SOFTWARE QUE NO NECESITA MANTENIMIENTO

ING. HUGO HESSE

Este es un título que despierta interés y duda. Nadie que conozca de programación y que haya trabajado en este campo, admitiría que se puede desarrollar un programa o un conjunto de programas, para ser usados a lo largo del tiempo y ser útiles sin requerir mantenimiento.

La utilización de programas está ligada al mantenimiento de éstos, en la misma forma en que la utilización de un vehículo está ligada a su mantenimiento.

Sin embargo, la idea en el área de sistemas de información no es necesariamente nueva. Más de una vez muchas personas se habrán preguntado si no exite la posibilidad de desarrollar un software que tuviese esta característica.

Dado lo interesante del tema el Ing. Hesse comenta en esta oportunidad, un artículo publicado en la revista ***Computer world*** *del 24 de febrero de 1992, volumen XXXVI, No. 8, página 87, cuyo autor es el Sr. Steve Hearn.*

El Sr.Hearn ha trabajado en este campo cerca de 20 años. Actualmente es el gerente del Centro de Cómputo de la Cía. Arco Products Co.

El autor sostiene que desde que se creó la programación, automáticamente se creó el mantenimiento de programas. En su opinión, si bien el grupo de programadores que desarrolla programas puede hacer un excelente trabajo reduciendo la carga de mantenimiento, la acción de reducir la carga de mantenimiento no sería ya el objetivo. En su opinión, ha llegado el tiempo para pensar en eliminar totalmente el trabajo de mantenimiemto de software.

Lo que el Sr. Hearn quiere decir, es que el objetivo sería desarrollar software libre de mantenimiento, esto es, "Software que no es modificado durante toda su existencia como software de producción" .

Existe ya, aunque en casos aislados, software libre de mantenimiento. Casos como estos serían software que tienen un período de vida muy corto o que son ejecutados en trabajos de producción (ciertas rutinas de error), si es que ellos no son modificados durante su existencia.

Otros ejemplos, según el autor, podrían incluir software que realiza una actividad específica y está protegido especialmente para que no haya cambios (por ejemplo una subrutina de conversión de temperatura).

Si pensamos en los ejemplos del autor, podemos con relativa facilidad, imaginarnos o recordar rutinas y aún programas que no son cambiados durante su existencia, pero aún así, no dejan de ser casos aislados.

Esto es, los ejemplos pequeños no nos dicen mucho. El verdadero objetivo sería producir software que es estructurado y construido de tal manera que no necesitaría mantenimiento durante toda su existencia y respecto de su función. La idea es, libre de mantenimiento por diseño y no por accidente.

El autor nos da seis características del software diseñado para trabajar sin-mantenimiento y algunas ideas de cómo lograrlo. El software no necesitaría tener todas estas características para ser considerado libre de mantenimiento, en su lugar, podría tener sólo una. Pero nos hace una prevención, hay que tener en cuenta que ninguna de ellas puede lograrse fácilmente y todavía no existe la tecnología disponible para hacer que todas ellas sean viables.

Estas características son interesantes porque si bien no son necesariamente viables hoy en día, son características que deberíamos tener en cuenta dentro del concepto de sin-mantenimiento cuando diseñemos nuevas aplicaciones.

Dado que las seis características que señala el Sr. Hearn son la parte medular del artículo y es interesante conocerlas no sólo en su expresión sino con el sustento que él les da, me he permitido hacer una traducción libre de esta parte del artículo para que el lector tenga una visión más clara del tema.

"QUOTE

Primera.- El software sin-mantenimiento está limpio cuando es pasado a producción.

Bajo este concepto se entiende que la codificación del programa ha capturado los verdaderos requerimientos del usuario, de tal manera que no existan razones para "arreglar" o "modificar" el software.

El software debe estar libre de errores (o si existen errores deben ser de tal naturaleza que no requieran modificación del programa) .

Para llegar a este punto, el personal debe enfatizar la correcta definición de requerimientos y conducir una rigurosa validación de los requerimientos y supuestos. Se requiere por lo tanto, un salto monumental hacia adelante en las técnicas de pruebas, en la exploración de códigos reusables y en el empleo de técnicas, todavía no desarrolladas, que impidan o eviten la introducción de errores. El software debe cumplir con el rendimiento, la portabilidad y facilidad de uso requeridas por el usuario al momento de su instalación.

Segunda.- El software sin-mantenimiento es autorreparable. El software debe ser capaz de detectar automáticamente y rectificar cualquier error que exista y sea de tal significancia que amerite corrección.

Estas posibilidades existen hoy en día en forma muy limitada. Por ejemplo, existen paquetes de software que interceptan condiciones de falta de espacio en el file y automáticamente asignan más espacio para estos files, eliminando la necesidad de intervención de los programadores u operadores.

Sin embargo, estas posibilidades son un atisbo de lo que será posible en el futuro. Estas habilidades de detección y corrección requerirán avances significativos, más allá de donde la tecnología actual ha llegado.

Tercera.- El software sin-mantenimiento es sensitivo a los cambios en requerimientos y se modifica asimismo automáticamente.

Si el conjunto de requerimientos para toda la vida del software no puede ser estipulado al momento de desarrollar el software, éste debe reconocer y adaptarse a los cambios de requerimientos de tal modo que no signifique modificación del software. En el futuro, estas características de percepción y adaptación obtendrán su mecanismo de cómo los organismos vivientes aprenden y se adaptan a los cambios en el medio ambiente. Las técnicas de inteligencia artificial podrían ser útiles en esta área.

Sin embargo, el software de hoy día necesita emplear otras técnicas. Una de tales técnicas es el uso de parámetros, sistema bajo el cual la primera importancia radica en los datos paramétricos más que en los datos de la aplicación misma.

Estos datos paramétricos definen los datos de la aplicación y la interface de la aplicación a los usuarios y al medio ambiente. El software de intercambio y traducción electrónica de datos, por ejemplo, representa software que está aislado en cierto grado de cambios sobreentendidos en el flujo de datos de entrada. Usa datos embebidos acerca de como traducir la transacción, y en efecto, el software se convierte en un utilitario cuya función es interpretar parámetros y dirigir los procesos de acuerdo a éstos.

Tal software está protegido de cambios, porque las alteraciones en el medio ambiente son reflejadas como cambios en los parámetros y no en el software.

De está manera, el software en un sistema financiero, por ejemplo, podría ser considerado sin-mantenimiento si éste, automáticamente buscara en una remota base de datos de leyes económicas y valores, la misma que es actualizada conforme las leyes y valores cambian.

El aspecto del software sin-mantenimiento, dirigido por los parámetros, implica

que las reglas del negocio de una empresa están representadas en forma computacional, a la cual las aplicaciones se refieren cuando es necesario. Las diferencias en las preferencias del usuario, tales como, qué fechas deberían aparecer, qué moneda debería usarse o qué color debería aparecer en las pantallas, deben ser representadas en una forma similar. Por lo tanto, los cambios en los requerimientos son manifestados como cambios a los parámetros en lugar de cambios en la codificación .

Esta sensibilidad al ambiente debe incluir la habilidad para acomodar la adición y el retiro de aplicaciones de interconección y datos pasados entre aplicaciones interconectadas. La eventual implementación de la tecnología de repositorio (almacén de rutinas probadas) podría ser útil para acomodar cambios de este tipo.

Este es un apartamiento dramático de la actual tecnología de enfocar las aplicaciones desde un punto de vista centralista y requerirá de la estandarización de la especificación y significado de paramétros. Pero este es un cambio que puede limitar dramáticamente el cambio en el software. De hecho, existe un sistema de ajuste de reclamos que es usado por muchas compañías aseguradoras hoy día, en la cual más del 95% de los cambios en los beneficios no requieren cambios de ninguna especie en el software. Esto es, porque se usó parametrización en el diseño de la aplicación.

Cuarta.- Si el software sin mantenimiento no es sensitivo a cambios en el ambiente, entonces debe estar muy bien aislado de cambios en el ambiente que interfieran con su habilidad de sobrevivir intacto.

El software que se ajusta a esta definición típicamente reproduciría una función muy estrecha (tal como la subrutina de conversión de temperatura) y sería bien protegidapor medio de programación por partes.

La programación por partes aisla el software de los cambios. Por ejemplo, una parte asignada a la representación gráfica de datos numéricos, puede estar bien aislada del software de un sistema de base de datos para gerencia, que suministra datos primarios y que puede aparecer y desaparecer tanto como el sistema de base de datos cambia.

El particionamiento, sin embargo, requerirá un esquema (similar en concepto al "Open System Interconect Model" para comunicaciones) que defina el software particionado y las interfaces entre las partes, el que asumirá importancia crítica en la arquitectura de software.

Quinta.- El software sin-mantenimiento es más fácil de reemplazar que de modificar.

Si el software es descartado y reemplazado por algo nuevo, el software original no es modificado porque es sin-mantenimiento por definición. El código descartado, producido por cierto generador de aplicaciones, está de acuerdo a estos requerimientos, porque los cambios son hechos a los requerimientos de entrada al generador y no al código generado. La idea de código que puede ser descartado implica que tiene código sin-mantenimiento en sus raíces.

Tome nota de que la especificación no es que el software sin-mantenimiento es facilmente reemplazo, sino que su remplazo es más fácil que su modificación.

Sexta.- El software sin-mantenimiento es casi imposible de modificar.

En el extremo de la definición, el software simplemente no puede ser modificado. Por ejemplo, no hay método o herramienta a emplearse para modificar el software. En un caso menos riguroso, el software necesita estar seguro contra los intentos de programadores impertinentes

o un cambio accidental. Las actuales técnicas de seguridad y de control de cambios pueden facilmente prevenir que el software de producción sea alterado.

CAMBIO DE MENTALIDAD EN IS (Sistema de Información)

Software sin-mantenimiento es una idea atractiva, pero sólo será realidad si IS cambia la manera como piensa acerca del software –cómo trabaja y qué cosa enfatiza—. Este cambio de pensamiento debe también extenderse a como los usuarios e IS encajan en el proceso.

Con todos los cambios, el mantenimiento cada vez más estará en las manos de las masas. Los paramétros serán controlados por el usuario final, mientras que otros parámetros serán controlados por grupos centrales tales como organizaciones IS y oficinas del gobierno.

Las compañías se están moviendo hacia una etapa en que los usuarios tendrán la habilidad de ser sus propios programadores (probablemente vía comandos verbales). Los usuarios cotidianamente dirigirán el software básico de soporte sobre como comportarse, dentro de los confines de lo que el software original fue diseñado para hacer.

Sin embargo, habrá un grupo de programadores maestros responsables del software básico, que será diseñado para ser sin-mantenimiento y muy sólido en el concepto de aceptar cambios de parámetros.

Estos programadores maestros, requerirán muy altos niveles de habilidad y sofisticación para manejar el desarrollo de software. El software desarrollado para ser sin-mantenimiento será complejo y existirá en un ambiente complejo, de tal manera que la potencia de desarrollo de software debe aumentar.

Los programadores deben crear software libre de errores (o bug-tolerant) en forma rutinaria, de tal manera que ellos buscarán avances importantes en técnicas de diseño y pruebas. La arquitectura del software en sí misma deberá ser mucho más científica, y las herramientas disponibles para el programador deberán ser más avanzadas. Las corrientes actuales sobre software estratificado son prometedoras, con los enfoques de orientación a objetos, diseños funcionales e interfaces estandard, trayendo abajo el mito de la programación monolítica. En realidad estos cambios reflejan la maduración del desarrollo del software en una ciencia.

Aunque esto no sucederá por algún tiempo, el mantenimiento como lo vemos ahora, será visto en el futuro por las nuevas generaciones de IS, como una reliquia de los estados iniciales de la evolución del software.

UNQUOTE"

Las ideas del Sr. Hearn nos hacen pensar en dos áreas, los desarrollos futuros, que veremos relacionados con programación y las técnicas de programación, que deberíamos ir pensando en adoptar, si no queremos quedar obsoletos por el rápido avance de esta tecnología.

En cuanto a los desarrollos futuros

que ya estamos viendo, podemos mencionar a la llamada Fussy-logic o lógica no exacta, técnica desarrollada en los Estados Unidos en la década del 60/70 por el profesor Zadec y que en norteamérica no logró muchos adeptos. Sin embargo, los japoneses han venido desarrollando esta técnica y han logrado considerables avances. Hoy día, también en Estados Unidos se viene aplicando esta técnica y ya se han hecho pruebas de programación, obteniendo resultados sorprendentes en cuanto a velocidad de programación y calidad (error-free).

El Sr. Hearn habla de programación orientada a los objetos, ésta también es una técnica que viene adquiriendo mayor velocidad de desarrollo. Otra de las tecnologías que viene avanzando es la de reconocimiento de la voz, técnica que permitiría un considerable avance en lo que respecta a la facilidad de programación.

Debemos además considerar la técnica de repositorios, técnicas avanzadas para corregir errores y el uso de herramientas CASE para considerar las posibilidades que en el cercano futuro se nos presentan.

Pero quizás, lo más importante que debemos rescatar del artículo del Sr. Hearn, es el cambio de mentalidad con que debemos ir enfocando nuestro trabajo de programación.

Es necesario que pensemos más en la estandarización de la programación en los diferentes niveles a los que podemos aplicarla.

Así, podemos hablar de estandarización de nuestros métodos de programación, estandarización de técnicas de programación y por último de estandarización de los procesos de programación.

Podemos pensar en un sistema de programación por medio del cual, todos los programas terminan en el mismo estandard de salida irrelevante de quien es el usuario o que requiere el usuario. Bajo este concepto todos los programas quedarían desligados del problema de definición del output, haciendo más fácil la definición y la programación.

Por otro lado, desarrollaríamos lo que sería un programa maestro, estandar de producción de outputs, que teniendo un solo tipo de entrada, la que mencionamos en el párrafo anterior, tendría la opción de crear a requerimiento del usuario, los outputs que éste quisiera diseñar para atender sus necesidades.

Debemos pensar en la mecanización de áreas aún no pensadas, como las de normas y políticas de las compañías, métodos de hacer negocios y métodos de toma de decisión. Indudablemente estamos hablando aquí de un futuro no muy cercano, particularmente para algunas compañías, pero es algo que ya no puede estar fuera de nuestros pensamientos si queremos estar en la línea de avanzada. ●