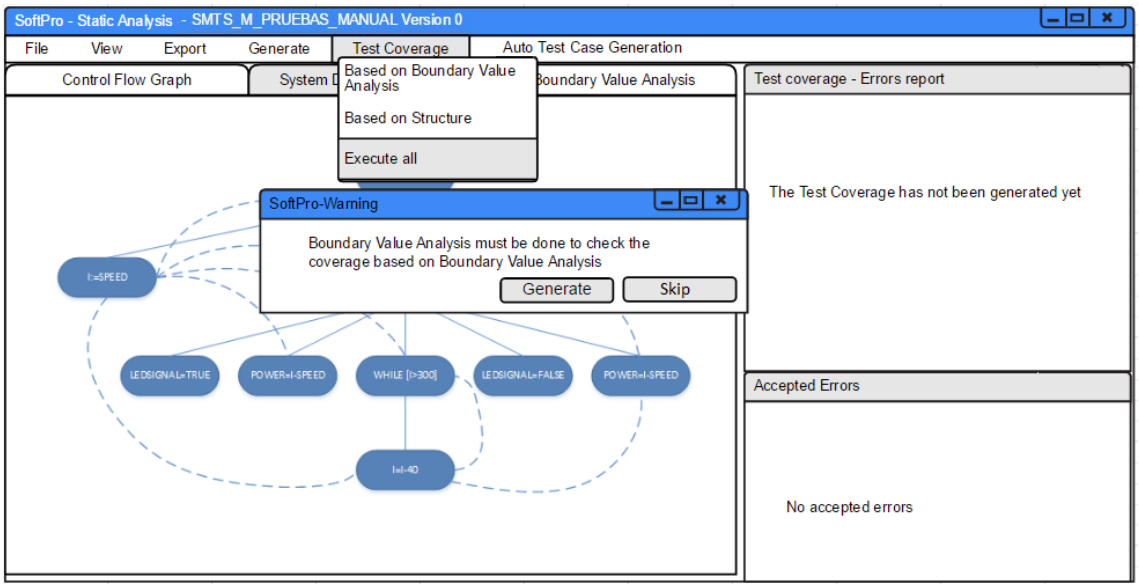

Ilustración 8 Cobertura del test, menú y aviso

Esta opción va a permitir calcular la cobertura de los casos de prueba respecto a los valores límite, es decir comprobar que se han ejecutado durante los test todos los valores límite de las variables de entrada y además comprobar que se han ejecutado todas las rutas posibles del código. Por ello en el menú contextual aparecen las opciones "Base don Boundary Value Analysis" y "Base on Structure" además de la posibilidad de hacer las dos a la vez.

Se muestra también un aviso que saldrá en caso de que se intente comprobar la cobertura de los casos de prueba basadas en un análisis que no se haya generado

previamente, desde aquí permite generar el análisis y continuar comprobando la cobertura o saltarse ese paso.

Una vez se ejecute este paso, deben mostrarse en la ventana de la derecha los errores que se hayan detectado con respecto a la cobertura de los test. Es decir, que valores límite no se han ejecutado en los test o que ramas del programa nunca se han recorrido durante la ejecución de los casos de prueba.

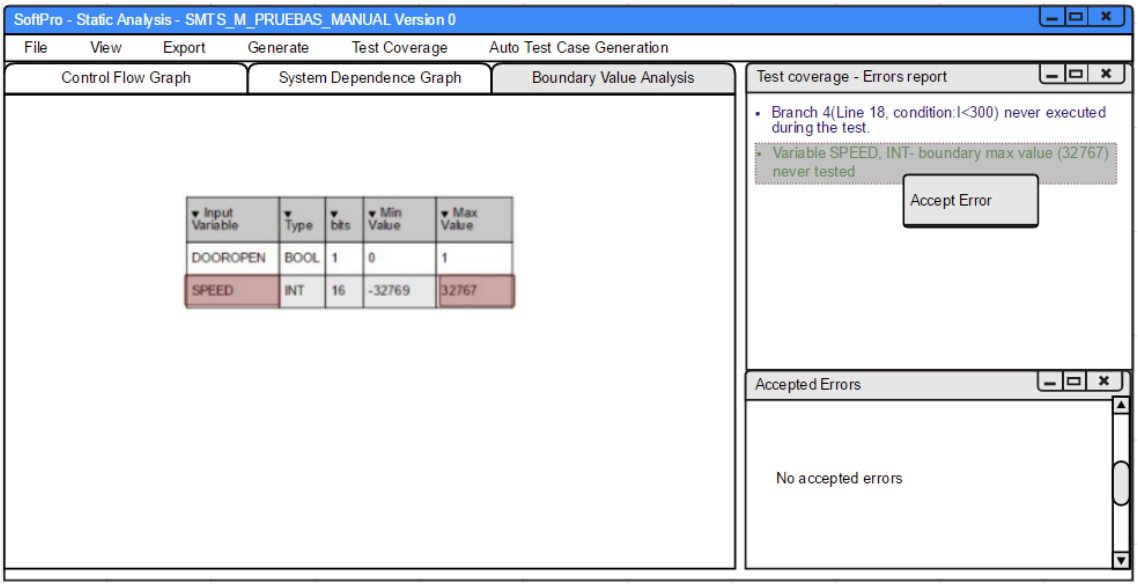

Ilustración 9 Cobertura del test, Informe de errores

Como se puede apreciar si seleccionamos un error este se muestra de forma visual en la representación del análisis al que corresponde además haciendo click derecho en él, da la posibilidad de aceptar el error como válido. Para esto debe salir un nuevo cuadro de diálogo.

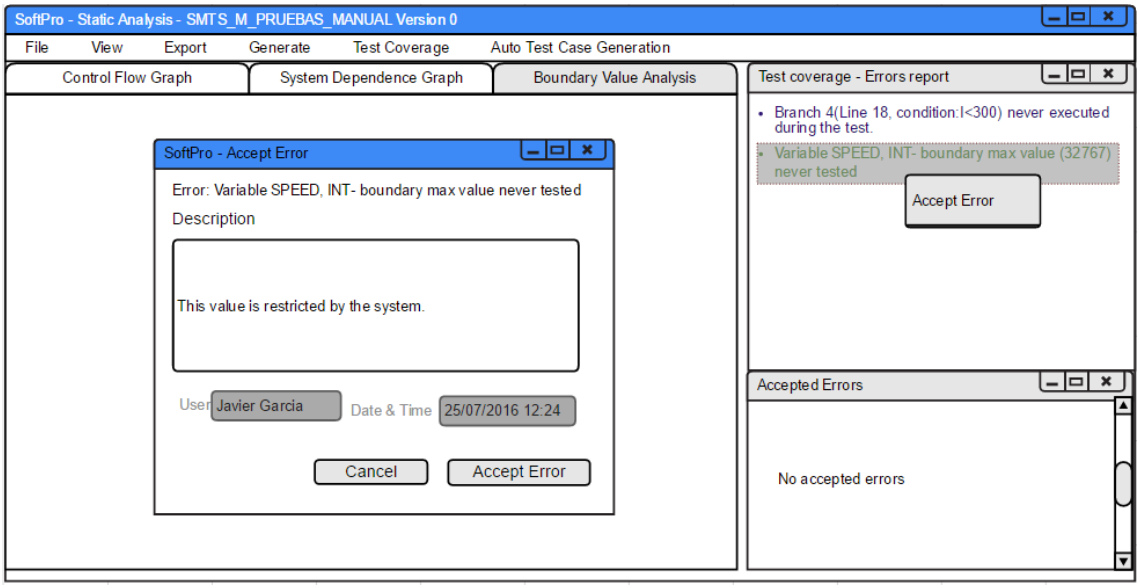

Ilustración 10 Cobertura del test, Ventana de aceptación de errores

En esta ventana aparecen distintos elementos, el cuadro de texto “Description” ha de ser una descripción de porque el error de cobertura es aceptado, explicar porque no se puede comprobar ese valor o ejecutar determinado camino. Bajo aparecen como campos no editables, el nombre del usuario que acepta el error y la fecha.

Una vez aceptado el error este se elimina de la ventana de informe de errores y aparecerá en la ventana de errores aceptados.

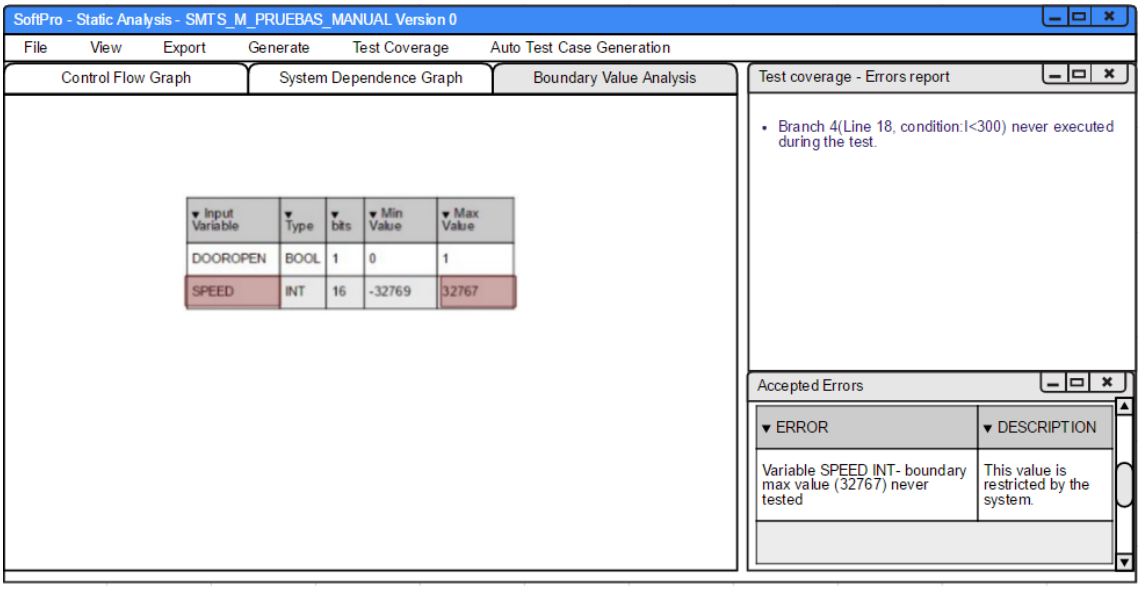

Ilustración 11 Cobertura del test, lista de errores aceptados

De igual forma que se pueden visualizar errores de cobertura del test respecto a los valores límite de forma gráfica, los errores de cobertura basados en la estructura del programa también se pueden mostrar en el diagrama de control de flujo donde se marcará en rojo la ruta que no se haya ejecutado nunca durante los casos de prueba.

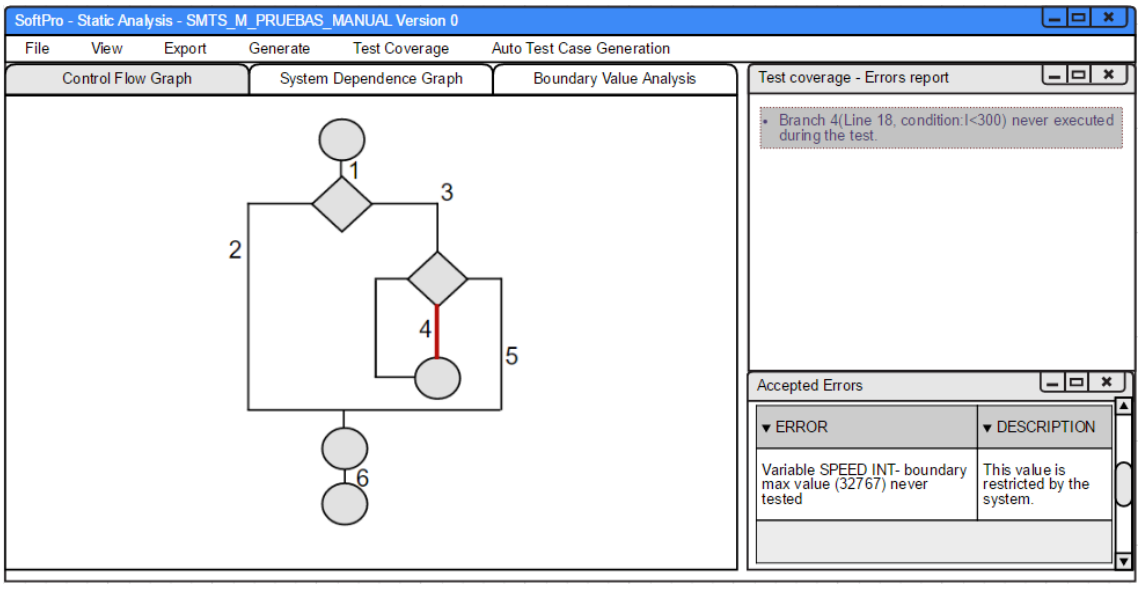

Ilustración 12 Cobertura del test, error basado en la estructura

La última opción de la barra de herramientas de esta ventana es “Auto Test Case Generation” la utilidad de esta es la de generar casos de prueba con la estructura usada por SoftPro basados en los análisis estáticos generados.

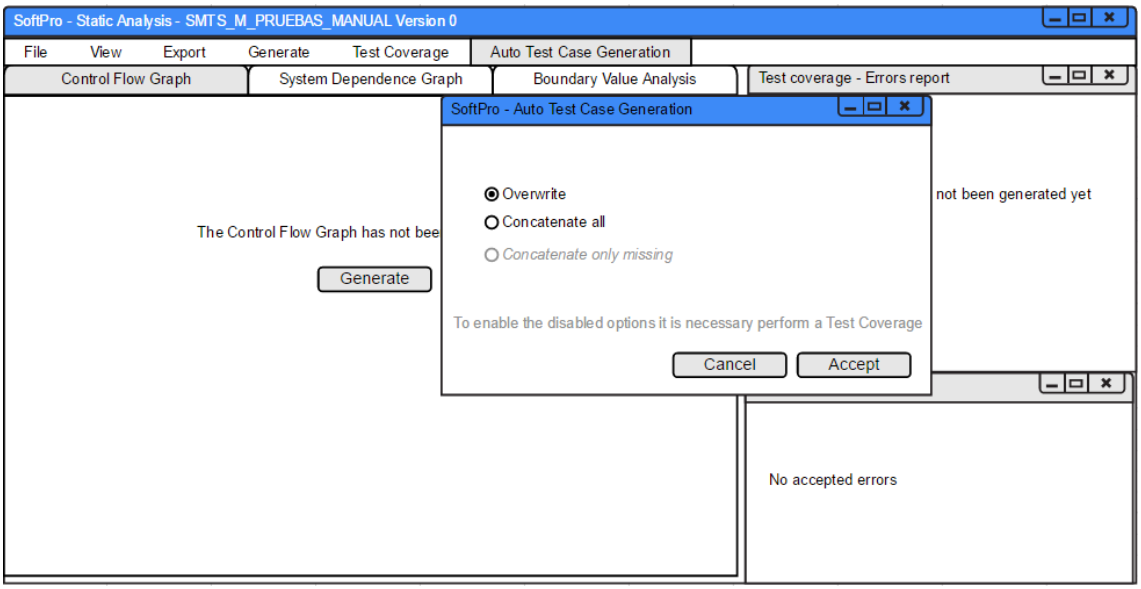

Ilustración 13 Generación automática de casos de prueba

Esta opción permita distintas posibilidades, en caso de que ya existieran casos de prueba, sobrescribir los nuevos que van a generarse automáticamente a los ya existentes o añadirlos sin borrar los anteriores. Existe otra opción que permitirá concatenar aquellos casos de prueba necesarios para corregir los errores de cobertura en caso de que los haya. Para poder ejecutar esta funcionalidad, es necesario previamente haber generado una comprobación de la cobertura de los test. Una vez haya sido hecha esta opción se habilitará.

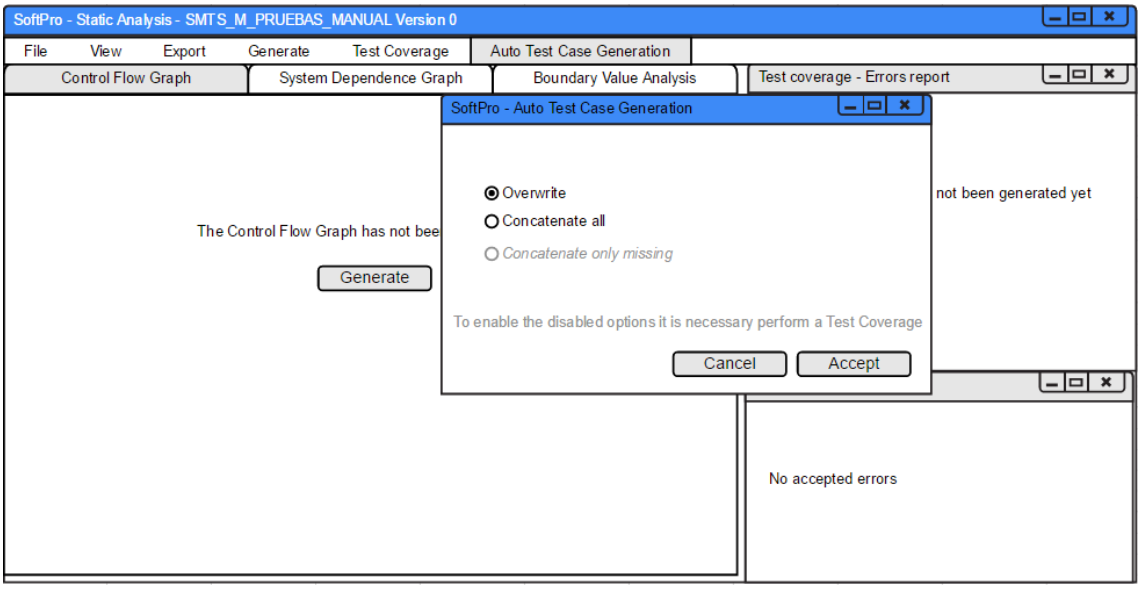

Ilustración 14 Generación automática de casos de prueba 2

Es importante destacar que esta generación automática de casos de prueba, sólo será posible siempre y cuando se haya llevado a cabo todos los análisis estáticos. Si no, no será posible generar los casos de prueba.

8.2.2. ADECUACIÓN DE LOS CASOS DE PRUEBA

Volviendo a la ventana principal del test, la última funcionalidad que queda por comentar es la adecuación de los casos de prueba. Como se ha mencionado previamente el objetivo es verificar la adecuación de los casos de prueba al código. Para ellos lo que se va a hacer es permitir introducir errores en el código y ejecutar los casos de prueba sobre estos códigos erróneos con el objetivo de comprobar si los test detectan todos los errores o siguen dando resultados positivos cuando en realidad debería fallar. Para ello una vez se seleccione el botón “Suitable Test” aparecerá una nueva ventana.

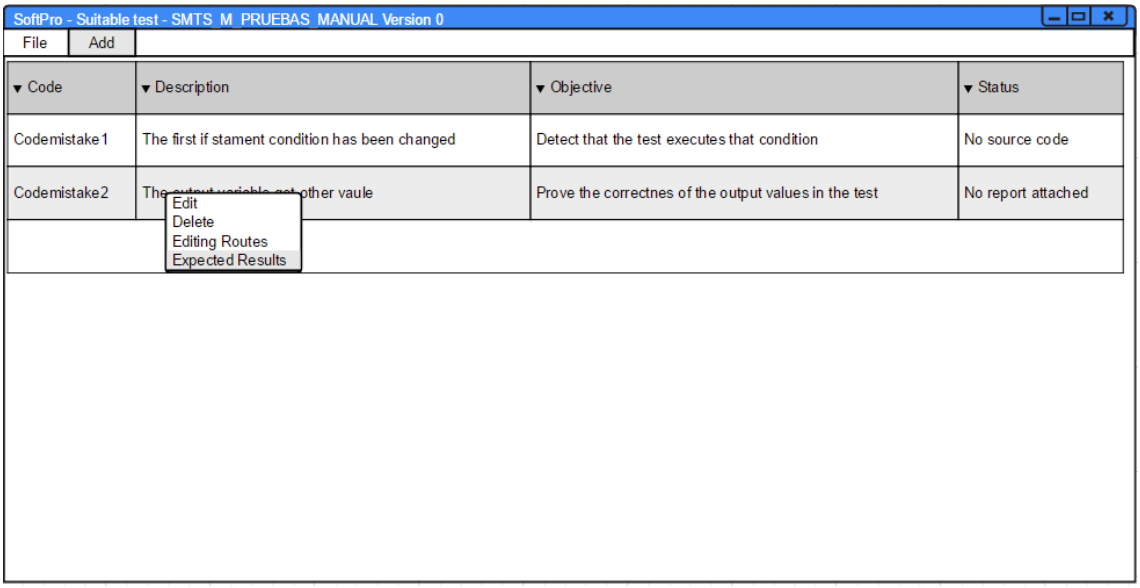

Ilustración 15 Adecuación del test, listado de códigos erróneos7

En esta ventana lo que se muestra es una lista de códigos a los cuales se les ha insertado errores y que están asociados al módulo al cual está asociado el test del cual partimos. La lista debe incluir todos los códigos con errores que existan asociados al módulo que estamos comprobando. Es decir, si previamente en otro test que tenía asociado exactamente el mismo módulo se le añadieron códigos con errores, estos deberían aparecer ahora en esa ventana.

Cada elemento de la lista ha de tener un nombre para el código fuente, que es la primera columna de la lista, además de una descripción en la cual se ha de decir que errores se han introducido en el código, así como otro campo que describa los objetivos que se pretenden demostrar con esos errores. Por último, también se ha de indicar el estado de cada elemento de la lista. Para cada código con errores añadido existen distintos estados que son los siguientes.

- Código fuente no asociado.

- Informe HTML del resultado no asociado.
- No se han definido los resultados esperados.
- Ni el informe de los resultados ni los resultados esperados se han definido.
- El informe de los resultados no coincide con los resultados esperados
- Los resultados del informe coinciden con los esperados.

Existe la posibilidad de añadir nuevos códigos erróneos pulsando sobre la opción "Add" de la barra de opciones.

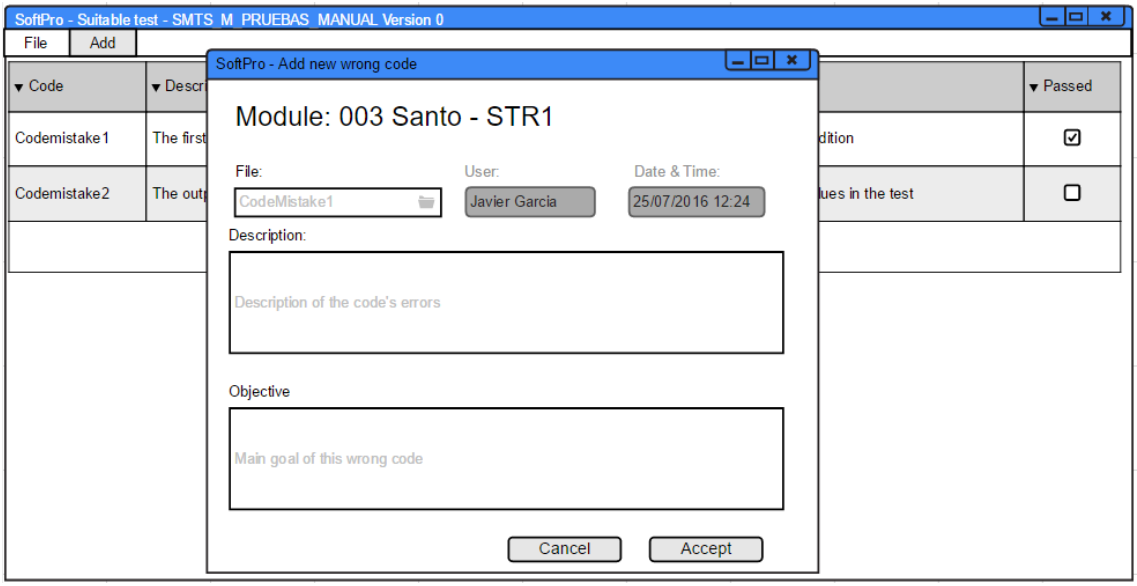

Ilustración 16 Adecuación del test, añadir nuevo código erróneo

Al añadir un nuevo caso de código erróneo, hay que introducir la ruta del fichero en la que se encuentra el código erróneo, la descripción y los objetivos, que son los campos descritos arriba.

Una vez hecho esto y como se muestra en la ilustración 15, haciendo click derecho sobre cada elemento de la lista, se pueden llevar a cabo varias opciones.

- "Edit"- Debe permitir editar los campos de la descripción y objetivos para ese elemento.
- "Delete"- Eliminará la entrada de la lista.
- "Editing Routes"- Abrirá una nueva ventana que permitirá a los usuarios añadir la ruta tanto al código fuente erróneo como la ruta al informe de ejecución de este código fuente con las condiciones definidas en el test asociado.
- "Expected Results"- Debe abrir una nueva ventana en la que se han de definir los resultados que se espera obtener ejecutando el código erróneo bajo las condiciones del test.

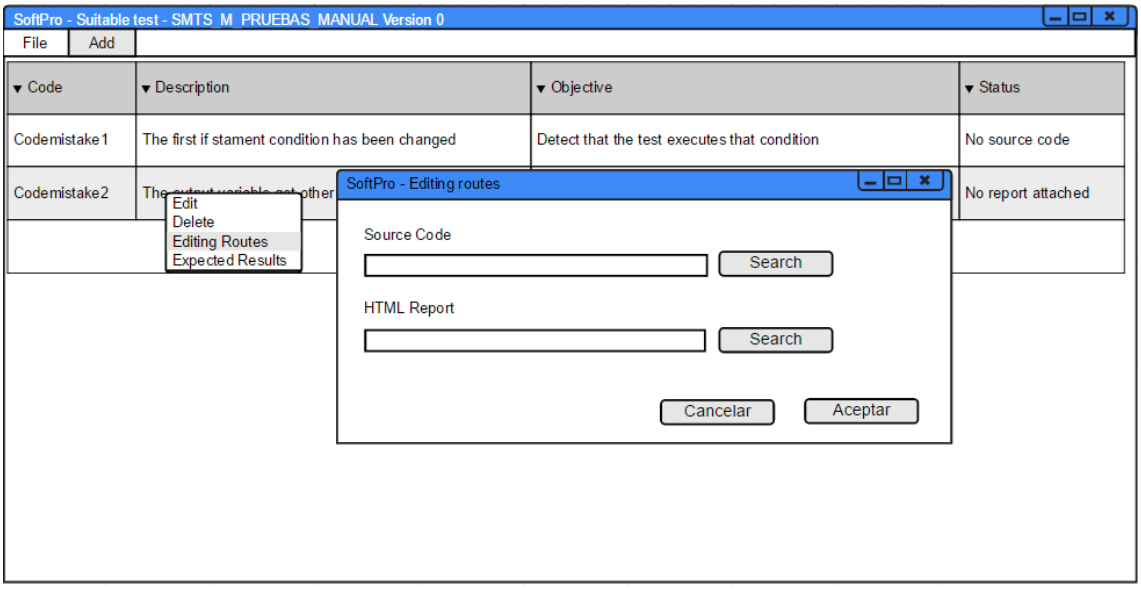

Ilustración 17 Adecuación del test, Editar rutas

Como se ha explicado arriba al seleccionar la opción "Editing routes" una nueva ventana emergente aparece en la cual se puede seleccionar el archivo del código fuente y el informe con los resultados del test ejecutado sobre el código erróneo.

Cuando se selecciona la opción "Expected Results" aparecerá una nueva ventana. Esta ha de tener una estructura similar a la ventana principal del test. Para poder ver todos los pasos y acciones que construyen un test y ahí definir si cada paso ha de fallar o no con respecto a los errores introducidos en el código.

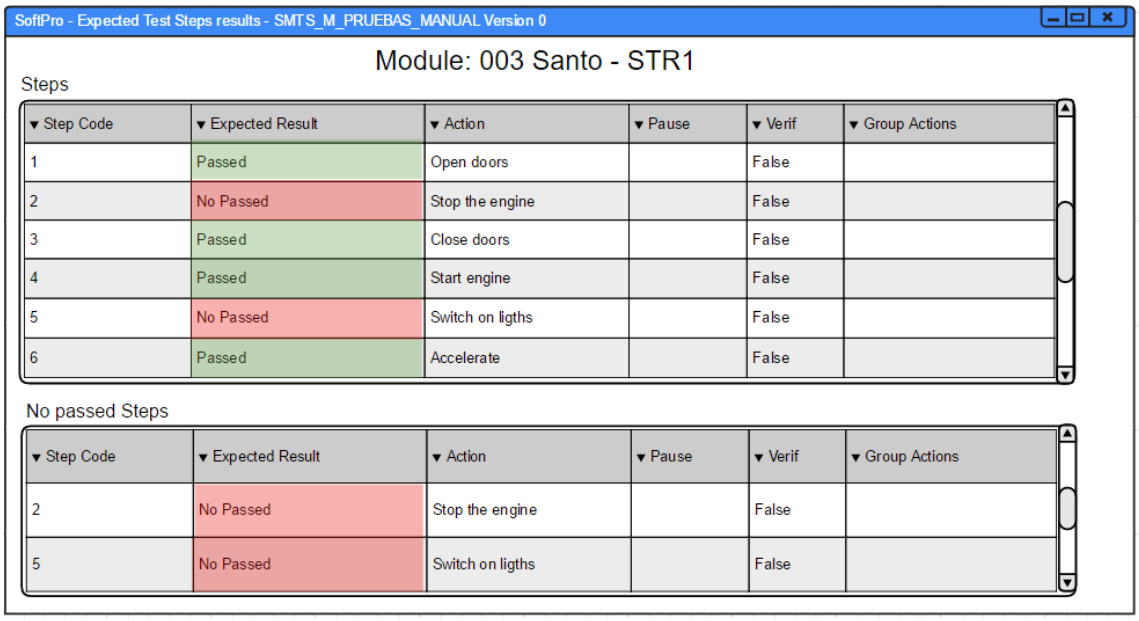

Ilustración 18 Adecuación del test, Resultados esperados

El objetivo de esta ventana es mostrar todos los pasos o acciones que lleva a cabo el test y determinar si los resultados que se obtengan con el código erróneo han de coincidir o no con el resultado que debería dar la ejecución del test con el código correcto. Por ello para cada acción del test hay una opción que permite definir si el

resultado esperado es que pasa el test o que no lo pase. Además, aunque en el prototipo no se muestra, sería interesante mostrar los valores de entrada que debe tener cada paso y los valores de salida que se esperan. Esto sí que se muestra en la ventana principal del test.

Estas serían todas las utilidades nuevas que se plantean en este trabajo para ser introducidas en la nueva herramienta SoftPro para así conseguir una mayor adecuación a la normativa europea. No es objetivo de este trabajo, al menos en esta fase, determinar cómo se llevarían a cabo todas estas funcionalidades mostradas. Simplemente se pretende enseñar de una forma visual un primer prototipo de la herramienta con sus mejoras y ver como se adapta a la estructura y que podría ser algo factible de realizar con el tiempo y los recursos suficientes.

9. CONCLUSIONES

Una vez abordadas todas las cuestiones clave que plantea el tema, creo conveniente realizar una serie de consideraciones a modo de conclusión. La realización de este trabajo final de máster en colaboración con la empresa Stadler Rail, me ha servido para aproximarme al mundo laboral, permitiéndome conocer las formas de trabajar en un entorno no académico al cual estoy mucho más acostumbrado. Esto sirve para poder ver la puesta en práctica de muchos conceptos que se han estudiado de forma teórica, pero en muchas ocasiones no hemos llegado a utilizar. Además, te permite ver la deficiencia en muchos puntos del desarrollo software debido a falta de conocimiento o inversión como por ejemplo la verificación.

Este proyecto en concreto me ha resultado de gran utilidad para conocer normativas sobre el desarrollo Software. Ya que gran parte del trabajo consiste en estudiar y analizar la normativa europea EN 50128, esto me ha servido para familiarizarme con este tipo de documentos, su estructura, vocabulario, funcionamiento, aplicación, etc. Además, de la importancia que estas tienen, ya que regulan el desarrollo de software consiguiendo así que esto sea un proceso con mayor rigurosidad y mayor reconocimiento. Este tipo de normativas, son un puente entre el mundo académico y el mundo industrial, ya que en ella se definen muchas técnicas o medidas que no se suelen aplicar en ámbitos profesionales pero que han sido creadas en ámbitos académicos y permiten mejorar muchos procesos.

El proceso de búsqueda de herramientas e información sobre un tema en concreto, me ha servido para poner en práctica algunas técnicas estudiadas durante el master para conocer el estado del arte de un cierto tema. Durante esta tarea he mejorado mi capacidad de lectura de artículos de investigación y ser capaz de distinguir aquellos que serán de utilidad de aquellos que no se ajustan tanto al tema concreto. También me dio la oportunidad de contactar con empresas o entidades que estaban desarrollando sus herramientas y tener conferencias con ellos.

En líneas generales la experiencia de desarrollar un trabajo de ámbito más profesional me ha permitido conocer el ritmo de trabajo dentro de una empresa, la relación entre los distintos empleados, por lo que me ha permitido complementar mi formación.

Por último, remarcar la satisfacción personal alcanzada al verme capaz de acabar este proyecto, que será útil para continuar mejorando ciertos procesos del desarrollo y el cumplimiento de las normativas europeas, gracias a mi esfuerzo y dedicación, a los conocimientos obtenidos durante mi formación académica, la ayuda de mi tutor y la colaboración con Stadler Rail.

10. BIBLIOGRAFÍA

- Normativa EN 50128, Aplicaciones ferroviarias; Sistemas de comunicación, señalización y procesamiento; Software para sistemas de control y protección del ferrocarril.
- Apuntes de la asignatura Transformación, Validación y Depuración del MITSS

-PLC

- http://www.microautomacion.com/capacitacion/Manual061ControladorLgicoProgramablePLC.pdf
- https://es.wikipedia.org/wiki/Controlador_l%C3%B3gico_programable
- http://recursostic.educacion.es/observatorio/web/gl/component/content/article/502-monografico-lenguajes-de-programacion?start=2

-ITRIS AUTOMATION

- http://www.itris-automation.com/
- Software factory for industrial automation: focus on the PLC software quality.Denis Chalon, Wafaa Ait-Cheik-Bihi, ITRIS AUTOMATION, Grenole France.

-PLC Verif

- https://wikis.web.cern.ch/wikis/display/EN/PLCverif/
- PLCverif: A TOOL TO VERIFY PLC PROGRAMS BASED ON MODEL CHECKING TECHNIQUES. D. Darvas, B. Fernández Adiego, E. Blanco Viñuela, CERN, Geneva, Switzerland.

-ARCADE PLC

- https://arcade.embedded.rwth-aachen.de/
- Arcade.PLC: A Verification Platform for Programmable Logic Controllers. Sebastian Biallas, Jörg Brauer, Stefan Kowalewski.

-REFLEX STUDIO

- http://www.en.artics.fr/#!products/c1rfn