Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 7, No. 3, diciembre, 2011

Web de la editorial: www.ati.es
Web de la revista: www.ati.es/reicis
E-mail: calidadsoft@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos de Informática (ATI), Via Laietana, 46, 08003 Barcelona.

Secretaría de dirección: ATI Madrid, C/Padilla 66, 3º dcha., 28006 Madrid

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica

Eduardo Lluna
Instituto Tecnológico de Informática (ITI)
elluna@iti.es

Resumen

El software es un elemento clave de los actuales sistemas de control incluidos los de seguridad crítica, en los que un fallo puede causar daños irreparables a personas o el entorno. Puesto que el software ni envejece ni se estropea, la calidad de éste dependerá principalmente de los defectos que se introduzcan en la fase de codificación. Por lo tanto cualquier técnica que permita eliminar estos defectos en la fase de creación permitirá aumentar la calidad a un coste más reducido. Las técnicas de Análisis Estático realizan esa función permitiendo localizar defectos sin ejecutar el código. Existen diversas técnicas y no siempre se pueden aplicar todas por razones de coste y tiempo. En este artículo se presenta una selección de las técnicas de análisis estático mínimas requeridas para un sistema de seguridad crítica en base a una norma y, puesto que estas técnicas son más eficientemente aplicadas por herramientas automáticas, se presenta un proceso de selección de estas herramientas en función de requisitos del proyecto.

Palabras clave: Calidad del software, Seguridad crítica, Análisis estático, EN-50128.

Static code analysis in the development cycle of safety critical software

Abstract

The software is a key element of control systems, including safety-critical, where failure could cause irreparable damage to persons or the environment. Software does not get aged or broken so its quality will largely depend on the number of defects introduced in the coding phase. Therefore any technique to avoid defects in the coding phase will increase software quality at a lower cost. Static Analysis techniques perform this function locating defects on the code without running it. There are several techniques but all of them cannot always be applied due to cost and time reasons. This article presents a minimum selection of static analysis techniques required for a safety critical system according a norm and, since these techniques are more efficiently applied by automated tools, a tool selection process based on project requirements is presented.

Key words: Software quality, Safety critical, Static analysis, EN-50128.

Lluna, E. "Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica", REICIS, vol. 7, no.3, 2011, pp. 24-**. Recibido: 22-11-2011; revisado:1-12-2011; aceptado: 19-12-2011.

1. Introducción

Hoy en día el software es un elemento clave en los sistemas de control, incluidos los de seguridad crítica, aquellos cuyo fallo puede causar daños irreparables a personas, bienes o medio ambiente. Esta dependencia ha hecho que el nivel de fiabilidad requerido para este tipo de software sea muy elevado. La forma de conseguir un software de calidad suficiente es realizando desarrollos guiados por una serie de normas y estándares que fuercen el uso de prácticas seguras y controles de forma que se minimicen las posibilidades de error y permitan un seguimiento de la evolución del mismo. Estas normas dependen del área de aplicación final y afectan tanto al hardware como el software del sistema.

Este artículo presenta la técnica del Análisis Estático de código y selecciona un conjunto mínimo de técnicas específicas que puedan ser aplicadas en el contexto de los sistemas de seguridad crítica. Esta técnica, que permite detectar defectos en el código desarrollado sin necesidad de ejecutarlo, aparece recomendada en la mayoría de las normas de desarrollo de este tipo de sistemas. El hecho de basarse en estas normas, las cuales a su vez se basan en la experiencia y lecciones aprendidas a lo largo de muchos años de actividad, nos facilita el proceso de elección mediante la adopción de una serie de criterios predefinidos. El uso del Análisis Estático de código es altamente recomendable no sólo en el desarrollo de sistemas de seguridad crítica sino de cualquier tipo.

2. Desarrollo de sistemas de seguridad crítica

Las normas en uso para el desarrollo de software de sistemas de seguridad crítica dependen de la aplicación final. En el campo de la aeronáutica se usa en todo el mundo principalmente la norma DO-173B de la RCSA [1]. En Europa para el software de sistemas electrónicos en general la norma base es la IEC-61608-3 [2] a partir de la cual se derivan algunas otras normas para sistemas específicos como la EN-50128 [3] para sistemas ferroviarios, la norma IEC-61513 [4] para sistemas de centrales nucleares y la IEC-26262-6 [5] para el sector de la automoción. En temas de espacio, la NASA usa la norma propia NASA 8739.8 [6] mientras que otras agencias espaciales, incluida la ESA, usan la norma ECSS-ST-40C [7] de la ECSS para sus sistemas críticos.

Las normas mencionadas son sólo algunas de las existentes y, pese a la diversidad, todas comparten una estructura común forzando un ciclo de desarrollo y definiendo la

documentación requerida así como una serie de técnicas y buenas prácticas a seguir en cada una de las fases del desarrollo.

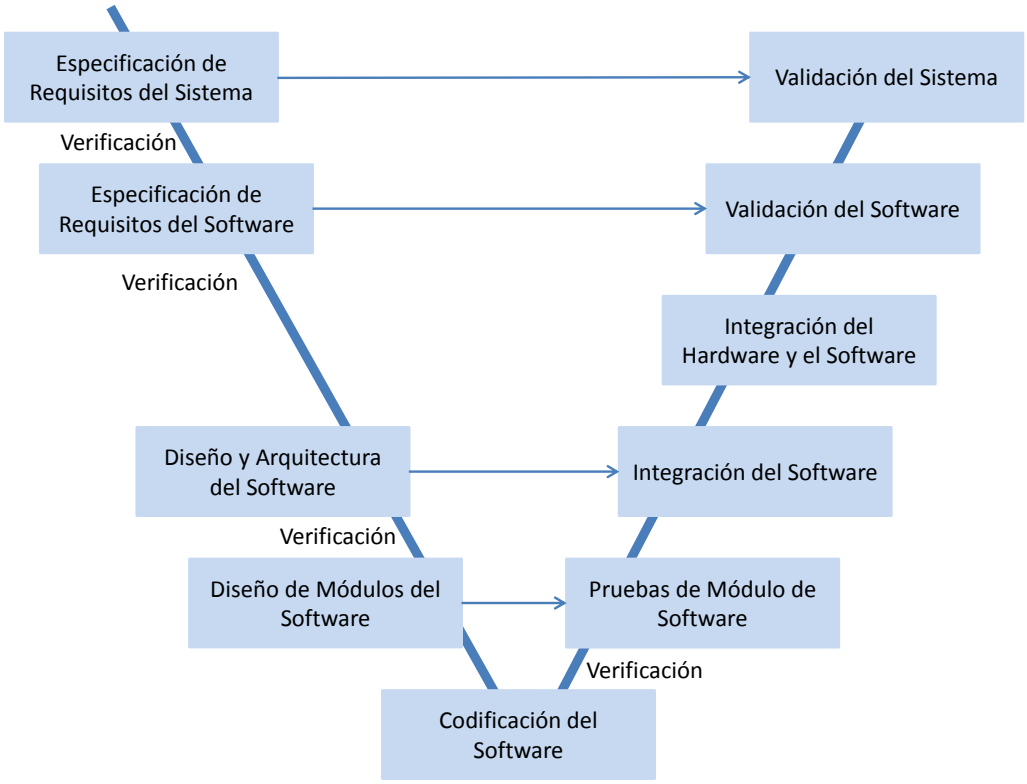

Figura 1: Ciclo de desarrollo del software definido por EN-50128.

Todos los sistemas no tienen la misma criticidad, por lo que las normas también definen una serie de niveles que van desde la ausencia de requisitos de seguridad hasta niveles máximos. Por ejemplo, la norma IEC-61608 define cuatro niveles de 1 a 4 de lo que llama *Safety Integrity Level* (SIL), siendo 1 el nivel mínimo de criticidad y 4 el máximo. Nivel 0 equivale a la ausencia de criticidad. Estos niveles de seguridad realmente se refieren a la seguridad del sistema como tal y se centran principalmente en el hardware definiendo tiempos medios antes de fallo. Desde el punto de vista del software estos niveles representan la peligrosidad o nivel de riesgo que produciría un fallo de software. En función del nivel de seguridad requerido las normas aplican unas técnicas u otras, por lo tanto un aspecto muy importante en el diseño es la elección adecuada del nivel de seguridad de la aplicación.

Vamos a centrarnos en la norma EN-50128 para aplicaciones ferroviarias la cual se basa en un ciclo de desarrollo en V (*V-model*) [8]. La Figura 1 muestra de forma esquemática este ciclo.

El ciclo de desarrollo incluye etapas habituales de especificación, diseño, codificación del software, integración de módulos, integración con el hardware y validación del software y del sistema. Pero un aspecto clave de la norma es el proceso de Verificación que se realiza siempre al terminar una de las fases y antes de comenzar la siguiente. Conviene tener clara la diferencia entre Validación y Verificación. De acuerdo con la definición de ambos términos que aparece en la norma ISO-9000 [9], la validación es la confirmación mediante la aportación de evidencia objetiva de que se han cumplido los requisitos para una utilización específica prevista, mientras que la verificación es la confirmación mediante la aportación de evidencia objetiva de que se han cumplido los requisitos especificados. Básicamente la validación tiene que ver con el uso final del sistema en una aplicación específica mientras que la verificación comprueba que en cada momento se ha hecho lo que se había dicho que se iba a hacer y los entregables de salida de una fase son los que se esperan en la siguiente.

3. Verificación del software

El software es un elemento del sistema que ni se estropea ni envejece. Eso significa que las posibles causas de fallo vendrán o bien de un fallo del hardware (sobre el que se ejecuta o de los periféricos que usa) o por un defecto en su desarrollo que hace que este no se comporte como se espera en función de las especificaciones en una situación particular. Para el caso de los fallos del hardware se emplean técnicas de tolerancia a fallos que principalmente se basan en redundancias de los componentes críticos y en las que no entraremos. La segunda fuente de fallos es más difícil de tratar y se basa en reducir al máximo posible, puesto que es prácticamente imposible erradicarlo al 100%, los defectos en el software. Es importante tener en mente que estamos asumiendo que la lógica especificada es correcta y que los problemas vienen de defectos que hacen que el software no siga la lógica definida. Si tenemos problemas en cuanto a la lógica del sistema (especificaciones de lo que debe hacer) nos encontramos en un caso de errores de diseño, los cuales deberían haberse detectado y subsanado en las etapas de diseño.

Por lo tanto es posible eliminar defectos del sistema verificando que el código desarrollado hace exactamente lo que se ha especificado que debe hacer. De esa forma en el uso real, cuando se den esas condiciones, hará lo que se espera de él.

Existen diferentes técnicas de verificación de software y la norma EN-50128 propone principalmente las siguientes: ensayos formales, ensayos probabilísticos, análisis estático y análisis dinámico. Para un nivel SIL4, el más estricto, todas estas

técnicas aparecen como Altamente Recomendables, mientras que para un nivel SIL1 sólo el análisis estático y el dinámico tienen esa consideración.

Aunque el uso de una única técnica de análisis no permite localizar todos los defectos de un sistema [13], en las siguientes secciones vamos a centrarnos en el análisis estático, ya que es una técnica que por sí misma es capaz de encontrar un número elevado de defectos en el software siendo una técnica relativamente sencilla de aplicar, por lo cual presenta un gran valor y es recomendada independientemente del nivel de seguridad.

4. Análisis Estático

El Análisis Estático es una evaluación del código generado para buscar defectos con la particularidad de que se realiza sin necesidad de ejecutar ese código.

Un aspecto importante para este análisis es el lenguaje de programación usado. La norma EN 50128 presenta una lista de lenguajes recomendables. En general son preferibles lenguajes altamente estructurados y muy restrictivos como el Ada, Modula-2 o Pascal, pero en la realidad, debido a la gran difusión del C/C++ en el sector industrial la mayoría de los desarrollos se realizan en este lenguaje y en Ada. La norma desaconseja el uso de C/C++ sin restricciones pero sí que permite el uso de estos lenguajes usando un subconjunto de los mismos y aplicando una serie de normas de codificación. El subconjunto más usado es el MISRA-C o MISRA-C++ [10] definido por la *Motor Industry Software Reliability Association* el cuál se considera seguro para aplicaciones de seguridad críticas.

En cuanto a las técnicas que pueden aplicarse para realizar este análisis la norma EN 50218 define un conjunto de éstas y entre las que marca como *Altamente Recomendables* para el nivel SIL4 se encuentran el análisis de valores extremos, el análisis del flujo de control y de datos, las revisiones de diseño y la ejecución simbólica del código. Estas técnicas tratan de revisar áreas que son conocidas como fuente de errores. Además hay que realizar una comprobación de las reglas de codificación que se adopten.

El análisis de valores extremos busca defectos en el manejo de las variables en los extremos de su rango de validez o en valores típicamente propensos a error como el cero en el caso de las divisiones o el uso de punteros. Igualmente se comprueban los accesos a *arrays* y elementos con un límite, fuentes habituales de problemas. El análisis del flujo de control busca problemas en la estructura lógica del programa los cuales

pueden ser reflejo de defectos. Por ejemplo, no debe haber fragmentos de código inalcanzables, los cuales se pueden deber a defectos en decisiones que impiden entrar en ciertas partes. También hay que evitar fragmentos de código de complejidad innecesaria puesto ésta puede ocultar problemas y verificar que todos los bucles tienen condición de salida. El análisis del flujo de datos busca errores en las estructuras de datos y en los accesos a las mismas. Los tipos de problemas habituales que deben comprobarse son, por ejemplo, la lectura de variables que no han sido previamente escritas, lo cual puede llevar a comportamientos indeseados, que no haya lecturas o escrituras a una misma variable seguidas, lo cual puede significar que falta código entre accesos. En general hay que comprobar cualquier operación con datos que pueda ser susceptible de enmascarar un problema. Las revisiones de diseños son procesos formales en los que un grupo de revisores comprueban el código usando un conjunto de casos de ensayo que son probados de forma manual sobre el código. Existen normas que definen los procesos de revisión formal y uno de los más conocidos, y sugerido por la norma, son las inspecciones de Fagan [11]. La ejecución simbólica consiste en sustituir a lo largo del software las expresiones del lado derecho e izquierdo de todas las asignaciones manteniendo nombre de variables en lugar de valores de forma que se obtenga al final una expresión para cada una de las variables. Esa expresión resultante se compara con la especificación para ver si coincide. Este proceso normalmente es largo y muy complejo por lo que esta técnica se limita a código relativamente simple.

La tabla 1 muestra a modo de resumen para cada una de las técnicas mencionadas, a excepción de la revisión formal y la ejecución simbólica, los aspectos más importantes que hay que comprobar siguiendo las recomendaciones de la norma EN-50128 para un sistema SIL4.

Técnica	Actividad
Reglas de codificación	Las que vengan definidas en la norma que se aplique (por ejemplo MISRA-C/C++)
Análisis de valores extremos	División por cero
	Uso de punteros nulos
	Uso del mayor valor posible de una variable
	Uso del menor valor posible de una variable
	Accesos fuera de rango en *arrays*
	Uso correcto de listas y *arrays* vacios
	Comprobación de rangos de parámetros en las funciones.

Análisis del flujo de control	Código accesible
	No existe código anudado (simplificable)
	Todos los bucles tienen condición de salida alcanzable
Análisis del flujo de datos	No se leen variables antes de ser iniciadas (salvo volátiles)
	No se escriben variables más de una vez antes de ser leídas (salvo volátiles)
	No se escriben variables que luego no se leen (salvo volátiles)

Tabla 1. Técnicas de Análisis Estático recomendadas por la norma EN 50128.

5. Métricas

Las métricas son indicadores cuantitativos del grado en que un componente, en este caso el software, posee un atributo dado. Al ser un indicador cuantitativo, un valor numérico, permite una comprobación fácil de si el valor está dentro de unos rangos y por lo tanto, de definir de forma clara un criterio de aceptación. Normalmente las métricas no identifican directamente defectos pero, por ejemplo, las métricas de complejidad del código, pueden dar una idea de las probabilidades de que haya más o menos defectos ocultos puesto que a mayor complejidad mayor probabilidad de que hayan defectos ocultos. Para cada métrica hay que definir también un rango de valores que se considera aceptables, los cuales dependerán de la aplicación y del nivel de seguridad requerido. Existen muchas métricas definidas y en uso pero para un sistema de seguridad crítica las más interesantes, por los motivos mencionados, son las de medida de la complejidad del código.

Métrica	Descripción	Rango
DCOM	Densidad de comentarios	>0.2
STCYC	Complejidad ciclomática	1-5
LEVEL	Nivel de anidamiento de funciones	0-4
CALLING	Número de veces que una función es llamada	0-5
CALL	Número de funciones llamadas por una función	0-5
PARAM	Número de parámetros de una función	0-5
RETURN	Número de puntos de retorno de una función	0-1
STNRA	Número de funciones con recursividad	0
NGTO	Número de instrucciones goto	0

Tabla 2. Métricas aplicables.

La tabla 2 muestra un conjunto de métricas a aplicar en sistemas críticos basada en la propuesta del *Hertseller Initiative Software* [12] para sistemas de automoción. Las

métricas seleccionadas miden principalmente la complejidad del código puesto que es el factor principal de cara a la presencia de defectos pero también incluye algunas relacionadas con el uso de técnicas de programación difíciles de seguir y depurar como la recursividad o de estructuras que incrementan la desorganización del código como el *goto*. La tabla también incluye los rangos válidos propuestos para cada métrica en sistemas con nivel SIL4.

El mantenimiento de las métricas dentro de los valores aceptados, si bien no evitan los errores, nos dan una indicación directa del nivel de complejidad y por lo tanto de la probabilidad de tener errores.

6. Proceso de selección de herramientas

El análisis estático puede realizarse de forma manual, de hecho, las inspecciones formales se realizan de esa forma. Aspectos como la comprobación de reglas de codificación y los análisis de valores extremos, flujo de control y datos se comprueban de forma más eficiente mediante herramientas de software que evitan que se cometan errores y aceleran el proceso.

Existen en el mercado una gran variedad de herramientas de análisis estático, tanto de código libre como comerciales pero para sistemas de seguridad crítica es necesario que estén certificadas, por lo que el número de estas se reduce considerablemente y por otra parte el coste de las mismas crece de forma importante. Por lo tanto, en base al proyecto en el que se vaya a aplicar, la elección de la herramienta adecuada es muy importante puesto que por una parte afectará a la cantidad de pruebas que haya que realizar de forma manual, lo cual impacta en la duración del proyecto, y por otra, al tener un coste elevado pueden llegar a ser una parte importante del presupuesto del proyecto.

Para realizar la selección de la herramienta de análisis se ha definido un proceso que se muestra en la Figura 2.

Lo primero y fundamental es tener perfectamente definido el uso de la herramienta, lo que incluye el lenguaje de programación, el nivel de seguridad requerido, las reglas de codificación y el conjunto de técnicas de análisis requeridas. En un proyecto de seguridad crítica la norma de aplicación marcará las pautas a seguir en estas decisiones. Una vez está clara esta información se prepara un plan de evaluación el cual incluye la definición de unas métricas que permitirán tomar la decisión final. En nuestro caso las métricas son el porcentaje de cobertura de comprobación de las reglas

de codificación y el porcentaje de cobertura de las técnicas de análisis incluidas en la tabla 1.

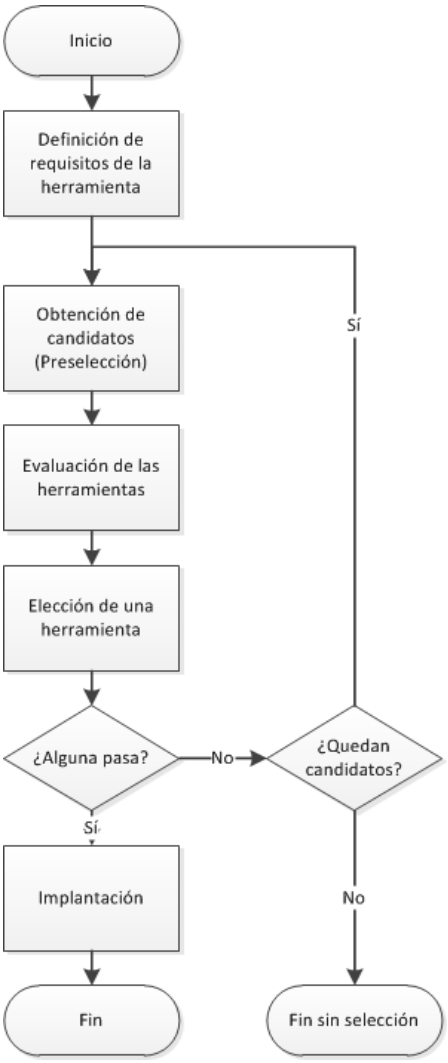

Figura 2: Proceso de selección de la herramienta

Métrica	Descripción	Rango
PTAE	Porcentaje de técnicas de Análisis Estático soportadas	>75%
PTAD	Porcentaje de técnicas de Análisis Dinámico soportadas	>75%
PRMC	Porcentaje de reglas MISRA C++ comprobadas	>75%
PMAE	Porcentaje de métricas de Análisis Estático calculadas	>80%
PMAD	Porcentaje de métricas de Análisis Dinámico calculadas	>80%
MCCY	Cálculo de complejidad ciclomática	Sí

Tabla 3. Métricas para la elección de la herramienta.

Para cada métrica también se definirá un rango de valores aceptables y el criterio de selección consistirá en seleccionar aquella herramienta que obtenga mayores valores en las métricas pero teniendo en cuenta que estas también tienen un valor mínimo de

aceptación y si no se llega a ese valor la herramienta en cuestión no puede ser tenida en cuenta. Estas métricas y sus valores se muestran en la tabla 3.

El siguiente paso es la selección de las herramientas que aparentemente pueden cumplir los requisitos básicos (preselección) y sobre las que se realizará la evaluación. Por ejemplo, sólo por el lenguaje de programación o por la certificación de seguridad requerida hay herramientas que se descartarán directamente. Por motivos de coste, principalmente en tiempo, lo normal es que no sea posible evaluar todas las herramientas que podrían servir al pasar la preselección, por lo que algunos candidatos potenciales pueden quedar sin evaluar. En esta preselección, cuando hay muchos candidatos y claramente hemos de quedarnos sólo con unos pocos, pueden aplicarse criterios de selección más o menos subjetivos en base a prestigio de marcas o experiencia previa. Es importante poder realizar la evaluación sobre una versión real de la herramienta puesto que hay una serie de factores 'no medibles' que pueden influir, principalmente ligados a la usabilidad de esta, por lo que es necesario en las herramientas comerciales obtener una licencia de evaluación. Puesto que el coste de las herramientas suele ser alto, es habitual que los fabricantes accedan a proporcionar versiones de evaluación de duración limitada.

Una vez conseguidas las herramientas se efectúa el proceso de evaluación mediante la ejecución del plan de pruebas definido con cada una de las herramientas preseleccionadas y se calculan las métricas. Esta etapa es la que más tiempo requiere puesto que las personas que realizan las pruebas deben de pasar por el periodo de aprendizaje inicial de las herramientas.

Finalizadas las pruebas y calculadas las métricas el proceso de selección es sencillo y se basa simplemente en la aplicación del criterio de selección previamente definido. Puesto que existen unos valores mínimos es posible que ninguna de las herramientas evaluadas supere la evaluación, en ese caso, si quedan candidatos potenciales no evaluados, se procederá a una nueva selección con estos. En el caso de que no queden candidatos, en principio no sería posible encontrar una herramienta con nuestros requisitos, por lo que se podría repetir el proceso de toma de decisión modificando el criterio de aceptación si decidimos que es preferible usar una herramienta con una cobertura limitada a no usar ninguna.

Con la herramienta seleccionada ya sólo queda la compra de la licencia final y la implantación de la misma. El proceso de implantación también puede llevar su tiempo puesto que todos los miembros del equipo que vayan a participar en el análisis deben

entrenarse en el uso de la nueva herramienta. Para ello es muy conveniente que en la fase de evaluación se vaya preparando una guía de cada herramienta que vaya recogiendo las dificultades que el evaluador ha ido encontrando en su propio proceso de aprendizaje. Si bien esto puede hacer que el proceso de evaluación tome más tiempo, permitirá acelerar la implantación y la difusión de la herramienta entre los miembros del equipo final.

Este proceso de selección se ha aplicado en un proyecto de seguridad crítica con nivel SIL4 desarrollado en C y C++ de acuerdo a la norma EN 50128. Los criterios de selección de la herramienta fueron los siguientes: certificada para SIL4, soporte del lenguaje C/C++, uso de reglas de codificación MISRA C/C++ y soporte de las técnicas de análisis estático mencionadas en la tabla 1. También se introdujeron una serie de requisitos para el análisis dinámico que no se ha incluido en este artículo. Con lo anterior se preparó el plan de evaluación y las métricas para tomar una decisión. El criterio de aceptación corresponde a los valores de la Tabla 3. La ejecución del plan consistió en realizar los análisis requeridos usando las herramienta bajo prueba a un proyecto interno bien conocido para comprobar si es posible obtener con ella los parámetros requeridos.

Se preseleccionaron cuatro herramientas comerciales que cumplían los requisitos previos y de las cuales fue posible obtener una licencia de evaluación. Se dedicó una única persona a la ejecución del plan de evaluación de forma que las evaluaciones se realizaron en serie. El desarrollo del plan de evaluación requirió 6 semanas de trabajo. Este periodo también puede ser considerado como de formación del evaluador en el manejo de esas herramientas y para la redacción de la documentación de uso de la misma. Esta documentación preparada mientras se realiza la evaluación es usada posteriormente para la formación de los demás miembros del grupo. Finalizada la evaluación, la selección de la herramienta más adecuada se realizó de forma objetiva e inmediata aplicando las métricas.

7. Conclusiones

En este artículo se ha visto el papel de la verificación en el ciclo de desarrollo en V típicamente usado en los sistemas de seguridad crítica en el caso particular de la norma EN 50128. En particular la verificación del código generado tiene una gran importancia puesto que las principales causas de fallo del software, descartadas las debidas a problemas con el hardware o el diseño, son defectos introducidos en la fase de creación

del mismo. Esta verificación es vital para eliminar la mayor parte de defectos en la fase más temprana posible y poder asegurar los niveles de seguridad y fiabilidad requeridos por la aplicación.

Dentro de las técnicas de verificación de código el análisis estático es una técnica relativamente sencilla y fácilmente automatizable, incluida como altamente recomendable en todas las normas de sistemas de seguridad crítica y que permite mejorar la calidad del software desde el momento mismo de su creación. Es conocido que el coste de reparación de un defecto crece conforme avanzamos en el ciclo de desarrollo, por lo que encontrar los defectos en la misma fase de la escritura del código es el momento menos costoso. Aunque una única técnica no garantiza el descubrir todos los problemas, la técnica tratada es una de las de más fácil adopción y que más ventajas aporta.

Existen multitud de herramientas que permiten realizar de forma automática este análisis de código, pero cuáles pueden ser usadas en un proyecto específico depende de ciertos parámetros del proyecto, como son el lenguaje de programación usado y el nivel de seguridad. Algunas de estas herramientas, sobre todo las que están certificadas para aplicaciones de seguridad crítica, son caras. Luego debido a la gran variedad y al coste de las mismas, la elección de la herramienta adecuada es importante para el éxito del proyecto. Se ha presentado un proceso de selección sencillo pero que permite realizar esta selección de una forma rigurosa.

Para sistemas que no sean de seguridad crítica, el análisis estático es también una herramienta importante para aumentar la calidad de los productos de software a un coste relativamente bajo por lo que debería estar incluida en cualquier ciclo de desarrollo de software.

Referencias

[1] RTCA, *DO-178B. Software Considerations in Airborne Systems and Equipment Certification*, RTCA, 1992.

[2] IEC, *IEC-61508-3 Seguridad funcional de los sistemas eléctricos, electrónicos y electrónicos-programables relacionados con la seguridad. Parte 3: Requisitos del Software (soporte lógico)*, IEC, 2004.

[3] AENOR, *UNE-EN-50128 Aplicaciones ferroviarias. Sistemas de comunicación, señalización y procesamiento. Software para sistemas de control y protección de ferrocarril*, AENOR, 2002.

[4] IEC, *IEC-61513 Nuclear power plants. Instrumentation and control for systems important to safety. General requirements for systems*, IEC, 2011.

[5] IEC, *IEC-26262-6 Road vehicles – Functional safety. Part 6. Product development. Software Level*, IEC, 2009.

[6] NASA. *NASA 8739.8 Software assurance standard*, NASA, 2004.

[7] ECSS. *ECSS-ST-40C Space Engineering. Software*, ECSS, 2009.

[8] Broekman, B. y Nothenboom, E., *Testing Embedded Software*, Addison-Wesley, 2003.

[9] AENOR. *UNE-EN ISO 9000 Sistemas de gestión de la calidad. Fundamentos y vocabulario*, AENOR, 2005.

[10] MISRA. *Motor Industry Software Reliability Association: Guidelines for the use of the C language in critical systems*. MISRA, 2004.

[11] Fagan, M.E., "Advances in Software Inspections", *IEEE Transactions on Software Engineering*, vol. 12, nº 7, pp 744-751, 1986.

[12] HIS, *Source Code Metrics*, Herfseller Initiative Software, 2008.

[13] Zheng, J., Nagappan, N., Hudepohl, J.P., Vouk, M.A., On the value of static analysis for fault detection in software. *IEEE Transactions on Software Engineering*, vol. 32, nº 4, pp 240-253, April 2006.