Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 5, Número 2 (especial XI JICS), septiembre, 2009

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

Asegurar que el software crítico se construye fiable y seguro

Patricia Rodríguez Dapena
SoftWcare SL
rodriguezdapena@softwcare.com

Resumen

El software falla y estos fallos pueden tener efectos catastróficos o afectar directamente a la fiabilidad y disponibilidad de los sistemas que utilizamos frecuentemente. Varios ejemplos ocurridos recientemente muestran que, incluso sistemas desarrollados siguiendo requisitos muy estrictos y fuertes medidas de control, fallan y causan daños irreparables (ej, pérdida de vidas humanas [9]). La seguridad 'safety' y la dependabilidad son características no-inherentes y son cada vez más importantes en más áreas y dominios. Hay que implementarlas en los sistemas y sub-sistemas explícitamente, como cualquier otro requisito. Existen diferentes técnicas y métodos que se utilizan para implementarlas, para los que no hay mucha experiencia en su utilización, ni aseguran el 100% de fiabilidad o de seguridad. El desarrollo de estos productos críticos resulta más caro, pero los riesgos de no desarrollarlos y mantenerlos adecuadamente son más caros aun: nos afectan a todos y no parece que haya nadie aun velando por nuestra seguridad y satisfacción. Este artículo presenta requisitos que exigen diferentes normas internacionales para implementar software crítico y analiza sus beneficios, inconvenientes y riesgos.

Palabras clave: Seguridad, dependabilidad, software crítico, SIL, niveles de criticidad, riesgo.

Assuring critical software is developed reliable and safe

Abstract

Software fails, and these failures can have catastrophic consequences o affect the reliability and availability of these critical systems we use everyday. Recent examples show that systems that have been developed following very strict requirements and strong control measures fail and may cause irreparable damage (e.g. loss of human lives [9]). Safety and dependability are non-inherent characteristics and they are very important in increasingly number of areas and daily live domains. They have to be consciously and explicitly implemented in systems and sub-systems, as any other requirement. In particular they are implemented using different techniques and methods, for which there is not much expertise and in practice use, and which do not ensure 100% reliability or security. The development of these critical software products is expensive, but the risks of not using the specific requirements and methods and techniques for their implementation are even more expensive: they affect all of us and it seems there is no one (yet) looking after us about our safety and satisfaction using them. This article presents safety and dependability requirements of different critical software-related standards and analyses some benefits, inconveniences and risks they have.

Key words: Safety, dependability, critical software, SIL, criticality levels, risk.

Rodríguez, P.,"Asegurar que el software se construye seguro y fiable", REICIS, vol. 5, no.2, 2009, pp.38-48. Recibido: 22-6-2009; revisado: 6-7-2009; aceptado: 31-7-2009

1. Introducción

El software en sí mismo, cuando falla no puede causar un accidente. Cuando se utiliza para controlar sistemas potencialmente peligrosos es cuando el software es crítico respecto a su seguridad (*safety*). El software en sistemas críticos se ha duplicado en pocos años en todas las facetas de nuestra vida cotidiana, además de ser cada vez más complejo, por lo que aumenta la posibilidad de que influya más directamente en la 'seguridad' y fiabilidad y disponibilidad de los sistemas que controla. En sistemas de consumo no considerados críticos respecto a su seguridad, la cantidad de software también se está duplicando casi cada año (ej, una televisión 'top-of-the-line' contiene software complejo cuando hace pocos años no tenía software). El usuario requiere cierto grado de disponibilidad y fiabilidad que no siempre se asegura.

Implementar, verificar y validar requisitos funcionales, operacionales, de interfaz es conocido y realizado en mayor o menor medida de una manera sistemática (el uso de la ISO 12207 [2] es cada vez más frecuente en los desarrollos software). Pero el concepto de implementar y verificar las características de seguridad *safety* y dependabilidad en un producto software aun no se realiza sistemáticamente en muchos campos donde sí se debería considerar crítico. Se requiere que los sistemas críticos con alto contenido de software funcionen correctamente en el entorno para el que han sido diseñados. El software que implementa funciones críticas del sistema, o que detecta peligros potenciales y/o que mitiga la consecuencia severa de los posibles accidentes, se exige que funcione correctamente (sea *dependable* o confiable) y que, además pueda reaccionar de manera segura y controlada incluso a cambios del entorno no esperadas. Además, el caso específico del software empotrado de control de sistemas críticos en tiempo real, del que no se tiene baja visibilidad de su funcionamiento en tiempo de ejecución, y para el que aun no hay tecnología para permitir su adecuada verificación, validación ni mantenimiento, son características mucho más complicadas de implementar.

Este artículo presenta brevemente diferentes requisitos de ciclo de vida del software crítico, diferentes mecanismos que se pueden/requieren utilizar y cuándo, lo imprescindibles que son, la poca experiencia existente en su aplicación, lo caro que puede resultar implementarlos, y los riesgos asumidos si se limita su uso a software crítico.

2. Software crítico y fallos de software

2.1 Seguridad y dependabilidad

Las características de dependabilidad y la seguridad se relacionan, aunque no son lo mismo. La seguridad *safety* se define como "esperanza de que un sistema en condiciones definidas no llega aun estado en el cual se ponga en peligro ni vidas humanas, ni la salud, ni las propiedades o el entorno" [8]. También se define como "estar libre de riesgos inaceptables" [3]. La seguridad (*safety*) se relaciona con el efecto final de cualquier fallo o evento. Es un aspecto a ser definido y controlado a nivel sistema. Los subsistemas y en particular el software que lo pueda controlar serán críticos según influyan en los aspectos de seguridad (*safety*) a nivel sistema. La dependabilidad en cambio se refiere a los fallos en sí mismos. En general, se define en diferentes estándares para dominios diferentes ([6], etc.) de diferente forma, pero se contabiliza la frecuencia y cantidad de fallos en sí mismos, sin evaluar sus consecuencias peligrosas. El término colectivo usado para describir cómo es la disponibilidad y los factores que la influencian: fiabilidad, mantenibilidad y soporte a la mantenibilidad [7]. Es a nivel sistema, subsistema o elemento. Por tanto, decir *software crítico* se puede referir a software cuyos aspectos de dependabilidad son importantes, o bien que tienen directa influencia en un sistema crítico respecto a su seguridad 'safety'.

De todos modos, es difícil imaginarse el desarrollar software crítico que sea seguro ('safe') pero no fiable. Si fallara mucho, pasaría a estado 'safe mode', sin hacer nada, y no sería un producto muy útil. Se necesita tener productos de software críticos seguros y fiables. Por tanto, aumentar la dependabilidad o la seguridad del software es una cuestión de enfoque en el control/eliminación o tolerancia de fallos del software y qué partes del software son las afectadas. El término criticidad se utilizará en este artículo como sinónimo de la seguridad y/o de la dependabilidad.

2.2 Software crítico

Un sistema puede realizar funciones que son críticas y entonces tiene que protegerse de eventos y/o fallos críticos para no causar daños importantes y estar disponible cuando se necesita. Para ello se diseñan dispositivos y/o mecanismos que mitiguen o controlen estos peligros y/o fallos. Estos eventos y/o fallos se representan en la Figura 1 como "Comportamiento inesperado" del sistema y los mecanismos para su control y mitigación corresponde a las zonas marcadas con un círculo como 'requisitos de seguridad y dependabilidad del sistema'. Tanto estos mecanismos de mitigación como otras funcionalidades críticas del sistema pueden ser implementados por software. Este software crítico debe funcionar correctamente, es decir, ni debe fallar en implementar las funciones críticas del sistema (debe controlar su propio comportamiento inesperado) ni debe ser la causa de tener que ponerse en funcionamiento los mecanismos de mitigación (esto corresponde a las zonas marcadas con un círculo como 'requisitos de seguridad y dependabilidad del software' en la [8]) y, además, no debe fallar en implementar mecanismos de mitigación de daños a nivel sistema que estén bajo su responsabilidad.

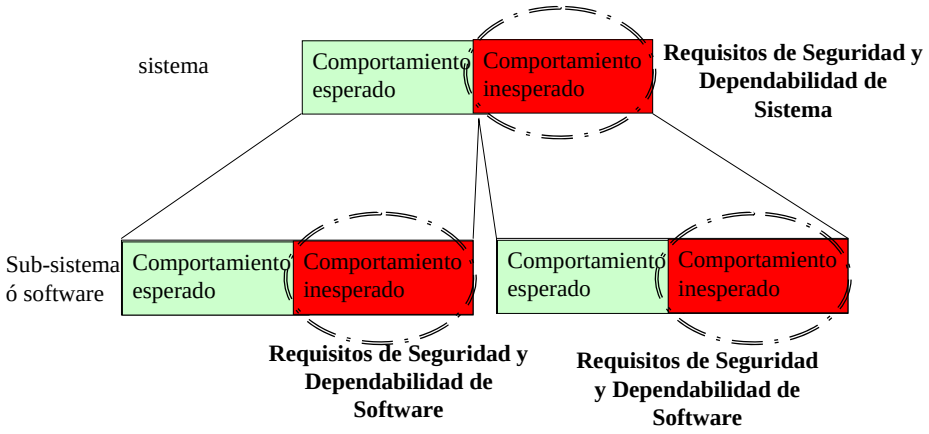

Figura 1. Requisitos y limitaciones de seguridad y dependabilidad del sistema y del software [1]

En general, el software crítico en un sistema crítico existe cuando:

- el software no falla de modo que cause o contribuya a un estado peligroso del sistema.
- el software no falla en la detección o la corrección estado peligroso del sistema.
- el software no falla en la mitigación o reducción del impacto del posible efecto si ocurre el accidente.

Los fallos de software son fallos sistemáticos puesto que se relacionan determinísticamente con una causa que sólo se puede eliminar modificando el diseño o código (o manuales de usuario/operaciones, etc.) [3]. El nivel de criticidad del software

depende únicamente del nivel de severidad del impacto que el fallo pueda ocasionar (el cálculo de las probabilidades de fallos de software es inmaduro y no se calcula en la práctica en la mayoría de los dominios para definir la criticidad).

La seguridad (*safety*) y dependabilidad de software están relacionados con los términos *failures*, *errors* y *faults*. Es necesario distinguir diferentes significados del término fallo de software, pues los mecanismos/medidas a implementarse para mitigar, prevenir estos 'fallos' de software se refieren a alguno de estos tres términos en particular:

1. *Failures* (fallo) se define como la terminación de habilidad de un elemento para realizar una función requerida [3]. Un software *failure* es la manifestación de un *error* [6].
2. *Error* es una discrepancia entre el valor o condición calculada, observada o medida y el valor o condición real, especificado o teóricamente correcto [3]. El error en el uso del software puede ser también la causa del *failure*.
3. *Fault* (fallo) es la causa de un *error* [4][6].

Implementar la dependabilidad y seguridad de software se centra en prevenir/eliminar/tolerar o predecir los fallos ('failure', 'error'y/o 'fault') de software.

3. Requisitos y procesos de ciclo de vida de software crítico

La seguridad y la dependabilidad del software, al ser características que hay que implementar, verificar, validar y evaluar explícitamente a través de la implementación de diferentes requisitos y mecanismos que se derivan del análisis de los requisitos y restricciones de seguridad y dependabilidad del sistema (ver Figura 1 anterior). Estos requisitos y mecanismos de deben planificar desde las primeras fases del ciclo de vida del software.

En cada una de las fases del desarrollo habrá que implementar, verificar y/o validar algún aspecto de estas características, como se muestra en la Figura 2. Existen cuatro grupos de métodos, para prevenir/eliminar/tolerar o predecir los fallos (*failure*, *error* y/o *fault*) de software.

Los tres primeros tipos de mecanismos son los más utilizados, mientras que el último aun es un grupo de métodos y técnicas inmaduras y aun en fase de investigación:

- Prevención de fallos - ¿cómo prevenir la aparición e introducción de fallos?

- Tolerancia a Fallos - ¿cómo proporcionar un servicio que cumpla con las especificaciones aún cuando se está produciendo un error?
- Eliminación de fallos - ¿cómo reducir la presencia de fallos, en relación con la cantidad y severidad de los fallos?
- Previsión de fallos - ¿cómo estimar la aparición de fallos?

Existen múltiples métodos y técnicas de cada tipo, y desgraciadamente no hay una lista estándar de cuáles utilizar en cada caso, ni qué combinación de técnicas es la mejor. Lo que sí es cierto es que la implementación de las características de seguridad y dependabilidad en el software puede resultar muy costosa dependiendo de los mecanismos, organización y requisitos que haya que implementar.

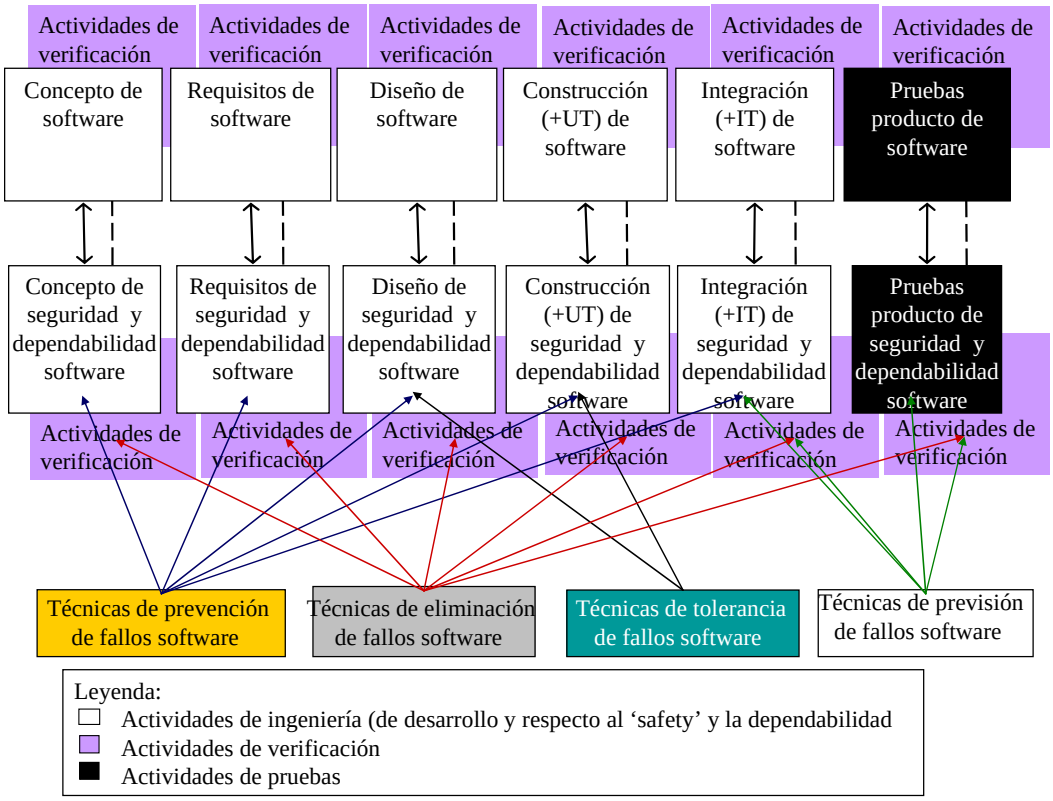

Figura 2. Mecanismos para la seguridad y dependabilidad del software 0

Los diferentes estándares internacionales existentes al respecto definen diferentes requisitos y limitaciones para la implementación de la seguridad o la dependabilidad del software y su aplicabilidad se requiere según lo que se denomina "nivel de criticidad" del software. El SIL o Nivel de Criticidad es un indicador del grado de confianza requerida para un producto o sistema, por lo que determinan qué medidas hay que poner para que, o

bien la función de mitigación sea fiable o que el fallo (*failure*) que pueda ocasionar un peligro no ocurra o se reduzca la severidad del efecto [8]. Los estándares donde se menciona la seguridad y/o la dependabilidad del software utilizan diferentes definiciones de niveles de criticidad e indicadores para definirlos. Además requieren:

- El uso de diferentes técnicas (prevención, tolerancia y/o análisis y eliminación) en las diferentes etapas del ciclo de vida del software para cada nivel de criticidad.
- Que las mismas actividades, técnicas y métodos se utilicen para el software configurable y sus datos, pues hay sistemas críticos que reutilizan software crítico genérico, pero configurable a través de sus datos, y éstos son críticos también.
- Diferentes niveles de independencia del personal para realizar algunas actividades, respecto del personal que lo desarrolla.
- Exigencias respecto a la confianza necesaria respecto a productos ya existentes que se quieren reutilizar como parte del software crítico.
- Exigencias respecto a las herramientas a ser utilizadas para el desarrollo, la verificación y la validación del software crítico, que pueden afectar a la calidad del producto final.

Estas diferencias se pueden ver al comparar estándares como el DO178B [4] (que contiene guías (utilizado en la práctica como requisitos) para la implementación y demostración de la seguridad-'safety' del software de control a bordo de los aviones, para las autoridades de certificación); o la norma NASA 8739.8 y su guía (NASA-GB-8719.8 [7]) (que contiene requisitos para el desarrollo de software crítico (seguridad) en sistemas espaciales para NASA); o la norma IEC 61508 [3] (que establece requisitos para todas las actividades relacionadas con el ciclo de vida de seguridad de los sistemas que incluyan componentes eléctricos y/o electrónicos y/o electrónicos programables (E/E/PES) que se utilizan para realizar las funciones de seguridad) y que es base para la futura ISO 26262 para dispositivos electrónicos en sector automoción; o la norma EN50128 [3], basada en la IEC 61508 0 antes mencionada, (especifica procedimientos y requisitos técnicos para el desarrollo de sistemas electrónicos programables para uso en aplicaciones de control y protección del ferrocarril); o la norma DEF-STAN-00-55 [5] (con requisitos de seguridad de software para sistemas de defensa del Reino Unido tiene requisitos tales como el uso de

lenguajes de descripción formal y verificaciones y pruebas formales para el software crítico).

Existen más ejemplos de estándares para software crítico donde se definen requisitos, métodos, técnicas, organizaciones, etc. para diferentes niveles de criticidad del software. La sección siguiente analiza los efectos de estos requisitos en las organizaciones y desarrollos software, además de los riesgos de no implementarlos.

4. Análisis de costes y riesgos asociados

Los diferentes requisitos para la implementación de la seguridad y dependabilidad del software en sistemas críticos requieren:

- Que los desarrolladores de software tengan conocimientos de los mecanismos específicos de seguridad y dependabilidad, tanto para la ingeniería (tolerancia a fallos y prevención de fallos - métodos más o menos formales para el desarrollo, uso de estándares limitados de programación, etc.), como para la verificación y análisis del software (eliminación de fallos). Es necesario además que sepan seleccionar cuáles son las técnicas a utilizarse y combinarse, valorando otros requisitos de rendimiento, tiempo real, ocupación de memoria, operacionales, etc. y, en la mayor parte de los casos, a ser propuestas y negociadas con el cliente. Por tanto, en las organizaciones:
    1. Posible formación necesaria
    2. Limitación del personal que podrá desarrollar estos productos
- Que las organizaciones planifiquen roles más o menos independientes para cada proyecto: a veces los que desarrollan deberán ser independientes de los que verifican, de los que validan y de los que evalúan la seguridad, por tanto:
    1. Más compleja organización aumentando la gestión de los proyectos
    2. Más compleja organización implica más formalidad en el proyecto, que puede conllevar a más duración y más esfuerzo
- Cambios en las herramientas que se utilicen para los desarrollos: no cualquier herramienta puede ser utilizada, por tanto:
    1. Compra y formación de nuevas herramientas de desarrollo, o
    2. Cualificación y pruebas de las herramientas existentes

- Que la reutilización de software ya existente se realice cumpliendo los mismos requisitos que el software crítico nuevo, requiere realizar actividades extra.

Por tanto, el desarrollo de software crítico es caro de implementar. Pero:

- ¿Qué riesgos se asumen y quién los asume si no se implementan los sistemas críticos con alto contenido en software con estos requisitos?
- ¿Cómo se asegura mayor fiabilidad y seguridad del software empotrado en los 100 ECUs de un automóvil cuando hoy día, en general, no se exige que demuestren el cumplimiento de este tipo de requisitos? (la norma ISO/IEC 26262 para seguridad funcional en vehículos de carretera aún no se ha publicado y todavía no es de obligado cumplimiento).
- ¿Cómo asegurar que los dispositivos médicos, cada vez más electrónicos (controlados por software), cumplan los más estrictos requisitos de dependabilidad y de seguridad?
- ¿Y el software del cajero automático? ¿y el software de control semafórico?...
- ¿Quién debe decidir qué requisitos son los que se implementarán y después se verificarán y validarán en estos sistemas críticos? ¿Quién será el responsable de su aseguramiento?

En muchos dominios, hoy día, son los fabricantes de estos sistemas los que deciden el riesgo a ser sumido por los usuarios. Estos riesgos, en muchas ocasiones, se miden por datos de mercado y ventas, más que por datos de riesgos de seguridad para los usuarios. Pero, por otro lado, ¿Se sabe el riesgo que asume al utilizar o depender de sistemas críticos controlados por software? Además, las reparaciones y mantenimiento de estos sistemas y de su software son cada vez más sofisticados. Hasta hace pocos años, los coches se reparaban en los talleres mecánicos y se conocían los mecanismos hidráulicos que tenían, o las lavadoras se arreglaban con ‘chapucillas doméstica’ que conseguían que siguiera funcionando. Ahora, con los subsistemas electrónicos, más sofisticados y complejos, ya no se puede corregir el software más que de los fabricantes directamente, que nos cambian las piezas completas (cajas ‘negras’) por nuevas que contienen nuevas versiones del software.

¿Quién asegura la seguridad y satisfacción de los usuarios de estos sistemas? ¿Es necesaria más regulación y legislación para exigir a los fabricantes cumplir unos mínimos de calidad (por ejemplo, que en las homologaciones de coches se revise el software y sus

características, etc.)? ¿Serán los seguros más baratos cuanto más fiable y seguro sea el software que va a bordo de los automóviles pues menos se tendrá que ir a repararlo?

5. Conclusiones

La implementación de las características de seguridad (*safety*) y dependabilidad del software en sistemas críticos es exigida a través de estándares en diferentes campos de aplicación e implica la utilización de técnicas de prevención, tolerancia y eliminación de fallos de software – aunque no se utilizan mucho ni con total rigor. Requieren tanto la realización de actividades 'extra' respecto de las mínimas definidas por ejemplo en ISO/IEC 12207 [2], como el uso de técnicas y métodos específicos para asegurar el desarrollo, la verificación, validación y evaluación de la seguridad y dependabilidad, siendo más estrictos cuanto más crítico sea el software. Además, se suele requerir una organización específica, donde ciertos roles (por ejemplo, evaluador de la seguridad, pruebas de validación, etc.) sean más o menos independientes de los desarrolladores. Hay además otros requisitos adicionales a ser cumplidos: los relativos tanto a software que se quiera reutilizar en un producto crítico, como a la cualificación de las herramientas de desarrollo, verificación y validación (pues pueden afectar a la 'calidad' del producto resultante), a los sistemas configurados por datos, etc.

Además de que estos requisitos no se utilizan sistemáticamente, su uso no asegura el 100% de fiabilidad ni de seguridad, son caros de implementar, la duración y complejidad del proyecto son mayores, etc. Hoy, en muchos de estos sistemas, son los fabricantes de estos productos/sistemas críticos los que definen los riesgos que asumirá el usuario final, y los definen en muchas ocasiones en base factores de ventas y mercado y no a los riesgos de seguridad. Por otro lado, los usuarios desconocen estos sistemas tan sofisticados y complejos, difíciles de entender, de mantener y los riesgos que conlleva su utilización. ¿Quién deberá salvaguardar estos mínimos de seguridad y satisfacción? ¿Es posible definir más regulación y legislación al respecto?

Referencias

[1] Rodriguez Dapena, P., *Software Safety Verification in Critical Software*. Tesis doctoral, Universidad Técnica de Eindhoven, Holanda, 2002.

[2] ISO. *ISO/IEC 12207 Systems and software engineering - Software life cycle processes*, ISO, 2008.

[3] IEC. *IEC 61508-3: Seguridad funcional de los sistemas eléctricos electrónicos electrónicos programables relacionados con la seguridad. Parte 3: Requisitos del software (soporte lógico)* Primera edición 1998-12 + Corrig. 1999-04.

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

[5] UK MOD. *DEF STAN 00-55(PART 1). "Requirements for Safety Related Software in Defence Equipment", Part 1: Requirements.* UK MOD 1 Agosto 1997.

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

[7] NASA. *NASA-GB-8719.13. NASA Software Safety Guidebook*. Prepared by NASA Safety and Mission Assurance Office. Marzo 31, 2004.

[8] ISO. *ISO/IEC 15026:1998 Information technology -- System and software integrity levels*. ISO/IEC Edition: 1, 1998.

[9] Risks digest. Forum On Risks To The Public In Computers And Related Systems. ACM Committee on Computers and Public Policy. http://catless.ncl.ac.uk/Risks/