Métodos Formales: Una Disciplina Emergente para Desarrollar Software

José de Jesús Lavalle Martínez
jlavalle@aleteya.cs.buap.mx
Grupo de Investigación en Métodos Formales
http://www.aletaya.cs.buap.mx
Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla
Puebla, Puebla, México

Junio de 2004

Resumen

En los inicios de la programación podemos considerar que no había una distinción clara entre programador y usuario; esto es, el software era usado por, y casí exclusivamente, su creador. En medida que se van automatizando soluciones a problemas cada vez más complejos, surge la necesidad de contar con herramientas más poderosas para construir software.

Un primer intento fue proveer a los lenguajes de programación con mecanismos que implementan los ahora llamados *principios de lenguajes de programación* (etiquetar, transportar, abstraer, encapsular, compartir comportamiento, etc.) con el afán de disminuir la brecha entre la forma en que resolvemos problemas mentalmente y la forma en que le damos ordenes a una computadora.

Poco después de que el desarrollo de software se convierte en una actividad lucrativa, surge la *crisis del software*, la cual a grandes rasgos ha significado pérdidas de millones de dolares y clientes insatisfechos. La primera estrategia para resolver la crisis fue adaptar procedimientos existentes en otros procesos ingenieriles al proceso de desarrollo de software, naciendo así la *ingeniería de software* o el estilo *catedral* para desarrollar software. Con el advenimiento de internet, la filosofía *GNU* y el éxito alcanzado con Linux, surge un cambio de actitud en la manera de desarrollar software, denominado estilo *bazar*.

Independientemente de sus ventajas y desventajas, de sus seguidores y detractores, ambos estilos adolecen del mismo problema: de entrada se asume que el sistema no estará libre de errores. Por lo anterior los *métodos formales* se han reforzado como una alternativa que pretende construir sistemas que están libres de errores. Lo cual ahorra dinero y tiempo, tanto en la construcción como en el mantenimiento del sistema, además deja a los usuarios más satisfechos con su funcionamiento.

En esta plática se aborda qué son los métodos formales, cuál es su rango de aplicación, se analizan algunos mitos que surgen alrededor de ellos, se ofrecen algunos datos interesantes sobre qué tan caro es cometer errores, también se presentan algunos resultados del grupo de investigación en métodos formales de la Universidad Autónoma de Puebla, finalmente se esbozan algunas oportunidades de aplicación e investigación.

1. Antecedentes

- En el principio sólo existian aplicaciones personales, se programaba en:
  - Ceros y unos,
  - Nemotécnicos o Lenguajes Ensambladores (Principio de etiquetar), las instrucciones están en correspondencia biunívoca con la arquitectura de la computadora.
- Escensialmente los cálculos eran numéricos,
  - Era deseable no tener que reprogramar (Principio de portar), las instrucciones de los lenguajes de programación empiezan a ser independientes de las arquitecturas.
- Se empieza a compartir código entre la comunidad de programadores,
  - Crisis de la instrucción GOTO (Principio de estructurar programas).
- En medida que el tiempo transcurre, se aspira a automatizar actividades más cotidianas y más complejas,
  - Hace falta modelar datos no numéricos (Principio de abstraer datos).
- El desarrollo de software se convierte en una actividad lucrativa.
  - Al mezclar código que funciona por separado el código resultante no hace lo esperado (Principio de encapsular datos y código),
  - Es deseable reusar código y en el peor de los casos enriquecer su comportamiento (Principio de compartir comportamiento).

2. Crisis del software

Poco después de que el desarrollo de software se vuelve una actividad lucrativa surgen los siguientes problemas con los sistemas:

- Nunca son entregados según lo pactado,
- No hacen exactamente lo que el usuario final esperaba, porque:
  - No se entendió su especificación,
  - Fue entregado tan tarde que las condiciones originales de funcionamiento ya habían cambiado.

Por lo tanto el software así construido resultaba ser (sobre todo si se le comparaba con los avances en la industria del hardware):

- Caro,
- Poco fiable,
- Escaso.

3. Ingeniería de software: desarrollo centralizado o el estilo catedral

Ciclo (Clásico) de Vida del Software:

- Análisis,
- Diseño,
- Implementación,
- Pruebas,
- Mantenimiento.

Dijkstra advierte que la no detección de "pulgas"no implica su ausencia.

La práctica muestra que, en el ciclo de vida del software, entre más temprano se detecten los errores menos costosa será su corrección.

Brooks establece que la relación hombre-mes, cuando se trata de desarrollo de software, no es inversamente proporcional.

4. Ingeniería de Software: desarrollo distribuido o el estilo bazar

Este enfoque ha sido viable gracias a:

- El advenimiento de internet,
  - Acceso "inmediato"al código fuente,
  - Muchos ojos de co-desarrolladores observando y corrigiendo la misma "pulga".
- La filosofía del proyecto *GNU* de Richard Stallman en particular, y del software de fuente abierta en general,
  - Derecho a compartir el software,
  - Derecho a cambiar el software,
  - Derecho a ejecutar el software.
- El éxito alcanzado por Linus Torvalds en el desarrollo de Linux (de acuerdo a Eric Steven Raymond quien acuñó el término bazar).
  - Empezar con código que sea ejecutable y que se pueda probar para empezar a jugar con él,
  - Liberar rápida y frecuentemente las mejoras al código,
  - Delegar todo lo que se pueda,

- Mantenerse abierto hasta el punto de la promiscuidad.

Independientemente de sus ventajas y desventajas, de sus seguidores y detractores, ambos estilos adolecen del mismo problema: de entrada se asume que el sistema no estará libre de errores.

5. Ingeniería de Software (IS) contra Otras Ingenierías (OI).

Diferencias:

- Los artículos propios de la IS no son "tan" tangibles como los artículos de OI,
- El esfuerzo para producir artículos de IS, a diferencia de los artículos producidos en OI, es más intelectual que material,
- Modificar los artículos de la IS requiere "menos" esfuerzo físico y material, además de ser "mucho" menos costoso,
- En su mayoría hay poco o nulo sustento formal en la producción de artículos de IS, a diferencia del sustento en modelos físicos o químicos que existen en OI.

6. Métodos formales

Son la síntesis de:

- Sistemas matemáticos,
  - Lógicas,
  - Álgebras,
  - Cálculos.
- Demostradores (semi)automáticos de teoremas.

Se usan para:

- Razonar sobre las propiedades que un sistema debe cumplir, o
- Verificar formalmente que la solución propuesta de un problema (su especificación formal) está libre de errores, o
- Demostrar que la expresión, (en determinado formalismo) que codifica la solución de un problema se sigue de la conjunción de expresiones que codifican el dominio del problema.

Garantizan que las especificaciones sean:

- No ambigüas, claras y concisas;
- Completas;
- Consistentes, tanto aislada como integralmente;
- Tratables;
- Correspondientes con las implementaciones.

7. Aplicabilidad

El uso de los métodos formales en el desarrollo de software particularmente se justifica en sistemas que son:

Complejos Muchos sistemas caen en esta categoría, y la tendencia es producir sistemas aún más complejos;

Concurrentes Estos sistemas exhiben patrones complejos de comportamiento que potencialmente interferirán y que por tanto deben entretejerse; la concurrencia surge en sistemas distribuidos, de tiempo real, en el diseño de hardware y en el procesamiento paralelo;

Críticos En cuanto a la:

calidad Estos son sistemas cuya falla no es dañina pero cuya confiabilidad es sumamente importante; algunos ejemplos son: aplicaciones financieras, telecomunicaciones y sistemas operativos;

vitalidad Las computadoras controlan sistemas vitales en actividades tales como defensa, medicina, industía nuclear, señalamiento en vias ferreas, telecomunicaciones, y administración de vuelo de las aeronaves;

seguridad Con el uso extendido de la tecnología de la información, prevenir el acceso no autorizado a la información o a las instalaciones de cómputo puede ser escencial por razones de seguridad nacional, secretos comerciales, o privacidad personal.

8. Mitos

De acuerdo a Hall:

1. Los métodos formales garantizan que el software sea perfecto y eliminan la necesidad de hacer pruebas;
2. Los métodos formales sólo sirven para probar que los programas son correctos,
3. Los métodos formales sólo son útiles en sistemas que son críticos en cuanto a la vitalidad;
4. Para aplicar los métodos formales se requieren matemáticos altamente entrenados;
5. Aplicar métodos formales incrementa los costos de desarrollo;
6. Los métodos formales nos son aceptados por los usuarios;
7. Los métodos formales no son usados en los sistemas reales de gran escala.

De acuerdo a Bowen y Hinchey:

1. Los métodos formales retrasan el proceso de desarrollo;
2. No existen herramientas de apoyo para los métodos formales;
3. Los métodos formales inducen al abandono de los métodos de diseño de la ingeniería tradicional;

4. Los métodos formales sólo se aplican al software;
5. No se requiere de métodos formales;
6. No hay soporte para los métodos formales;
7. La gente de métodos formales siempre usa métodos formales.

9. Datos interesantes

- Blake Stone (Jefe científico en Borland) afirma que los programadores emplean **muchísimo** más tiempo componiendo errores en código existente que en escribir código nuevo;
- El Instituto Nacional de Estándares y Tecnología de Estados Unidos (NIST) estima que, el 80 % de lo que cuesta desarrollar software en un proyecto típico se gasta identificando y corrigiendo errores;
- Djenana Campara (Directora en Jefe de Tecnología en Clockwork, Ottawa) expresa que componer un error en el escritorio de un programador cuesta $1.00. Pero, componer el mismo error cuando éste se ha incorporado a un programa completo cuesta $100.00, y algunos miles de dólares si se identifica el error cuando el programa ha llegado al cliente;
- NIST en el 2002 reportó que, los errores de software son tan comunes que le cuesta a la economía de Estados Unidos 60 miles de millones de dólares al año, o alrededor del 0.6 % de su producto interno bruto.

10. Principales resultados

- Caracterización de Redes de Petri y Sistemas de Reescritura de Términos mediante Reescritura Regulada;
- Especificación Formal de un Sistema de Recuperación de Información Modelado con UML;
- Especificación Formal de Sistemas Distribuidos mediante Cálculo-π;
- Especificación y Verificación de Circuitos Lógicos mediante el Álgebra de procesos Circal;
- Semántica Denotacional del Modelo de Actores;
- Redes de Petri para la Especificación de Sistemas Concurrentes;
- Intérprete de ML y Demostración Automática de Teoremas en Lógica de Alto Orden;
- Prototipo para la Notación Z;
- Especificación Formal de Sistemas Interactivos mediante Lógica Lineal.

11. Oportunidades

Una de las mayores riquezas de los métodos formales es que se puede trabajar con ellos en diferentes niveles (siempre hay quehacer), los cuales además se retroalimentan, a saber:

Concreto Especificar y verificar formalmente sistemas;

Técnico Creación de herramientas de apoyo, en particular: demostradores automáticos, verificadores de tipo, trazadores de errores, etc.;

Teórico Existen diversos métodos para la especificación formal de sistemas del mismo tipo, mostrar su equivalencia directamente es muy tedioso, por lo tanto es necesario usar alguna teoría matemática con la cual podamos analizar dichos métodos, para compararlos en el mismo nivel de abstracción. A nosotros nos gusta la teoría de categorías para tal propósito.

12. Conclusiones

- Los sistemas son cada vez más complejos y su uso se ha masificado (gracias a Internet), lo cual los sitúa dentro de la categoría de sistemas suceptibles a ser tratados con métodos formales;
- Pensemos en gobierno electrónico, tu banca en línea, la ley de transparencia en la información, etc;
- El único detalle es que el uso de métodos formales prefiere de una formación sólida y rigurosa en:
  - Matemáticas Discretas,
  - Lógica Matemática,
  - Teoría de la Computabilidad,
  - Teoría de Tipos,
  - Programación Declarativa.