Revista Antioqueña de las
Ciencias Computacionales y la Ingeniería de Software

ISSN: 2248-7441

www.fundacioniai.org/raccis/index.htm
raccis@fundacioniai.org

Application of Formal Methods in Software Engineering

Aplicación de los Métodos Formales en la Ingeniería de Software

Adriana Morales[1], Miguel A. Cifuentes[2]
[1] UNIVA. Vallarta, México. *aMora.vallarta(AT)univa.mx*
[2] UNIVA. Vallarta, México. mCifu.*vallarta(AT)univa.mx*

INFORMACIÓN DEL ARTÍCULO

Tipo de artículo
Investigación

Historia del artículo
Recibido: 12-07-2011
Correcciones:
Aceptado: 15-11-2011

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification – *formal methods.*

General Terms
Design, Reliability, Security, Verification.

Keywords
Correctness, Software Engineering, Formal Methods, Proof, Software Products.

Palabras clave
Exactitud, Ingeniería de Software, Métodos Formales, Pruebas, Productos Software.

ABSTRACT
The purpose of this research work is to examine: (1) why are necessary the formal methods for software systems today, (2) high integrity systems through the methodology C-by-C –Correctness-by-Construction–, and (3) an affordable methodology to apply formal methods in software engineering. The research process included reviews of the literature through Internet, in publications and presentations in events. among the Research results found that: (1) there is increasing the dependence that the nations have, the companies and people of software systems, (2) there is growing demand for software Engineering to increase social trust in the software systems, (3) exist methodologies, as C-by-C, that can provide that level of trust, (4) Formal Methods constitute a principle of computer science that can be applied software engineering to perform reliable process in software development, (5) software users have the responsibility to demand reliable software products, and (6) software engineers have the responsibility to develop reliable software products. Furthermore, it is concluded that: (1) it takes more research to identify and analyze other methodologies and tools that provide process to apply the Formal Software Engineering methods, (2) Formal Methods provide an unprecedented ability to increase the trust in the exactitude of the software products and (3) by development of new methodologies and tools is being achieved costs are not more a disadvantage for application of formal methods.

RESUMEN
El propósito de este trabajo de investigación consiste en examinar: (1) por qué son necesarios los Métodos Formales para los sistemas software de hoy, (2) la alta integridad de sistemas a través de la metodología C-by-C –Correctness-by-Construction– y (3) una metodología asequible para aplicar los Métodos Formales en la Ingeniería de Software. El proceso investigativo incluyó revisiones a la literatura a través de Internet, en publicaciones y en ponencias en eventos. Entre los resultados de la investigación se encontró que: (1) cada vez es mayor la dependencia que tiene las naciones, las empresas y las personas de los sistemas software, (2) existe una demanda creciente a la Ingeniería de Software para que incremente la confianza social en los sistemas software, (3) existen metodologías, como C-by-C, que pueden proporcionar ese nivel de confianza, (4) los Métodos Formales constituyen un principio de las Ciencias Computacionales que se puede aplicar a la Ingeniería de Software para llevar a cabo procesos confiables de desarrollo de software, (5) los usuarios del software tienen la responsabilidad de exigir productos software fiables y (6) los ingenieros de software tienen la responsabilidad de desarrollar productos software confiables. Además, se concluye que: (1) se necesita más investigación para determinar y analizar otras metodologías y herramientas que proporcionan procesos para aplicar los Métodos Formales en la Ingeniería de Software, (2) los Métodos Formales proporcionan una capacidad sin precedentes para incrementar la confianza en la exactitud de los productos software y (3) mediante el desarrollo de nuevas metodologías y herramientas se está logrando que los costos no sean más una desventaja de aplicación de los Métodos Formales.

1. INTRODUCCIÓN

Estudio de caso 1: Es justo después de la 17:00 horas de un viernes en cualquier ciudad del planeta. Los trabajadores en toda la ciudad regresan a sus casas para disfrutar de un fin de semana tranquilo. Algunos deciden tomar el metro, otros los taxis, sus autos o los autobuses y

otros simplemente caminan. Todos esperan una típica tarde de regreso a casa: largas filas y aglomeraciones en el metro, atascamientos en las autopistas y calles atestadas de personas, autos y autobuses. Para sorpresa y asombro de todos, el regreso está lejos de ser un viaje típico, de hecho toda la ciudad entrará en un paro total. ¿Por qué? La ciudad es Nueva York y la fecha es 14 de agosto de 2003; ese día, una gran falla en el fluido eléctrico afectó la mayor parte del noreste de los EE.UU. No sería hasta finales de noviembre de ese año cuando el grupo de trabajo, encargado de investigar la causa del apagón, pudo determinar que uno de los factores más significativos que lo provocó fue una "falla de software en First Energy Corp.", uno de los mayores proveedores de energía para esa zona del país [1].

Estudio de caso 2: Es febrero de 1991 en la Operación Tormenta del Desierto. EE.UU. emplea el sistema de misiles Patriot de la Armada para combatir el bombardeo de misiles SCUD, que lanza el ejército de Saddam contra sus tropas en Arabia Saudita a través de la frontera iraquí, lo mismo que contra Kuwait e Israel. Este sistema, diseñado originalmente como un sistema software de guía "track-via-missile" para proporcionar defensa de corto alcance contra los aviones y misiles soviéticos, tuvo que ser adaptado para trabajar como un arma defensiva de largo alcance para contrarrestar los misiles SCUD [2]. A pesar de los intentos por actualizar el software del sistema para que funcionará correctamente más allá de los límites originales de diseño, el 25 de febrero de 1991 una batería de misiles Patriot, que había funcionado por más de 100 horas, disparó un misil para interceptar un SCUD entrante y falló. Como resultado, el misil Patriot cayó en un cuartel del ejército matando a 28 estadounidenses [2].

Estudio de caso 3: El 25 de enero de 2003, el mundo de la informática experimentó los efectos del virus informático más rápido en la historia, el gusano Sapphire, también conocido como el gusano Slammer [3]. Este gusano, utilizó un desbordamiento de búfer en Microsoft SQL Server y SQL software Desktop Engine, que se había descubierto en julio de 2002 [3]. Los efectos de Sapphire incluyeron: parálisis en Corea del Sur, alteración de 13.000 cajeros automáticos en los EE.UU. y desactivación de los servicios de emergencia en Seattle, así como la interferencia en las elecciones americanas y provocó retrasos en el transporte aéreo [3, 4].

Estos estudios de caso son algunos ejemplos de fallas en la Ingeniería de Software, que resultaron en la pérdida de vidas humanas o en fracasos de misión crítica como la pérdida de ingresos a un nivel extremadamente alto, de una misión importante, o de la capacidad empresarial nacional. Los sistemas software, que se pensaba no eran sistemas críticos, se han integrado de tal forma en la cultura empresarial y social que el mundo, simplemente, no puede funcionar sin ellos, o con ellos en un estado degradado. Por ejemplo, el producto interno bruto de los EE.UU. depende de ellos en más del 98% [4]. Bill Wulf, presidente de la Academia Nacional de Ingeniería de EE.UU., en su discurso en la segunda Cumbre Nacional Anual de Software, comentó que en ese país existe un miedo latente, a punto de explotar, debido a la criticidad que arrastran los sistemas software, a la falta de calidad en el desarrollo y a la complacencia y reducción de la propia capacidad de la Ingeniería de Software nacional [5].

Una debilidad común en estos ejemplos, como lo destacó el Sr. Wulf, es la falta de verdaderos procesos de Ingeniería de Requisitos, de Verificación y Validación y de Análisis de los sistemas software. Esto, por supuesto, es sólo uno de tantos puntos débiles en los estudios de caso descritos. En un esfuerzo por hacer frente a las falencias antes mencionadas en la industria, Tony Hoare promulgó el "Gran Desafío" para la Investigación en las Ciencias Computacionales: El compilador de Verificación [6]. El Gran Desafío consiste en construir un compilador que utilice razonamiento lógico-matemático automatizado para comprobar, mediante Verificación formal, la precisión de los programas que compila. Para lograrlo, el Gran Desafió propone que se incluyan procesos de especificación formal en la Ingeniería de Requisitos [6]. El presente trabajo de investigación se centra en: (1) la Verificación formal del software, particularmente en por qué se requieren los Métodos Formales para los sistemas actuales, (2) en examinar la metodología Correctness-by-Construction C-by-C y (3) en analizar cómo aplicar los Métodos Formales a los sistemas software.

2. POR QUÉ LOS MÉTODOS FORMALES

Los métodos formales son técnicas basadas en matemática para especificar, desarrollar y verificar sistemas software y hardware [7]. Pueden proporcionar mecanismos para demostrar, matemáticamente, que un sistema o componente software hace lo que tiene que hacer, nada más y nada menos [7]. Estos mecanismos se proporcionan por medio de lenguajes de especificación formal como Z, B, Vienna Development Method VDM, Communicating Sequential Processes CSP, Larch y Formal Development Methodology FDM [8]. Estos lenguajes de especificación se utilizan normalmente para desarrollar modelos, libres de ambigüedades, que describen cómo funcionará el sistema [8]. Esos modelos, de sistemas específicos, se pueden utilizar para aplicarles pruebas matemáticas con el objetivo de asegurar su validez [8]. El resultado de la prueba puede aportar suficiente evidencia de Verificación como para que el desarrollador compruebe si el sistema, cuyo modelo fue especificado formalmente bajo un lenguaje de especificación, es correcto.

A pesar de que los Métodos Formales pueden parecer la "bala de plata" para la crisis del software, es necesario comprender que no existe tal cosa y que su utilización en la Ingeniería de Software puede incrementar el costo de los productos [7]. Las pruebas de correctitud requieren mucho tiempo para producir una utilidad limitada diferente a la de garantizar exactitud y, por tanto, normalmente se reservan para campos ingenieriles donde los beneficios de tales pruebas permiten que los Métodos Formales agreguen valor a los recursos [7].

Si los Métodos Formales se reservan para sistemas de seguridad y de misión crítica, entonces se debe identificar claramente cuáles son los sistemas que entran en esas categorías. Fácilmente podrían identificarse sistemas como los aeroespaciales y los militares –donde actualmente se aplican como se señala en el estudio de caso Sapphire. Muchos de los sistemas actuales poco a poco y en silencio arrastran procesos críticos de negocio, como los responsables de los procesos electorales, los sistemas bancarios y muchos otros. Por lo tanto, éstos también deben ingresar en la categoría de sistemas de misión crítica. Una vez que estos sistemas se identifiquen como de misión crítica se puede implementar un análisis de riesgos más apropiado, para comprender la relación entre el costo de verificar formalmente su funcionalidad frente a las consecuencias si se pierde esa funcionalidad debido a una falla del sistema o, en este caso, a un defecto de seguridad. A continuación se detalla una lista de las pérdidas de ingresos por hora estimadas en la industria debido a la inactividad de sus sistemas [4]:

- Transporte: US$ 28.000
- Ventas Teleticket: US$ 69.000
- Reservaciones Aéreas: US$ 89.000
- Hogar y decoración: US$ 113.000
- Pay-per-view: US$ 150.000
- Ventas con tarjetas de crédito: US$ 2.650.000
- Mercados Financieros: US$ 6.450.000

Como se observa claramente, sólo el impacto de una hora de inactividad, que fue aproximadamente la misma cantidad de tiempo antes que se produjeran las primeras respuestas al gusano Sapphire, puede ser muy costoso e incrementar significativamente la capacidad de la misión de impacto [3].

3. METODOLOGÍA CORRECTNESS-BY-CONSTRUCTION

C-by-C es una metodología de Ingeniería de Software desarrollada por Praxis High Integrity Systems, que se orienta principalmente hacia la industria aeroespacial y sus sistemas de seguridad crítica [1]. Sin embargo, esta metodología también tiene un impacto significativo en una variedad de industrias, en las que los sistemas de seguridad y de misión crítica son de gran interés [10]. La empresa Praxis HIS ha documentado [11] los logros de aplicación de la metodología de la siguiente manera: (1) Alta productividad: ahorros percibidos durante las fases de pruebas, (2) Baja cantidad de fallos: quedan muy pocos errores después de la entrega, (3) Software garantizado: emisión de estándares por parte de Praxis para garantizar el software, (4) Bajos costos de soporte: la calidad es fácil de mantener y (5) Clientes satisfechos: se alcanza con éxito los objetivos de los negocios subyacentes del cliente. En estudios de caso de la utilización de la metodología, como el Multi-Application Operating System Proyect, MULTOS –un sistema operativo de 100KLOC para tarjetas inteligentes–, los resultados obtenidos fueron: una tasa de 0,04 fallas por KLOC, 4 en total; altos niveles de productividad, 28 líneas de código por día y, lo más importante, el sistema satisface a sus usuarios, funciona bien y es fiable [12].

La metodología C-by-C se compone de siete principios fundamentales: (1) Esperar requisitos para el cambio, (2) Saber por qué se está probando, (3) Eliminar errores antes de la prueba: la prueba es la forma más cara de encontrar errores, (4) Escribir software que sea fácil de verificar: reducir el peso de la Verificación con un mayor esfuerzo al escribir buen código, (5) Desarrollo incremental: un poco de código, un poco de Verificación, (6) Algunos aspectos del desarrollo de software simplemente son difíciles: no existe ninguna bala de plata y no se debe esperar una herramienta o método para hacer que todo sea fácil y (7) El software no es útil por sí mismo: hay que desarrollar un software de acuerdo con y en respuesta a la arquitectura de la empresa, los procesos de negocio, los manuales de usuario y la documentación de diseño, entre muchas otras [11].

La metodología también tiene cinco características distintivas: (1) Utiliza Verificación estática, (2) Utiliza pequeños pasos de diseño verificables, (3) Genera evidencias de certificación y de evaluación como resultado secundario al proceso de desarrollo, (4) Aplica apropiadamente la formalidad y (5) Utiliza notaciones y herramientas adecuadas para el trabajo. La capacidad para eliminar los errores antes de la prueba, o Verificación estática, es lo más complicado de estos principios y características. La capacidad para llevar a cabo esta Verificación, es decir, técnicas de Verificación que no incluyen la ejecución de los programas, depende en gran medida del lenguaje o de la notación en la fase de desarrollo [13]. Praxis recomienda utilizar su herramienta SparkAda, especialmente desarrollada para este propósito y que es un subconjunto del lenguaje de programación Ada, en el que se eliminan ciertas funciones y capacidades para reducir considerablemente el número de posibles formas en que una determinada función se puede llevar a cabo, lo que permite eliminar ciertas ambigüedades [14]. Sin embargo, la capacidad única que proporciona SparkAda, para ayudar al demostrador de teoremas a verificar formalmente el código, es las notaciones semánticas estandarizadas. Estas anotaciones semánticas se basan en el lenguaje de especificación formal Z y en la teoría axiomática de conjuntos [14].

4. APLICACIÓN Y DISCUSIÓN DE RESULTADOS

La metodología Correctness by Construction requiere una estricta coherencia entre los procesos de Ingeniería de Software formalizados. Entre otras cosas, incluye los pasos típicos del desarrollo de sistemas, el análisis de la arquitectura empresarial y todas las formas de disposición del sistema. Para aplicar los métodos formales a través de C-by-C se debe seguir el proceso estrictamente y de principio a fin. No importa cuánto se intente aplicar los métodos formales en un análisis, si la Validación del sistema falla entonces todos los demás esfuerzos fueron infructuosos porque probablemente se tendrá que modificar totalmente. Por lo tanto, sin un proceso de Ingeniería de Software sólido no hay manera económica de aplicar los métodos formales.

De acuerdo con esta metodología, si los procesos de Ingeniería de Software formalizados cuentan con una

estricta coherencia entonces un enfoque válido para adoptar métodos formales consiste en aplicarlos de forma selectiva, mediante algún análisis de riesgo, a los componentes individuales del sistema más grande. Con esa estricta coherencia, los requisitos y los errores de diseño se reducen ampliamente y se validan fácilmente, lo que le permite a los desarrolladores poder concentrarse en la Verificación y en la Validación del código. Luego, aplicando el análisis de riesgos del sistema y los estudios de costo-beneficio, los desarrolladores pueden determinar cuáles son las funciones y las áreas del sistema que están en mayor riesgo, y que pueden beneficiarse de un análisis de métodos formales para garantizar su exactitud. Por ejemplo, un análisis de riesgo del sistema puede demostrar que las interfaces –hombre, máquina e Internet– son de alto riesgo, mientras que un algoritmo interno para ordenar los datos y su código de control asociado es de bajo riesgo. En este ejemplo, los desarrolladores concentrarán las herramientas de los métodos formales en las interfaces del sistema, mientras que, a pesar de mantener la calidad del software, el algoritmo interno y su código de control no tienen que ser analizados formalmente.

Si para los estudios de caso que se examinaron en la introducción se tratara de aplicar una medida asequible de métodos formales, primero se tendría que reformular el ciclo de vida de la Ingeniería de Software, hasta el punto del análisis de la arquitectura empresarial y del concepto de exploración/desarrollo. Suponiendo que el proceso transcurra sin incidentes a través del desarrollo de requisitos, se puede realizar entonces un análisis de riesgos del sistema para cada sistema en particular, con el objetivo de identificar las áreas con mayor riesgo de causar un fallo o degradación del sistema y en un nivel lo suficientemente alto como para causar fallas en la misión empresarial. En el caso del apagón de New York, debido a la importancia y al impacto que tiene una falla en el sistema eléctrico, se puede concluir que era aceptable un análisis formal para todo el sistema software. Para el caso de los misiles Patriot, el código de tiempo y otros códigos, relacionados con la trayectoria y el seguimiento de los objetivos, se consideraban de misión crítica en las partes del sistema y, por lo tanto, debieron analizarse formalmente para asegurar su corrección. Sin embargo, es importante señalar que este caso también pone de relieve una falla en el proceso de Ingeniería de Sistemas al momento de adaptar el sistema a un nuevo entorno, porque era necesario re-evaluar desde el principio el proceso de desarrollo y elicitar los requisitos nuevamente. Finalmente, el caso del gusano Sapphire es algo diferente y mucho más complejo.

No existe un sistema o persona responsable que pueda determinar dónde aplicar de mejor forma el análisis de los métodos formales. En este caso, las empresas, las organizaciones y los usuarios deben identificar sus procesos de misión crítica y las soluciones de TI que soportan esos procesos. Una vez identificados, deben exigir a sus proveedores de soluciones de TI que les proporcionen soluciones probadas y garantías del nivel de responsabilidad del desarrollador, en lo que tiene que ver con la exactitud de sus componentes de software. Los proveedores de soluciones de TI deben conocer a sus clientes, saber cómo se utilizarán sus soluciones y realizar análisis de riesgo y de costo-beneficio, para determinar dónde sería apropiado aplicar los métodos formales en el desarrollo de las soluciones individuales.

5. RECOMENDACIONES

La aplicación económica de los métodos formales, como los métodos inherentes a la metodología C-by-C, es posible y está disponible. Mediante una carga frontal, el esfuerzo de desarrollo para garantizar la validación de requisitos antes de su implementación y, aún más, para seleccionar componentes críticos para Verificación formal, reduce considerablemente el costo necesario para realizar la prueba y el mantenimiento del sistema. Sin embargo, antes de esto se pueden aplicar metodologías como C-by-C, para que los ingenieros cumplan estrictamente los procesos formalizados de ingeniería, de tal manera que se valide el sistema en cada fase durante todo el ciclo de vida.

Para lograrlo, los ingenieros de software y sus clientes tienen responsabilidades específicas. Primero, los ingenieros de software deben conocer a sus clientes, debe entender cómo se utilizan sus sistemas y qué partes son esenciales para sus clientes; deben respetar la ingeniería formal de los procesos de desarrollo; deben implementar sus sistemas con el análisis formal apropiado, tal como fue identificado en el análisis de riesgos y de costo-beneficio del sistemas; además, deben asegurar que sus clientes estén informados y que participen en el análisis de riesgos del sistema y del análisis de equilibrio. Segundo, los clientes deben conocer sus empresas y qué partes de sus procesos de negocio son los más críticos para lograr o mantener el éxito de la misión; deben asegurarse que los sistemas software que utilizan son construidos con las fortalezas apropiadas para soportar sus procesos de negocio críticos y deben exigir garantías de que el software cumple con las necesidades documentadas.

6. CONCLUSIONES

Los métodos formales, cuando se utiliza adecuadamente, proporcionan una capacidad sin precedentes para crear confianza en la exactitud de un sistema o componente. A través del desarrollo de metodologías como C-by-C y de herramientas como SparkAda, se reducen los costos cada vez más, lo que es ventajoso para la aplicación de estos métodos en el ciclo de vida de la Ingeniería de Software.

Debido a que el nivel crítico de los sistemas de TI aumenta constantemente, de la misma forma se debe incrementar la confianza en que estos sistemas funcionen como se espera. Los clientes de sistemas software, como el gobierno, las empresas y todos los demás usuarios de TI, deben exigir que sus sistemas se entreguen con un nivel de corrección o de confianza probado y acorde con la criticidad de la función que realizan. Los ingenieros de software deben contar con las herramientas y las capacidades para brindar ese nivel de confianza.

La actual dependencia de los sistemas software ha alcanzado un nivel extremadamente alto, al punto que se están re-evaluando los argumentos del pasado acerca de que los métodos formales eran demasiado costosos en tiempo y dinero. El hecho es que cuesta mucho más, en tiempo de inactividad y en mantenimiento, no probar formalmente la corrección de estos sistemas [7].

Como se discutió en los estudios de caso, los sistemas involucrados son fundamentales para la seguridad de la vida, el éxito de la misión, el éxito de un negocio esencial o la capacidad nacional. Estos sistemas se entregaron sin un nivel medible de corrección y, finalmente, fallaron causando impactos significativos a sus propietarios. Los nuevos sistemas deben requerir, medir y evaluar la corrección, lo que sólo se puede lograr verdaderamente a través de una adecuada aplicación de los métodos formales.

La habilidad y la capacidad para aplicar los métodos formales a la Ingeniería de Software ya están disponible para los ingenieros, y se está reduciendo la relación costo-beneficio de la aplicación de los mismos. Hasta que disminuya la dependencia de los sistemas software para el cumplimiento de los objetivos estratégicos empresariales o metas personales, seguirá creciendo la necesidad de confiar en estos sistemas.

7. REFERENCIAS

[1] Verton, D. 2003. Software failure cited in August blackout investigation. ComputerWorld Online. [April 2011].

[2] Morgan, T. & Roberts, J. 2002. An Analysis of the Patriot Missile System. Online [April 2011].

[3] Moore, D. et al. 2003. The Spread of the Sapphire/Slammer Worm. Online [April 2011].

[4] Chen, J. 2004. The Enterprise: Unwired. Center for National Software Studies. Online [May 2011].

[5] Wulf, W. A. 2004. Do we have a "creeping crisis" and are we losing our [Edge]? Proceedings of the 2nd National Software Summit, Reston, USA. Online [May 2011].

[6] Hoare, C. A. R. 2003. The Verifying Compiler: a Grand Challenge for Computing Research. Journal of the ACM, 50(1), 63-69.

[7] Serna, M. E. 2010. Métodos Formales e Ingeniería de Software. Revista Virtual Universidad Católica del Norte, 30, 158-184.

[8] Wing, M. J. 1990. A specifier's introduction to formal methods. Computer, 23(9), 8-23.

[9] Amey, P. 2002. Correctness by Construction: Better can also be cheaper. Crosstalk Magazine, March, 24-28.

[10] Chapman, R. & Hall, A. 2002. Correctness by Construction: Developing a Commercial Secure System. IEEE Software, Jan/Feb, 18-25.

[11] Praxis High Integrity Systems.2008. Correctness by Construction Approach. Online [April 2011].

[12] MAOSCO Ltd. 2008. MULTOS Case Study. Online [April 2011].

[13] Chapman, R. 2004. SPARK, an Intensive Overview. Proceedings of the SIGADA Conference, San Diego, USA. 14-18 November.

[14] Knight, J. C. 2004. Correctness by Construction: An Introduction to SPARK Ada. Online [April 2011].