CAPÍTULO X

REINGENIERÍA DE PROGRAMACIÓN [18]

1. Introducción

Uno de los problemas que enfrenta el campo de la computación es el alto costo que representa el desarrollo del software. El equipo, con todos sus avances en desempeño, velocidad y capacidad, resulta inútil sin una colección de programas para operarlo, que van desde sistemas operativos, compiladores, paquetes de aplicación, hasta sistemas expertos y programas hechos a la medida. Las técnicas de programación se están depurando y se está promoviendo una mayor disciplina, lo cual se traduce en disminución de costos y tiempo de desarrollo, a la vez que se aumenta la calidad del producto.

Sin embargo, los problemas con los productos de programación no se limitan a fallas durante la ejecución, sino que incluyen a aquellos que surgen en el desarrollo, la administración y el mantenimiento del creciente volumen de software existente, aunado a los problemas de tipo social derivados por el contacto con las personas involucradas. Los problemas asociados con el desarrollo de software tienen sus raíces en la planeación deficiente, agravada por un control de calidad inadecuado durante las fases del ciclo de vida de éste. Estas deficiencias subsisten y se manifiestan durante las etapas de desarrollo y mantenimiento.

Durante años se esperó que los problemas de mantenimiento del software y el envejecimiento de los sistemas, desaparecieran con la introducción de nuevas tecnologías de desarrollo y el reemplazo gradual de las ya existentes. Sin embargo, la historia y la experiencia nos muestran que ésta es una falsa esperanza.

En 1989 un estudio de 862 departamentos de desarrollo en los Estados Unidos arrojó que 63% del tiempo de los empleados se destinaba al mantenimiento del software o en actividades relacionadas con éste (MC92). En un artículo de febrero de 1994, Kristin Marks afirma que las actividades de soporte, actualización y entrenamiento ocupan una proporción de tres cuartas partes del ciclo de vida del software (MA94). Un artículo de la revista Software Development, agosto de 1994, estima que el mantenimiento de un programa de un millón de líneas puede alcanzar un costo anual de dos millones de dólares, situación que ha causado reacciones, tales como la creación del nuevo Estándar 1219-1993 de la IEEE (LE94).

---

18 Reingeniería de Programación: Una propuesta para la solución de los problemas de mantenimiento de software. Tesis de Ingeniero en Computación.
Autores: Héctor Francisco Bautista González, Cuauthemoc Freyre Mercado y Norma Susana Zavala Carrasco. FI-UNAM 1995.

Aunque no contamos con algún estudio que nos indique la gravedad del problema en México, podemos esperar peores proporciones si consideramos que tardamos más tiempo en adoptar nuevas tecnologías de desarrollo.

A pesar de la introducción de nuevas tecnologías de desarrollo a principios de los noventa, la mayoría de los esfuerzos siguen concentrados en el mantenimiento del software. Confiar en que las estrategias de desarrollo van a resolver el problema no revertirá la tendencia. El desarrollo de tecnologías de software sólo puede tener un impacto limitado en el mantenimiento; hay dos razones para esto.

La primera es que la causa del mantenimiento del software no está entendida. La mayor parte del trabajo de mantenimiento es originada por cambios en los requerimientos del sistema, no por defectos en el software. Los programas utilizados en la industria sufren modificaciones constantemente, con objeto de adaptarse a las nuevas tecnologías y a las necesidades del usuario.

Aun el sistema que es totalmente confiable, que cubre totalmente las necesidades del usuario y que está bien estructurado, requiere de mantenimiento. A menos que las nuevas tecnologías, como CASE, nos ayuden a construir sistemas que acepten cambios fácilmente y sin perder calidad, el mantenimiento del software continuará siendo una actividad consumidora de tiempo y dinero. La clave para reducir el mantenimiento es hacer más fáciles y seguros los cambios en los sistemas.

Segundo, es muy improbable que los sistemas existentes sean reemplazados en corto tiempo por sistemas nuevos, más fáciles de mantener. No solamente fue muy grande la inversión de dinero para crear estos sistemas, sino que también es muy grande la inversión necesaria para su cambio.

De acuerdo con esto, la acción idónea para reducir los problemas en mantenimiento de software es adoptar una serie de medidas más agresivas y directas, como las siguientes:

- Anticipar y enfrentar eficientemente los problemas que surgirán al realizar cambios en el software.
- Instituir buenas prácticas de dirección, control y planeación para el mantenimiento.
- Aumentar la eficiencia y calidad de las actividades de mantenimiento.
- Mejorar la aptitud de los sistemas existentes para recibir mantenimiento.

La *reingeniería* es una medida directa auxiliada con herramientas automáticas para el mantenimiento del software. En principio, es un reconocimiento de que el problema existe y de que ignorarlo no va a resolverlo. La reingeniería nos brinda una perspectiva en la que el software existente se concibe como un activo que debe de ser administrado y protegido adecuadamente, que puede ser utilizado nuevamente. Incorpora una visión integral del ciclo de vida del software en el cual mantenimiento y desarrollo son actividades conjuntas, dado que comparten las mismas metodologías y los mismos tipos de herramientas.

Una parte esencial de cualquier estrategia de ataque directo es el uso de herramientas automáticas para completar el ciclo de vida del software, proceso denominado "automatización del software". En la actualidad contamos con dos herramientas automáticas: CASE, para las actividades relacionadas con el desarrollo de programas, y la reingeniería, enfocada al mantenimiento de software.

Sin embargo, es muy importante destacar que la reingeniería no es un procedimiento rígido y lineal, sino un conjunto de estrategias que pueden ser aplicadas total o parcialmente en un problema dado, de acuerdo con las características de éste y al criterio del grupo de trabajo que esté encargado de realizar la tarea de mantenimiento.

2. Mantenimiento del software

Los problemas de mantenimiento de un producto de programación han crecido gracias a que se cree que el mantenimiento es más sencillo de llevar a cabo que el desarrollo, por lo que puede ser realizado por personal con menos experiencia, herramientas y dirección. Por lo contrario, el mantenimiento del software generalmente presenta más retos que el desarrollo. ¿Por qué? ¿Qué es exactamente lo que hacen las personas de mantenimiento?

2.1. Definiciones de mantenimiento

El mantenimiento es el último proceso en el ciclo de vida del software; a través de los años se han propuesto muchas maneras de definirlo, a continuación se listan varios ejemplos representativos (MC92).

- Mantenimiento es la modificación de un producto de software para mejorar su desempeño y demás atributos o adaptarlo para el cambio a un nuevo ambiente (ANSI/IEEE Std. 729-1983).

- Mantenimiento es hacer depuraciones del software.
- Mantenimiento es el proceso de modificar un software existente dejando sus funciones intactas.
- Mantenimiento es el mecanismo para combatir el deterioro del software que con el tiempo tienda a ser no estructurado, ilegible o resistente a cambios.
- Mantenimiento se refiere a modificar un programa, actualizando funciones para agregar nuevas construcciones o herramientas.
- Mantenimiento es realizado como respuesta a una serie de cambios requeridos en procesamiento de datos para eliminar ineficiencias y facilitar futuros cambios.
- Mantenimiento incluye actualizaciones a las aplicaciones existentes.
- Mantenimiento consiste en hacer cambios a un programa de software después de que ha sido puesto en producción.
- Mantenimiento es adaptar el software constantemente para las necesidades del negocio.

Estas definiciones identifican tres actividades fundamentales:

- ❑ Corregir los errores.
- ❑ Revisar los requerimientos originales.
- ❑ Mejorar el funcionamiento, aumentar su desempeño e introducir funciones nuevas.

3. Naturaleza del trabajo de mantenimiento

Frecuentemente el mantenimiento se relaciona con la corrección de errores, el argumento es que el mantenimiento correctivo es la actividad que más se realiza. Sin embargo, muchos estudios muestran que esto es incorrecto. Por ejemplo, un estudio de principios de los ochenta realizado por Lientz y Swanson sobre 487 organizaciones de procesamiento de datos, enseñó que sólo 20% del trabajo de mantenimiento es el de corrección. Jorge DiNardo en un estudio más reciente de 25 organizaciones con equipo IBM, demostró que sólo 17% del trabajo de mantenimiento es de corrección (MC92). Ambos estudios reportan que el mayor esfuerzo de mantenimiento es el desarrollar adaptaciones y mejoras (aunque los estudios tiene muy diferentes puntos de vista de cómo los esfuerzos se dividen en mantenimiento de adaptación y de perfeccionamiento).

La mayor parte de los trabajos de investigación en mantenimiento han identificado al trabajo de mantenimiento adaptivo y perfectivo como las actividades dominantes en las organizaciones.

4. Factores que afectan el esfuerzo de mantenimiento

Los factores que incrementan el esfuerzo de mantenimiento son:

- El tamaño del sistema.
- La edad.
- La familiaridad del personal con el sistema.
- El nivel de experiencia.

Los sistemas de software más grandes requieren de mayor esfuerzo de mantenimiento que los sistemas más pequeños. Existe una curva de aprendizaje asociada con los sistemas grandes; entre más grandes son, las funciones son más complicadas y complejas. También, los sistemas viejos requieren de mayor mantenimiento que los sistemas nuevos. Los sistemas de software tienden a crecer con el tiempo, a encontrarse menos estructurados al ser modificados, y ser menos entendibles cuando se cambia el grupo de personas que lo manejan. Una gran cantidad de trabajo de mantenimiento se refiere al mantenimiento de corrección, especialmente rutinas de corrección de errores para programas grandes y viejos.

Así mismo, una gran parte del trabajo de mantenimiento se gasta en mantenimiento correctivo por caídas del sistema. Personas de mantenimiento que no formaban parte del grupo original que desarrolló el sistema, dedican más tiempo tratando de entender el sistema, ya que no están familiarizados con la estructura de éste ni con la forma de trabajar de los autores originales.

Estos factores predicen un incremento en el mantenimiento del sistema conforme éste va creciendo o habiendo cambios en el personal.

5. Mejoramiento del mantenimiento durante el desarrollo

Muchas actividades realizadas durante el desarrollo del software mejoran el mantenimiento del producto. Algunas de ellas se listan y se analizan a continuación.

5.1. Actividades de análisis

- Desarrollar estándares y guías.
- Fijar logros en los documentos de apoyo.
- Especificar procedimientos de control de calidad.
- Identificar probables mejoras del producto.
- Determinar recursos requeridos para el mantenimiento.
- Estimar costos de mantenimiento.

5.2. Actividades de diseño arquitectónico

- Subrayar la claridad y modularidad como criterios de diseño.
- Diseñar para facilitar probables mejoras.
- Usar notaciones estandarizadas para documentar flujos de datos, funciones, estructura e interconexiones.
- Observar los principios de encapsulamiento de información, abstracción de datos y descomposición jerárquica hacia abajo.

5.3. Actividades de diseño detallado

- Uso de notaciones estandarizadas para especificar algoritmos, estructuras de datos y procedimientos para especificar interfaces.
- Especificar efectos colaterales y manejo de excepciones para cada rutina.
- Proporcionar directorios en referencia cruzada.

5.4. Actividades de implementación

- Usar estructuras de una sola entrada y una sola salida.
- Introducir sangrado estándar en las estructuras.
- Emplear un estilo de codificación simple y claro.
- Usar constantes simbólicas para asignar parámetros a las rutinas.
- Proporcionar margen de recursos.
- Habilitar prólogos estándar de documentación en cada rutina.
- Apegarse a las guías de comentarios estándar internos.

5.5. Otras actividades

- Elaborar una guía de mantenimiento.
- Desarrollar un juego de pruebas.
- Proporcionar la documentación del juego de pruebas.

6. Reingeniería

La reingeniería es la manera de realizar el mantenimiento de una forma automática, aplicando herramientas, técnicas y metodologías para extender la vida útil de un sistema a un bajo costo. La reingeniería de programación considera el mejoramiento del proceso de mantenimiento al sugerir una estrategia a largo plazo, en lugar de ejecutar los cambios como se van presentando.

La reingeniería es el proceso de examinar un sistema de software ya existente (programa) y modificarlo con la ayuda de herramientas automáticas para:

- Incrementar la disposición del software que recibirá mantenimiento.
- Incrementar su nivel tecnológico.
- Extender sus expectativas de vida.
- Capturar sus componentes en una biblioteca donde las herramientas CASE se puedan utilizar para su soporte.
- Incrementar la productividad en el mantenimiento.

6.1. Propósitos de la reingeniería

- Mejorar el manejo de los sistemas existentes.
- Proveer asistencia automatizada para el mantenimiento.
- Reducir errores y costos de mantenimiento.
- Hacer sistemas fáciles de entender, modificar y analizar.
- Otorgar sistemas de conversión y migración.
- Forzar a utilizar estándares en los desarrollos (nuevos y antiguos).
- Mejorar el tiempo de respuesta a las solicitudes de mantenimiento.
- Mejorar el mantenimiento.
- Proteger y extender la vida de los sistemas.
- Utilizar CASE para soportar los sistemas actuales.
- Reutilizar los componentes de los sistemas actuales.

La reingeniería puede ayudarnos a entender sistemas existentes y a descubrir componentes (arquitectura, estructuras de datos) de software que sean comunes a lo largo de los mismos sistemas. Estos componentes pueden ser usados –o reusados– en la fabricación de nuevos sistemas, disminuyendo así el tiempo de desarrollo.

7. Tipos de reingeniería

7.1. Reestructuración

Es el proceso de cambiar la forma del software (e. g., nombres de datos y definiciones, y fuentes de códigos del programa) sin alterar su funcionalidad. La razón principal de la reestructuración es hacer que el programa sea más fácil de entender.

7.2. Ingeniería inversa

Es el proceso de analizar el sistema para construir una descripción de sus componentes y de sus interrelaciones entre sí. El resultado es una descripción de alto nivel (diagramas de flujo, diagramas entidad relación, código fuente, etc.) del programa a partir de sus niveles más bajos de información (en muchos casos el código fuente). El propósito de la ingeniería inversa es el de actualizar la documentación o volver a documentar el sistema y descubrir su diseño como una ayuda para hacer el programa más entendible o para migrarlo a una nueva tecnología.

7.3. Administración de configuración y cambios

Es la actividad de generar y organizar la información referente a la evolución de los programas que se encuentran en las etapas de desarrollo o de mantenimiento. Permite administrar las bibiliotecas que almacenan estos datos y controlar los cambios que se efectúan a las diferentes versiones del producto.

Aunque los conceptos envueltos en la reingeniería (reestructuración, herramientas de medición, CASE, etc.) no son nuevos, *la idea de redesarrollar los sistemas viejos y actualizarlos dentro de una nueva estrategia, utilizando los mismos conceptos que en el desarrollo sí lo es*. Los elementos aplicados a la reingeniería incorporan varias de las mejores ideas del pasado y del presente de la ingeniería de software, lo que permite que se puedan aplicar a una gran variedad de empresas, no importando la metodología, el hardware, ni mucho menos el lenguaje utilizado en sus sistemas. Sin embargo, sí deberá existir una metodología de desarrollo y un ciclo de vida de software, así como políticas y procedimientos para aceptar y controlar los cambios realizados. Al contar con estos requisitos, la labor de reingeniería puede ser implementada utilizando las mismas técnicas que se emplean actualmente en el desarrollo de nuevos sistemas, pero aplicándolos a los sistemas existentes (RA92).

8. Tipos de herramientas de reingeniería

Las herramientas son una parte importante de la reingeniería al cambiar sustancialmente la productividad y calidad del trabajo de mantenimiento. En la siguiente tabla se muestran ocho tipos básicos de herramientas. Estas no solamente ayudan a los programadores a realizar pruebas más eficientes, sino también mucho más completas e intensivas de lo que es posible con técnicas manuales. Pruebas como las de descubrir códigos muertos o variables inutilizables en programas largos y complejos antes eran imposibles y ahora con la ayuda de herramientas de reingeniería se pueden realizar fácilmente. Al final del capítulo se proporciona una tabla de 57 herramientas de reingeniería. Estas herramientas se encuentran clasificadas como sigue:

8.1. Analizadores de programa

- Rastreadores de lógica y de datos
- Referencias cruzadas
- Delineadores (*Profilers*)

8.2. Métricas

- Programas monitores de estándares
- Programas analizadores de calidad
- Programas controladores de complejidad

8.3. Reestructuración

- Reestructuradores de procesos lógicos
- Estandarización de nombres de datos y definiciones

8.4. Ingeniería inversa

- Ingeniería inversa de datos
- Ingeniería inversa de lógica

8.5. Pruebas

- Generadores de datos para pruebas
- Analizadores de alcance para pruebas
- *Debuggers*
- Comparadores

8.6. Convertidores

- De lenguaje

8.7. Manejadores de configuración y cambios

- Manejadores de control de cambio
- Manejadores de librerías
- Generadores de código

8.8. Herramientas de redocumentación

- Referencias cruzadas
- Generadores de diagramas

9. El puente a nuevas tecnologías

Aunque el eje principal de la reingeniería se utiliza en el mantenimiento del software, la reingeniería es más que simplemente una ayuda para el mantenimiento de éste. Es el puente de técnicas antiguas y obsoletas a tecnologías nuevas que se deben de implementar en las organizaciones para responder a los cambios constantes en los negocios. Para muchas organizaciones la reingeniería no es una opción, es una necesidad si quieren proveerse de software con costos moderados que les ayuden a alcanzar una ventaja competitiva.

Probablemente el beneficio más importante que la reingeniería lleve consigo, es el unir las funciones de desarrollo del software y de mantenimiento en una sola, esto se logra incorporando las mismas herramientas y técnicas en las dos fases del ciclo de vida del sistema. Información acerca del sistema, de sus componentes y de las interrelaciones entre sí después de aplicar la reingeniería pueden ser utilizados por herramientas CASE para proveer un soporte en el futuro.

10. Reingeniería contra reemplazo

Sistemas frágiles que son altos candidatos para la reingeniería, son sistemas que:

- Son de notable importancia para la corporación.
- Son el blanco de frecuentes trabajos de mantenimiento y requieren de un gran porcentaje de los recursos asignados al mantenimiento.
- Solamente uno o unas cuantas personas selectas entienden el sistema y puedan realizar cambios en él.
- Contiene errores que nadie puede encontrar.
- Se requiere actualizarlos o incrementar su tecnología.

A continuación se presenta un resumen de las razones para reingeniería y para el reemplazo; indicándose las labores por realizar cuando un programa muestra determinadas características.

10.1. Reestructuración de código del programa
- Violación de estándares
- Código incongruente
- Pésima documentación
- Nombres sin sentido
- Lógica compleja

10.2. Reestructuración de datos
- Datos mal organizados
- Definición de datos no estándar
- No existe diccionario de datos

10.3. Ingeniería inversa, migración
- Tecnología obsoleta
- Lenguaje y DBMS antiguos
- Falta de especificaciones de diseño

10.4. Reemplazo total/parcial
- Algoritmo pésimo
- Ilegible
- Funcionalmente incompleto o incorrecto
- Diseño de BD defectuoso

11. Reconocimiento de los sistemas frágiles

El concentrar las tareas de reingeniería en los sistemas frágiles permite a las organizaciones reducir el mantenimiento y disminuir su costo asociado. Por supuesto, lo anterior depende de poder reconocer a un sistema frágil.

La mayoría de las organizaciones mantienen una hoja de registro de los sistemas que poseen. Los registros contienen varios puntos para enumerar cada sistema como los que se muestran a continuación:

- Número de líneas de código.
- Número de funciones.
- Tiempo.
- Lenguaje y versión del lenguaje.
- Procesamiento por lotes o en línea.
- Ambiente operativo (hardware, software, DBMS, TP monitor).
- Costo de mantenimiento por año.
- Estimar la producción de errores.

- Costo y tiempo promedio de corrección de errores.
- Costo y tiempo promedio de corrección de cambios.
- Número de errores corregidos por año.
- Número de errores pendientes.
- Número de cambios requeridos por año.
- Número de cambios requeridos pendientes.
- Requerimientos personalizados.
- Satisfacción del usuario.
- Opinión de la persona de mantenimiento sobre la disposición del sistema para recibir mantenimiento.

Los registros son utilizados para dar información sobre las características y el comportamiento de los sistemas existentes para hacer una comparación entre sí, y pueden ser utilizados para sugerir sistemas candidatos para la reingeniería.

Sin embargo, estos registros por lo general están escritos a mano, incompletos, inexactos, y desactualizados. No necesariamente el sistema más antiguo, o el más largo, o el menos estructurado, es el más frágil. Tal vez sólo algunos módulos, no todo el sistema, son la causa de grandes problemas de mantenimiento. Con la ayuda de herramientas de reingeniería y de registros de mantenimiento se pueden detectar los sistemas frágiles y aquellas porciones de código que son más complejas y con posibilidad de tener errores.

12. Analizadores de código y herramientas de medición

Los herramientas de medición proporcionan una medida aceptable de la disposición del software para recibir mantenimiento. Diferentes herramientas de medición miden diferentes características de calidad de los programas como son: la capacidad de ser probado, lo entendible que es, y la facilidad que tiene de modificarse. No se puede controlar lo que no se puede medir.

La mayoría de las herramientas de medición leen el código fuente del programa para producir:

- **Medidas de complejidad**
  - Métricas de tamaño
  - Métricas de flujo
- **Reportes analíticos**
  - Listados
  - Cálculos diversos

12.1. Qué tan entendible es el software

Para dar mantenimiento a un programa es necesario entenderlo, la decisión de dónde reemplazar o reestructurar muy frecuentemente se basa en cuánto se entiende el programa.

12.2. Factores de entendimiento

Son muchos los factores que hacen más fácil o difícil de interpretar un programa en particular. Estos factores se pueden catalogar en dos grupos: habilidad para programar y forma del programa.

Los *programas claros* se pueden caracterizar por ciertas propiedades como las siguientes:

- Modularidad.
- Consistencia de estilo.
- Sin *trucos* o código intrincado.
- Uso de datos con significados específicos y nombres claros en los procedimientos.
- Estructuración.

El estructurar un programa tiende a incrementar la claridad, ya que se establece un patrón en el programa. Sin embargo, Boehm sugiere que además de una buena estructura, un programa debe ser conciso, consistente y completo.

Un programa *conciso* es aquel en el que no se presenta un exceso de piezas. En un programa conciso cada instrucción debe ser ejecutada al menos una vez.

Un programa *consistente* es aquel que es escrito con un estilo coherente y sigue una línea de diseño. La consistencia en el estilo del código implica que el programa contenga notación, terminología y simbología consistente, respetando los nombres convencionales corporativos y estándares.

Un programa *completo* es aquel en el que se disponen de todos sus componentes o funciones. Si se carece de alguna parte del código fuente o de la especificación del diseño, el programador pierde confianza en su capacidad para corregirlo.

13. Métricas de complejidad

Existen diferentes propuestas para medir las características que hacen que un programa sea complejo. Estas propuestas se pueden dividir en dos grandes grupos: métricas de tamaño y métricas de control de flujo.

Las métricas de tamaño se basan en la premisa que cuanto más *largo* es un programa, más complejo es.

Las métricas de control de flujo miden qué tan complejo es un sistema basándose en el número de decisiones, sentencias como if, for, while, case, etc. Algunas medidas utilizan la teoría de gráficas para visualizar y abstraer el flujo en un programa.

Las dos métricas más divulgadas son la *Ciencia Halstead del Software* (métrica de tamaño) y la *Complejidad Ciclomática de McCabe* (métrica de control de flujo).

13.1 Complejidad ciclomática de McCabe

La complejidad ciclomática es una teoría de gráficas que clasifica los caminos que se forman con las trayectorias únicas.

La medida McCabe no sólo mide lo complejo del programa, sino que además identifica el número exacto de caminos lógicos, indicando el número mínimo de pruebas por realizar (Una por cada camino).

Para determinar la complejidad ciclomática de un programa, se utiliza la siguiente fórmula:

$$v(G) = e - n + 2$$

Donde *e* representa las líneas o trayectos entre cada instrucción y *n* representa nodos o puntos. Es decir:

$$Complejidad\ ciclomática = caminos - declaraciones + 2$$

Existe otro método de calcular el número de complejidad ciclomática:

$$v(G) = regiones\ encerradas\ en\ la\ gráfica\ de\ flujo + 1$$

Finalmente, ya que una declaración condicional produce una región encerrada, de ésta resultan por lo menos dos caminos, por lo que la complejidad ciclomática puede ser determinada simplemente contando las declaraciones condicionales (IF):

$$v(G) = conteo\ de\ IF + 1$$

13.2. Extensión de la complejidad ciclomática

En vez de simplemente contar los IFs, esta medida también cuenta los ANDs y los ORs, con lo que se incluye la dificultad de las declaraciones condicionales IF <condición> AND <condición> OR <condición> .

13.3. Complejidad esencial

El cálculo de la estructura se realiza contando el número de veces que un camino realiza un *brinco* fuera de un módulo y no regresando a su punto de partida. En un programa perfectamente estructurado, todos los códigos se ejecutan en ese momento o por vía de llamadas a procedimientos, garantizando que todos los brincos tienen un regreso automático.

El porcentaje de estructuración puede ser calculado como sigue:

$$Porcentaje\ de\ estructuración = \frac{número\ de\ divisiones\ que\ regresan}{divisiones\ totales}$$

El porcentaje de estructuración debe ser 100%.

En la Tabla 1 se tiene un compendio de las métricas de complejidad de McCabe.

Complejidad ciclomática	Cuenta el número de caminos lógicos en un módulo. Este debe ser menor o igual a 10.
Extensión de la complejidad ciclomática	Cuenta el número de caminos lógicos en un módulo, lo suma al número de AND y OR
Complejidad esencial	Mide el grado de estructuración de un módulo al contar el número de GO TO. El resultado debe ser 1.

Tabla 1. Métricas de complejidad de McCabe

14. Ciencia Halstead del software

La teoría de Halstead se basa en contar los operadores y operandos en un programa:

- *Operadores* son palabras reservadas para el lenguaje como: ADD, GREATER THAN, MOVE, READ, IF, CALL, PERFORM; operadores aritméticos como: **+**, **-**, *****, **/**; y operadores lógicos como: GREATER THAN o EQUAL TO.
- *Operandos* son los datos variables y las constantes en el programa.

Halstead distingue entre el número de operadores únicos y el número de operadores totales. Por ejemplo, un programa puede tener un READ, siete MOVE, y un WRITE; por lo tanto, tendrá tres operadores únicos, pero nueve operadores totales.

Para operandos se aplica el mismo tipo de razonamiento que para los operadores. En notación científica de Halstead:

n_1 = *operadores únicos*
N_1 = *total de operadores*
n_2 = *operandos únicos*
N_2 = *total de operandos*

14.1. Tamaño

El *tamaño N* de un programa se calcula como:

$$N = N_1 + N_2$$

Entre mayor sea el valor de la *N*, más difícil es de entender y mantener el programa.

El tamaño estimado, *N'*, se calcula con la siguiente fórmula:

$$Tamaño\ estimado\ N' = (n_1 * log_2(n_1)) + (n_2 * log_2(n_2))$$

Elshoff encontró que el tamaño aproximado, *N'*, se acerca más al tamaño actual, *N* –en programas bien estructurados–. Basado en este descubrimiento, se utiliza una comparación de *N* a *N'* como una prueba de estructuración. Esta medida se conoce como el radio puro y se calcula de la siguiente manera:

$$radio\ puro = N' / N$$

14.2. Volumen

Si dos programas tienen el mismo largo *N*, pero uno posee un mayor número de operadores únicos y de operandos, esto lo hace más difícil de entender, entonces este programa tendrá mayor volumen. La fórmula es:

$$volumen\ V = N * log_2(n)\ y\ n = n_1 + n_2$$

14.3. Esfuerzo

El esfuerzo es una medida del trabajo requerido para desarrollar un programa. La fórmula es:

$$esfuerzo\ E = V / L$$

El nivel del lenguaje *L* indica qué tan poderosas son las instrucciones de un lenguaje:

$$L = (2*n_2)/(n_1 *N_2)$$

A continuación se indican las ventajas de las medidas científicas de software de Halstead:

- Son fáciles de calcular y de automatizar, y no requieren regresar al análisis de las características del programa como el detalle de la gráfica de flujo del programa.
- Son aplicables a cualquier lenguaje de programación.
- Muchos estudios estadísticos de diferentes programas demuestran la validez como pronóstico del esfuerzo de programación y de encontrar el número de errores en un programa.

Las medidas científicas de software se encuentran resumidas en la siguiente tabla:

Longitud	$N = N_1 + N_2$
Tamaño estimado	$N' = (n_1 * log_2(n_1)) + (n_2 * log_2(n_2))$
Volumen	$V = N * log_2(n)\ donde\ n = n_1 + n_2$
Esfuerzo	$E = V / L$
El nivel del lenguaje (nivel de abstracción del lenguaje)	$L = (2*n_2)/(n_1*N_2)$

Tabla 2. Métricas ciencia Halstead del software

15. Reestructuración

Tanto la lógica como los datos (nombres de las variables, de los procedimientos, etc.) del programa pueden ser reestructurados para mejorar la comprensión.

> **Reestructurar es el proceso de estandarizar las variables y los nombres de los procedimientos, así como el mejorar la estructura lógica y la modularidad del programa para aumentar la productividad en el mantenimiento del software.**

15.1. Objetivos de la reestructuración

- Mejorar la claridad y simplificar la lógica.
- Disminuir las pruebas y tiempo de depuración.
- Obligar a la programación con estándares.
- Simplificar los cambios en el programa.
- Reducir los costos del mantenimiento.
- Mejorar la respuesta a las solicitudes de mantenimiento.
- Aumentar la calidad de los programas.
- Incrementar la satisfacción de los usuarios hacia los programas.
- Reducir la dependencia hacia una persona para realizar el mantenimiento.
- Prepararse para la migración del software hacia otra plataforma o tecnología.
- Conservar al software frágil.

El primer objetivo de la reestructuración es el mejorar la *claridad* del programa. Los propósitos a corto plazo de la reestructuración son: el mejorar de manera general la calidad del software e incrementar su valor para la corporación. Propósitos a largo plazo son: preparar los sistemas para la migración a nuevas tecnologías, elaborar diccionarios de datos, sistemas idóneos para la ingeniería inversa, etc.

15.2. Reestructuración de la lógica del programa

La reestructuración de la lógica de un programa es el proceso de reordenar el código fuente de acuerdo con las reglas de programación estructurada.

Las tres funciones básicas de la reestructuración de la lógica del programa son:

- Reordenar la lógica del programa.
- Cambiar el código fuente.
- Normalizar el uso del lenguaje de programación.

15.3. Programas bien estructurados

La clave para controlar la complejidad es el normalizar la estructura del programa.

♦ Definición:

> La programación estructurada es un método para construir programas, utilizando un conjunto de reglas que requieren de un formato estricto de estilo, estructura de control modular y jerárquica, y un juego estricto de estructuras de control.

♦ Objetivos para crear un programa estructurado:

- Mejorar la facilidad de lectura.
- Minimizar la complejidad.
- Simplificar el mantenimiento.
- Incrementar la productividad en el desarrollo.
- Proveer de una disciplina para la programación.

♦ Medidas para cumplir con los objetivos de la programación estructurada:

- Minimizar el número de caminos lógicos en el programa.
- Limitar los caminos patrones del programa a través de reestructurar las estructuras de control utilizadas.
- Regresar el control del programa al camino lógico principal después de hacer una llamada a un procedimiento.

♦ Resumen de las propiedades de un programa bien estructurado:

- El programa es dividido en un conjunto de módulos con un ordenamiento jerárquico.
- Cada módulo representa una función lógica.

- Las estructuras de control del programa son: secuencia, selección y repetición.
- La ejecución del programa es restringida a un sistema en el cual el control entra en el módulo, se procesan las instrucciones, se libera el control en la salida del módulo, y se regresa al lugar en que fue invocado el módulo.
- El procesamiento de errores es seguido por un control normal, excepto en los casos de datos irrecuperables, donde un proceso normal no puede continuar.
- Cada variable del programa sirve sólo para un propósito y el alcance de la variable es aparente y limitado.

15.4. Cuándo reestructurar

Resumen de las características de los programas que son candidatos para ser reestructurados.

- Pésima calidad de código.
- Código imposible de leer, cambiar, y analizar.
- Alto índice de errores, tiempo de corrección y costo.
- Gran número de requerimientos especiales.
- Sistemas que son importantes, caros y modificados frecuentemente.

15.5. Herramientas de reestructuración lógica del programa

Las herramientas de reestructuración lógica leen el código fuente del programa, analizan y reestructuran los caminos de control, creando así un código fuente nuevo, estructurado, formateado y funcionalmente equivalente al programa original.

16. Reestructuración de datos

Nombres con significado y definiciones estándares para los datos mejoran la facilidad de lectura de un programa. Por ejemplo, considérese las dos declaraciones siguientes de COMPUTE COBOL:

COMPUTE GASTOS=TRANSPORTACION + COMIDAS + ALOJAMIENTO

COMPUTE E = E1 + E2 + E3

La primera expresión da sustancialmente mayor información acerca del programa, gracias a los nombres seleccionados para los datos.

Debajo hay tres variables con nombres diferentes para los números del empleado y tres definiciones de tipo.

¿Cuál es correcto y cuál el que debe utilizar el encargado de mantenimiento?

NUMERO_DE_EMPLEADO	*PIC X(4)*
CP_EMP_NO	*PIC 9999*
EMPL_NUM	*PIC 9(9)*

El estandarizar los nombres de los datos y definiciones disminuye la confusión, el esfuerzo y los errores en el mantenimiento.

Como la reestructuración lógica del programa, la reestructuración de los datos mejora el mantenimiento de los sistemas existentes a través de mejorar su entendimiento. La reestructuración lógica estandariza la lógica del programa; la reestructuración de datos estandariza los datos y definiciones del programa.

Definición:

> **La reestructuración de datos es el proceso de normalizar los nombres de las variables y las definiciones de los datos o tipos a lo largo del sistema.**

Beneficios de la reestructuración de datos:

- Mejora el entendimiento de los sistemas existentes.
- Disminuye los costos de mantenimiento.
- Incrementa la productividad de mantenimiento.
- Respalda la incorporación de estándares y convenciones en la programación.
- Elimina inconsistencias en los nombres de variables, longitudes de campos y redundancias en los datos (varias variables que realizan la misma función).
- Posibilita la creación de un diccionario de datos.
- Provee de una documentación de todos los datos utilizados.
- Coloca al sistema en posibilidad de migrar a otra tecnología.

16.1. Candidatos para la reestructuración de datos

Sistemas con las siguientes características son candidatos para la reestructuración de datos:

- Nombres de variables sin sentido o no estandarizados.
- Definiciones inconsistentes de tipos de datos.
- Nombres de procedimientos inconsistentes.
- Código difícil de auditar.

16.2. Herramientas de reestructuración de datos

Como la reestructuración lógica del programa, la reestructuración de datos puede realizarse de manera más eficiente y efectiva con la ayuda de herramientas automáticas de reingeniería. Las herramientas de reestructuración de datos reúnen, analizan y modifican los nombres de las variables y sus definiciones de tipo a través del programa, a fin de asegurarse de que todas las variables en el sistema se encuentran definidas de igual manera (del mismo tipo) y con el mismo nombre.

16.3. Ingeniería inversa

La *ingeniería inversa* es un proceso mediante el cual se examinan las descripciones físicas de los programas, tales como el código fuente o las descripciones de las tablas, para obtener o construir las especificaciones de alto nivel que ilustran la lógica del proceso y los datos de dichos programas.

Esta técnica se ve apoyada por herramientas automatizadas que juegan un papel muy importante en la calidad de los resultados, por lo que deben ser consideradas como parte integral de todo el procedimiento.

Es parte fundamental de este procedimiento el transformar una representación de bajo nivel como lo es el código fuente o el lenguaje de descripción de datos (DDL) en una descripción equivalente de un nivel de abstracción más alto, conservando la esencia del sistema. Para llegar a una descripción de alto nivel no basta con herramientas automáticas y código fuente. Es indispensable la actuación de una persona, por lo que se debe concebir la ingeniería inversa como una práctica interactiva que requiere intervención humana y autómata.

Durante el proceso de ingeniería inversa, se examina la descripción en nivel físico para producir representaciones en un nivel superior de abstracción. El siguiente paso es utilizar herramientas CASE para realizar la tarea de codificar sin preocuparnos por la confección del nuevo sistema, mecanismo que se denomina *ingeniería progresiva* y completa la actividad de mantenimiento.

La ingeniería inversa se divide en dos grandes áreas: datos y lógica.

16.4. Ingeniería inversa de datos

La *ingeniería inversa de datos* consiste en extraer entidades, relaciones atributos y especificaciones de diseño a partir del código fuente, los diccionarios de datos y los DDL. Incluye mecanismos de creación, modificación, verificación y consolidación de las bases de datos, además de presentación y manipulación de diagramas. También ofrece la recreación de DDL desde los diagramas acualizados.

El método principia por transformar el código fuente, las descripciones de datos y los DDL en diagramas de especificaciones físicas, que son depositados en un almacén. Estos resultados pueden ser modificados y utilizados para generar nuevo DDL. También pueden ser transformados en un nivel más alto de abstracción, como diagramas entidad-relación, que pueden ser introducidos en el almacén, modificados y utilizados para generar diagramas de especificaciones físicas, y después obtener un nuevo DDL con los cambios físicos y lógicos. El siguiente paso es realizar una verificación de diagramas y una recomendación de sistemas expertos, así como generar la documentación y los archivos correspondientes a la descripción de la base de datos normalizada.

16.5. Ingeniería inversa de lógica

La *ingeniería inversa de lógica* es el proceso mediante el cual se detectan las estructuras de control y la jerarquía de éstas a partir del código fuente de un programa, constituyendo unas especificaciones de diseño que pueden ser utilizadas para regenerar programas, para documentarlos y eventualmente migrarlos a nuevas plataformas.

Esta técnica es muy similar al análisis de código porque utiliza la misma materia prima y produce la misma documentación. No obstante, el alcance de la ingeniería inversa es más extenso, ya que incluye mecanismos para manipular los resultados. Estas herramientas utilizan tres almacenes: de diseño físico, de diseño lógico y de código fuente.

El almacén de diseño físico es pieza central de la herramienta, dado que su nivel de detalle, revisión de errores y dispositivos de documentación determinan la naturaleza de este proceso. Éste se alimenta de la ingeniería inversa del código fuente o de la ingeniería progresiva del almacén de diseño lógico, y contiene típicamente la siguiente información: las estructuras de datos, la carta jerárquica, los diagramas HIPO, la lógica de control y los componentes de hardware. Este almacén tiene las siguientes prestaciones: lenguaje de diseño, modificación de diseño, revisión de errores, documentación y generación de código (ingeniería progresiva).

El almacén de diseño lógico suele estar integrado dentro de un ambiente CASE y captura las especificaciones de más alto nivel de abstracción, como lo son los diagramas de flujo de datos (DFD), las descripciones de los procesos y el diccionario de datos. Estas herramientas pueden tiener un ambiente gráfico para editar los diagramas, así como implementos de revisión de errores acordes con este nivel de abstracción. Pueden generar reportes con la documentación correspondiente a los diagramas que maneja, además de efectuar ingeniería progresiva para alimentar el almacén de diseño físico.

El almacén de código fuente puede residir separadamente o dentro del almacén de diseño físico. Cuenta con un editor de texto que permite revisar o modificar el código fuente, un generador de reportes para generar la documentación, e inclusive servicios de revisión de errores. De particular interés resultan los reportes de retroalimentación, que comparan el código modificado con el diseño físico original e identifican modificaciones en interfaces, nombres, llamadas y flujo de parámetros. La utilidad de este reporte es asegurar que no se introduzcan errores inadvertidamente cuando se modifica el código en forma manual.

16.6. Alcances y tendencias

Las herramientas de ingeniería inversa de datos y de lógica difieren en la naturaleza del problema que están atacando. Hacer un tratamiento de lógica es una tarea mucho más complicada que hacer el tratamiento de los datos y esto explica las desigualdades entre dichas herramientas.

Las herramientas de datos pueden efectuar ingeniería inversa y progresiva desde el código e inclusive desde el nivel de diseño físico. Es un proceso nítido y acotado, con una serie de ayudas gráficas que facilitan las modificaciones del mantenimiento.

La parte de lógica es más difícil y complicada. El generar código fuente es un resultado parcial y la aplicación de la herramienta para efectuar la ingeniería inversa requiere de la participación y el esfuerzo del factor humano. El responsable de efectuar el mantenimiento tendrá que llenar lagunas que ninguna herramienta puede completar, que involucran información que no se encuentra en el código sino en el ámbito donde se ejecuta la aplicación.

El beneficio de estas herramientas reside en su habilidad para producir documentos que faciliten la comprensión del sistema, más que en su capacidad para rehacerlo.

La tendencia será involucrar nuevas tecnologías, como la inteligencia artificial, que permitan manipular la lógica de un programa en un nivel de detalle similar al que actualmente se tiene en datos. También se dirige hacia la integración con ambientes de desarrollo, de tal manera que existan vínculos más estrechos entre el código fuente y el pseudocódigo utilizado para representarlo. Así mismo, las nuevas técnicas de programación orientada a objetos (OOP) facilitan el modelado, aunque las herramientas que las incorporan no prescinden de la intervención del ser humano.

16.7. Administración de configuración y cambios

Durante el mantenimiento del software, se requiere un plan de administración, de la configuración y herramientas para dicha administración de manera que se pueda seguir la pista y controlar las distintas versiones de los productos de trabajo que constituyen un producto de software.

El rastreo y control de las múltiples versiones de un producto de software son un aspecto significativo para su mantenimiento. Las herramientas de software, para apoyar la administración de la configuración, incluyen las bases de datos para esta administración y los sistemas de biblioteca para el control de las versiones. Una base de datos para el control de la configuración puede proporcionar información concerniente a la estructura del producto, el número de revisión en curso, el estado en curso y la historia de las solicitudes de cambios para cada versión del producto.

Una biblioteca para el control de versiones puede ser parte de una base de datos para la administración de la configuración, o se puede usar como una herramienta independiente. Una base de datos de este tipo proporciona una visión extendida de una familia de productos, mientras que la biblioteca ya mencionada controla los distintos archivos que constituyen las diferentes versiones de un producto de software. Un sistema de biblioteca para el control de versión no es una base de datos. Entre las entidades en una biblioteca para el control de versiones pueden encontrarse el código fuente, el código objeto relocalizable, los comandos para el control de trabajos, los archivos de datos y los documentos de apoyo. Cada entidad en esta biblioteca debe tener una tarjeta de identidad que muestre el número de versión, la fecha, la hora y la identidad del programador. Las operaciones que se realizarán en una biblioteca son producto de la biblioteca, adición y borrado de componentes, preparación de copias de respaldo, edición de archivos, listado de estadísticas de resumen, así como compilación y ensamblado de versiones especificadas del sistema.

Los sistemas de manejo efectivo de la configuración integran herramientas de apoyo como las bases de datos para la administración de la configuración y los sistemas de biblioteca para el control de versiones dentro del marco de trabajo del control de modificaciones.

Actualmente, las herramientas de control de configuración llevan a cabo dos funciones principales: administración de bibliotecas y control de cambios. La administración de bibliotecas crea y opera la información del almacén de software; el control de cambios manipula esa información.

16.8. Administración de bibliotecas

La administración de bibliotecas provee de clasificación, almacenamiento y acceso a los componentes de un programa, de manera que protege la integridad del sistema y facilita el trabajo del desarrollador del sistema, así como el mantenimiento. La administración de bibliotecas tiene cinco dispositivos:

- *Identificación de componentes*. Los nombres físicos del conjunto de datos de los componentes y su tipo se guardan en el directorio interno de la herramienta.
- *Representación lógica*. Después de identificar los componentes físicos de un sistema, la administración de bibliotecas permite al usuario definirlos. Los componentes físicos se clasifican por jerarquía (programas, subsistemas, sistemas, etc.), grupos de trabajo, o tipo de actividad (requisiciones, mejoras o proyectos).
- *Control de revisión*. Esta herramienta maneja la denominación y el almacenamiento de cada componente original y de todas sus revisiones. Típicamente las revisiones son identificadas con números (e.g., V.1.2), cambiando los números izquierdos al hacer revisiones mayores y los derechos al realizar revisiones menores. La mayoría de estas herramientas utilizan un método *delta* para almacenar las revisiones que consiste en comparar la nueva versión del componente con la versión antigua, identificando los cambios, y guardando solamente las modificaciones.
- *Control de versión*. Comúnmente existe la necesidad de agrupar las revisiones específicas de ciertos componentes en una versión. Esto requiere de un nivel de identificación superior al del control de revisión, que permite a los desarrolladores del programa y al personal de mantenimiento trasladar versiones enteras entre las bibliotecas con la confianza de que todo será incluido correctamente.
- *Desarrollo paralelo (ramificación).* Esta herramienta es muy parecida al control de versiones, con la diferencia de que provee medios para denominar una derivación y distinguirla de la revisión principal, además de brindarnos la capacidad de agrupar la derivación en una versión paralela. Algunas herramientas permiten a estas derivaciones reincorporarse a las revisiones principales.

16.9. Control de cambios

Así como la administración de bibliotecas maneja el inventario de los componentes de los programas, el control del cambio opera la modificación del sistema. El control del cambio autoriza, controla y explora las modificaciones a los componentes, y las lleva a la implantación. El control de cambios tiene cuatro dispositivos:

- *Seguridad*. La seguridad controla el acceso de los usuarios a los componentes de software. El administrador del proyecto identifica los componentes envueltos en un desarrollo, o mejora y brinda acceso a los programadores.
- *Registro de entradas y salidas*. Cuando un usuario autorizado registra la salida de un programa, la herramienta de administración de configuración genera una copia de trabajo de la revisión más reciente del programa, registra información referente al cambio (identificación de usuario, fecha y hora), y le pone un semáforo de modificación al programa. Una vez modificado el programa, se registra su entrada y se incorporan las modificaciones.
- *Elaboración de sistemas*. Este dispositivo controla la compilación, el ligado y el traslado de los componentes modificados a las diferentes bibliotecas. Así, no sólo lleva un producto de la fase de desarrollo a la de producción, sino que recupera versiones en forma automática cuando la nueva versión tiene problemas. Esta herramienta utiliza una *lista de elaboración*, una especie de guía del sistema que identifica todos los componentes, sus interdependencias y la secuencia de acciones necesarias para construir el sistema.
- *Reportes*. El generador de reportes trabaja sobre el archivo de actividades y otra información de la administración de configuración. Los usuarios pueden generar reportes predeterminados, reportes de SQL y consultas en línea. Algunos reportes típicos son: reportes de actividad, de versión, de interdependencia, de análisis del impacto del cambio, de cambio en la actividad, de cambio en la historia, etc.

16.10. Beneficios

En principio, el uso de estas herramientas hace más sistemática la labor de mantenimiento. Las herramientas de control de cambios no sólo manejan la logística del mantenimiento y cambio del software, sino que también refuerzan un proceso completo y definido. Su principal aportación es hacer del desarrollo y del mantenimiento del software actividades más baratas y menos propensas a errores.

Las funciones del manejo de la configuración son una parte integral del entorno de las herramientas CASE. En este entorno, la noción del manejo de la configuración debe incluir no sólo el manejo en el nivel físico de los componentes del software, sino también el manejo del nivel lógico de los componentes. La administración de la configuración se necesita durante el proceso de desarrollo y el de mantenimiento para operar cambios a través del ciclo de vida entero. Desafortunadamente, aunque estén disponibles herramientas poderosas de configuración y de control de cambios, no se han integrado al medio ambiente de las herramientas CASE. Esta situación deberá cambiar en la presente década, es seguro que las funciones de configuración y cambio se van a volver parte de los servicios del almacén.

17. Tabla de productos clasificados sobre herramientas de reingeniería

Como se mencionó en el punto 8, a continuación se ofrece una tabla-listado de 57 productos clasificados sobre herramientas de reingeniería conforme a la siguiente estructura y listados en orden alfabético.

- **Analizadores de programa**
  1. Rastreadores de lógica y de datos
  2. Referencias cruzadas
  3. Delineadores (*Profilers*)
- **Métricas**
  1. Programas monitores de estándares
  2. Programas analizadores de calidad
  3. Programas controladores de complejidad
- **Reestructuración**
  1. Reestructuradores de procesos lógicos
  2. Estandarización de nombres de datos y definiciones

- **Ingeniería inversa**
  1. Ingeniería inversa de datos
  2. Ingeniería inversa de lógica
- **Pruebas**
  1. Generadores de datos para pruebas
  2. Analizadores de alcance para pruebas
  3. *Debuggers*
  4. Comparadores
- **Convertidores**
  1. De lenguaje
- **Manejadores de configuración y cambios**
  1. Manejadores de control de cambio
  2. Manejadores de librerías
  3. Generadores de código
- **Herramientas de redocumentación**
  1. Referencias cruzadas
  2. Generadores de diagramas [18]

[18] Reingeniería de Programación: Una propuesta para la solución de los problemas de mantenimiento de software. Tesis de Ingeniero en Computación.
Autores: Héctor Francisco Bautista González, Cuauthemoc Freyre Mercado y Norma Susana Zavala Carrasco. FI-UNAM 1995.

Productos listados en orden alfabético	Analizadores de programas	Métricas	Reestructuración	Ingeniería Inversa	Pruebas	Convertidores	Manejo de configuración y cambios	Herramientas de redocumentación
Aide-De-Camp Software Configuration Management System (V.9.0)							1,2	1
AISLE/QualGen	1,2	1,2,3						1,2
AIX Configuration Management and Version Control (CMVC) (V.2.2)							1,2	
Answer:Comparex/CDF-15,20					4			1
C Set++ Debugger	1				3			
C-DOC Documentation Tools for C and C++ (V.5.0)								1,2
C-DOC Professional (V.5.0)								1,2
C-Metric		1,2,3						
CA-Panl.CM Configuration Manager							1,2	
CCC/Manager (Change and Configuration Control) (V.2.1)							1,2	
CISLE/CDADL	1,2	1,2						
COBOL ANALYST (V.3.0)	1,2							1,2
COBOL Structuring Facility (V.2.0)	1,2	1,2	1,2					
CodeBreaker (V.4.1)				1,2	4			
Codecheck/2 (V.5.0)	1,2	1,2,3						
Coverage	1,2							
CrossView Debugger for Windows	1				3			
DCD-PC	1,2			1,2				1,2
ETi Source Compare					4			1
For Struct (V.2.0)			1,2					
Language Translators						1		
Legacy Workbench (V.5.1)	1,2	1,2,3	1,2	1,2				1,2
Legacy Workbench for Windows (V.5.1)	1,2	1,2,3	1,2	1,2				1,2
LISP to C Translator (Rel.3.2)						1		
Logiscope	1,2	1,2,3	1,2	1,2				
MASM to C Translator						1		
McCabe Slice Tool (V.4.1)	1,2				2			
Microsoft Source Profiler (V.1.0)	2,3				4			
Microsoft Source Profiler for Windows (V.1.0)	2,3				4			
MultiScope Debuggers for Windows (V.2.01)	1				3			
Non-IBM COBOL to IBM COBOL/COBOL II Translator						1		
PC/MC/UX/VX-Metric (V.4.0)	1,2	1,2,3						1
PL/1 to C Translator						1		

Tabla 3. Productos clasificados sobre herramientas de reingeniería

256

Productos listados en orden alfabético	Analizadores de programas	Métricas	Reestructuración	Ingeniería Inversa	Pruebas	Convertidores	Manejo de configuración y cambios	Herramientas de redocumentación
PL/1 to COBOL/COBOL II Translator						1		
PL/M to C Translator						1		
Pro Struct			1,2					
Product Configuration Management System (V.4.0.5)							1,2	
Profile Code (V.3.20)	3							
PVCS Configuration Builder (V.5.1)							1,2	1
PVCS for Software Configuration Management							1,2	
PVCS Version Manager (V.5.1)							1,2	
Q/Auditor COBOL (V.2.4)	1,2	1,2,3						
Q/Auditor COBOL for Windows (V.2.4)	1,2	1,2,3						
QA FORTRAN (V.6.0)		1,2,3	1,2		1,4			
Software Configuration Management Supervisor/2 (SCMS/2)							1,2	1
Software Profile Management Facility	3							
Software Refinery (V.4.0)	1,2			1,2	1,2,4		3	1,2
Source Program Compare					4			1
STW/Advisor	1,2	1,2,3						
STW/Advisor for Windows	1,2	1,2,3						
Turbo Debugger and Tools (V.3.0)	1,2,3				3			
VIA/Insight2	1,2			1,2				
VIA/Recap	1,2	1,2,3						
Visual Debugger-14	1							
Xinotech COBOL Composer (V.2.0)	1,2			1,2				
XperCASE (V.3.5)			1,2	1,2				1,2
XPONENT Configuration Management (XCM) for Windows							1,2	1

Tabla 3. (continuación)

257