Revista Virtual Universidad Católica del Norte
ISSN: 0124-5821
asanchezu@ucn.edu.co
Fundación Universitaria Católica del Norte
Colombia

Serna Montoya, Edgar
Métodos formales e Ingeniería de Software
Revista Virtual Universidad Católica del Norte, núm. 30, mayo-septiembre, 2010, pp. 1-26
Fundación Universitaria Católica del Norte
Medellín, Colombia

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

Métodos formales e Ingeniería de Software[1]

Formal Methods and Software Engineering

Méthodes formelles et génie logiciel

Edgar Serna Montoya
Candidato a Magíster en Ingeniería de Sistemas
Universidad Nacional, Medellín
Grupo de investigación SISCO (FUNLAM)
Profesor auxiliar Fundación Universitaria Luis Amigó
edgar.sernamo@amigo.edu.co, eserna@gmail.com

Tipo de artículo: Revisión resultado de investigación
Recepción: 2010-04-08
Revisión: 2010-04-19
Aprobación: 2010-05-15

Contenido

[1] Artículo resultado del proyecto de investigación: "Estructuración de una metodología genérica para la realización de pruebas de Caja Negra en los sistemas de información", realizado por el grupo de investigación SISCO de la Facultad de Ingenierías de las Fundación Universitaria Luis Amigó, en la línea de investigación: "Sistemas de Información y Sociedad del conocimiento - SISCO".

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

Resumen. Los métodos formales surgieron como puntos de vista analíticos con los que es posible verificar el desarrollo de sistemas mediante la lógica y las matemáticas, lo que aporta grandes ventajas para mejorar la calidad de los programas y por tanto la Ingeniería de *Software*. En este campo del conocimiento, la especificación formal es una de las más importantes fases del ciclo de vida, labor que requiere mucho cuidado ya que su función es garantizar que tanto el funcionamiento como el desempeño del programa sean correctos, bajo cualquier situación. En el futuro, los métodos formales deberían estar presentes como principios esenciales en el desarrollo de *software*, ya que se convierten en la base para aplicar las técnicas de prueba y, dado su principio matemático, en potencialmente automatizables.

Palabras clave: Especificación, Ingeniería de *software*, Métodos formales, Validación, Verificación.

Abstract. Formal methods have emerged as analytical points of view that allow checking systems development through logic and mathematics, providing significant benefits to improve the quality of programs, and therefore the Software Engineering. In this field of knowledge, formal specification is one of the most important life cycle stages, a task that requires great care because its function is to ensure that both the operational and program performance will be correct in any situation. In the future, formal methods should be present as essential principles in software development since they become the basis for applying the testing techniques and, because of its mathematical principle, they are potentially automatable.

Keywords: Formal Methods, Software Engineering, Specification, Validation, Verification.

Résumé. Les méthodes formelles ont émergé comme points de vue analytiques qui rendent possible la vérification du développement de systèmes grâce à la logique et les mathématiques, en apportant grandes avantages pour l'amélioration de la qualité des logiciels et par conséquent la Génie Logiciel. Dans ce champ de connaissance, la spécification formelle est une des plus importantes phases de cycle de vie, travail qui exige beaucoup d'attention puisque son fonction est garantir que le fonctionnement et la performance du logiciel soient correctes dans tous les situations. Dans le futur, les méthodes formelles devraient rester comme principes essentiales dans le développement de logiciels, parce qu'ils deviennent la référence pour appliquer las techniques de preuve et,

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

à cause de son principe mathématique, ils ont la potentialité de être automatisées.

Mots-clés: Spécification, Génie Logiciel, Méthodes Formelles, Validation, Vérification

1. Introducción

El concepto de métodos formales involucra una serie de técnicas lógicas y matemáticas con las que es posible especificar, diseñar, implementar y verificar los sistemas de información (Monin, 2003). Hasta hace pocos años el desarrollo industrial de sistemas, usando dichas técnicas, era considerado un complejo ejercicio teórico e inviable en problemas reales. Las notaciones "*oscuras*" tomadas de la lógica, sin las suficientes herramientas de soporte, no podían competir con los lenguajes de cuarta generación y los ambientes de desarrollo rápido de los años 80 y 90; sin embargo, el mundo industrial cambió su actitud frente a los métodos formales. Por una parte, porque los lenguajes de especificación son cada vez más cercanos a los lenguajes de programación, las técnicas de desarrollo se adaptan mejor a los nuevos paradigmas, y las herramientas comerciales que soportan la calidad del *software* son métodos que se distribuyen comercialmente (Sobel & Clarkson, 2002). Además, porque a medida que los sistemas informáticos crecen en complejidad las pérdidas causadas por fallas son cada vez mayores. Cuando "*corrección certificada*" se traduce en dinero, los métodos formales atraen a la industria, ya que su aplicación ayuda a lograr los estándares de calidad que la sociedad exige.

La importancia de los métodos formales en la Ingeniería de *Software* se incrementó en los últimos años: se desarrollan nuevos lenguajes y herramientas para especificar y modelar formalmente, y se diseñan metodologías maduras para verificar y validar. Los modelos que se diseñan y construyen de esta forma, desde las fases iniciales del desarrollo de *software*, son esenciales para el éxito del futuro proyecto; ya que en la actual Ingeniería de *Software* constituyen la base que sustenta las subsiguientes fases del ciclo de vida, y porque los errores surgidos en ella tienen gran impacto en los costos del proyecto (Perry, 2006).

Desde hace varias décadas se utilizan técnicas de notación formal para modelar los requisitos, principalmente porque estas notaciones se pueden

verificar “*fácilmente*” y porque, de cierta forma, son más “*comprensibles*” para el usuario final (Felleisen et al., 2001). Además, el paradigma Orientado por Objetos en la programación parece ser el más utilizado en la industria, y una forma de incrementar la confiabilidad del *software* y, en general de los sistemas, es utilizar los métodos formales en la ingeniería aplicada (Kuhn et al., 2002).

Pero, aunque los métodos formales tienen un amplio recorrido, y su utilidad y eficiencia en desarrollos críticos están demostradas, todavía falta más trabajo para que la mayoría de ingenieros los conozcan y apliquen. En parte esta labor la deben realizar las facultades que deben incluirlos en sus contenidos académicos; los profesores, quienes se deben formar e investigar mejor esta área del conocimiento; y los estudiantes, quienes deben tener una formación más solida en matemáticas y lógica (Dijkstra, 1989). Sólo con un trabajo mancomunado se podrá lograr que la labor del ingeniero de *Software* sea verdadera Ingeniería, y que el usuario final acepte los productos *software* como confiables.

Los propósitos de los métodos formales son: sistematizar e introducir rigor en todas las fases de desarrollo de *software*, con lo que es posible evitar que se pasen por alto cuestiones críticas; proporcionar un método estándar de trabajo a lo largo del proyecto; constituir una base de coherencia entre las muchas actividades relacionadas y, al contar con mecanismos de descripción precisos y no ambiguos, proporcionar el conocimiento necesario para realizarlas con éxito.

Los lenguajes de programación utilizados para desarrollar *software* facilitan la sintaxis y la semántica precisas para la fase de implementación, sin embargo, la precisión en las demás fases debe provenir de otras fuentes. Los métodos formales se inscriben en una amplia colección de formalismos y abstracciones, cuyo objetivo es proveer un nivel de precisión comparable para las demás fases. Si bien esto incluye temáticas que actualmente están en desarrollo, algunas metodologías ya alcanzaron un nivel de madurez suficiente para que se utilicen ampliamente.

Existe una tendencia discutible a fusionar la matemática discreta y los métodos formales en la Ingeniería de *Software*. Efectivamente, en muchas temáticas de ésta se apoya la Ingeniería de *Software*, y no es posible ni deseable evitarlas cuando se trabaja con métodos formales; pero no es posible pretender que por el simple hecho de tomar algunos enfoques de la Matemática Discreta y emplearlos en aquella, se apliquen métodos formales.

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

El principal objetivo de la Ingeniería de *Software* es desarrollar sistemas de alta calidad y, con los métodos formales, en conjunción con otras aéreas del conocimiento, se puede lograr.

En este artículo se describe esa conjunción, y se estructura de la siguiente forma: en la segunda parte se definen los métodos formales; en la tercera se describe su utilidad en la Ingeniería de *Software*, su base matemática y se presenta un ejemplo ilustrativo; en la cuarta se detallan sus ventajas, y en la quinta se presentan las tendencias de la investigación en esta área del conocimiento.

2. ¿Qué son los métodos formales?

Aunque el término se utiliza ampliamente, no parece existir una clara definición acerca de qué son (Hehner, 2006). En muchas ocasiones el término se emplea simplemente para indicar la utilización de un lenguaje de especificación formal, pero no incluye una descripción de cómo se utiliza o la extensión de su uso. Incluso el término "*lenguaje de especificación formal*" no es preciso, ya que no es claro si los lenguajes "*específicamente*" tienen como objetivo diseñar la implementación, en lugar de especificar el problema. En este trabajo se utiliza el término "*métodos formales*" para describir cualquier enfoque que utilice un lenguaje de especificación formal, pero que describa su función en el proceso del desarrollo de *software*. Utilizar métodos formales necesariamente implica un proceso formal de refinamiento, razonamiento y prueba. El término "*lenguaje de especificación formal*" se utiliza aquí para referenciar un lenguaje en el que es posible especificar completamente la funcionalidad de todo o parte de un programa, de tal forma que sea susceptible de razonamiento formal, y que se fundamenta en una sólida base matemática, más en que si es lo suficientemente abstracto.

La base teórica de los lenguajes de especificación formal varía considerablemente, la más común es la de los métodos formales. Los lenguajes de este tipo más utilizados en la industria son *The Vienna Development Method* -VDM- (Jones, 1990) y Zed -Z-, que se basa en la teoría de conjuntos y la lógica de predicados de primer orden de Zermelo Fraenkel (Spivey, 1992). VDM se utilizó primero en proyectos de gran tamaño, pero Z parece ganar popularidad recientemente. Aunque su sintaxis es diferente, ambos lenguajes están estrechamente relacionados y se basan, principalmente, en la teoría matemática de conjuntos (Hayes et al., 1994). Según Gödel (1934), un sistema matemático formal es "*un sistema de*

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

símbolos con sus respectivas reglas de uso", todas recursivas. Este último requisito es importante ya que permite desarrollar programas para comprobar si un determinado conjunto de reglas se aplica correctamente (Zeugmann & Zilles, 2008); sin embargo, garantizar que se puede lograr utilizando métodos formales tiene un precio, ya que para muchas aplicaciones resulta extremadamente costosos.

Las nociones fundamentales de especificación y verificación formal de programas se desarrollaron en los años 70, entre otros por Hoare (1969) y Dijkstra (1976): "*el primer paso en la construcción de un sistema es determinar un modelo abstracto del problema que se va a resolver sobre el cual razonar*". Como no todos los aspectos del problema son relevantes en el sistema que se desarrolla es necesaria una observación cuidadosa para abstraer sus características más importantes, y utilizar un lenguaje formal para describir el modelo -especificación formal-. En las ciencias computacionales los "*métodos formales*" adquieren un sentido más lineal, y se refieren específicamente al uso de una notación formal para representar modelos de sistemas, y en un sentido aún más estrecho se refieren a la formalización de un método para desarrollar sistemas (Flynn & Hamlet, 2006).

El término "*métodos formales*" se refiere entonces al uso de técnicas de la lógica y de la matemática discreta para especificar, diseñar, verificar, desarrollar y validar programas. La palabra "*formal*" se deriva de la lógica formal, ciencia que estudia el razonamiento desde el análisis formal -de acuerdo con su validez o no validez-, y omite el contenido empírico del razonamiento para considerar sólo la forma -estructura sin materia-. Los métodos formales más rigurosos aplican estas técnicas para comprobar los argumentos utilizados para justificar los requisitos, u otros aspectos de diseño e implementación de un sistema complejo.

En la lógica formal como en los métodos formales el objetivo es el mismo, "*reducir la dependencia de la intuición y el juicio humanos para evaluar argumentos*" (Kiniry & Zimmerman, 2008). Los métodos menos rigurosos enfatizan en la formalización y renuncian a la computación, definición que implica un amplio espectro de técnicas, y una gama igualmente amplia de estrategias. La interacción de las técnicas y estrategias de muchos métodos formales, en determinados proyectos, se limita por el papel que interpretan y por los recursos disponibles para su aplicación (García et al., 2006).

3. Los métodos formales en la Ingeniería de *Software*

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

En Ingeniería de *Software*, un método formal es un proceso que se aplica para desarrollar programas, y que explota el poder de la notación y de las pruebas matemáticas. Además, los requisitos, la especificación y el sistema completo deben validarse con las necesidades del mundo real (Kuhn et al., 2002). En la Ingeniería de *Software* los métodos formales se utilizan para:

- *Las políticas de los requisitos*. En un sistema seguro se convierten en las principales propiedades de seguridad que éste debe conservar -el modelo de políticas de seguridad formal-, como confidencialidad o integridad de datos.
- *La especificación*. Es una descripción matemática basada en el comportamiento del sistema, que utiliza tablas de estado o lógica matemática. No describe normalmente al *software* de bajo nivel, pero sí su respuesta a eventos y entradas, de tal forma que es posible establecer sus propiedades críticas.
- *Las pruebas de correspondencia entre la especificación y los requisitos*. Es necesario demostrar que el sistema, tal como se describe en la especificación, establece y conserva las propiedades de las políticas de los requisitos. Si están en notación formal se pueden diseñar pruebas rigurosas manuales o automáticas.
- *Las pruebas de correspondencia entre el código fuente y la especificación*. Aunque muchas técnicas formales se crearon inicialmente para probar la correctitud del código, pocas veces se logra debido al tiempo y costo implicados, pero pueden aplicarse a los componentes críticos del sistema.
- *Pruebas de correspondencia entre el código máquina y el código fuente*. Este tipo de pruebas raramente se aplica debido al costo, y a la alta confiabilidad de los compiladores modernos.

Por tanto, los métodos formales en la Ingeniería de *Software* son técnicas matemáticamente rigurosas que se utilizan para describir las propiedades del sistema, y proporcionan marcos de referencia para especificarlo, desarrollarlo y verificarlo de forma sistemática, en lugar de hacerlo *ad hoc* (Jones et al., 2006). Un método es formal si posee una base matemática estable, normalmente soportada por un lenguaje de especificación formal, que permite definir, de manera precisa, nociones como consistencia y completitud y, aún más relevantes, la especificación, la implementación y la correctitud. Al utilizar notaciones y lenguajes formales es posible estructurar claramente los requisitos del sistema, y generar las especificaciones que

permitan definir su comportamiento de acuerdo con el "*qué debe hacer*", y no con el "*cómo lo hace*" (Bolstad, 2004).

Matemáticamente rigurosas significa que las especificaciones en los métodos formales son declaraciones gramaticalmente correctas en una lógica matemática, y que la verificación formal es una deducción rigurosa en ella. Es decir, que cada paso se deriva de una regla de inferencia que se puede comprobar mediante un proceso automático (Liu & Venkatesh, 2005). Los métodos formales en la Ingeniería de *Software* difieren en la forma como se aplican, y en los tiempos necesarios para cada fase del ciclo de vida; su estructuración y utilización requiere mayor tiempo y trabajo para desarrollar la especificación, pero su utilización garantiza, más "*formalmente*", que la construcción de los diseños es correcta.

3.1 Fundamentos matemáticos de los métodos formales

La matemática es una ciencia de la cual se pueden aprovechar muchas de sus propiedades para el desarrollo de los grandes y complejos sistemas actuales, en los que, idealmente, los ingenieros deberían estructurar su ciclo de vida de la misma forma que un matemático se dedica a la matemática aplicada: "*presentar la especificación matemática del sistema y elaborar una solución con base a una arquitectura de software que haga posible su implementación"* (Drechsler, 2004). Aunque la especificación matemática de sistemas no es tan concisa como las expresiones matemáticas simples, dado que los sistemas *software* son mucho más complejos, no sería realista pensar que es posible especificarlos de la misma forma matemática. En todo caso, la ventaja de utilizar matemáticas en la Ingeniería de *Software* radica en que "*suaviza*" la transición entre sus actividades, ya que es posible expresar la especificación funcional, el diseño y el código de un programa, mediante notaciones. Esto se logra gracias a que la propiedad fundamental de la matemática es la abstracción, una excelente herramienta para modelar debido a que es exacta, y a que ofrece pocas probabilidades de ambigüedad, lo que permite verificar matemáticamente las especificaciones, y buscar contradicciones e incompletitudes.

Con las matemáticas es posible representar, de forma organizada, los niveles de abstracción en la especificación del sistema; es una buena herramienta para modelar, ya que facilita el diseño del esquema principal de la especificación; permite a analistas e ingenieros verificar la funcionalidad de esa especificación y a los diseñadores ver los detalles suficientes para llevar a cabo su tarea desde las propiedades del modelo. También

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

proporciona un nivel elevado de verificación, ya que para probar que el diseño se ajusta a la especificación y que el código refleja exactamente el diseño, se puede utilizar una demostración matemática (Langari & Pidduck, 2005). Como se expresó antes, el término formal se caracteriza por utilizar una serie de categorías de modelos o formalismos matemáticos, estos modelos no sólo incluyen los fundamentos matemáticos, también a los procesos para la producción de *software*. Entre estos fundamentos básicos se cuentan nociones como lenguajes formales, sintaxis, semántica, modelos, gramática, teorías, especificación, verificación, validación y pruebas; temáticas a las que deberían tener acceso los Ingenieros de *Software*, para contar con bases sólidas al momento de seleccionar un determinado formalismo (Wirsing, 1991), (Lewis & Papadimitriou, 1997), (Meldenson, 1997), desde los cuales es posible, de forma más acertada, tener los primeros acercamientos a conceptos como requisitos, especificación formal e informal, validación, validez, completitud, correctitud y nociones sobre pruebas del *software*.

Los formalismos matemáticos de los métodos formales son:

- *Lógica de primer orden y teoría de conjuntos* (Manna & Waldinger, 1985). Se utilizan para especificar el sistema mediante estados y operaciones, de tal forma que los datos y sus relaciones se describan detalladamente, sus propiedades se expresen en lógica de primer orden, y la semántica del lenguaje tenga como base la teoría de conjuntos. En el diseño y la implementación del sistema los elementos, descritos matemáticamente, se pueden modificar, pero siempre conservarán las características con las que se especificaron inicialmente. En esta lógica se establece, mediante diversos sistemas de deducción, la definición del lenguaje empleado -lógica de predicados de primer orden-, y de la prueba. Tanto las reglas de inferencia, el diseño de las pruebas, como el estudio de las teorías de igualdad e inducción, representan aspectos claves de este formalismo (Milner, 1990), (Mendelson, 1997).

- *Algebraicos y de especificación ecuacional* (Ehrig & Mahr, 1990). Describen las estructuras de datos de forma abstracta para establecer el nombre de los conjuntos de datos, sus funciones básicas y propiedades -mediante fórmulas ecuacionales-, en las que no existe el concepto de estado modificable en el tiempo. Tanto el cálculo ecuacional -pruebas mediante ecuaciones-, como los sistemas de reescritura, constituyen el soporte para realizar las deducciones necesarias (Duffy, 1991). También se utilizan para especificar sistemas de información a gran escala, tarea

en la que es necesario estudiar los mecanismos de extensión, de parametrización y de composición de las especificaciones (Ehrig & Mahr, 1990).

- *Redes de Petri* (Murata, 1989), (Reising, 1991), (Manna & Pnueli, 1992), (Bause & Kritzinger, 2002), (Desel & Esparza, 2005), (Juhás et al., 2007). Establecen el concepto de estado del sistema mediante lugares que pueden contener marcas, y hacen uso de un conjunto de transiciones -con pre y post-condiciones-, para describir cómo evoluciona el sistema, y cómo produce marcas en los puntos de la red. Utilizan características semánticas de entrelazado o de base en la concurrencia real. En la primera, los procesos paralelos no deben ejecutar las instrucciones al mismo tiempo, mientras que en la segunda existe esa posibilidad. Estas caracterizaciones son fundamentales para poder representar conceptos como estado y transición en los sistemas.

- *Lógica temporal* (Manna & Pnueli, 1992). Se utiliza para describir los sistemas concurrentes y reactivos, poseen una amplia noción de tiempo y estado. Sus especificaciones describen las secuencias válidas de estados -incluyendo los concurrentes- en un sistema específico. Para aplicar este formalismo es necesario establecer inicialmente una clasificación de los diferentes sistemas de lógica temporal de acuerdo con los criterios de proposicionalidad vs. primer orden, tiempo lineal vs. tiempo ramificado, de evaluación instantánea o por intervalos, y tiempo discreto vs. tiempo continuo. Una vez que se establece la formalización del tiempo es posible estudiar la aplicación del formalismo.

Los métodos formales proporcionan un medio para examinar simbólica y completamente el diseño digital –sea *hardware* o *software*-, y establecer las propiedades de correctitud o seguridad. Sin embargo, en la práctica raramente se logra -excepto en los componentes de seguridad de sistemas críticos-, debido a la enorme complejidad de los sistemas reales.

Se ha experimentado con los métodos formales en varios proyectos para lograr el desarrollo de sistemas reales con calidad:

- Para automatizar la verificación tanto como sea posible (Wunram, 1990).
- En requisitos y diseños de alto nivel, en los que la mayor parte de los detalles se abstraen de diversas formas (Rushby, 1993).
- Sólo a los componentes más críticos (Lutz & Ampo, 1994).

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

- Para analizar modelos de *software* y *hardware*, en los que las variables se discretizan y los rangos se reducen drásticamente (McDermid, 1995).
- En el análisis de los modelos de sistemas de manera jerárquica, de tal forma que se aplique el "*divide y vencerás*" (Zhang, 1999).

Aunque el uso de la lógica matemática sea un tema de unificación en los métodos formales, no es posible indicar que un método sea mejor que otro; cada dominio de aplicación requiere métodos de modelado diferentes, lo mismo que el acercamiento a las pruebas. Además, en cada dominio en particular, las diferentes fases del ciclo de vida se pueden lograr mejor con la aplicación combinada de diferentes técnicas y herramientas formales (Pressman, 2004).

3.2 Un ejemplo ilustrativo

El siguiente ejemplo ilustra la utilización de los métodos formales en la Ingeniería de *Software*. Se trata de especificar de forma diferente a la tradicional, y de utilizar el lenguaje matemático para hacerlo con los requisitos de los sistemas. El ejemplo muestra la especificación de las funciones de inserción y borrado en una estructura pila.

Tabla 1. Especificación en lógica formal de las funciones de insertar y borrar elementos en una estructura pila

▪ **L.push**(e:stelement): Inserta *e* después del último elemento insertado
▪ **PRE**: ∃L ∧ L ≠ {NULL} ∧ L = {PRF} ∧ tamaño(L) = n ∧ ¬existe(L,clave(e))
▪ **POST**: L = {PRF,e} ∧ tamaño(L) = n + 1
▪ **PRE**: ∃L ∧ L = {NULL}
▪ **POST**: L = {e} ∧ n = 1
▪ **L.pop**(): borra el elemento recientemente insertado
▪ **PRE**: ∃L ∧ L = {PRF, e} ∧ tamaño(L) = n
▪ **POST**: L = {PRF} ∧ tamaño(L) = n – 1

Nótese como, en lugar de utilizar el lenguaje natural para especificar las funciones, se recurre a la notación Z de la lógica formal para hacerlo. Haciéndolo de esta forma no se tienen ambigüedades y es posible automatizarlas.

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

4. Ventajas de los métodos formales

La utilidad de los métodos formales en la Ingeniería de *Software* es un tema que se debate desde hace varias décadas. Recientemente, con el surgimiento de la Ingeniería de Conocimiento en la Sociedad de la Información, y la aplicación de los métodos formales en los procesos industriales, nuevamente surge el debate. A continuación se detallan algunas de las ventajas de utilizarlos.

Especificación. Uno de los problemas más ampliamente reconocido en el desarrollo de *software* es la dificultad para especificar claramente el comportamiento que se espera del *software*, problema que se agudiza con el actual desarrollo basado en componentes, ya que el ingeniero tiene sólo una descripción textual de los requisitos, procedimientos, entradas permitidas y salidas esperadas. Asegurar que este tipo de *software* sea seguro es un problema, no sólo por su tamaño y complejidad, sino porque el código fuente no suele estar disponible para los componentes que se adquieren. Una forma de ofrecer "*garantía rigurosa*" del producto es definir, con precisión, el comportamiento esperado del *software* (NASA, 1995). La especificación formal proporciona mayor precisión en el desarrollo de *software*, y los métodos formales brindan las herramientas que pueden incrementar la garantía buscada. Desarrollar formalmente una especificación requiere conocimiento detallado y preciso del sistema, lo que ayuda a exponer errores y omisiones, y por lo que la mayor ventaja de los métodos formales se da en el desarrollo de la especificación (Clarke & Wing, 1996). En la formalización de la descripción del sistema se detectan ambigüedades y omisiones, y una especificación formal puede mejorar la comunicación entre ingenieros y clientes.

Los métodos formales se desarrollaron principalmente para permitir un mejor razonamiento acerca de los sistemas y el *software*. Luego de diseñar una especificación formal, se puede analizar, manipular y razonar sobre ella, de la misma forma que sobre cualquiera expresión matemática. Una diferencia significativa entre una especificación formal de *software* y una expresión matemática de álgebra o cálculo, es que típicamente es mucho más grande -a menudo cientos o miles de líneas-. Para tratar con el tamaño y la complejidad de estas expresiones se desarrollan herramientas de *software* que se pueden agrupar en dos grandes categorías: *probadoras de teoremas* y *verificadoras de modelos* (Anderson et al, 1998).

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

Las primeras ayudan al usuario a diseñar pruebas, generalmente para demostrar que las especificaciones cumplen con las propiedades deseadas. Para usarlas, es necesario que el ingeniero tenga cierto grado de habilidad y conocimiento, pero pueden manipular especificaciones muy grandes con propiedades complejas. Los desarrollos recientes en métodos formales introdujeron las verificadoras de modelos, que exploran, hasta cierto punto, todas las posibles ejecuciones del programa especificado. Pueden verificar si cumple con una propiedad específica mediante la exploración de todas las posibles ejecuciones, o producen contraejemplos en los que la propiedad no se cumple. Aunque estas herramientas pueden ser completamente automáticas, no pueden resolver problemas tan grandes o variados como las probadoras de teoremas. Las herramientas más sofisticadas combinan aspectos de ambas, aplican las verificadoras a algunas partes de la especificación, pero confían en el usuario para probar las propiedades complicadas.

Verificación. Para asegurar la calidad de los sistemas es necesario probarlos, y para asegurar que se desarrollan pruebas rigurosas se requiere una precisa y completa descripción de sus funciones, incluso cuando en la especificación se utilicen métodos formales. Una de las aplicaciones más interesantes de éstos es el desarrollo de herramientas, que pueden generar casos de prueba completos desde la especificación formal.

Aunque gran número de herramientas para automatizar pruebas se encuentran disponibles en el mercado, la mayoría automatiza sólo sus aspectos más simples: generan los datos de prueba, ingresan esos datos al sistema y reportan resultados. Definir la respuesta correcta del sistema, para un determinado conjunto de datos de entrada, es una tarea ardua, que la mayoría de herramientas no puede lograr cuando el comportamiento del mismo se especifica en lenguaje natural. Debido a que la respuesta esperada del sistema se puede determinar únicamente mediante la lectura de la especificación, los ingenieros esperan que a las pruebas automatizadas se les adicione este faltante y crítico componente (Dan & Aichernig, 2002). La gran ventaja de las herramientas, que generan pruebas con base en los métodos formales, es que la especificación formal describe matemáticamente el comportamiento del sistema, desde la que se puede generar la respuesta a un dato de entrada en particular, es decir, la herramienta puede generar casos de prueba completos.

Las técnicas de verificación formal dependen de especificaciones matemáticamente precisas y, desde un punto de vista costo-beneficio,

generar pruebas desde la especificación puede ser uno de los usos más productivos de los métodos formales. Aproximadamente la mitad del tiempo del equipo de trabajo, en un desarrollo típico de *software* comercial, se invierte en esfuerzo para desarrollar pruebas, e “*incluso con este nivel de esfuerzo sólo se eliminan los errores más evidentes*” (Saiedian & Hinchey, 1996). Algunas mediciones empíricas demuestran que las pruebas, generadas con herramientas automatizadas, ofrecen una cobertura tan buena o mejor que la alcanzada por las manuales, por lo que los ingenieros pueden elegir entre producir más pruebas en el mismo tiempo, o reducir el número de horas necesarias para hacerlas (Gabbar, 2006).

Validación. Mientras que la verificación se puede realizar semiautomáticamente y las pruebas mecánicamente, la validación es un problema diferente. Una diferencia específica entre verificación y validación es que la primera responde a si “*se está construyendo el producto correctamente*”, y la segunda a si "*se está construyendo el producto correcto*" (Dasso & Funes, 2007). En otras palabras, la verificación es el conjunto de actividades que aseguran que el *software* implementa correctamente una función específica, y la validación es un conjunto de actividades diferentes que aseguran que el *software* construido corresponde con los requisitos del cliente (Amman & Offutt, 2008).

Desde el conjunto de requisitos es posible verificar, formal o informalmente, si el sistema los implementa; sin embargo, la validación es necesariamente un proceso informal. Sólo el juicio humano puede determinar si el sistema que se especificó y desarrolló es el adecuado para el trabajo. A pesar de la necesidad de utilizar este juicio en el proceso de validación, los métodos formales tienen su lugar, especialmente en grandes y complejas aplicaciones, como en modelado y simulación. Una de sus aplicaciones más prometedoras es en el modelado de requisitos, ya que, al diseñarlos formalmente, el teorema provisto en la herramienta de prueba se puede utilizar para explorar sus propiedades, y a menudo detectar los conflictos entre ellos. Este método no sustituye al juicio humano, pero puede ayudar a determinar si se especificó el "*sistema correcto*", por lo que es más fácil determinar si las propiedades deseadas se mantienen (Flynn & Hamlet, 2006).

Una diferencia significativa entre validar sistemas de modelado y simulación, y los de control o cálculo, es que los primeros tienen dos tipos de requisitos de validación: deben modelar y predecir el comportamiento de alguna entidad del mundo real, problema que se conoce como "*validación*

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

operacional", y deben "*validar el modelo conceptual*", para asegurar que la hipótesis en la que se sustenta es correcta, y que su lógica y estructura son adecuadas para el modelo que se propone (Sargent, 1999). Debido a que el modelo conceptual describe lo que debe representar la simulación, es necesario incluir supuestos acerca del sistema, su entorno, las ecuaciones, los algoritmos, los datos, y de las relaciones entre las entidades del modelo. Aunque los algoritmos y las ecuaciones son declaraciones necesariamente formales, los supuestos y las relaciones se describen normalmente en lenguaje natural, lo que introduce potenciales ambigüedades e incomprensiones entre ingenieros y usuarios.

Una tendencia relativamente reciente en los métodos formales, conocida como "*métodos formales ligeros*" (Jackson, 2001), demuestra tener potencial para detectar errores importantes en la declaración de requisitos, sin el costo de una verificación diseñada formalmente. La premisa básica de este enfoque es el uso de técnicas formales en el análisis de los supuestos, las relaciones y las propiedades de los requisitos, indicadas en su declaración o en el modelo conceptual. Se puede aplicar a especificaciones parciales o a un segmento de la especificación completa, proceso que se realiza en tres fases: 1) reafirmar los requisitos y el modelo conceptual en una notación formal -o semiformal-, típicamente en una tabla de descripción de estados; 2) identificar y corregir las ambigüedades, conflictos e inconsistencias; y 3) utilizar un verificador de modelos o un probador de teoremas para estudiar el comportamiento del sistema, demostrar sus propiedades y graficar su comportamiento. Los ingenieros y usuarios pueden utilizar estos resultados para mejorar el modelo conceptual (Davis, 2005).

Un aspecto particularmente interesante de este enfoque es que se ha utilizado para modelar y analizar el comportamiento del *software*, del *hardware* y de las acciones humanas en los sistemas (Mazzola et al, 2006). Agerholm y Larsen (1997) describen su aplicación en un sistema de actividad extra-vehicular de la NASA; y Lutz (1997) describe la validación de requisitos de los monitores de errores a bordo de una nave espacial. Un detalle importante de este proyecto es que los ingenieros utilizaron el modelo de requisitos para un segundo proyecto, que se desarrolló a partir del primero, como una construcción en serie. Janssen et al (1999) describen la aplicación de un verificador de modelos para analizar procesos de negocios automatizados, como el procesamiento de reclamaciones de seguros.

5. El futuro de los métodos formales

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

La industria del *software* tiene una larga y bien ganada reputación de no cumplir sus promesas y, a pesar de más de 60 años de progreso, tiene años -incluso décadas- por debajo de la madurez necesaria que requiere para satisfacer las necesidades de la naciente Sociedad del Conocimiento. Goguen (1997) cita algunas estimaciones de los gastos que generan los fracasos del desarrollo de *software*, los calculó en 81 mil millones de dólares para 1995 y en 100 mil millones para 1996; posteriormente (1999), llamaba la atención sobre la cancelación del contrato de 8 mil millones de dólares a *The International Business Machines* -IBM- por la FAA -*The Federal Aviation Administration*-, para el diseño de un sistema de control aéreo para toda la nación; y del contrato del DOD -*United States Department of Defense*-, a la misma IBM, por $2 mil millones para modernizar su sistema de información; del fallo del *software* para la entrega en tiempo real de datos en las Olimpiadas de 1996; y del año y medio de retraso en el sistema para el manejo automatizado de equipaje en el aeropuerto de Denver para United Airlines, con un costo de $1,1 millones diarios. En su libro, Neumann (1994) revela que estos problemas no son en absoluto nuevos, aunque parece que se incrementan; incluso señala que algunos de ellos ya provocaron muertes de personas, por ejemplo la sobredosis de radiación en un sistema de terapia a mediados de los 80 (Gowen & Yap, 1991). Es claro que aún no es posible, con la tecnología actual, asegurar el éxito de los proyectos de *software*, y que para proyectos grandes y complejos el enfoque *ad hoc* ha demostrado ser insuficiente.

La falta de formalización en los puntos clave de la Ingeniería de *Software* la hace sensible a problemas que son inevitables en actividades altamente técnicas y detalladas como la creación de *software*. Las buenas prácticas en Ingeniería deben aplicarse en todo el proceso del desarrollo de sistemas, pero, aunque el desarrollo tecnológico aporta mucho material para alcanzarlo, todavía no se logra este objetivo. Incrementar la precisión y el control riguroso es esencial, y es el principal objetivo de los métodos formales, que utilizan esencialmente formalismos lógicos buscando mejorar el *software* y el *hardware*, en áreas como confiabilidad, seguridad, productividad y reutilización. Los ejes principales de su accionar son la verificación del código y el diseño, así como la generación de programas y casos de prueba desde las especificaciones.

Los métodos formales deberían estar presentes como principios esenciales de las técnicas de prueba. Gaudel (1995) lo estableció como un tema importante de investigación; Hoare (2002a) describió el uso de aserciones

formales no para probar el programa, sino para diseñar las pruebas; y Hierons et al. (2008) desarrollaron una investigación en aspectos formales de las pruebas. Los métodos formales se utilizan en el mantenimiento del *software* (Younger et al., 1996) y en su evolución (Ward & Bennett, 1995), y tal vez su más amplia aplicación sea en el mantenimiento de código heredado (Hoare, 2002b).

Los métodos formales son un área de investigación muy activa, y se espera que cada día se incrementen las colaboraciones. Existen varias revistas especializadas, como *Formal Aspects of Computing* y *Formal Methods in System Design*, que hacen hincapié en sus aplicaciones prácticas, así como en la teoría. Conferencias como la *NASA Formal Methods Symposium* y la *Computer-Aided Verification* están dedicadas al tema y tienen procesos de selección de aportes muy competitivos. Otras conferencias importantes son *Principles of Programming Languages, Logic in Computer Science*, y *Conference on Automated Deduction*. Existen talleres y conferencias especializadas más pequeñas, algunas enfocadas en herramientas y técnicas, como la ABZ, que cubre las notaciones Alloy, ASM, B y Z, y talleres de refinamiento. Unas debaten cuestiones teóricas específicas, como *Integrated Formal Methods*, y otras cubren áreas de aplicación, como *Formal Methods for Open Object-based Distributed Systems* y *Formal Methods for Human-Computer Interaction*. *The Computer Science Bibliography* (www.informatik.uni-trier.de/~ley/db/, enero 2010) contiene referencias a más de un millón de artículos, indexados por metadatos, entre los que existen gran cantidad acerca de los métodos formales, especificación, verificación y validación, muchos de edición reciente, lo que presupone que para esta área la producción está en incremento.

Se estima que en EE.UU. existen por lo menos 1.000 investigadores en verificación, cerca de 300 profesores y 200 estudiantes graduados de pregrado, y alrededor de 250 investigadores en la industria –Microsoft, Intel, Cisco, IBM, Cadence, Synopsys, Mentor Graphics-, y 50 en el gobierno –NASA, NSA, NRL– (Woodcock et al, 2009). *The UK Engineering and Physical Science Research Council* desarrolla por lo menos 400 proyectos de investigación en Ingeniería de *Software*, y *The Fundamentals of Computing* dedica US $144 millones a patrocinios. Además, se estima que en Europa existen más de 1.000 investigadores, en China 250, en los países bajos 500, en Australia, Brasil, Canadá, Nueva Zelanda y África, unos 1.000.

Se espera que una pesquisa futura -por lo menos en 10 años-, y de acuerdo con las proyecciones aquí mostradas, demuestre un notable incremento en

"Revista Virtual Universidad Católica del Norte". No. 30, (mayo - septiembre de 2010, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias, Latindex, EBSCO Information Services y Actualidad Iberoamericana

investigación, desarrollo y aplicación; y entonces será posible una mayor aceptación en la industria, mayor participación de las facultades de Ingeniería y la academia en general, y más trabajo práctico y experimental alrededor de los métodos formales. Con esto se podrá concretar el apoyo decidido a la visión de Hoare (2003, 2007) de "*un mundo en el que los productos software siempre serán fiables, y que la labor de los ingenieros de software sea realmente Ingeniería*".

6. Conclusiones

- Los métodos formales son técnicas matemáticas, a menudo soportadas por herramientas, para el desarrollo de sistemas *software* y *hardware*. Su rigor matemático le permite a los ingenieros analizar y verificar sus modelos en cualquier parte del ciclo de vida del desarrollo; y dado que la fase más importante en estos procesos es la Ingeniería de Requisitos, son útiles para elicitarlos, articularlos, representarlos y especificarlos (George & Vaughn, 2003). Sus herramientas proporcionan el soporte automatizado necesario para la integridad, trazabilidad, verificabilidad, reutilización, y para apoyar la evolución de los requisitos, los puntos de vista diversos y la gestión de las inconsistencias (Ghose, 2000).

- Los métodos formales se utilizan en la especificación de *software*, y se emplean para desarrollar una declaración precisa de lo que el *software* tiene que hacer, evitando al mismo tiempo las restricciones del cómo se quiere lograr. La especificación es un contrato técnico entre el ingeniero y el cliente, que proporciona un entendimiento común de la finalidad del *software*; el cliente la utiliza para orientar la aplicación del software, y el ingeniero para guiar su construcción. Una especificación compleja se puede descomponer en sub-especificaciones, que describen un sub-componente del sistema que se pueden delegar a otros ingenieros -diseño por contrato- (Meyer & Mandrioli, 1992).

- Los sistemas de *software* complejos requieren una cuidadosa organización de la estructura arquitectónica de sus componentes, un modelo del sistema que suprima detalles de la implementación, y que permita al arquitecto concentrarse en los análisis y decisiones más importantes para estructurar el sistema y satisfacer sus requisitos (Allen & Garlan, 1992), (Lamsweerde, 2003). Darwin (Magee & Kramer, 1996) y Wright (Allen, 1997) son ejemplos de lenguajes de descripción arquitectónica basados en la formalización del comportamiento abstracto de los componentes y conectores arquitectónicos.

- Los métodos formales se utilizan en el diseño de *software*, donde el refinamiento de datos incluye la especificación de máquinas de estado, funciones de abstracción y pruebas de simulación (Hoare, 1975); cumplen un rol protagónico en métodos como VDM y Z, y en el cálculo de refinamiento de programas (Dijkstra, 1975).

- En la fase de implementación, los métodos formales se utilizan para verificar el código, ya que toda especificación tiene explícito un teorema de correctitud con el que, si se cumplen ciertas condiciones, el programa deberá conseguir el resultado descrito en la documentación. La verificación del código es un intento por demostrar este teorema, o al menos de encontrar por qué falla. El método de verificación por aserción inductiva de programas fue desarrollado por Floyd (1967) y Hoare (1969), y consiste en anotar el programa con aserciones matemáticas, o sea las relaciones entre las variables y los valores iniciales de entrada.

- Más de cuatro décadas de investigación y experimentación demostraron que los métodos formales son, hasta el momento, el único medio práctico para exponer la ausencia de comportamientos no deseados en los programas, una propiedad esencial en los sistemas críticos. Los modelos para probar calidad industrial, y los probadores de teoremas avanzados permiten, de forma automática o semiautomática, hacer análisis complejos de las especificaciones formales, por lo que estas herramientas son atractivas para uso comercial.

- La capacidad para generar casos de prueba completos desde la especificación formal, representa un ahorro sustancial a pesar del costo de su desarrollo. La experiencia demuestra que las técnicas formales se pueden aplicar productivamente, incluso en las pruebas más completas. El proceso para desarrollar una especificación es la fase más importante de la verificación formal, y el enfoque de los "*métodos formales ligeros*" permite analizar formalmente las especificaciones parciales, y la definición temprana de requisitos.

7. Lista de referencias

Agerholm, S. & Larsen P. G. (1997). “Modeling & Validating SAFER in VDM-SL”. *The Fourth NASA Langley Formal Methods Workshop*. Hampton, Virginia, USA, 65-73.

Allen, R. B. & Garlan D. (1992). “A formal approach to software architectures”. *The IFIP 12th World Computer Congress on Algorithms, Software, & Architecture*. Madrid, Spain, 1, 134-141.

Allen, R. B. (1997). “A formal approach to software architecture”. Ph.D. thesis, School of Computer Science, Carnegie Mellon University. Issued as CMU Technical Report CMU-CS-97-144.

Amman, P. & Offutt J. (2008). *Introduction to software testing.* UK: Cambridge University Press.

Anderson, R. J., Beame P., Burns S., Chan W., Notkin D. & Reese J. D. (1998). “Model checking large software specifications”. *IEEE Transactions on Software Engineering*, 24 (7), 498-520.

Bause, F. & Kritzinger P. S. (2002). *Stochastic Petri Nets*. London: Friedrich Vieweg & Sohn Verlag.

Bolstad, M. (2004). “Design by Contract: A simple technique for improving the quality of software”. *The 2004 Users Group Conference*. Williamsburg, Virginia, USA, 319-323.

Clarke, E. M. & Wing J. (1996). “Formal Methods: State of the Art and Future Directions”. *Special ACM 50th-anniversary issue: strategic directions in computing research*, 626-643.

Dan, L. & Aichernig B. K. (2002). “Automatic Test Case Generation for RAISE”. *Report 273, UNU-IIST -United Nations University, International Institute for Software Technology*. Macau, China.

Dasso, A. & Funes A. (2007). *Verification, validation and testing in software engineering*. USA: Idea Group Publishing.

Davis, J. F. (2005). “The Affordable Application of Formal Methods to Software Engineering”. *ACM SIGAda Ada Letters*, 25 (4), 57-62.

Desel, J. & Esparza J. (2005). *Free Choice Petri Nets - Cambridge Tracts in Theoretical Computer Science*. Cambridge: Cambridge University Press.

Dijkstra, E. W. (1975). “Guarded commands, nondeterminacy and formal derivation of programs”. *Communications of the ACM*, 18 (8), 453-457.

Dijkstra, E. W. (1976). *A discipline of programming*. New York: Prentice-Hall.

Dijkstra, E. W. (1989). “On the cruelty of really teaching computing science”. *Communications of the ACM*, 32 (12) 1398-1404.

Drechsler, R. Ed. (2004). *Advanced formal verification*. USA: Kluwer Academic Publishers.

Duffy, D. (1991). Principles of Automated Theorem Proving. New York: John Wiley & Sons.

Ehrig, H. & Mahr B. (1990). “Fundamental of Algebraic Specification II: Specifications and constraints”. Berlin: Springer-Verlag.

Felleisen, M., Findler R. B., Flatt M. & Krishnamurthi S. (2001). *How to design programas: An Introduction to Programming and Computing*. USA: The MIT Press.

Floyd, R. (1967). “Assigning meanings to programs”. *Proceedings of Symposium in Applied Mathematics*. Providence, Rhode Island, USA, 19-32.

Flynn, S. & Hamlet D. (2006). “On formal specification of software components and systems”. *Electronic Notes in Theoretical Computer Science*, 161, 91-107.

Gabbar, H. A. (ed). (2006). *Modern Formal Methods and Applications*. Netherlands: Springer.

García, J., Amescua A. & Velasco M. (2006). “TOP 10 de factores que obstaculizan la mejora de los procesos de verificación y validación en organizaciones intensivas en software”. *Revista Española de Innovación, Calidad e Ingeniería del Software,* 2 (2), 18-28.

Gaudel, M. C. (1995). “Testing can be formal, too”. *The 6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, Aarhus, Denmark, 82-96.

George, V. & Vaughn R. (2003). "Application of lightweight formal methods in requirement engineering". *Crosstalk: The Journal of Defense Software Engineering*, 16 (1), 30-38.

Ghose, A. (2000). "Formal methods for requirements engineering". *International Symposium on Multimedia Software Engineering*, Taipei, Taiwan, 13-16.

Gödel, K. (1934). "On undecidable propositions of formal mathematical systems". *Princeton lectures*, *Collected works I*, 346-369.
Goguen, J. A. (1997). "Formal methods: promises and problems". *IEEE Software*, 14 (1), 73-85.

Goguen, J. A. (1999). "Hidden algebra for software engineering". *Combinatorics, Computation and Logic,* 21 (3), 35-59.

Gowen, L. D. & Yap M. Y. (1991). "Traditional software development's effects on safety". *The Sixth Annual IEEE Symposium*, Amsterdam, Netherlands, 58-63.

Hayes, I. J., Jones C. B. & Nicholls J. E. (1994). "Understanding the Differences between VDM and Z". *ACM SIGSOFT Software Engineering Notes*, 19 (3), 75-81.

Hehner, Eric C. (2006). *A practical theory of programming*. New York: Springer-Verlag Publishers.

Hierons, R. M., Bowen, J. P. & Harman, M., Eds. (2008). "Formal Methods and Testing An Outcome of the FORTEST Network". *Revised Selected Papers*. Berlin: Springer.

Hoare, C. A. R. (1969). "An axiomatic basis for computer programming". *Communications of the ACM*, 12 (10), 576-580.

Hoare, C. A. R. (1975). "Proof of correctness of data representations". *Lecture Notes In Computer Science*, 46, 183-193.

Hoare, C. A .R. (2002a). "Assert early and assert often: Practical hints on effective asserting". *Presentation at Microsoft TechFest 2002*. Redmond, Washington, USA.

Hoare, C. A. R. (2002b). “Assertions in modern software engineering practice”. *The 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, Oxford, UK, 459-462.

Hoare, C. A. R. (2003). “The verifying compiler: A grand challenge for computing research”. *Journal of the ACM*, 50 (1), 63-69.

Hoare, C. A. R. (2007). “The ideal of program correctness: Third Computer Journal Lecture”. *Computer Journal*, 50 (3), 254-260.

Jackson, D. (2001). “Lightweight Formal Methods. Formal Methods for Increasing Software Productivity”, Berlin, Germany, 1-18.

Janssen, W., Mateescu R., Mauw S., Fennema P. & Stappen P. (1999). “Model Checking for Managers”. *The 5th-6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, Toulouse, France, 92–107.

Jones, C. B. (1990). *Systematic software development using VDM.* New York: Prentice-Hall.

Jones, C., O’Hearn P. & Woodcock J. (2006). “Verified software: A Grand Challenge”. *Computer*, 39 (4), 93-95.

Juhás, G., Lehocki F. & Lorenz R. (2007). “Semantics of Petri nets: a comparison”. *The 39th conference on Winter simulation*, Washington, USA, 617-628.

Kiniry, J. R. & Zimmerman D. M. (2008). “Secret Ninja formal methods”. *The 15th international symposium on Formal Methods*, Turku, Finland, 214-228.

Kuhn, D. R., Chandramouli T. & Butler R. W. (2002). “Cost effective use of formal methods in verification and validation”. *Foundations '02, a Workshop on Modeling and Simulation Verification and Validation for the 21st Century*, Laurel, Maryland, USA, 106-144.

Lamsweerde, A. van. (2003). “From system goals to software architecture”. *Lecture Notes in Computer Science*, 2804, 25-43.

Langari, Z. & Pidduck A. B. (2005). "Quality, cleanroom and formal methods". *The third workshop on Software quality*, St. Louis, Missouri, USA, 1-5.

Lewis, H. L. & Papadimitriou C. H. (1997). *Elements of the Theory of Computation*. New York: Prentice Hall International.

Liu, Z. & Venkatesh R. (2005). "Methods and tools for formal software engineering". *Verified Software: Theories, Tools, Experiments. First IFIP TC 2/WG 2.3 Conference*, Zurich, Switzerland, 31-41.

Lutz, R. R. & Ampo Y. (1994). "Experience report: Using formal methods for requirements analysis of critical spacecraft software". *The 19th Annual Software Engineering Workshop*, Greenbelt, Maryland, USA, 231-236.

Lutz, R. R. (1997). "Reuse of a Formal Model for Requirements Validation". *The Fourth NASA Langley Formal Methods Workshop*, Hampton, Virginia, USA, 75-85.

Magee, J. & Kramer J. (1996). "Dynamic Structure in Software Architectures". *ACM SIGSOFT Software Engineering Notes,* 21 (6), 3–14.

Manna, Z. & Waldinger R. (1985). *The Logical Basis for Computer Programming, Volume I: Deductive Reasoning*. London: Addison-Wesley Professional.

Manna, Z., & Pnueli A. (1992). *The Temporal Logic of Reactive and Concurrent Systems: Specification*. New York: Springer-Verlag.

Mazzola, G. B., Milmeister G. & Weissmann J. (2006). "Comprehensive Mathematics for Computer Scientists 1: Sets and Numbers, Graphs and Algebra, Logic and Machines, Linear Geometry". Berlin: Springer.

McDermid, J. A., Nicholson M., Pumfrey D. J. & Fenelon P. (1995). "Experience with the Application of HAZOP to Computer-Based Systems". *The 10th Annual Conference on Computer Assurance*, Gaithersburg, Maryland, 37-48.

Mendelson, E. (1997). *Introduction to Mathematical Logic*. UK: Chapman & Hall/CRC.

Meyer, B. & Mandrioli D. Eds. (1992). *Advances in Object-Oriented Software Engineering.* New York: Prentice Hall.

Milner, R. (1990). "Operational and algebraic semantics of concurrent processes". *Handbook of theoretical computer science, Volume B: Formal models and semantics*, 1201-1242.

Monin, J-F. (2003). *Understanding formal methods*. New York: Springer.
Murata, T. (1989). "Petri nets: Properties, analysis and applications". *Proceedings of the IEEE*, 77 (4), 541-580.

NASA Office of Safety and Mission Assurance. (1995). "Formal Methods Specification and Verification". *Guidebook for Software and Computer Systems. Volume I: Planning and Technology Insertion*. Washington, NASA-GB-002-95.

Neumann, P. G. (1994). *Computer Related Risks*. New York: Addison-Wesley Professional.

Perry, W. E. (2006). *Effective methods for software testing*. Indianapolis: Wiley Publishing, Inc.

Pressman, R. S. (2004). *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill Science.

Reising, W. (1991). "Petri nets and algebraic specifications". *Theoretical Computer Science*, 80 (1), 1-34.

Rushby, J. (1993). "Formal Methods and Digital Systems Validation for Airborne Systems". *SRI-CSL-93-07. Technical Report: cr4551*. NASA Langley Technical Report Server.

Saiedian, H. & Hinchey M. G. (1996). "Challenges in the successful transfer of formal methods technology into industrial applications". *Information and Software Technology*, 38 (5), 313-322.

Sargent, R. G. (1999). "Validation and Verification of Simulation Models". *The 31st conference on Winter simulation: Simulation a bridge to the future*. Phoenix, Arizona, USA, 39-48.

Sobel, K. A. E. & Clarkson R. M. (2002). "Formal methods application: An empirical tale of software development". *IEEE Transactions on software engineering*, 28 (3), 308-320.

Spivey, J. M. (1992). *The Z Notation: A reference manual*. New York: Prentice-Hall.

Ward, M. P. & Bennett K. H. (1995). "Formal methods to aid the evolution of software". *International Journal of Software Engineering and Knowledge Engineerin,* 5, 25-47.

Wirsing, M. (1991). "Algebraic Specification". *Handbook of Theoretical Computer Science, Formal Models and Semantics*, 675-788.

Woodcock, J. J., Larsen P. G., Bicarregui J. & Fitzgerald J. (2009). "Formal Methods: Practice and Experience". *ACM Computing Surveys*. 41 (4), Article No. 19.

Wunram, J. (1990). "A Strategy for Identification and Development of Safety Critical Software Embedded in Complex Space Systems". *The 41st International Astronautical Congress*, Dresden, Germany, 35-51.

Younger, E. J., Luo Z., Bennett K. H. & Bull T. M. (1996). "Reverse engineering concurrent programs using formal modelling and analysis". *International Conference on Software Maintenance*, Monterey, California, 255-264.

Zeugmann, T. & Zilles S. (2008). "Learning recursive functions: A survey". *Theoretical Computer Science*, 397 (1-3), 4-56.

Zhang, Z., Basili V., & Shneiderman B. (1999). "Perspective-based Usability Inspection: An Empirical Validation of Efficacy". *Empirical Software Engineering*, 4 (1), 43-69.