UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

REINGENIERIA DE PROGRAMACION: UNA PROPUESTA PARA LA SOLUCION DE LOS PROBLEMAS DE MANTENIMIENTO DE SOFTWARE

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

PRESENTAN:

HECTOR FRANCISCO BAUTISTA GONZALEZ
CUAUHTEMOC FREYRE MERCADO
NORMA SUSANA ZAVALA CARRASCO

DIRECTOR: ING. ELOISA DAVALOS PAZ

FACULTAD DE INGENIERIA
U N A M

México, D. F. 1995

FALLA DE ORIGEN

A mi madre, María Elva, y a mi hermano, Sergio Arturo, por su cariño y su apoyo.

A la ingeniera Eloísa Dávalos por su asesoría profesional.

A mis profesores y amigos.

Gracias.

Héctor.

A mis padres que incodicionalmente me han dado su apoyo, cariño y comprensión.

A mis hermanos, por su cariño y alegría.

A mi novio, por el impulso que me dió para la realización de la presente tesis.

A la ingeniera Eloísa Dávalos por su asesoría profesional.

A mis profesores y amigos.

Gracias.

Norma.

A mi hermana María Elsa Freyre y mi cuñado Enrique Esquer, por su apoyo y afecto.

A mis sobrinos, Paulino y Rodrigo Esquer, por su cariño.

A mi novia Paola García, por su asistencia y apoyo.

A la ingeniera Eloísa Dávalos, por su asesoría profesional.

A mis profesores y amigos.

Gracias.

Cuauhtémoc

CONTENIDO

INTRODUCCIÓN IV

PROBLEMAS ACTUALES VI
AUTOMATIZACIÓN DEL MANTENIMIENTO DE SOFTWARE VII

MANTENIMIENTO DEL SOFTWARE 1

ACTIVIDAD DE MANTENIMIENTO 1
DEFINICIÓN 2
Definiciones de mantenimiento 2
CAUSAS DEL MANTENIMIENTO DEL SOFTWARE 4
LA PERSPECTIVA DE CAMBIO 5
NATURALEZA DEL TRABAJO DE MANTENIMIENTO 6
FACTORES QUE AFECTAN EL ESFUERZO DE MANTENIMIENTO 8
SOFTWARE FRÁGIL 10
MEJORAMIENTO DEL MANTENIMIENTO DURANTE EL DESARROLLO 12

REINGENIERÍA 17

DEFINICIÓN 18
TIPOS GENERALES DE REINGENIERÍA 19
REQUISITOS 20
TIPOS DE HERRAMIENTAS DE REINGENIERÍA 21
EL PUENTE A NUEVAS TECNOLOGÍAS 22
MANEJO DE SISTEMAS EXISTENTES 23
Sistemas frágiles 23
Reingeniería contra reemplazo 24

Reconocimiento de los sistemas frágiles 25

ANALIZADORES DE CÓDIGO Y HERRAMIENTAS DE MEDICIÓN 27

QUÉ TAN ENTENDIBLE ES EL SOFTWARE 28
Factores de entendimiento 29
ANALIZADORES DE CÓDIGO DE LOS PROGRAMAS 30
GRÁFICAS DE FLUJO 30
GRÁFICA DE FLUJO DE RIGUROSIDAD MATEMÁTICA 30
Asistentes de prueba 31
DELINEADORES (PROFILERS) 32
MÉTRICAS DE COMPLEJIDAD 33
Métricas de tamaño y de flujo 34
COMPLEJIDAD CICLOMÁTICA DE MCCABE 34
Extensión de la complejidad ciclomática 36
Complejidad esencial 36
Número óptimo de la complejidad de McCabe 36
CUENTA DE NODOS (KNOT COUNT) 37
CIENCIA HALSTEAD DEL SOFTWARE 37
Tamaño 39
Volumen 39
Esfuerzo 40
CONTAR Y LISTAR 41
REGLAS DE MANTENIMIENTO DE SOFTWARE 41

REESTRUCTURACIÓN 43

ESTRUCTURA Y ESTILO 44
REESTRUCTURACIÓN DE LA LÓGICA DEL PROGRAMA 46
MODULARIDAD 46
Abstracción 46
Ocultamiento de la información 47
Tipos de módulos 48
Independencia de módulos 49
Cohesión 49
Acoplamiento 51
PROGRAMAS BIEN ESTRUCTURADOS 54
Definición 54
CUANDO REESTRUCTURAR 56
PROCESO DE REESTRUCTURACIÓN DE LA LÓGICA DEL PROGRAMA 57
BENEFICIOS Y EFECTOS LATERALES 58
MÉTODOS DE REESTRUCTURACIÓN 58
HERRAMIENTAS DE REESTRUCTURACIÓN LÓGICA DEL PROGRAMA 60
ANÁLISIS Y DOCUMENTACIÓN 61
Medidas y conteos 61
Análisis lógico — Rastreo y Problemas 61
Cartas jerárquicas 61
Apoyo de Modularización 61
Técnicas de Reingeniería Conocidas 62
REESTRUCTURACIÓN DE DATOS 62

Definición 62
Candidatos para la reestructuración de datos 63
Nombres 64
Herramientas de reestructuración de datos 65
ANÁLISIS Y ACTUALIZACIÓN DE LAS VARIABLES Y LOS TIPOS A LO LARGO DEL PROGRAMA 66
RASTREO Y ACTUALIZACIÓN DE UN *ELEMENTO DE DATO* A LO LARGO DEL PROGRAMA 66
ACTUALIZACIÓN A UN DICCIONARIO DE DATOS 66
REPORTES DE ANÁLISIS 67

INGENIERÍA INVERSA 68

DEFINICIÓN 68
EL CÓDIGO FUENTE 68
INGENIERÍA INVERSA E INGENIERÍA PROGRESIVA 69
INGENIERÍA INVERSA DE DATOS 69
INGENIERÍA INVERSA DE LÓGICA 72
Almacén de diseño físico 73
Almacén de diseño lógico 75
Almacén de código fuente 76
ALCANCES Y TENDENCIAS 76
BENEFICIOS 77
LA CUESTIÓN LEGAL 78

ADMINISTRACIÓN DE CONFIGURACIÓN Y CAMBIOS 80

ADMINISTRACIÓN DE BIBLIOTECAS 82
Identificación de componentes 83
Representación lógica 83
Control de revisión 84
Control de versión 84
Desarrollo paralelo (ramificación) 84
CONTROL DE CAMBIOS 85
Seguridad 85
Registro de entradas y salidas 85
Elaboración de sistemas 86
Reportes 87
BENEFICIOS 88

CONCLUSIONES 90

APÉNDICE 94

BIBLIOGRAFÍA 136

INTRODUCCIÓN

En la actualidad muchos países estamos inmersos en un proceso de modernización que busca reducir la distancia que nos separa de las naciones más prósperas. Parte de esta estrategia de modernización consiste en abrir las economías, de tal forma que otros países inviertan capitales en industrias y en empresas, provocando un florecimiento del comercio internacional, un incremento en la producción, un aumento de las fuentes de trabajo, una transferencia de tecnologías, pero también significa una disminución en las oportunidades de aquellos que no incorporen métodos de trabajo más eficientes.

Entre los factores que influyen en la distancia que nos separa de esos países, podemos mencionar el grado de industrialización, el apoyo que se brinda a la investigación y los avances tecnológicos. Es muy claro que tienen una preeminencia en el plano económico porque sus empresas aprovechan estos factores con miras a incrementar la productividad, abatir costos y tiempos, crear mercados cautivos que dependen de sus productos. Comercializar productos manufacturados redunda en mayores beneficios de los que se obtienen cuando se opta por el comercio de materias primas.

Desde esta perspectiva, el presente año envuelve un profundo cambio en las políticas que adoptamos en lo que respecta a la industria, porque nos encontramos en un proceso de integración con una de las economías más grandes del planeta, la de Norteamérica, proceso que involucra la

competencia de empresas nacionales con extranjeras sin el cobijo del proteccionismo. Algunos estadounidenses piensan en México como un país con atrasos profundos en tecnología de servicios, con problemas con la lengua inglesa y con profesionistas dedicados pero poco confiables (BE94). La nueva regla del juego es enfrentarse a las empresas extranjeras en una lucha de igual a igual, donde sobrevivirán aquellos que se adapten a las características de un mercado que tendrá acceso a las mejores alternativas de esta parte del continente. Una variedad de empresas extranjeras buscará establecerse en nuestro territorio para aprovechar un mercado ávido de productos de calidad, a precios competitivos, con mano de obra barata. Ofrecerán un desafío a los establecimientos actuales en todas las áreas que podemos mencionar, desde el sector de la industria, la educación, el comercio, hasta la esfera de las telecomunicaciones y la banca. Los norteamericanos esperan beneficios del tratado en materia de tarifas, esperando reducir los impuestos a productos de computación de 10% ó 20% a 0% o 3.9%, lo cual redundará en precios más ventajosos. También esperan una regulación más estricta en derechos de propiedad intelectual, preocupación recurrente dentro de una industria que invierte miles de millones de dólares en investigación y desarrollo. El TLC terminará con las restricciones en inversiones de capital extranjero, permitiéndoles establecerse en las condiciones que ellos prefieran. Igualmente tienen los ojos puestos en el área de servicios de telecomunicaciones y procesamiento de datos (HA93).

Lo anterior tiene implicaciones para los profesionistas nacionales, en particular para los egresados de las carreras relacionadas con computación, ya que los recursos tecnológicos para procesamiento de datos que habrán de introducir los inversionistas extranjeros serán los más modernos y complejos de la actualidad, además que ofrecerán una infraestructura bastante más sólida que la nacional. Por otra parte, ya sea que se pretenda competir contra los extranjeros o asociarse con ellos, se requerirá tener un nivel de calidad comparable.

En particular, para los profesionistas de nuestra carrera, podemos destacar los productos de software. Se puede decir que un producto de software, en contraposición con el software *casero*, es un programa que tiene una gran cantidad de usuarios y muchas veces tiene muchos programadores y personas de mantenimiento. En la mayor parte de los casos, son personas distintas quienes desarrollan los productos, quienes los usan y quienes les dan mantenimiento. El desarrollo y el mantenimiento de productos de programación requiere un enfoque más sistemático que el necesario en el desarrollo de programas caseros.

La fuente principal de software para la mayoría de los usuarios, sobre todo en las fases iniciales del establecimiento de un sistema basado en

computadoras, es la paquetería de uso común como lo son Word, Excel, DBase, etc.. Sin embargo, tarde o temprano se decidirá que no existe un paquete comercial que resuelva las necesidades particulares de cualquier empresa. Es entonces cuando debe enfrentarse a la decisión de desarrollar su propio software o solventar el problema de otra manera.
El desarrollar software consiste en algo más que escribir líneas de código en Clipper, Excel, C, Pascal, o cualquier otro lenguaje o paquete. El software bien desarrollado se plantea, justamente como debería plantearse cualquier otro proyecto de ingeniería correctamente diseñado. La planeación se supone tiene lugar antes de que el software se escriba, además el software debe estar adecuadamente documentado y esmeradamente probado.

Problemas actuales

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 del mismo. Estas deficiencias subsisten y durante las etapas de desarrollo y mantenimiento se manifiestan de la siguiente manera:

- Frecuentemente, los proyectos de software se emprenden con sólo una vaga indicación de los requerimientos del usuario, lo cual conduce a la insatisfacción del mismo con el producto terminado. Lo anterior es una indicación de que debe mejorarse la comunicación entre el usuario y los constructores del producto.
- Los presupuestos y los programas para el desarrollo de software son a menudo demasiado imprecisos, por lo que los sobrecostos y las entregas tardías del producto son frecuentes.

- El software no se desarrolla en forma adecuada a pesar de existir métodos probados para su planeación, especificación, diseño, codificación y prueba.
- El software no se documenta en forma adecuada. Un producto de programación debe contar con la documentación en todas sus fases la cual debería escribirse durante el proceso de desarrollo y no después. Estos documentos ayudan al administrador en el proceso de control y permiten evaluar el progreso del proyecto.
- La calidad del software es pobre. Existen técnicas probadas para asegurar la calidad del producto, que no se aplican en forma consistente.
- Los programas existentes son difíciles de mantener. Actualmente el mantenimiento del software consume la mayoría de los recursos que se destinan a este rubro. Por lo que se debe hacer énfasis en la facilidad de mantenimiento como criterio importante para la aceptación de los productos de programación.

En lo que respecta a los aspectos sociales, observamos que:

- La mayoría de los jefes de los centros de cómputo se comprometen a entregar resultados en tiempo récord, sin tomar en cuenta a quienes realmente hacen el trabajo.
- Si no son los jefes quienes se comprometen, por lo general tienen que aceptar los tiempos marcados por los directivos, pues aquellos son impuestos sin considerar el grado de dificultad de los trabajos ni los recursos con los que se cuenta.
- Los técnicos aprenden sobre la marcha, de una manera por completo empírica y sin planes de capacitación previamente definidos. Muchos de los técnicos se inscriben por iniciativa propia en escuelas técnicas, costeando sus propios estudios.
- Todo el tiempo se encuentran saturados de trabajo por falta de planeación efectiva, razón por la cual difícilmente realizan labor de investigación, necesaria para la mejor explotación de los recursos de cómputo.
- Con frecuencia, los usuarios se sienten insatisfechos por un nivel de servicio que no es acorde con sus necesidades de información, y en ocasiones están en completa discordia con el centro de cómputo
- Existe gran escasez de personal con experiencia, y la mayoría de las empresas no pueden contratar el personal suficiente.
- No hay metodologías de trabajo, y donde llegan a existir, rara vez son respetadas.

Automatización del mantenimiento de software

¿Por que a través de los años las nuevas tecnologías de software han tenido tan poco impacto en la crisis que está sufriendo éste? y ¿porqué existe incapacidad para desarrollar sistemas de software de alta calidad,

con la suficiente rapidez y el bajo costo para satisfacer una demanda que crece 12% al año? Desde que se reconoció la crisis en el software a mediados de los años 60's, el consenso general señala que el desarrollo de software es una labor intensiva y que hay una carestía de profesionales expertos en el área. Por esto, la mayoría de las soluciones que se han propuesto enfatizan el mejoramiento de las metodologías y herramientas de desarrollo, enfocándose en estas fases del ciclo de vida del software.

Durante los últimos treinta años se crearon herramientas poderosas como ingeniería de programación, metodologías estructuradas, herramientas de cuarta generación y más recientemente ingeniería de programación auxiliada por computadora (CASE). Desafortunadamente, estas tecnologías son grandes soluciones para el problema equivocado. Una mirada más cercana a la crisis del software revela que la etapa de desarrollo solamente representa la punta del iceberg. El mantenimiento del producto domina el ciclo de vida de la programación, y es ahí donde se gastan la mayor parte de los recursos.

Una forma de detectar las causas de la crisis del software es observando los fracasos pasados y las tendencias de los costos actuales en el desarrollo de software. Sabemos que es difícil desarrollar programas de alta calidad. Además, la mayoría de los proyectos de programación de cualquier magnitud vienen acompañados por problemas y fallas, por lo que usualmente toman más tiempo para llevarse a cabo y cuestan más dinero de lo previsto. Cuando buscamos las razones encontramos algunos hechos importantes que debemos reconocer.

Los usuarios están insatisfechos con los sistemas de software, no por defectos en ellos, sino por falta de documentación, interfaces no amigables, y programas endebles que fallan cuando se introducen cambios. Además, los proyectos de software fracasan no por problemas técnicos sino por fallas en la dirección y en el control. Los costos van en aumento por la dificultad que envuelve mantener el creciente número de sistemas existentes, que deben de ser cambiados con frecuencia para atender las demandas de los negocios.

Hay varias posibilidades a considerar cuando tratamos de mejorar las actividades de mantenimiento del software:

1. Colocar más personas en el desarrollo de sistemas y en el equipo de mantenimiento para reducir los retrasos.
2. Atacar de manera indirecta al mantenimiento del software, enfocándonos en desarrollar tecnologías innovadoras para crear sistemas más sencillos de mantener, que gradualmente reemplacen a los existentes.
3. Adoptar un plan más agresivo para reescribir los sistemas existentes tan pronto como sea posible.

4. Hacer un ataque directo al mantenimiento del software usando herramientas automáticas para mejorar el mantenimiento y la tecnología de los sistemas existentes, y la eficiencia en el proceso de 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 empleaba en mantenimiento del software o en actividades relacionadas con el mantenimiento (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) (Figura 1). 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). 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.

Figura 1. El mantenimiento consume tres veces más tiempo que el desarrollo (MA94)

A pesar de la introducción de nuevas tecnologías de desarrollo a principios de los 90's, la mayoría de los esfuerzos sigue concentrados en el mantenimiento del software. Confiar en que las estrategias de desarrollo

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 ésto.

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 el objeto de adaptarse a las nuevas tecnologías y a las necesidades del usuario. Aún el sistema que es totalmente confiable, que cubre totalmente los requerimientos 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.

Tampoco se ha resuelto la pregunta de qué lenguaje se debe de utilizar para reescribir los sistemas. La mayoría de las herramientas de CASE generan aplicaciones en COBOL. Pero con estos nuevos sistemas en COBOL, ¿realmente será más fácil y barato el mantenimiento que con sus predecesores? Porque muchos de estos sistemas, si bien no están perfectamente estructurados, trabajan satisfactoriamente y no son enteramente inaceptables para sus usuarios, no es un hecho que deben de ser reemplazados, especialmente si el reemplazo está escrito en COBOL.

Muchas compañías ven a sus sistemas existentes como valores que deben preservar, y no como algo que se debe arrojar al cesto de la basura. Quieren extender la vida de estos sistemas mejorando su estructura y su documentación, y actualizándolos para aprovechar las nuevas tecnologías.

Otro factor que obstaculiza el cambio a corto plazo de los sistemas es el tiempo que tarda introducir y adoptar las nuevas tecnologías. Frecuentemente las nuevas ideas se encuentran con una resistencia, ya que es un factor humano oponerse al cambio. Se necesita más tiempo del esperado para que los avances tecnológicos penetren completamente en la comunidad. Desde que se concibe una tecnología hasta que se adopta ampliamente en el mercado, transcurren entre quince y veinte años. Tenemos un ejemplo en la programación estructurada y los diccionarios de datos, que empezaron a introducirse a principios de los 70's, y no

estaban muy difundidos una década después. Nos encontramos a mediados de los 90's y solamente una pequeña cantidad de los programadores que laboran en México usan metodologías estructuradas en sus labores, como las herramientas CASE, que han sido más complicadas en la práctica de lo que se esperaba porque no han encontrado un ambiente adecuado para su uso. Llegaremos al año 2000 antes de que CASE se convierta en una tecnología madura, utilizada en la mayoría de las empresas.
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.
Al adoptar estas herramientas durante nuestras labores como programadores, podemos esperar un aumento de productividad y una reducción en nuestros costos. Además, podemos mejorar nuestras tareas de mantenimiento utilizando herramientas que separen, por ejemplo, porciones de programas que pueden mejorarse y conservarse, de aquellas que deben reemplazarse.

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 a las características del mismo y al criterio del grupo de trabajo que esté encargado de realizar la tarea de mantenimiento. Por esta razón el presente trabajo no pretende sino describir las diferentes metodologías que componen la reingeniería.

CAPÍTULO 1

MANTENIMIENTO DEL SOFTWARE

Actividad de mantenimiento

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[1]. Por lo contrario, el mantenimiento del software generalmente presenta más retos que el desarrollo. ¿Porqué? ¿Qué es exactamente lo que hacen las personas de mantenimiento?

En el proceso de mantenimiento se llevan a cabo una variedad de análisis y pruebas de programación, como son las siguientes:

- Estudiar especificaciones y diseños del sistema
- Entrevistar a los usuarios
- Examinar programas y documentación asociada
- Rastrear las fuentes de los errores y deficiencias del programa
- Diseñar el cambio del programa
- Modificar el programa
- Revalidar el programa
- Poner al día la documentación del programa

---

[1] Por ejemplo, Microsoft señala "*...por regla general los programadores experimentados escriben código nuevo y los nóveles mantienen el código.*" Steve Maguire, *Código sin errores*, McGraw Hill, 1993, p. 146

Existen tres procedimientos básicos para cambiar el software:

1. Entender los cambios a realizarse
2. Modificar para incorporar los cambios
3. Revalidar los cambios realizados

Para cambiar un sistema se necesita entender el propósito, la estructura interna, y los requerimientos de operación de los programas asociados. Si el encargado de realizar el mantenimiento no entiende realmente el programa, se corre el riesgo de introducir errores. Una serie de cambios hechos al azar en el programa pueden rápidamente destruir su estructura, provocando un desastre de mantenimiento.
Modificar un programa consiste en crear o cambiar estructuras de datos y/o procesos lógicos de programas ya existentes actualizando la documentación. El encargado del mantenimiento se debe fijar que los cambios no afecten la integridad del programa o lo vuelvan más complejo. Igualmente, se debe de estar pendiente de posibles efectos laterales que se deriven del cambio.
En el momento en que se modifique el software, las correcciones deben de ser revalidadas. Se tendrán que hacer pruebas constantes para estar seguros de que los cambios se realizaron de manera correcta.

Definición

Cualquier ataque celebrado con éxito a los problemas del mantenimiento del software, presupone comprender qué es el mantenimiento. De cualquier manera, en la mayoría de las organizaciones se pone poca atención y se desconoce acerca del mantenimiento. ¿Cuál es la fuente de casi todo trabajo de mantenimiento? ¿Cuáles son las dificultades del trabajo de mantenimiento del software? ¿Qué hace a un sistema de software difícil o fácil de mantener?
Primero, hay que entender qué actividades y dificultades se encuentran envueltas en el mantenimiento e identificar qué factores intervienen para el alto costo. Con este conocimiento se pueden seleccionar técnicas y herramientas más efectivas para reducir el trabajo de mantenimiento.

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:

1. Corregir los errores
2. Revisar los requerimientos originales
3. Mejorar el funcionamiento, aumentar su desempeño e introducir funciones nuevas

Las actividades de mantenimiento implican corregir problemas, mejorar los productos de software, y adaptarlos a nuevos ambientes. La corrección de problemas implica la modificación y revalidación del software para corregir los errores, algunos de los cuales requieren atención inmediata, otros se pueden corregir con base en un calendario periódico, y los menos se conocen, pero nunca se corrigen. La mejora de los productos de software puede dar como resultado proporcionar nuevas capacidades funcionales, mejorar los despliegues al usuario y los modos de interacción, revalorar los documentos externos y la documentación interna, o revalorar las características del desempeño de un sistema. La adaptación de un producto de software a un nuevo ambiente puede provocar el traslado del software a una máquina distinta, o por ejemplo, modificar el software para instalar un nuevo protocolo de comunicaciones o adaptarlo de trabajar en un ambiente centralizado a uno distribuido.

Definiciones estrictas de mantenimiento incluyen solamente la primer operación, corrección de errores.

Cambios a un sistema de software ya existente que necesiten de modificaciones de los requerimientos originales no se incluyen como

mantenimiento, pero sin embargo, se clasifican como proyectos de desarrollo o redesarrollo.
Algunas organizaciones clasifican actividades como el mantenimiento o el desarrollo con base en qué personas realizan la actividad. Si la función se elabora por el grupo de desarrollo, es una actividad de desarrollo; y si se realiza por el grupo de mantenimiento, es una actividad de mantenimiento.
Otras organizaciones clasifican una tarea como de mantenimiento o de desarrollo con base en el tamaño o la complejidad de la tarea. Por ejemplo, si la tarea necesita más de tres meses para llevarse a cabo, se considera inmediatamente como una tarea de desarrollo y se asigna a el grupo de desarrollo.
Se deben asignar todas las definiciones de mantenimiento, considerando las tres operaciones antes mencionadas como partes fundamentales. No se debe describir una actividad como de mantenimiento o de desarrollo fundamentados en quién hace la actividad, en qué tipo de operación se lleva a cabo, o en la magnitud del esfuerzo. Lo que se debe de considerar como mantenimiento de software está definido por el punto en el que se tome la actividad en el ciclo de vida del software y por el uso de las especificaciones y documentos asociados al sistema.

> **El mantenimiento del software es el proceso de mantener operando un programa (o varios programas) o de perfeccionarlo.**

Esta definición sugiere que el mantenimiento envuelve una variedad de actividades, algunas similares a los nuevos proyectos de desarrollo y otras específicas en la fase de mantenimiento.

Causas del mantenimiento del software

Una manera de entender las bases para proporcionar mantenimiento a un sistema, es en términos de las causas de este mantenimiento. Swanson divide las causas en tres categorías básicas:

1. Fallas
2. Cambios exteriores
3. Solicitudes por parte del personal de mantenimiento o de los usuarios

Fallas. Se atribuyen a errores en el software. Terminación anormal de un programa, resultados inválidos, ineficiencias en el desarrollo y violaciones de reglas de programación.
Cambios exteriores. Comúnmente ocurren en sistemas de software de organizaciones que los han utilizado por un largo periodo de tiempo. Existen dos tipos de cambios: en los datos, como son los cambios en la

estructura de la base de datos; y en el procesamiento, como son la instalación de un nuevo dispositivo de hardware, de un sistema operativo o el transferir un programa a una nueva plataforma.

Usuarios y personas del mantenimiento del software son también causas del mantenimiento. Se pueden pedir cambios a el sistema de software para incrementar su eficiencia de operación, para darle una nueva estructura a la salida de los datos, para agregar nuevas cualidades o características, para cambiar funciones existentes, o para aumentar su aptitud para recibir modificaciones.

Las actividades en respuesta a estas causas básicas, las define Swason como:

1. Mantenimiento correctivo.- Realizado para identificar y corregir:
   - Fallas de procesamiento: Terminación anormal del programa, pérdida de datos de entrada, salida incorrecta del programa.
   - Fallas de rendimiento: Tiempo de respuesta lento o procesamiento inadecuado.
   - Fallas de implementación: Estándares, violaciones e inconsistencias en el diseño.
2. Mantenimiento por adaptación. Realizado para adaptar cambios de software en el desarrollo del programa
   - Cambios del medio ambiente de los datos: Cambios en la estructura de la base de datos.
   - Cambiar el procesamiento de medio ambiente: Traspasar el sistema a una nueva plataforma de hardware o a un nuevo sistema operativo.
3. Mantenimiento de mejoramiento.
   - Mejorar el rendimiento
   - Cambiar o añadir nuevas características
   - Mejorar el futuro mantenimiento del programa

La perspectiva de cambio

Tal vez una mejor definición de lo que mantenimiento de software significa es la siguiente:

> **Mantenimiento es el proceso de cambio de los sistemas de software.**

El mantenimiento del software incluye todos los tipos de cambios que se hagan al sistema existente, incluyendo correcciones, aumento de atributos, y ampliaciones. Desde esta perspectiva, se puede notar que para mejorar las funciones de mantenimiento del software se tienen que encontrar herramientas, métodos, y técnicas de manejo para apoyar el cambio. Y para obtener sistemas que sean fáciles de mantener, se deben

desarrollar programas que sean fáciles de cambiar. Saber operar los cambios es una clave para mejorar el mantenimiento del software.

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 80s 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). Como se muestra en la Figura 2, 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.

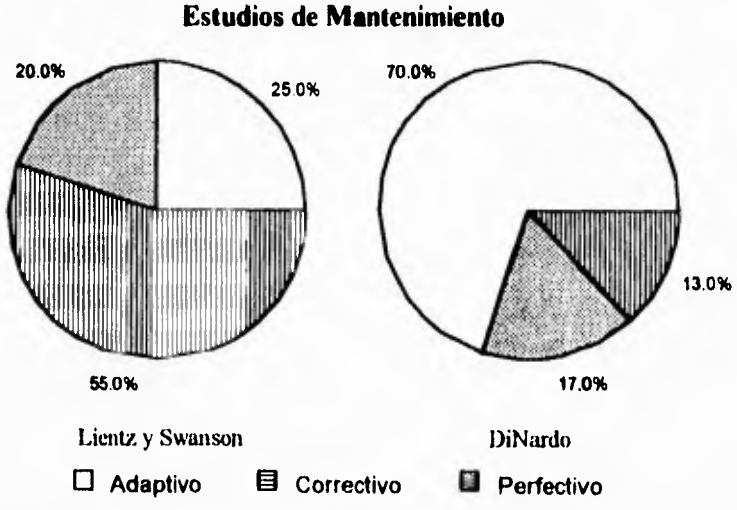

Figura 2. El mayor esfuerzo de mantenimiento es el desarrollar adaptaciones y mejoras (MC92)

Los mayores requerimientos de mantenimiento son *las demandas de los usuarios para el mejoramiento del sistema*, como reportó Lientz y Swanson en un estudio en 1980 y en otro realizado en 1989 por Swanson y Beath (ver Figura 3).

Los aspectos anteriores muestran que el objetivo principal del desarrollo del software debe ser la producción de sistemas que faciliten su propio mantenimiento. El mantenimiento , como todos los atributos de calidad de alto nivel, se puede expresar en términos de atributos construidos dentro del producto. Los atributos primarios del producto que contribuyen al mantenimiento son la claridad, la modularidad, y la buena documentación interna del código fuente, además de documentos de apoyo apropiados.
También se debe observar que el mantenimiento del software es un microcosmos del ciclo de desarrollo del software. El mejoramiento y la adaptación del software reinician el desarrollo en la fase de análisis, mientras que la corrección de un problema de software puede reiniciar el ciclo de desarrollo en la fase de análisis, en la fase de diseño, o en la de implantación. Por lo tanto, todas las herramientas y técnicas utilizadas para desarrollar el software son potencialmente útiles para el mantenimiento del software.

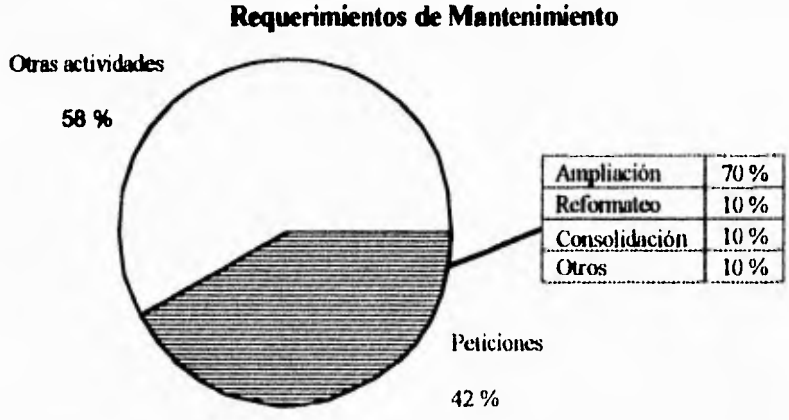

Figura 3. Los mayores requerimientos de mantenimiento son *las demandas de los usuarios para el mejoramiento del sistema* (MC92)

Las actividades de análisis durante el mantenimiento del software implican la comprensión del alcance y efecto de una modificación deseada, además de las restricciones para hacer la modificación. El diseño durante el mantenimiento supone rediseñar el producto para incorporar los cambios deseados. Entonces éstos deben implantarse, la documentación interna del código se debe actualizar, y se deben proyectar nuevos casos de prueba para evaluar la adecuación de la modificación. También, los documentos de apoyo (especificaciones de requisitos y diseño, plan de prueba, principios de operación, manual del usuario, directorios de referencias cruzadas, etc.) se deben poner al día para ilustrar los cambios. Las versiones modernizadas del software

(código y documentos de apoyo) deben distribuirse en las distintas instalaciones de los clientes, y deben actualizarse los registros de control de la configuración para cada sitio.
Todas estas tareas se deben efectuar mediante un enfoque sistemático ordenado que rastree y analice los requisitos de las modificaciones, y con un cuidadoso rediseño, reimplantacion, revalidación y redocumentacion de los cambios. De otro modo, el producto de software se degradará con rapidez como resultado del proceso de mantenimiento. No es raro que la versión inicial de un producto de software bien diseñado, implantado de manera apropiada, y adecuadamente documentado, se convierta en un programa difícil de modificar debido a procedimientos inconvenientes de mantenimiento. En estas situaciones resulta más fácil y menos costoso reimplantar un módulo o un subsistema que modificar la versión existente. Las actividades de mantenimiento del software no deben destruir su facilidad de mantenimiento. Un pequeño cambio en el código fuente requiere, a menudo, modificaciones extensas en el conjunto de pruebas y en los documentos de apoyo. No reconocer el costo verdadero de un *cambio pequeño* en el código fuente es uno de los problemas más significativos en el mantenimiento del software.

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.
Asi 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 va habiendo cambios en el personal.
En la Figura 4, el estudio de IBM realizado por Fjelstad y Hamlen enseña las primeras pruebas que realizan los encargados del mantenimiento.

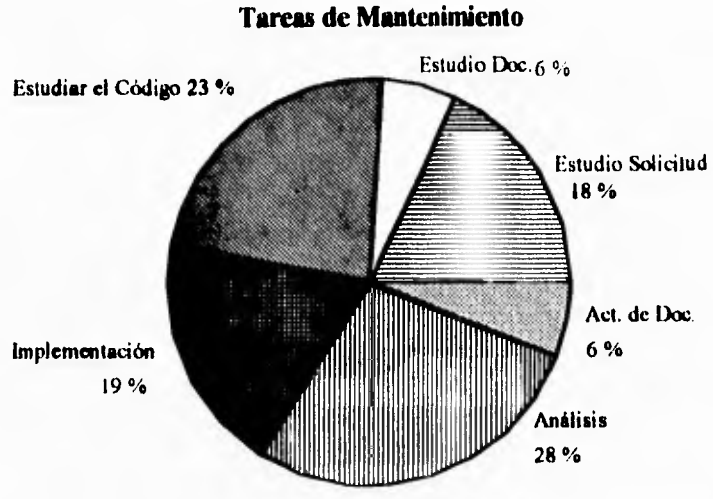

Figura 4. El estudio de IBM realizado por Fjelstad y Hamlen muestra las primeras pruebas que realizan los encargados del mantenimiento (MC92)

La Figura 5 enseña la misma información en términos de las tres funciones básicas de mantenimiento discutidas con anterioridad, claramente señalando que el entendimiento del software requiere del mayor esfuerzo. En la Figura 6, Currier da otra perspectiva de lo que hacen los encargados del software. El punto de vista de Currier además enseña que el mayor esfuerzo está dirigido al entendimiento del software (MC92).
Una de las principales causas de los problemas de mantenimiento es la dificultad de entender la intención de los programadores que realizaron el sistema. Debido a una estructura pobre, una lógica encontrada, nombres de programas sin sentido, ninguna norma en los nombres de los datos y definiciones, y poca documentación; el software es extremadamente difícil de entender.
Un software que es difícil de comprender, también es difícil y peligroso de cambiar. Por ejemplo, en 1983 Weimberg reportó que los 10 errores más caros de programación se debían a errores de mantenimiento. Los tres más altos, que envolvían cambios en solamente una línea de código,

costaron a sus compañías, $1.6 millones, $900 millones, y $250 millones de dólares.

Software frágil

Existe una alta posibilidad de introducir errores cuando se cambia el software. La probabilidad de introducir un error en 200 líneas de código varía entre 35% y 75%.

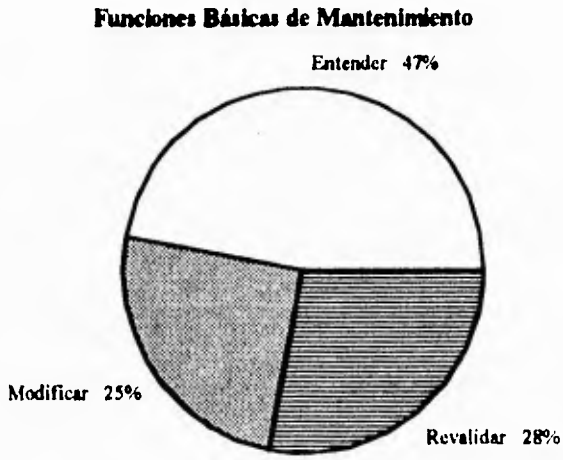

Figura 5. Estudio de IBM realizado por Fjelstad y Hamlen en términos de las tres funciones básicas de mantenimiento

Figura 6. El punto de vista de Currier además enseña que el mayor esfuerzo está dirigido al entendimiento del software (MC92)

El origen de los errores más comunes de software, conforme a un estudio realizado por James Collofello y Jeffrey Buck en 1987, es la creación de nuevos defectos en versiones anteriores al tratar de añadir funciones nuevas y de mejorar su funcionamiento. El estudio encontró que 53% de los errores eran consecuencia de modificar el sistema al tratar de perfeccionarlo (CO93).

El mundo del mantenimiento se encuentra lleno de software frágil, software que se *quiebra* fácilmente cuando se cambia.

El mantenimiento es difícil ya que, por una parte, hay un gran número de sistemas frágiles que deben de ser mantenidos, y en la otra, hay falta de herramientas de mantenimiento lo suficientemente poderosas. Existe la necesidad de herramientas que ayuden a comprender, cambiar, y revalidar sistemas ya existentes (Figura 7). Así como hay herramientas para mejorar la productividad y la calidad del software durante el desarrollo, deben de existir herramientas que mejoren la productividad y la calidad del software durante la fase de mantenimiento. La necesidad es aún mayor para el mantenimiento, ya que ahí es donde muchos de los profesionales en sistemas pasan la mayor parte de su tiempo (La productividad de desarrollo del software para la tercera generación de lenguajes es en promedio de 10 a 15 líneas de código por día, teniendo para el mantenimiento de una a dos líneas por día).

Herramientas que Ayudan al Mantenimiento

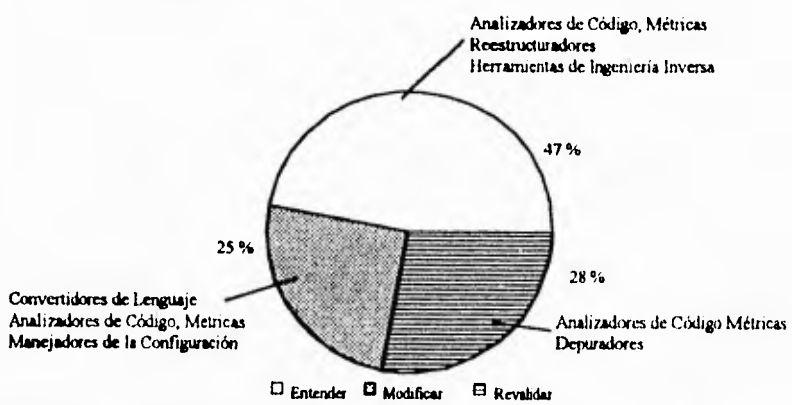

Figura 7. Las herramientas de reingeniería ayudan a comprender, cambiar, y revalidar los sistemas

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.

- **Actividades de análisis**
  - Desarrollo de 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
- **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
- **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
- **Actividades de implementación**
  - Usar estructuras de una sola entrada y una sola salida
  - Usar sangrado estándar en las estructuras
  - Usar un estilo de codificación simple y claro
  - Usar constantes simbólicas para asignar parámetros a las rutinas
  - Proporcionar margen de recursos
  - Proporcionar prólogos estándar de documentación en cada rutina
  - Apegarse a las guías de comentarios estándar internos
- **Otras actividades**
  - Desarrollar una guía de mantenimiento
  - Desarrollar un juego de pruebas
  - Proporcionar la documentación del juego de pruebas

Actividades de análisis. La fase de análisis del desarrollo del software se relaciona con la determinación de los requisitos y las restricciones del cliente, y el establecimiento de la factibilidad del producto. Desde el punto de vista del mantenimiento, las actividades más importantes sucedidas durante el análisis son señalar estándares y principios generales para el proyecto y para los productos de trabajo de modo que se garantice la uniformidad de los productos; establecer marcas de logros para asegurar que los productos de trabajo se produzcan en el tiempo programado; especificar los procedimientos de control de calidad para garantizar el desarrollo de documentos de alta calidad; identificar las mejoras del producto que probablemente ocurran después de la entrega inicial del sistema, y estimar los recursos (personal, equipo, espacio) requeridos para desarrollar las actividades de mantenimiento.

El mantenimiento del software puede realizarlo la organización responsable del desarrollo, el cliente, o un tercero por parte del cliente. En cualquier caso, el cliente debe recibir una estimación de los recursos requeridos y los costos probables que se ejercerán durante el mantenimiento del sistema. Estas estimaciones pueden influir fuertemente en la factibilidad de los requisitos del sistema, y pueden dar como resultado modificaciones a los requisitos. Una estimación de los recursos necesarios para mantenimiento permite planear y procurar las facilidades y el personal de mantenimiento que trabajará en el ciclo de desarrollo, además de prevenir sorpresas desagradables al cliente.

Estándares y directrices. Se pueden desarrollar varios tipos de estándares y principios generales para mejorar el mantenimiento del software. Los formatos estándar para los documentos de requisitos y las especificaciones de diseño, las convenciones de codificación estructurada, y los formatos estandarizados para los documentos de apoyo como el plan de prueba, los principios de operación, el manual de instalación y el del usuario contribuyen a la comprensión, y por lo tanto al mantenimiento del software. El grupo de control de calidad puede tener la responsabilidad de desarrollar y hacer cumplir los distintos estándares y principios generales durante el desarrollo del software. Los administradores pueden asegurar que las marcas de logros se estén cumpliendo, y que los documentos se estén desarrollando a tiempo, junto con las especificaciones de diseño y el código fuente.

Actividades de diseño. El diseño estructural se relaciona con el desarrollo de los componentes funcionales, de las estructuras conceptuales de los datos, y de las interconexiones en un sistema de software. La actividad más importante para mejorar el mantenimiento durante el diseño estructural es recalcar la claridad, la modularidad, y la facilidad de modificación como los principales criterios de diseño. Dadas las opciones para estructurar un sistema, los diseñadores elegirán una estructura particular sobre la base de ciertos criterios de diseño que pueden establecerse en forma explícita, o estar implícitamente comprendidos. Entre estos pueden encontrarse el acoplamiento y la cohesión de los módulos, consideraciones de eficiencia, las interfaces al software existente, las características en la arquitectura de la máquina, y otros factores. Buscar la claridad, la modularidad y la facilidad de modificación, suelen producir un sistema más fácil de darle mantenimiento que uno diseñado utilizando como principales criterios de diseño la eficiencia en el momento de ejecución y la minimización del espacio de memoria.

Los conceptos de diseño (encapsulamiento de la información, abstracción de datos y descomposición jerárquica descendente) son mecanismos apropiados para lograr una estructura de sistema clara, comprensible, modular y fácil de modificar. Para simplicidad de comprensión y de verificación de la perfección y consistencia del diseño, se deben emplear notaciones estandarizadas como los diagramas del flujo de datos, los de estructura o HIPO o una combinación de ellas. Estas formas de documentación del diseño ayudan al encargado del mantenimiento del software, quien debe comprender el producto de software lo suficientemente bien como para modificarlo y revalidarlo.

El diseño detallado se relaciona con la especificación de los detalles algorítmicos, las representaciones concretas de los datos y los detalles de las interfaces entre las rutinas y las estructuras de datos. Las notaciones estandarizadas se deben usar para describir algoritmos, estructuras de datos e interfaces. Las especificaciones de las interfaces de procedimientos deben describir los modos y los atributos del dominio del problema de los parámetros y las variables globales utilizadas por cada rutina. Además, se deben documentar las áreas de datos compartidos selectivamente, las variables globales, los efectos laterales y los mecanismos de manejo de excepciones para cada rutina que incorpore esas características. Se debe preparar una gráfica de llamados y un directorio de referencias cruzadas que indique el alcance del efecto de cada rutina; las gráficas de llamados y los directorios proporcionan la información necesaria para determinar qué rutinas y qué estructuras de datos se afectan con las modificaciones a otras rutinas.

Actividades de implantación. La implantación, al igual que el diseño, debe tener el objetivo principal de producir un software de comprensión y modificación sencillas. Se deben usar construcciones de una sola entrada y una sola salida del código; se debe observar la indentación estándar de las construcciones, y se debe adoptar un estilo de codificación simple. La facilidad en el mantenimiento se mejora mediante el uso de constantes simbólicas que parametricen el software, utilizando técnicas para el encapsulado de los datos, y por medio de márgenes adecuados en los recursos como tamaños de las tablas y pistas de sobreflujo en los discos. Además, los prólogos estándar en cada rutina deben proporcionar el nombre del autor, la fecha de desarrollo, el nombre del programador de mantenimiento, así como la fecha y el propósito de cada modificación. También, se deben documentar en el prólogo de cada rutina afirmaciones de entrada y salida, efectos laterales y excepciones, además de acciones para su manejo. Los comentarios internos en el código deben seguir los mismos principios generales en todos los programas.

Documentos de apoyo. Hay dos documentos de apoyo particularmente importantes que se deben preparar durante el ciclo de desarrollo del software para facilitar las actividades de mantenimiento. Estos documentos son la guía de mantenimiento y la descripción del conjunto de pruebas. La guía de mantenimiento brinda una descripción técnica de las capacidades operacionales del sistema completo, diagramas de jerarquía, gráficas de llamados y directorios de referencias cruzadas del sistema. En ella se debe especificar una descripción externa de cada módulo, incluyendo su propósito, las afirmaciones de entrada y salida, los efectos laterales, las estructuras de datos globales a las que se tuvo acceso, las excepciones y las acciones para su manejo.

Todo producto de software entregado debe ir acompañado de un conjunto de pruebas; esto es un archivo de casos de pruebas desarrollado durante las pruebas de integración del sistema y las de aceptación del cliente. Tal conjunto debe contener un grupo de datos y los resultados reales de dichas pruebas. Cuando se modifica el software, se añaden los casos de prueba al conjunto de éstas para validar las modificaciones, y éste se corre completo de nuevo, con el fin de certificar que las modificaciones no hayan introducido efectos laterales inesperados. La ejecución de un conjunto de pruebas después de la modificación al software se denomina prueba de regresión.

La documentación del mencionado conjunto debe especificar la configuración del sistema, las suposiciones, condiciones y razones para cada caso de prueba, los datos de entrada reales para cada prueba, y una descripción de los resultados esperados de cada prueba. Durante el desarrollo del producto, el grupo de control de calidad a menudo tiene

asignada la responsabilidad de preparar los conjuntos de pruebas de aceptación y de mantenimiento.

CAPÍTULO 2

REINGENIERÍA

En una empresa la toma de decisiones, se basa en la información. La incertidumbre ante el futuro y la falta de conocimiento sobre la situación actual hacen que el gerente busque información. Por lo tanto, la información puede definirse –informalmente– como datos organizados que reducen la incertidumbre en el momento de tomar decisiones. Un ambiente en constante transformación y el aumento de tamaño y complejidad de los sistemas han incrementado las necesidades de información por parte de los gerentes. El costo de las decisiones erróneas tiene dimensiones exorbitantes, pero a su vez ha crecido enormemente la utilidad conseguida con buenas decisiones estratégicas. Sin duda hacen falta buenos sistemas para suministrar a los gerentes información oportuna, adecuada y concisa en todo tipo de empresa (MU88).

En los últimos veinticinco años han evolucionado una variedad de tecnologías de desarrollo con el fin de incrementar la productividad en el desarrollo de sistemas de software. Los avances incluyen los lenguajes de tercera generación en los años 60s, metodologías y técnicas estructuradas en los 70s, e ingeniería de software con herramientas CASE en los 80s. En esta época, la programación orientada a objetos, la reusabilidad y generación del código, y la reingeniería de software prometen aumentar la productividad de manera considerable.

Sin embargo, muchas compañías cuentan con sistemas ya desarrollados, por lo que los gerentes deberán de invertir en su software para que éste

se adapte a las nuevas necesidades y requerimientos para ser más competitivos.
Cuando se planea originalmente un sistema de software, se piensa muy poco en la cantidad de trabajo de mantenimiento que se requerirá, por lo que el mantenimiento que se le tiene que aplicar al software toma por sorpresa a las organizaciones.
Como analogía: Cuando se compra una máquina industrial que cuesta N$300,000. se prevén gastos de mantenimiento preventivo para la misma. Sin embargo, cuando se invierte en un sistema de software que cuesta N$1,000,000. no se contempla gastar para su mantenimiento en forma regular, empezando la labor de mantenimiento cuando el sistema no satisface los nuevos requerimientos de la compañía.
Una vez que el sistema está produciendo, la organización reacciona a las solicitudes de mantenimiento conforme éstas van creciendo. De cualquier modo, ya que el mantenimiento del software es la actividad más común para la mayoría de las organizaciones, se debe de tratar como una función bien planeada y bien manejada, dentro del ciclo de vida del software, no como una sorpresa (BR94).
Se requiere una revolución en la ingeniería de software: los diseñadores deben de redefinir la manera de desarrollar sistemas, las nuevas herramientas como la reusabilidad de código, la programación orientada a objetos, y la reingeniería se deberán de tomar en cuenta para los nuevos desarrollos (RO94).
La reingeniería se basa en las siguientes premisas:

- Los sistemas existentes de software son elementos valiosos de los cuales depende la corporación, por lo que deben de manejarse apropiadamente
- El mantenimiento del software se puede realizar de manera más eficiente cuando es ayudado por herramientas poderosas

Definición

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. Ésta envuelve el mejorar el 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/o modificarlo con la ayuda de herramientas automáticas para:**

- **Incrementar la disposición del software a recibir 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**

Los propósitos de la reingeniería son tanto colocar a los sistemas existentes para que tomen ventaja de las nuevas tecnologías como dejar que nuevos desarrollos utilicen los sistemas ya existentes como base. La reingeniería tiene el poder de aumentar la calidad y la productividad del software en todo su ciclo de vida.
La reingeniería usualmente envuelve un cambio de forma (cambios en nombres de los archivos y definiciones, reestructuración de la lógica) en el programa y el mejoramiento de su documentación, la funcionalidad (comportamiento) del programa no cambia; sólo se modifica su forma. En otros casos, el proceso de reingeniería realiza modificaciones de forma y rediseña el programa para mejorar la funcionalidad del mismo, cubriendo así los requerimientos del usuario. A continuación se listan los 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 del mismo. Estos componentes pueden ser usados –o reusados– en la fabricación de nuevos sistemas, disminuyendo así el tiempo de desarrollo.

Tipos generales de reingeniería

Como se mencionó anteriormente, los sistemas existentes son elementos importantes para una empresa. El primer paso de la reingeniería es el de seleccionar el sistema que tendrá que ser modificado. Esta labor se realiza examinando los sistemas ya desarrollados para entender sus

componentes, su funcionamiento, y medir la calidad del mismo; identificando así a los mejores candidatos para la reingeniería.
Después de haber seleccionado el sistema a modificar, se podrán aplicar tres tipos de reingeniería:

1. Reestructuración
2. Ingeniería inversa
3. Administración de configuración y cambios

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.
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.
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.

Requisitos

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 utilizan actualmente en el desarrollo de nuevos sistemas pero aplicándolos a los sistemas existentes (RA92).

Tipos de herramientas de reingeniería

Las herramientas son una parte importante de la reingeniería al cambiar substancialmente la productividad y calidad del trabajo de mantenimiento. En la Tabla 1 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. En el apéndice se proporciona una lista de 57 herramientas de reingeniería. Estas herramientas se encuentran clasificadas con respecto a la tabla 1.

TIPOS DE HERRAMIENTAS DE REINGENIERÍA

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

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

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

Ingeniería inversa
- Ingeniería inversa de datos
- Ingeniería inversa de lógica

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

Convertidores
- De lenguaje

Manejadores de configuración y cambios
- Manejadores de control de cambio
- Manejadores de librerías
- Generadores de código

Herramientas de redocumentación
- Referencias cruzadas
- Generadores de diagramas

Tabla 1

Las herramientas de reingeniería pueden incrementar tanto el nivel de confianza de los encargados del mantenimiento, como la calidad del software a ser mantenido. Los encargados del mantenimiento no se tienen que preocupar por saber en donde deben de hacer los cambios ni cual será el impacto en el programa. Los analizadores de código pueden ayudar a el equipo de mantenimiento a entender mejor a un programa proporcionando automáticamente gráficas de flujo, código que nunca se ejecuta, y variables que nunca se utilizan. Por ejemplo, el encargado de mantenimiento puede saber con 100% de certeza y en cualquier parte del programa, donde está referida alguna variable, o donde se invoca algún procedimiento. El resultado es un incremento en exactitud y eficiencia en cualquier cambio realizado.

Con el uso de herramientas de prueba se pueden crear automáticamente datos para revalidar el código recién creado o cambiado. Como el sistema puede ser sometido a una prueba fuerte y completa, la probabilidad de introducir errores por un cambio realizado en el código fuente se reduce bastante.

Herramientas de medición, como analizadores complejos ofrecen una medida cuantitativa de mejoramiento de la calidad del programa y una manera de juzgar cuando un cambio arriesga o no la calidad del sistema.

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.

Como se muestra en la Figura 8, la reingeniería es el proceso de preparación del sistema para el rediseño y reemplazo, proceso durante el cual se pueden utilizar la ingeniería progresiva y las herramientas CASE.

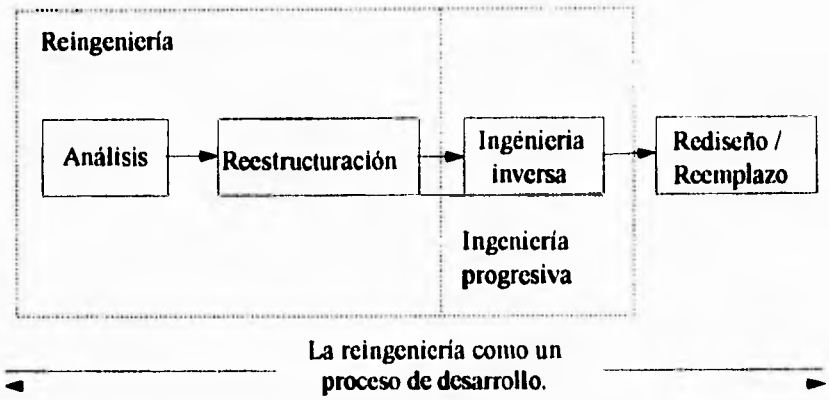

Figura 8. La reingeniería es el proceso de preparación del sistema para el rediseño y reemplazo

Manejo de sistemas existentes

Comúnmente la primera solución que se plantea en las organizaciones para disminuir el trabajo de mantenimiento, es el reemplazar a los sistemas existentes con nuevos sistemas que cubran mejor los requerimientos, utilizando en ellos tecnologías más avanzadas desarrollando así sistemas más fáciles de mantener. Tal vez esta sea la solución más apropiada para un sistema en particular, otras soluciones pueden ser más apropiadas dependiendo del sistema y de la organización. Existen varias opciones a seguir con los sistemas existentes: el reemplazo cae en un extremo de las opciones, el realizar el mantenimiento como se ha venido realizando se encuentra en el otro extremo, y la reingeniería del software cae en medio.

Sistemas frágiles

En la mayoría de las organizaciones 20% de los sistemas de software causan 80% de los problemas de mantenimiento. En otras ocasiones tan sólo 5% del código causa 80% del trabajo de mantenimiento. Los sistemas que se encuentra en cualquiera de los dos casos, representan los sistemas frágiles en las organizaciones, los cuales son difíciles de cambiar y que contienen muchos errores.

Estos sistemas comúnmente presentan características como las que se muestran a continuación:

- Obtención frecuente de fallas
- Problemas de desempeño
- Tecnología anticuada
- Problemas de integración con otros sistemas
- Pobre calidad de código
- Difícil (peligroso) de cambiar
- Difícil de probar
- Caro de mantener
- Aumento de problemas en el sistema

Aplicar el concepto de reingeniería a los sistemas frágiles puede reducir el trabajo de mantenimiento, ya que en el proceso se realizan cambios de forma en el sistema haciéndolo más fácil de entender y seguro de cambiar. Por otro lado, el dar mantenimiento a un sistema frágil como hasta ahora se ha venido realizando, seguramente traerá como consecuencia un aumento de tiempo (y de costos) ya que los sistemas frágiles tienden a convertirse más difíciles de mantener con cada cambio.

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
- Que 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

La reingeniería de estos sistemas es la mejor alternativa, ya que es barata, fácil, y segura. El costo rescribir el sistema manualmente es de $10 a $25 dólares por línea de código, el costo de la reingeniería es de $0.02 a $2.00 dólares.

Algunas veces, la reingeniería no es suficiente. Cuando un sistema es completamente indescifrable, el algoritmo utilizado es muy pobre, o se proporciona solamente una pequeña parte de la funcionalidad requerida, entonces es necesario reemplazar el sistema.

La Tabla 2 es un resumen de las razones para reingeniería y para el reemplazo.

REINGENIERIA CONTRA REEMPLAZO

Labores a realizar cuando un programa muestra determinadas características:

Reestructuración de código del programa

- Violación de estándares
- Código incongruente
- Pésima documentación
- Nombres sin sentido
- Lógica compleja

Reestructuración de datos

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

Ingeniería inversa, migración

- Tecnología vieja
- Lenguaje y DBMS antiguos
- Falta de especificaciones de diseño

Reemplazo Total / Parcial

- Algoritmo pésimo
- Ilegible
- Funcionalmente incompleto o incorrecto
- Diseño de BD defectuoso

Tabla 2

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 a 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 la disposición del sistema a 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. Por ejemplo, las herramientas de medición pueden hacer contribuciones a la calidad del programa o a un módulo en particular. Los analizadores de código pueden reportar las deficiencias del programa, como códigos muertos, recursiones, lógica confundida, falto de estructura, y violaciones al lenguaje de programación.
Con herramientas de reingeniería se pueden identificar los sistemas que deben de ser reemplazados, los que deben de perdurar, y aquellos a mejorar. El trabajo de mantenimiento puede ser planeado y manejado.

CAPÍTULO 3

ANALIZADORES DE CÓDIGO Y HERRAMIENTAS DE MEDICIÓN

Uno de los principales objetivos de la reingeniería, es el de mejorar la futura capacidad de mantenimiento. La capacidad de mantenimiento se define como el procedimiento más sencillo con el cual un sistema de software puede ser corregido descubriendo errores o deficiencias. En las primeras fases de mantenimiento se puede saber si el sistema va a ser fácil de entender, modificar, y probar. Pero ¿Cuándo se sabe que un sistema es mantenido correctamente con base en las características encontradas en las primeras fases? ¿Cómo se puede determinar qué capacidad tiene un sistema en particular para recibir mantenimiento?

Los herramientas de medición proporcionan una medida aceptable del la disposición del software a recibir mantenimiento. Diferentes herramientas de medición se han utilizado para medir objetivamente qué tan sostenible es un sistema. Existen varias compañías que utilizan instrumentos de medición, su uso da como resultado ahorros significativos en los costos del ciclo de vida del software, así como mejoras en la calidad de los sistemas. La habilidad de utilizar los instrumentos de medición trae consigo beneficios a largo plazo, prediciendo en qué parte de los sistemas es factible que se presenten errores y dificultades. Por ejemplo, se puede predecir el número de errores que contiene un sistema, así como cuándo se encuentra lo suficientemente probado.

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. Algunas métricas proveen importantes señales de las limitaciones que tienen los programas existentes; otras ayudan a disminuir la probabilidad de introducir errores en el momento de desarrollar nuevos programas; y otras diferentes como son las métricas de productividad ayudan a medir con qué rapidez se desarrolla un producto de software.

Las herramientas calculan de manera automática el tamaño y complexión del programa, ayudando a los encargados y directores del proyecto a entender, comparar y evaluar los programas. Las herramientas de medición son utilizadas para identificar los requerimientos para la reingeniería del software y para asegurarse de que la calidad del software no disminuya por los cambios realizados en el mantenimiento. La habilidad de medir automáticamente la calidad del sistema de una manera objetiva y cuantitativa es esencial para tomar el control del mantenimiento. 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

Por lo general cada herramienta métrica produce su propia fuente de información de manera gráfica y/o como reporte. Sin embargo, no existen herramientas de medición que puedan ser aplicadas en las especificaciones del diseño y ayuden a comprender la calidad en las primeras fases del ciclo de vida del programa analizado.

¿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 al mismo. Si un programa no se comprende, es virtualmente imposible de mantener de manera efectiva y eficiente. Por desgracia, existen varios programas difíciles de entender, siendo éste un factor preponderante en el alto costo del mantenimiento. Muchas veces la gente desiste de intentar cambiar un programa ajeno, después de una secuencia de intentos frustrados al tratar de interpretarlo.

La interpretación se define como la facilidad con la cual se comprende el propósito del programa, teniendo como fuente principal de información los siguientes documentos: el código fuente, la descripción de la base de datos, las especificaciones de los archivos y demás documentación asociada. Para que un programa sea interpretado, se debe contar con información adecuada que determine los objetivos, los supuestos, las salidas, las entradas, los componentes, los niveles existentes, y las relaciones con otros programas. La interpretación también implica poder entender el programa a varios niveles de detalle y de abstracción.

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. La experiencia del programador y el trato con un lenguaje en particular de programación afecta su habilidad para entender ciertos programas.
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. Este patrón impone restricciones en la forma de desarrollar el programa, en la modularidad y la documentación. Existen en el mercado programas que analizan automáticamente la estructura del sistema para asegurar el grado de estructura del mismo.
Sin embargo una buena estructura no asegura la interpretación completa de todos los aspectos del programa. Boehm sugieren 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. Por ejemplo, en un programa conciso cada instrucción debe ser ejecutada al menos una vez. Esta propiedad es considerada importante porque los códigos *no accesibles* o *muertos* implican un trabajo extra desconcertando a la persona de mantenimiento. Los analizadores automáticos de códigos que contengan rastreo lógico y rastreo de datos, pueden ser útiles para detectar códigos muertos así como elementos inútiles.

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. Consistencia en un diseño es a lo que Brooks llaman integridad conceptual. La integridad conceptual es conservada cuando un solo diseño arquitectónico básico es utilizado a lo largo de todo el programa. Es más fácil entender la estructura de un programa cuando éste contiene integridad conceptual.

Un programa **completo** es aquel 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.

Los analizadores de código y las herramientas de medición pueden reducir el tiempo y el esfuerzo necesario para entender el programa, especialmente cuando se encuentran escritos o modificados por otra persona. Sin estas herramientas, sería imposible entender largos y complicados sistemas de software.

Analizadores de código de los programas

Los analizadores de códigos, tales como los analizadores lógicos y los rastreadores de datos, proveen de ayuda automatizada para comprender el programa.

Los analizadores lógicos ayudan al personal de mantenimiento a comprender la forma con la que se utilizaron las estructuras de control (secuencia, decisión y repetición) en el programa, el flujo de la lógica, y a identificar problemas lógicos y otros defectos. Una vista de la estructura del programa puede verse en una gráfica, anotaciones en el código, o resaltando solamente aquellas líneas representativas en la jerarquía del programa.

Analizadores interactivos de datos pueden señalar el próximo lugar del programa a lo largo del código donde se cita una misma variable.

Gráficas de flujo

Estas gráficas son utilizadas para modelar el flujo de la lógica en un programa. La Figura 9 muestra un pequeño programa y su gráfica de flujo. Los puntos representan uno o más recorridos posibles de cada instrucción del programa y las líneas representan los trayectos entre cada instrucción. Las gráficas de flujo muestran todos los caminos posibles a través del programa (tres caminos en la Figura 9).

Gráfica de flujo de rigurosidad matemática

Las gráficas de flujo son herramientas poderosas para ayudar a entender el programa, ya que proporcionan de manera visual los caminos

La teoría de gráficas es la base de las herramientas métricas complejas, de reestructuración y de la programación estructurada.
En 1966, Boehm y Jacopini publicaron un artículo en el que demostraron que la secuenciación, la selección entre otras alternativas, y la iteración forman un conjunto suficiente de constructores para describir el flujo de control de todo algoritmo concebible.

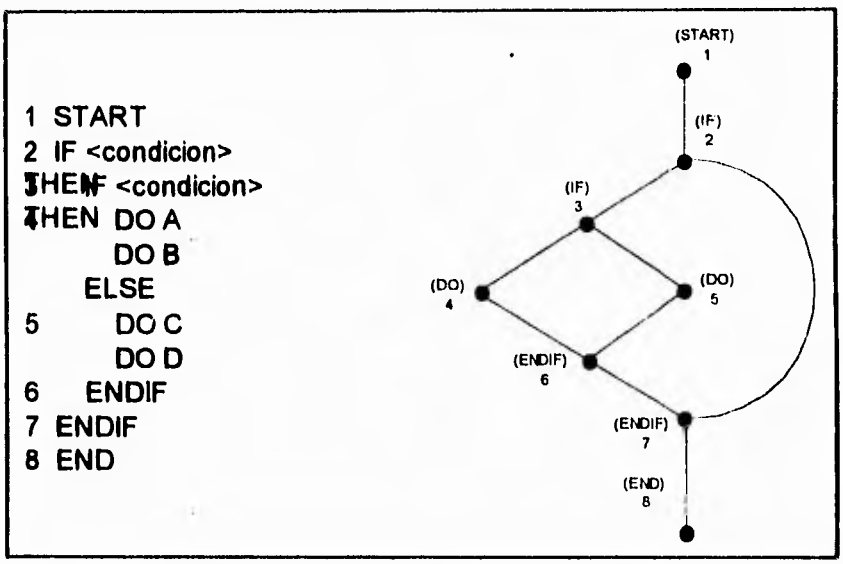

Figura 9. Ejemplo de un programa y su gráfica de flujo asociada

Una versión modificada del teorema de Bohm-Jacopini puede establecerse como sigue:

> **Cualquier segmento de programa con una entrada y una salida que tenga todas las proposiciones en algún camino de la entrada a la salida puede especificarse usando solamente secuenciación, selección e iteración** (FA87).

Asistentes de prueba

Las gráficas de flujo pueden ser muy útiles como auxiliares de prueba en un programa. Con la información de la lista de rutas, el encargado puede decidir que caminos tomar y las pruebas necesarias para cada uno.
Ya que las pruebas consumen una gran parte del esfuerzo en el ciclo de vida del software (20% a 40%), y del mantenimiento (28%) (FA87), las herramientas de prueba elevan la productividad y la calidad del software.

herramientas de prueba elevan la productividad y la calidad del software.
Si el lograr que los programas existentes sean fáciles de comprender es primordial para reducir los costos en el mantenimiento, entonces lo segundo más importante es automatizar las pruebas.
Si se posee un conjunto de fórmulas las cuales permitan calcular la complejidad de un módulo en especial, los resultados obtenidos pueden aportar información práctica; si se puede determinar cuáles módulos son los más complejos, se puede saber en que parte del código se tiene que poner más atención en el momento de probar o reescribir el programa. Cuando un programa contiene miles (o millones) de líneas de código, es casi imposible probar todo el programa (y mucho menos cuando existen varios programas de igual magnitud). Con herramientas que ayuden a identificar a que módulos debemos prestar mayor atención en el código fuente, se puede saber que partes del código se tienen que probar (CM93).

Delineadores (Profilers)

Un delineador es una herramienta para calcular el tiempo consumido en la ejecución de un programa.
Los delineadores permiten a los programadores deseubrir *cuellos de botella* que menguan el desempeño y mostrar los módulos que no son lo suficientemente rápidos y que necesitan ser reescritos.
Los delineadores permiten descubrir:

- En dónde utilizan su tiempo de ejecución los programas
- Cuántas veces se ejecuta una línea de código en particular
- Qué líneas se han ejecutado
- Cuántas veces se invoca a un procedimiento en particular y cuáles son los procedimientos que lo invocan

Las principales razones para analizar el tiempo de ejecución de un programa son:
Se necesita saber cuántas veces es llamada una función (o módulo en especial), ya sea por encontrarse en un ciclo o por otros motivos. Las funciones que son llamadas varias veces se les tendrá que prestar mayor atención que las que son llamadas una sola vez. Si una función es llamada varias veces y tiene un desempeño en tiempo malo, al volver a escribir el código para mejorar la ejecución se obtendrá un avance considerable en el rendimiento del programa.

Metricas de complejidad

Durante los últimos años, se han dedicado muchos esfuerzos al desarrollo de métricas para medir la complejidad del código fuente. La mayor parte de las métricas incorporan propiedades fáciles de calcular del código fuente, como el número de operadores y operandos, la complejidad de la gráfica del flujo de control, el número de parámetros y de variables globales en las rutinas, así como el número de niveles y formas de interconexión de la gráfica de llamados. El enfoque adoptado es calcular un número o un conjunto de números, que midan la complejidad del código. Por lo tanto, un programa con medida de 10 sería mas complejo que otro con una medida de cinco.

Una medida global de la complejidad del software, para cualquier técnica, debe considerar factores como el ambiente de computo, el área de aplicación, los algoritmos particulares implantados, los niveles requeridos de confiabilidad y eficiencia, además de las características de los usuarios del producto. Las medidas basadas exclusivamente en las propiedades del código fuente no toman en cuenta esos factores, por lo que son poco útiles al comparar la complejidad de dos programas por completo diferentes. Sin embargo, las medidas de la complejidad basadas en las propiedades del código fuente se pueden usar para comparar dos buenas versiones similares del mismo programa. Por lo tanto, las medidas de la complejidad del código fuente se pueden servir en la determinación de la complejidad de un programa antes y después de una modificación, y también para identificar las rutinas que probablemente tengan un mayor refinamiento y trabajo. Sin embargo, aun en esos casos se debe cuidar que los programadores no introduzcan complejidad de una naturaleza más oscura (en las estructuras de datos y en las técnicas de acceso a los datos, por ejemplo) para minimizar las propiedades medidas por las métricas particulares de complejidad en uso.

Dadas estas precauciones, y una comprensión detallada de lo que se mide y lo que no se mide por las métricas del código fuente, las métricas pueden ocuparse para indicar que algunos aspectos de la calidad del software están o no siendo degradados por las actividades de mantenimiento. Si la complejidad del código fuente se incrementa con cada modificación subsecuente, se puede alcanzar un punto en donde un producto de software que al principio estaba bien estructurado, era fácil de comprender y estaba documentado en forma adecuada, se vuelva difícil de mantener. La técnica de herramientas automatizadas para analizar el código fuente y calcular las medidas de la complejidad la hacen particularmente atractiva.

Métricas de tamaño y de flujo

Existen diferentes propuestas para medir las características que hacen que un programa sea complejo. De manera general, 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 éste. El tamaño del programa no necesariamente es el número de líneas que contiene. Una medida más aceptable del tamaño es la que se fundamenta en el número de procedimientos, número de operadores, número de variables, etc.

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.– que éste contiene. 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).

La complejidad es una medida de cuánto se entiende a un programa. Entre más complicado, más difícil es de entender. La complejidad del programa se encuentra en función de lo complicado que sea de programar, a su tamaño, y al número de caminos posibles que tome y lo difícil que sea seguirlos. Como ejemplo sencillo, un programa que posee sólo un camino de ejecución, no es complejo. Evitar que un programa sea complejo, da confianza y reduce el esfuerzo necesario para el desarrollo y el mantenimiento.

Se han propuesto varios métodos para medir lo complejo de un programa. Algunos se utilizan en el diseño, y la mayoría en el código fuente. Algunos miden qué tan complicado es un módulo y otros el programa entero. Se han sugerido diversas alternativas de uso de las herramientas:

1. **Herramienta de diseño:** Para evaluar qué tan bien se diseñaron los módulos y así sugerir cuando un módulo debe de ser subdividido
2. **Herramienta de prueba:** Para identificar cuál módulo es más difícil de probar y cuál es más susceptible a errores. Proporcionan criterios para seleccionar datos de prueba
3. **Herramienta de mantenimiento:** Para predecir cuáles módulos y programas van a ser más difíciles y peligrosos de modificar

Complejidad ciclomática de McCabe

La medida de la complejidad Ciclomática McCabe es la medida que más se utiliza hoy en día. Ha sido utilizada para mejorar programas en

FORTRAN, COBOL, PL/I, C, y PASCAL. Esta medida es la utilizada por varias herramientas de reestructuración y de CASE.
Thomas McCabe fue el primero en utilizar la teoría de gráficas y las gráficas de flujo para medir la complejidad en los programas. La complejidad Ciclomática es una teoría de gráficas que clasifica los caminos que se forman con las trayectorias únicas. Por ejemplo, en el programa de la Figura 9. Existen tres caminos lógicos posibles dependiendo de las condiciones IF:

Camino 1:	Declaraciones 1,2,3,4,6,7,8 (Si ambos IF son verdaderos)
Camino 2:	Declaraciones 1,2,3,5,6,7,8 (Si la primer IF es verdadera y la segunda es falsa)
Camino 3:	Declaraciones 1,2,7,8 (Si la primera IF es falsa)

Este programa tiene una complejidad Ciclomática de 3.
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 a 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

En el programa de la Figura 9. Existen tres trayectorias (líneas) y ocho nodos (puntos, declaraciones). Sustituyendo, se obtiene el siguiente resultado:

$$v(G) = 9 - 8 + 2$$

$$v(G) = 3$$

Que es el número de caminos lógicos en el programa.

Existe otro método de calcular el número de complejidad Ciclomática:

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

En la Figura 9, existen dos regiones encerradas o espacios en la gráfica: el espacio en forma de diamante y la área justo a la derecha de éste.
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 (IFs):

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

Extensión de la complejidad ciclomática

Una variante de la medida de complejidad Ciclomática de McCabe es la de Extensión de complejidad Ciclomática de Glenford Myers. En vez de simplemente contar los IFs, ésta 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> .

Complejidad esencial

La complejidad esencial McCabe mide el grado de estructura de un programa, no su complejidad lógica. 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. En efecto, la complejidad esencial se encuentra en función a los GO TOs en el programa.
El porcentaje de estructuración puede ser calculado como sigue:

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

El porcentaje de estructuración debe de ser 100%.
En la Tabla 3 se tiene un compendio de las métricas de complejidad de McCabe.

Número óptimo de la complejidad de McCabe

McCabe controla la complejidad controlando el número de caminos lógicos en el programa o en un módulo en especial.
McCabe recomienda un límite superior de 10 como la complejidad máxima para la gráfica de control de una rutina individual (Una excepción permisible es una proposición case grande con muchos casos

independientes). La complejidad ciclomática en el intervalo de tres a siete es normal en rutinas bien estructuradas.

MÉTRICAS DE COMPLEJIDAD DE McCABE

- **Complejidad ciclomática**: Cuenta el número de caminos lógicos en un módulo. Éste 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 ANDs y ORs
- **Complejidad esencial**: Mide el grado de estructuración de un módulo al contar el número de GO TOs. El resultado debe de ser 1

Tabla 3

En su artículo, McCabe informa una fuerte correlación entre la complejidad ciclomática, la facilidad de probar, y la confiabilidad de las rutinas. Curtis ha encontrado que la métrica de esfuerzo de Halstead se correlaciona más fuertemente con la facilidad de depuración y modificación que como lo hace el número ciclomático de McCabe.
Existen muchos programas disponibles que miden automáticamente la complejidad Ciclomática para códigos fuentes en FORTRAN, COBOL, C, PL/1, ADA, BASIC, y ENSAMBLADOR.
Una descripción detallada de los programas que utilizan la complejidad Ciclomática de McCabe se detallan en el Anexo.

Cuenta de nodos (Knot Count)

Como la medida de McCabe, la medida de cuenta de nodos mide la complejidad en función del grado de estructuración del programa.
Por ejemplo, en el programa de la Figura 10, los caminos de control se marcaron con líneas negras. Existe una línea de cada IF a cada ENDIF, y una línea de cada GO TO a su destino. Los *nodos* muestran donde los caminos de control se intersectan. Este programa tiene una cuenta de nodos de 10. Idealmente, la cuenta de nodos debe de ser 0.

Ciencia Halstead del Software

Durante los finales de los 70's en la Universidad de Purdue, Maurice Halstead desarrolló una serie de mediciones llamadas Ciencia Halstead del Software. Diversos estudios y experimentos tanto de la comunidad industrial como en la académica han demostrado que la ciencia Halstead asombrosamente satisface las medidas de complejidad del programa. Las mediciones de Halstead también se utilizan para medir la rentabilidad de un programa, para predecir el tamaño, y para estimar el esfuerzo de programación.

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 MOVEs, y un WRITE; por lo tanto tendrá tres operadores únicos, pero nueve operadores totales.

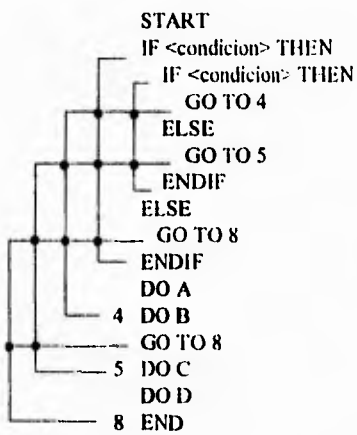

Figura 10. Los *puntos* muestran que el módulo no se encuentra estructurado

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**

Las medidas científicas de software pueden ser derivadas de estos cuatro valores básicos y son completamente independientes del lenguaje de programación utilizado.

Tamaño

Muchas teorías de complejidad se han desarrollado al relacionar estos números con las propiedades intrínsecas del programa, como son: *su tamaño, el volumen, y el lenguaje utilizado.* Por ejemplo, el **tamaño N** de un programa se calcula como:

$$N = N_1 + N_2$$

N es una medida simple del tamaño del programa. Entre mayor sea el valor de la **N**, más difícil es de entender y mantener el programa. **N** es una medida alterna de medir el tamaño del programa que simplemente contar las líneas.

En un estudio de 154 programas de PL/I en los laboratorios de investigación de General Motors, Elshoff fue capaz de predecir el tamaño de un programa de manera muy exacta (la correlación entre el largo actual y el predicho fue de 0.98) nada más con saber el número de operadores únicos y de los operandos en el programa. El tamaño estimado, **N'**, se calcula con la siguiente fórmula:

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

Esta fórmula muestra como Halstead estima que conforme el número de operadores únicos y operandos crece, el número total de operadores y operandos aumenta de forma logarítmica.

Elshoff encontró que el tamaño aproximado, **N'**, se acerca más a el 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:

$$\textbf{radio puro} = N' / N$$

Volumen

El tamaño **N** se utiliza en estimar la unidad de Halstead llamada volumen. Donde el tamaño es una simple cuenta (o estimación) del total de operadores u operandos, la medida del volumen da un peso extra a el número de operadores únicos y de operandos en el programa. Por ejemplo, si dos programas tiene 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:

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

Esfuerzo

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

$$\text{esfuerzo } E = V / L$$

donde el volumen **V** es dividido por el nivel del lenguaje **L**.
El nivel del lenguaje (el nivel de abstracción del lenguaje) es:

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

El nivel del lenguaje indica que tan poderosas son las instruciones de un lenguaje en particular. Por ejemplo, un procedimiento sencillo puede tener un valor de **L** de uno; COBOL debe tener un valor de **L** de 0.1, y en ensamblador un valor de **L** de 0.01. Por lo tanto, el esfuerzo **E** se incrementa, conforme se incrementa el volumen **V**, pero decrecer con el uso de un lenguaje de mayor nivel. Como esperábamos, el volumen decrece conforme un lenguaje va conteniendo juegos de instrucciones más poderosas. Las medidas de Halstead demuestran que un programa escrito en un lenguaje de cuarta generación se entiende más que los programas escritos con un lenguaje de nivel inferior.
De acuerdo a muchos estudios empíricos, el esfuerzo del programa, **E**, es una mejor medida para el entendimiento de éste que la **N**. En varios experimentos diferentes de programación, el esfuerzo de programación calculado utilizando medidas científicas de software se encontró que estaba muy cerca de los esfuerzos de programación actuales. En un estudio que envuelve FORTRAN, PL/1, y programas APL, Halstead predijo 22.51 horas. El tiempo verdadero fue de 20.15 horas (MC92).
Cuando se aplica a programas individuales, las medidas científicas de software no son siempre muy precisas. De cualquier modo, los resultados empíricos reportan que lo que más ayuda en el entendimiento del programa es examinar unas medidas elementales.
Las ventajas de las medidas científicas de software de Halstead son:

1. 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
2. Son aplicables a cualquier lenguaje de programación

3. 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 Tabla 4

MÉTRICAS CIENCIA HALSTEAD DEL SOFTWARE	
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 4

Contar y Listar

Otra medida simple de comparar dos programas diferentes es el contar en el programa:

- Líneas de código
- IF's
- Procedimientos
- Archivos
- Bases de datos relacionados con el programa
- Llamadas a procedimientos

O los *vicios de programación* –idealmente no debe de existir ninguno– como son:

- GO TOs
- Un grado de anidamiento de IFs grande (mayor a tres)
- Variables no inicializadas
- Código recursivo
- Procedimientos con baja cohesión
- Terminaciones ilegales en el programa
- Código que en ningún momento se ejecuta
- Violaciones de rangos (en apuntadores, arreglos, etc.)

Reglas de mantenimiento de software

Con analizadores de código se puede entender mejor un sistema. Con herramientas de medición, los encargados de mantenimiento pueden medir sistemas de una manera objetiva y cuantitativa. Pueden medir la complejidad, la estructura y el tamaño de cada programa individual o de

los módulos que los componen. Igualmente, se puede elaborar una lista de características y defectos del programa. Sistemas y librerías enteras pueden ser medidas utilizando estas mismas herramientas.
Las herramientas antes mencionadas son invaluables para la selección de programas y sistemas que son candidatos para la reestructuración o para la ingeniería inversa.

La Tabla 5 es una lista de analizadores de código de programas y de herramientas métricas.

ANALIZADORES DE CÓDIGO Y HERRAMIENTAS MÉTRICAS
Medidas de complejidad
• Gráficas de flujo
• Complejidad Ciclomática McCabe
• Medida Científica de Halstead
• Cuenta de Nodos
Contadores de programa
• Contadores generales (líneas de código, IF's, Procedimientos, funciones)
• Contadores de defectos (GO TO's, anidamiento excesivo IF's, fracasos, inicializado de datos, procedimientos traslapados)
• Lista de defectos (recursión, terminación ilegal, código no accesible, etc.)
Análisis lógico del programa
• Estructura del programa -cartas de estructura
• Trayectoria lógica del programa
Análisis de datos
• Rastreo de datos
• Registro del rastreo

Tabla 5

CAPÍTULO 4

REESTRUCTURACIÓN

El mantenimiento del software es muy caro y consume mucho tiempo, lo anterior se debe principalmente a que los sistemas existentes son difíciles de entender. La mayoría de los programas no se encuentran documentados, ni estructurados, ya que fueron desarrollados antes de la era de la ingeniería de programación (software engineering) o no se siguen metologías en su desarrollo. Una manera automática y poderosa para aumentar la productividad en el mantenimiento del software, y al mismo tiempo reducir el riesgo que se corre al cambiarlo; es estructurando y documentando al sistema. 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.**

El objetivo de reestructurar un programa es para mejorar la eficiencia y la efectividad del grupo de mantenimiento.A continuación se muestran los 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 incrementa 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.

Estructura y estilo

El estilo es la ruta consistente de elecciones hechas entre caminos alternos de lograr un efecto deseado.
El estilo de codificación se manifiesta en las rutas que usa el programador para expresar una acción o un resultado deseado. Los programadores que trabajan juntos pronto llegan a reconocer los estilos de codificación de sus colegas.
Durante los últimos años, se ha enfocado mucho la atención en el estilo de codificación (FA87). Se ha reconocido que un buen estilo de codificación puede superar muchas de las deficiencias de un lenguaje de programación primitivo, mientras que un estilo pobre puede frustrar los propósitos de un excelente lenguaje. El objetivo de un buen estilo de codificación es proporcionar un código fácil de comprender, sencillo y elegante.
No hay un conjunto único de reglas que se puedan aplicar en todas las situaciones; sin embargo, hay principios generales que son ampliamente aplicables.
Algunos de esos principios generales se listan en la Tabla 6 y Tabla 7 en las cuales se presentan los *sí* y los *no* de un buen estilo de codificación.

ACCIONES A SEGUIR PARA UN BUEN ESTILO DE PROGRAMACIÓN
Empléense unas cuantas construcciones estándar de control
Utilícense las estructuras GOTO de manera disciplinada
Introdúscanse tipos de datos definidos por el usuario para modelar entidades en el dominio del problema
Cúbranse las estructuras de datos bajo las funciones de acceso
Aíslense las dependencias de máquina en unas cuantas rutinas
Proporciónense prólogos estándar de documentación para cada subprograma y/o unidad de compilación
Examínense cuidadosamente las rutinas que tengan menos de cinco o más de 25 proposiciones
Utilícense sangrías, paréntesis, espacios y líneas en blanco, y márgenes alrededor de los bloques de comentarios para mejorar la legibilidad

Tabla 6

ACCIONES A EVITAR PARA UN BUEN ESTILO DE PROGRAMACIÓN
No hay que ser demasiado complicado
Evítense las proposiciones THEN nulas
Evítense las proposiciones THEN_IF nulas
No se anide en forma muy profunda
Evítense efectos colaterales oscuros
No se suboptimice
Examínense cuidadosamente las rutinas que tengan más de cinco parámetros formales
No se emplee un identificador para propósitos múltiples

Tabla 7

El estilo generalmente se considera como un asunto de preferencias personales y de marca de individualidad. La reestructuración clarifica, no oscurece, lo que el autor está comunicando. Para ello se ha de escribir el programa con sencillez para que el lector pueda entender con el mínimo de esfuerzo.

Las reglas de un buen estilo en programación son iguales que las reglas convencionales de escritura, ya que el programa va a ser mantenido y leído por personas. Si un programa está escrito utilizando un buen estilo de programación, desaparece una gran parte del problema de leer y entender. Tanto la reestructuración de la lógica, como la de datos, ofrecen un estilo fácil de entender dentro de los sistemas de software.

Existen dos tipos de reestructuración: reestructuración lógica y reestructuración de datos

Reestructuración de la lógica del programa

La reestructuración de la lógica de un programa es el proceso de reordenar el código fuente de acuerdo a 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.

Modularidad

Los sistemas modulares están formados por unidades claramente definidas y manejables, con interfaces perfectamente definidos entre sí. Un módulo esta constituido por una o varias instrucciones físicamente contiguas y lógicamente encadenadas, con puntos de inicio y termino perfectamente definidos. Los módulos se identifican mediante un nombre y pueden ser llamados por este, desde diferentes puntos de un programa. Un módulo puede ser un programa, una función o un procedimiento.
Se ha establecido que la modularidad es el atributo del software, que permite que un programa pueda entenderse en forma global. Los programas monolíticos son difíciles de comprender, debido al gran numero de trayectorias de control, al numero de variables y a la complejidad total involucrada. Es mas fácil resolver un problema complejo descomponiéndolo en partes que son fácilmente manejables y que posteriormente se pueden ensamblar para obtener el producto deseado, que pretender resolver el problema en forma global.
Un gran numero de módulos significan que serán de tamaño pequeño, sin embargo, a medida que aumenta el numero de módulos, aumenta el costo asociado a la creación de interfaces entre sí. Esta situación se ilustra en la gráfica de la Figura 11

Abstracción

Cuando se considera una solución modular para un problema, se pueden formular muchos niveles de abstracción. En los niveles altos de abstracción, se establece una solución en forma amplia, utilizando el lenguaje propio del ambiente del problema. En los niveles inferiores de abstracción, se hacen las consideraciones de procedimientos.

Cada paso del proceso de planeación y desarrollo es un refinamiento en los niveles de abstracción. Durante la definición de los requerimientos, la abstracción permite la separación de los aspectos conceptuales de los de procedimiento. Durante el diseño preliminar permite posponer las consideraciones estructurales y algorítmicas hasta que las características funcionales y las estructuras de datos han quedado definidas. En el diseño detallado se hacen las consideraciones estructurales y de proceso pertinentes. Este enfoque aminora la complejidad en cada etapa.

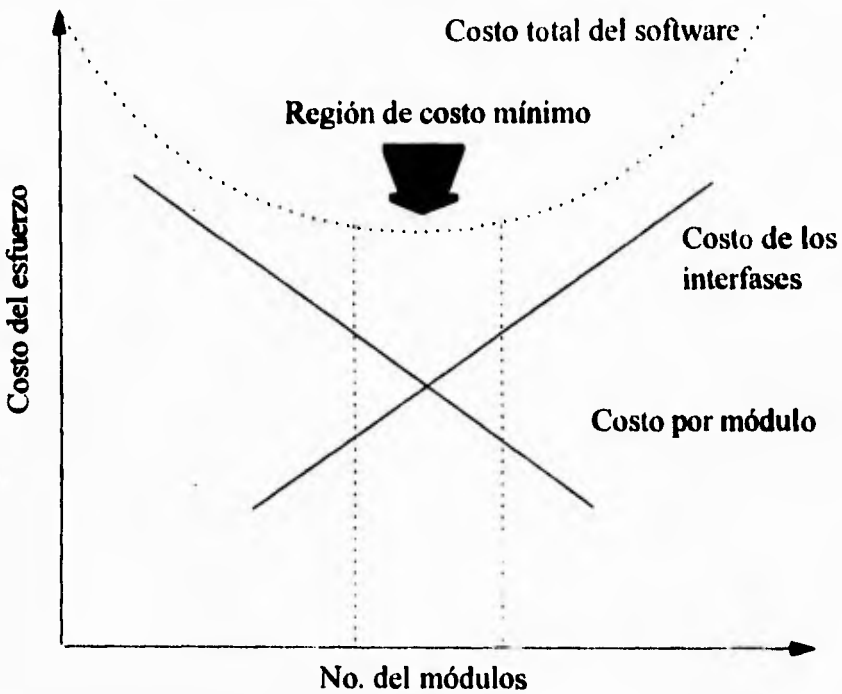

Figura 11. Costo del software en función de la modularidad (FA87)

Existen tres niveles de abstracción, los cuales son la abstracción funcional, la abstracción de los datos y la abstracción en el control. Los niveles antes mencionados permiten controlar la complejidad del proceso, avanzando sistemáticamente de lo abstracto a lo concreto.

Ocultamiento de la información

El concepto de modularidad conduce a preguntar como es posible obtener una solución que permita encontrar el mejor conjunto de módulos. El principio de ocultamiento de la información sugiere que cada módulo debería ser capaz de ocultar sus procesos internos a los demás. En otras palabras, los módulos deberían de elaborarse de tal

forma, que la información (procedimientos y datos) contenida dentro de un módulo sea inaccesible a todos los otros.
El término ocultamiento implica que la modularidad efectiva puede obtenerse definiendo un conjunto de módulos *independientes*, que se comunican entre sí, a través de la información necesaria para conseguir la función deseada. La abstracción ayuda a definir las entidades de procesamiento o de información que comprende el software. El ocultamiento define y refuerza las restricciones de acceso a los detalles de procedimiento y estructuras locales de datos utilizadas por un módulo.
El uso del ocultamiento de la información, como un criterio de diseño para sistemas modulares, proporciona grandes beneficios cuando se requieren modificaciones durante las pruebas y, más tarde, durante la fase de mantenimiento. Esto es posible, puesto que la mayoría de los procedimientos y datos se ocultan para los otros módulos del sistema. Los errores inadvertidos que se introducen durante las modificaciones, es menos probable que se propaguen a otros lugares dentro del sistema.

Tipos de módulos

La abstracción y el ocultamiento de la información se emplean para definir los módulos dentro de una estructura de software. Los atributos anteriores deben traducirse en características operacionales, para cada módulo, que permitan definir su incorporación en el tiempo, los mecanismos de activación y los patrones de control.
La incorporación en el tiempo se refiere al periodo en el cual un módulo se incluye dentro del programa fuente.
Los mecanismos de activación son dos. Generalmente un módulo se llama por referencia, por ejemplo, una instrucción CALL en FORTRAN. Sin embargo, en aplicaciones en tiempo real, un módulo puede invocarse por interrupción, esto es, un acontecimiento exterior causa una discontinuidad en el proceso, lo cual conduce a pasar el control a otro módulo. Los mecanismo de activación son importantes, dado que pueden afectar la estructura del software.
Los patrones de control del módulo describen la forma en que este opera internamente. Los módulos deberían tener una sola entrada y una sola salida, ajecutándose en forma secuencial. No obstante, algunas veces se requieren patrones de control mas sofisticados.
Dentro de una estructura de software se pueden clasificar a los módulos de la siguiente forma:

1 **Módulo secuencial**, aquel que se llama y ejecuta sin interrupción aparente del programa..

2. **Módulo de incremento**, es el que puede interrumpirse antes de su terminación, por medio del programa, y posteriormente restablecerse en el punto de interrupción.
3. **Módulo paralelo**, sise procesa en forma simultánea con otros módulos en ambientes de proceso concurrente.

Los módulos secuenciales son los de uso mas común, como ejemplo se tienen los subprogramas convencionales (procedimientos, funciones o subrutinas).
Los módulos de incremento, también conocidos como corutinas, mantienen un puntero de entrada que les permite reanudar la ejecución en el punto de interrupción.
Los módulos paralelos se utilizan cuando es necesario ejecutar un proceso a alta velocidad, el cual exige que dos o mas microprocesadores trabajen en paralelo. Por ejemplo, en líneas de proceso, sistemas de defensa, etc.
Una estructura de software típica no es posible cuando se utilizan corutinas o módulos paralelos. Tales estructuras no jerárquicas requieren enfoques de diseños especial, que se encuentran en las etapas iniciales de desarrollo.

Independencia de módulos

El concepto de independencia de módulos es una consecuencia de los conceptos de modularidad, abstracción y ocultamiento de la información. La independencia se logra haciendo que cada módulo realice una sola función y evitando la interacción excesiva entre sí.
Como se ha establecido, el objetivo es diseñar el software de forma que cada módulo efectúe una sola función especifica de los requerimientos y que tenga conexiones simples con otros módulos del sistema.
La independencia es importante, puesto que permite que los productos de programación sean fáciles de desarrollar y mantener. La independencia de los módulos es la clave para un buen diseño y el diseño es la clave para desarrollar un producto de calidad.
La independencia se mide por medio de dos criterios cualitativos cohesión y el acoplamiento. La cohesión es una medida de la efectividad funcional, relativa, del módulo, esto es, la coherencia interna del mismo. El acoplamiento mide la interdependencia, relativa, entre los módulos de una estructura.

Cohesión

La cohesión es una extensión natural del concepto de ocultamiento de la información. La cohesión mayor, para un módulo, se obtiene cuando este realiza una sola tareas y requiere poca interacción con los procedimientos que se llevan a cabo en otra parte del programa.

La cohesión puede representarse por medio de una escala como la mostrada en la Figura 12
Siempre debería buscarse una cohesión alta. Sin embargo, el punto medio de la escala es, por lo general, aceptable. La escala para medir la cohesión es no lineal, esto es, la separación entre los niveles bajos y los niveles medios de la escala, es mayor que la experiencia entre los niveles medios y superiores. Esta escala es parecida a la dureza de materiales que se utiliza en ingeniería.

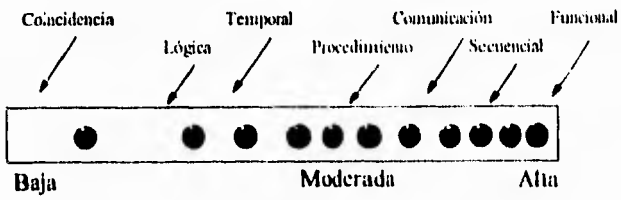

Figura 12. Escala para la cohesión

Un módulo que efectúa un conjunto de tareas que no tienen relación aparente entre ellas, tienen cohesión por coincidencia. Si el módulo ejecuta tareas que están relacionadas lógicamente, su cohesión es lógica, por ejemplo un módulo que produce todas las salidas independientemente de su tipo. Cuando un módulo contiene funciones que se relacionan por el hecho de que todas deben hacerse dentro del mismo periodo de tiempo, el módulo exhibirá cohesión temporal.
Los niveles medios de cohesión están muy relacionados entre ellos en el grado de independencia. Si los elementos de proceso de un módulo se llevan a cabo en un orden especifico, existe cohesión de procedimiento. Cuando todos los elementos de procesamiento se refiere al mismo conjunto de datos, se presenta la cohesión por comunicación.
La alta cohesión se caracteriza por módulos que realizan tareas perfectamente definidas. Si un módulo combina todas las transformaciones indicadas por sus fronteras, exhibe cohesión secuencial, esto es, cada elemento del proceso esta íntimamente relacionado con la misma función y deben ejecutarse en secuencia. Cuando un módulo procesa una y solo una función, el módulo posee cohesión funcional. En la Figura 13 y Figura 14 se muestran ejemplos de baja y alta cohesión.
Stevens, Myers y Constantine proporcionan un conjunto de normas para establecer el grado de cohesión. La aplicación de estas implica escribir una oración que describa la función o propósito del módulo y, posteriormente, evaluarla de acuerdo a las siguientes reglas:

1. Si la oración es completa, es decir, contiene comas o tiene más de un verbo, el módulo probablemente realiza mas de una función. Por lo tanto es probable que posea cohesión secuencial o de comunicación.
2. Si la oración contiene palabras alusivas al tiempo, tales como primero, siguiente, después, al iniciar, etc., entonces el módulo posiblemente tenga cohesión secuencial o temporal.
3. Si el complemento de la oración no contiene un objeto específico al cual aplicarle la acción del verbo, el módulo tal vez tenga cohesión lógica. Por ejemplo, edite todos los datos, tiene cohesión lógica, edite todas las instrucciones fuente, puede exhibir cohesión funcional.
4. La existencia de palabras tales como iniciales, ordene, limpie, etc., implica cohesión temporal.

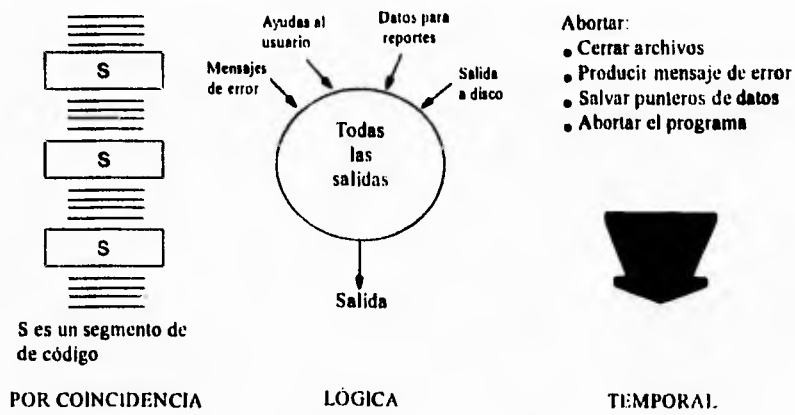

Figura 13. Baja cohesión

En la práctica no es necesario determinar el nivel preciso de la cohesión, no obstante, es importante esforzarse por obtener una alta cohesión, puesto que ésta hace posible que el producto pueda modificarse o ampliarse con facilidad.

Acoplamiento

El acoplamiento es una medida de la interconexión que existe entre los módulos de una estructura de software. Al igual que la cohesión, el acoplamiento se representa por medio de una escala, como se muestra en la Figura 15.

El acoplamiento depende de la complejidad de los interfaces existentes entre los módulos, el punto de entrada o llamada a un módulo y el tipo de datos que se intercambian a través del interfaz.

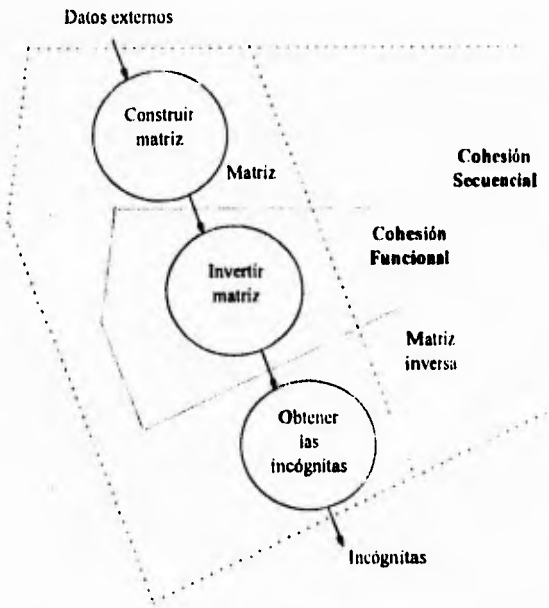

Figura 14. Alta cohesión

En el diseño del software se debe buscar el acoplamiento mas bajo. Las conexiones simples entre módulos, dan por resultado un producto que es fácil de entender y menos propenso a transmitir errores de un módulo a otro. En los niveles bajos de acoplamiento se tiene el acoplamiento de datos, en el cual se intercambian conjuntos de argumentos simples entre los módulos. Una variación del acoplamiento de datos, llamado acoplamiento estructural, se encuentra cuando se transfieren estructuras de datos entre los módulos (ver Figura 16).

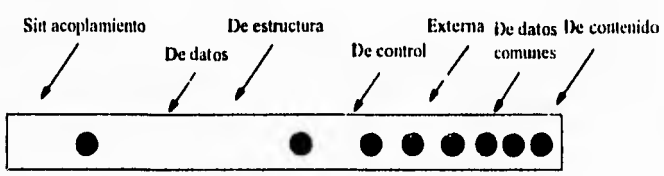

Figura 15. Escala para el acoplamiento

Los niveles moderados de acoplamiento se caracteriza por pasar instrucciones de control entre los módulos. El acoplamiento de control mas común, incluye el paso de banderas de control, ya sea como

parámetros o en forma global, entre los módulos, de forma tal que se transfieran el control de la secuencia del proceso de un módulo a otro (ver Figura 17).

Los niveles altos de acoplamiento ocurren cuando los módulos están ligados a un ambiente externo al producto. Por ejemplo, la entrada y salida de un módulo esta asociada a equipos específicos y protocolos de comunicación. El acoplamiento externo es esencial, pero debe limitarse a un pequeño numero de módulos dentro de una estructura.

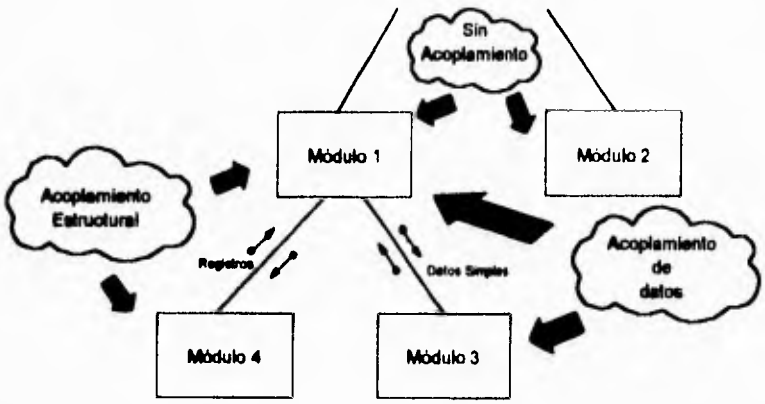

Figura 16. Acoplamiento bajo

Un acoplamiento alto, también ocurre cuando ciertos módulos hacen referencia a un conjunto de datos globales, este es el acoplamiento de datos comunes.

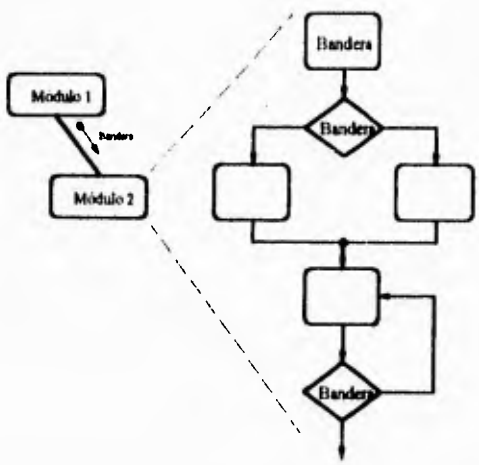

Figura 17. Acoplamiento moderado

El grado mayor de acoplamiento ocurre cuando un módulo usa datos o la información de control que se mantiene dentro de las fronteras de otro módulo, esto se conoce como acoplamiento de contenidos. Este tipo de acoplamiento puede y debe evitarse (ver Figura 18)

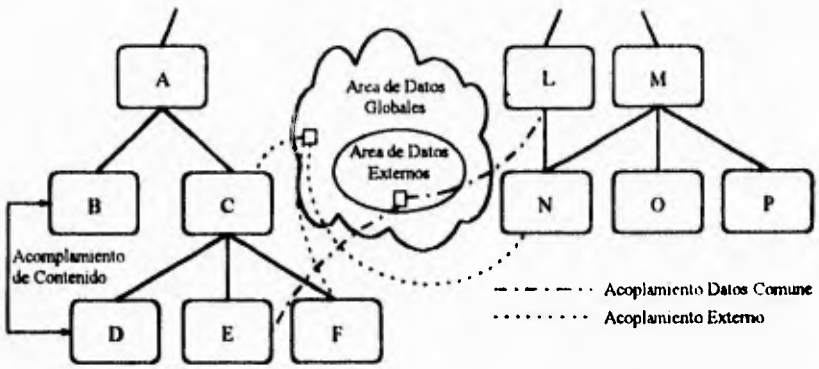

Figura 18. Acoplamiento alto

Programas bien estructurados

La teoría de la programación estructurada se desarrollo para resolver los problemas de complejidad. La clave para controlar la complejidad es el normalizar la estructura del programa. Como las medidas de complejidad, la programación estructurada posee una rigurosa base matemática, específicamente, en la teoría de gráficas. Como se mencionó anteriormente (capítulo tres), en los años 60's Bohm y Jacopini demostraron matemáticamente que todos los flujos lógicos posibles de control de un programa se pueden expresar utilizando tres estructuras básicas de control: secuencia, selección, e iteración. Estas tres estructuras de control son los cimientos para construir un programa bien estructurado.
Una versión modificada del teorema de Bohm-Jacopini pude establecerse como sigue:
Cualquier segmento del programa con una entrada y una salida que tengan todas las proposiciones en algún camino de la entrada a la salida puede especificarse usando solamente secuenciación, selección e iteracción

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.
Los objetivos para crear un programa estructurado son los siguientes:

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

Para cumplir con los objetivos de la programación estructurada se deberán de tomar las siguientes medidas:

- 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

Un programa estructurado es un programa modular, donde cada módulo representa una función única a ser realizada por el programa. Como se muestra en la Figura 19, los módulos están acomodados de manera jerárquica. Los módulos que llevan a cabo las tareas más generales se encuentran hasta arriba de la jerarquía; aquellos que hacen las tareas más específicas se encuentran abajo.

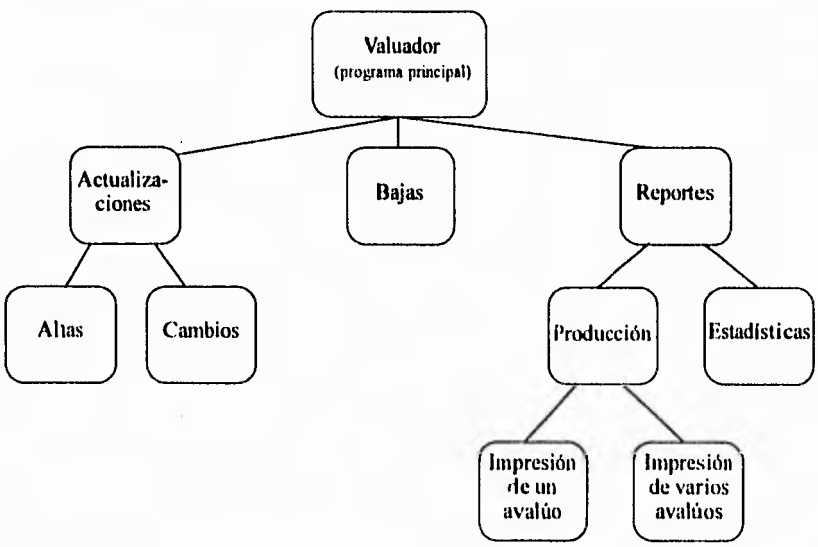

Figura 19. Un programa bién estructurado es modular y jerárquico

La forma normalizada de un programa bien estructurado da al lector un *mapa* de los caminos a seguir a través de cualquier programa,

dependiendo de su tamaño o de su complejidad inherente. Debido a la organización jerárquica de sus módulos, el lector puede entender al programa por niveles, trabajando con el siguiente *nivel de detalle* sólo cuando sea necesario. A continuación se tiene un 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, y se libera el control en la salida del módulo, regresando a el 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 solo para un propósito y el alcance de la variable es aparente y limitado.

La experiencia en la industria muestra que los programas bien estructurados, tienen un índice de error más bajo y un costo de mantenimiento menor a el equivalente en los programas no estructurados.

Cuando reestructurar

El modificar un programa para que se encuentre de acuerdo a las reglas de la programación estructurada lo puede hacer más sencillo de entender, y por consiguiente, más fácil de mantener. De cualquier manera, no todos los programas deben de estar bien estructurados. Programas que son familiares a el grupo de mantenimiento, que no se cambian frecuentemente, y que tienen un índice de error muy bajo; es mejor no alterarlos. Una regla básica del mantenimiento del software es: *Si no está roto, no lo arregles.*
Desafortunadamente, esta regla no se cumple con la mayoría de los programas existentes. La mayor parte de los programas se encuentran rotos, en el sentido de que son frágiles, difíciles de entender y de modificar. Por ejemplo, se estima que del 100% de líneas en COBOL, aproximadamente 70% se encuentran no estructuradas, ya que su lógica contiene:

- Uso excesivo de curvas de iteración
- Anidamientos excesivos
- Demasiadas constantes y literales

- Código auto-modificable
- Varios puntos de entrada y salida en los módulos
- Módulos con un número excesivo de líneas de código
- Varios módulos realizando la misma tarea

Los sistemas de software frágiles son los más beneficiados con la reestructuración. En general, programas frágiles, complejos, propensos a errores y sin estructura; son buenos aspirantes para la reestructuración. La siguiente lista resume 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

Proceso de reestructuración de la lógica del programa

El proceso de reestructuración de la lógica del programa transforma códigos fuentes no estructurados a códigos bien estructurados, esto es realizado con la ayuda de herramientas de reestructuración. El programa estructurado resultante es equivalente -funcionalmente- al programa original, ya que ambos programas realizan las mismas operaciones con los mismos datos. Claro que, cualquier error de lógica contenido en el programa original será transferido a el nuevo programa estructurado. Sin embargo, los errores serán más fáciles de detectar y de reparar en la versión estructurada.

La principal ventaja del proceso de reestructuración es que se puede elaborar por instrumentos automáticos, los cuales requieren una pequeña injerencia humana. El proceso de reestructuración consiste en seis pasos básicos:

1. Analizadores del código son utilizados para examinar los programas no estructurados y revelar los defectos, como son: códigos muertos o curvas infinitas.
2. El programa no estructurado se compila para identificar y corregir errores de compilación.
3. Se revisan los resultados de la compilación así como de los analizadores y se corrigen en el programa original (programa no estructurado).
4. La corrección del código fuente original se obtiene automáticamente y/o se rectifica con la ayuda de herramientas de reestructuración.

5. El código fuente estructurado es compilado con la ayuda de compiladores de optimización para aumentar las eficiencias del programa.
6. La estructura del programa se valida al comprobar que la nueva versión y la versión original sean funcionalmente equivalentes. Lo anterior se logra por medio de procesar los mismos datos en ambas versiones y comparar los resultados con comparadores automáticos.

Beneficios y efectos laterales

Diferentes estudios muestran que los programas reestructurados son la mitad o una tercera parte mas baratos de mantener que sus predecesores. De cualquier manera, se debe de tomar en cuenta los efectos laterales que se deriven de la reestructuración.

La curva de aprendizaje de los nuevos programas estructurados es uno de los factores predominantes en los costos de la reestructuración. A los encargados del programa les toma tiempo el adaptarse al nuevo modelo estructurado. Otro tipo de costo, es el asociado a la eficiencia del programa. Programas estructurados muchas veces son de mayor tamaño que sus antecesores, y no *corren* con la eficiencia esperada. Un estudio reportó que el incremento promedio en las líneas de código era del 26%, aunado a nueve por ciento más de tiempo requerido de CPU.

Además, el proceso de reestructuración puede darle nuevas secuencias a los módulos al incorporar nuevos párrafos o procedimientos con nombres que no son familiares para el personal de mantenimiento. Muchos de los comandos del programa viejo pueden perder su sentido o significado en la nueva versión reestructurada. Finalmente, ya que algunas de las herramientas de reestructuración no pueden manejar excepciones, pueden hacer la lógica del programa más complicada al introducir varios NOT'S y AND'S.

Métodos de reestructuración

La reestructuración lógica de los programas elimina los GOTO'S, algunas fallas de lógica, códigos muertos, recursividad, y las curvas infinitas creando así jerarquías en los módulos con programas bien estructurados. ¿Cómo se lleva a cabo esto? Muchos reestructuradores utilizan métodos de reestructuración de código basados en una aproximación por teoría de gráficas (ver
Figura 20).

Con la teoría de gráficas se pueden utilizar las gráficas de flujo para modelar la lógica de control del programa. La gráfica se encuentra formada por líneas y puntos, donde los puntos representan una o más declaraciones y las líneas representan los caminos de control.

Como se ve en la Figura 21, para reestructurar un programa, la herramienta de reestructuración lee la fuente del código y crea una gráfica de flujo del programa no estructurado.
El proceso inicia en el punto que representa la parte más baja de la red de las declaraciones IF de la gráfica (en el ejemplo, declaración 3), se empiezan a aplicar una serie de transformaciones de estructuración, como es la conversión de los GOTO'S a la rutina de PERFORM (COBOL). Durante esta transformación sólo se modifican las líneas de la gráfica, cambiando los caminos de control entre las declaraciones. Los puntos que representan las declaraciones permanecen sin cambio. Después la herramienta de reestructuración selecciona la declaración más baja que sigue (en el ejemplo, declaración 2) y realiza la misma transformación. Esta iteración continua hasta el IF más externo y todas las transformaciones se han completado.

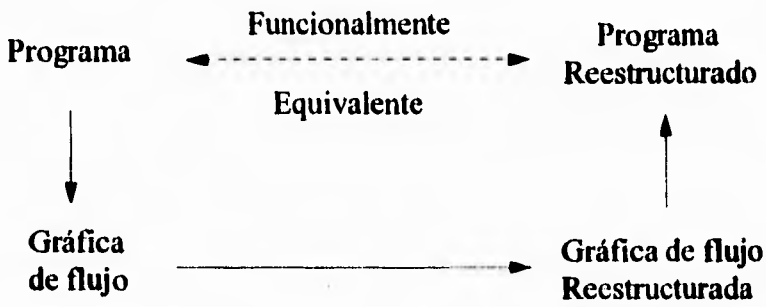

Figura 20. Los programas de reestructuración utilizan la teoría de gráficas

En el mismo instante que la modificación de los caminos de control se lleva a cabo, las herramientas de reestructuración también hacen modificaciones al código actual. Por ejemplo, se puede crear un procedimiento principal partiendo de la lógica esencial del programa y así aislar la *lógica cardinal del programa*; pueden aislar los procedimientos dependiendo de su función en el programa, procedimientos de E/S, procedimientos de cálculo, etc.; pueden dividir y combinar rutinas según sea necesario; pueden remover códigos muertos.
Las herramientas de reestructuración separan la forma del programa de la función del programa. Estas modifican la forma, pero dejan igual la función. Ya que el proceso de reestructuración esta basado en la teoría de gráficas, es posible demostrar de manera matemática que la versión sin estructurar y la estructurada son funcionalmente equivalentes.
Una vez que se han hecho todas las transformaciones y las modificaciones, la herramienta de reestructuración genera un nuevo código fuente. Este código refleja todas las transformaciones que se

hicieron en la gráfica de flujo, así como las modificaciones en el código sin estructurar.

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.

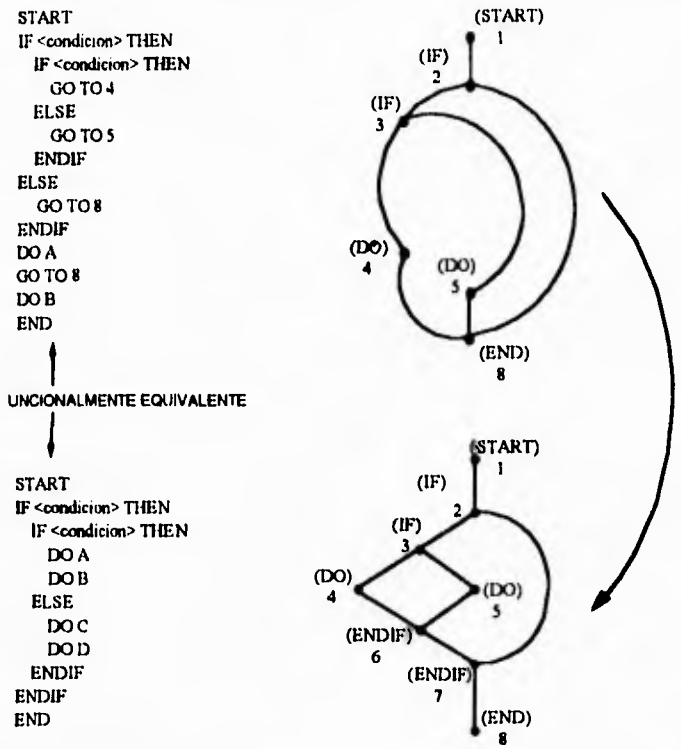

Figura 21. Las herramienta de reestructuración lee el código fuente y crea una gráfica de flujo del programa no estructurado

La mayoría de las herramientas de reestructuración soportan COBOL FORTRAN y PASCAL y son herramientas de procesos en lote. Su principal función es la de actualizar y convertir los códigos fuentes no estructurados a su forma estructurada. La versión estructurada de cada programa reemplaza la versión sin estructurar en la librería original (si el programa es parte de un sistema más complejo).

Las herramientas de reestructuración llevan a cabo las siguientes funciones:

- Reestructuran la lógica del programa
- Implementan varias opciones de estructuración
- Implementan varias opciones de formateo
- Producen reportes y documentación del análisis del programa

Análisis y documentación

El objetivo principal de la reestructuración lógica es el crear programas estructurados de unos no estructurados. De cualquier manera, estas herramientas también crean reportes de análisis y proporcionan documentación para que así los encargados de mantenimiento puedan actualizarse a la nueva versión estructurada del programa.
Las herramientas de reestructuración generalmente producen el siguiente tipo de análisis y documentación:

- Medidas y conteos (listas y conteos)
- Análisis lógico
- Cartas jerárquicas
- Apoyo de modularización

Medidas y conteos

Estos reportes muestran una métrica de la complejidad así como de varios conteos (e.g., GOTO, ALTER) para el programa tanto antes como después de la reestructuración. Son similares a las medidas y las cuentas descritas en el capitulo 3.

Análisis lógico — Rastreo y Problemas

Un rastreo lógico crea comentarios para cada párrafo en el programa recién estructurado. Los comentarios identifican para cada párrafo aquellos que van a ser llamados por éste. Los problemas lógicos que se pueden identificar son: recursividad, códigos muertos, y procedimientos sin regreso. El análisis lógico que se lleva a cabo aquí y es similar al descrito en el capítulo 3.

Cartas jerárquicas

Diagrama de módulos estructurados que muestran el nivel jerárquico de los procedimientos en el nuevo programa.

Apoyo de Modularización

Frecuentemente los programas seleccionados para la reestructuración no son solamente complicados sino además largos. El nuevo programa estructurado resultar muy largo, resultando más fácil de mantener si es subdividido en un conjunto de programas más pequeños. Identificando bloques del programa (procedimientos y todos los procedimientos que

estos llaman a su vez) que pueden ser removidos y relocalizados en un programa nuevo.
El apoyo a la modularización, ayuda a identificar que procedimientos tienen que ser repetidos para cada bloque de código a eliminar.

Técnicas de Reingeniería Conocidas

Las herramientas de reestructuración lógica son las herramientas de reingeniería más maduras y las más utilizadas.
Ejemplos de herramientas de reestructuración se especifican en el anexo.

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=TRANSPORTACIÓN+ COMIDAS+ ALOJAMIENTO

COMPUTE E = E1 + E2 + E3

La primer expresión da substancialmente 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 es el que debe de 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. El resultado de muchos experimentos de programación enseñan que entre más largos y más complejos sean los programas, más importante es el dar nombres estandarizados a las definiciones y a los datos. Muchos encargados de mantenimiento pierden gran parte de su tiempo intentando dar significados a programas llenos de nombres sin sentido y de definiciones de datos –tipos– inconsistentes.

Definición

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. Para mejorar el entendimiento, los datos utilizados en el sistema deben de estar definidos de igual manera a todo lo largo de éste.

> **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.**

La reestructuración de datos ayuda a preparar a los sistemas a la migración a nuevas tecnologías, como son los diccionarios de datos y los almacenes. Además, la reestructuración de datos ofrece otros beneficios como: el forzar a incorporar normas y convenciones, y reducir los esfuerzos y los costos del mantenimiento. El siguiente listado muestra los 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 realizando 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

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 dato
- Nmbres de procedimientos inconsistentes
- Código difícil de auditar

Algunas organizaciones han abandonado proyectos como el incrementar el tamaño del campo del código postal a nueve dígitos, después de determinar que el proyecto requeriría de un año de tiempo del equipo de mantenimiento realizar la corrección, cientos de programas serían afectados, y muchos programas podrían afectarse como resultado del cambio.

Un problema que existe en la actualidad es el relacionado con varios programas escritos en COBOL y que contienen variables de fecha con sólo dos dígitos. Estos programas no podrán diferenciar el año 2000 del 1900, ya que sus variables contendrán los dígitos 00. ¡En algunos programas las operaciones con fechas se realizarán con 99 años de diferencia!
La solución planteada por algunas compañías de Estados Unidos al problema anterior es contratar a programadores para que revisen millones de líneas y cambien las variables de dos dígitos por variables de cuatro (WH94).
A través de la reestructuración de datos y la utilización de herramientas de reingeniería –como los analizadores de códigos– se pueden realizar estos cambios de manera segura en unas pocas semanas.

Nombres

El escoger nombres mnemónicos contribuye a la claridad del programa. En la mayoría de los lenguajes la sintaxis de los identificadores no limitan el número de caracteres para sus nombres, por lo que el nombre a utilizar tendrá que ayudar a comprender el programa.
Una técnica utilizada es usar un prefijo (o sufijo) común para identificar un grupo de variables que se encuentran relacionadas lógicamente en el programa. Por ejemplo, si se realizan operaciones en varios archivos se pueden identificar de la siguiente manera:

archivo_principal archivo_transacción archivo_actualización

Otro método es el utilizar mayúsculas y minúsculas en los nombres para que sirvan como delimitadores visuales. Por ejemplo:

CostoAutobus TasaLibor PesoPromedio

No utilizar nombres no estándares o nombres personalizados que no sean claros, por ejemplo:

dps (días por semana) ineny (incremento en y)
mxdis (máxima dispersión)

Es recomendable utilizar caracteres extra para hacer más entendibles los nombres.

DíasPorSemana IncremY MaxDisp

Tener cuidado con respecto a dos nombres que puedan confundirse. Tener cuidado especial de no confundir 0 con O, 1 con l, y 2 con Z.

Contador	**Estado**
Contador2	**Estrato**

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, con el fin de asegurarse de que todas las variables en el sistema se encuentran definidas de igual manera (del mismo tipo) y bajo el mismo nombre.
Como se muestra en la Figura 22, las herramientas de reestructuración de código leen en el código fuente, el JCL, y en los tipos o descripciones de las variables; obteniendo como resultado nuevas variables con nuevos tipos. Además, crean reportes de análisis y archivos para exportar a un diccionario de datos, a herramientas CASE, y a almacenes.

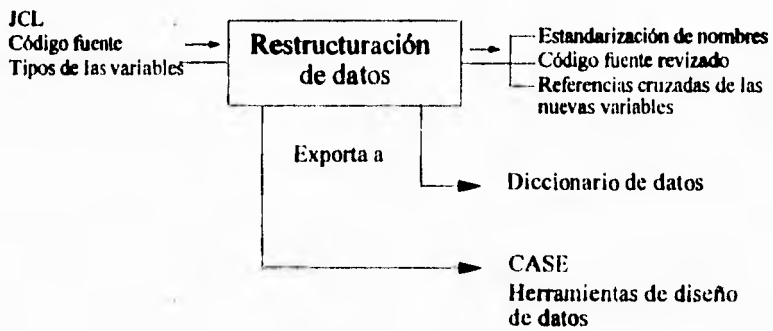

Figura 22. Orígenes y destinos de las herramientas de reestructuración

Las herramientas de reestructuración realizan las siguientes funciones:

- Análisis y actualización de las variables y los tipos a lo largo del programa
- Rastreo y actualización de un *elemento de dato* a lo largo del programa
- Actualización a un diccionario de datos
- Reportes de análisis

Análisis y actualización de las variables y los tipos a lo largo del programa

Las herramientas de reestructuración agrupan todas las variables y los tipos del programa y de todas sus unidades relacionadas. Realizan un listado que contiene los nombres de las variables, sus tipos, y sus longitudes; agrupadas por nombre o por longitud. Con esta información se pueden seleccionar las variables que realizan la misma función en el sistema y asignarle nuevos nombres estándares.
La herramienta de reestructuración se encargará de extender los nuevos nombres de las variables o los nuevos nombres de los tipos a lo largo del sistema.
Después de realizar los cambios se tendrán que probar que los sistemas funcionan correctamente y que no se han introducido nuevos errores al modificarlo.

Rastreo y actualización de un *elemento de dato* a lo largo del programa

Esta función permite identificar, delinear, y actualizar las variables relacionadas entre sí. Por ejemplo, si se tiene una constante llamada TASA_LIBOR que contiene un valor de 0.5 y existe una variable con nombre TASA la cual se le asigna el valor de TASA_LIBOR, entonces TASA_LIBOR y TASA son dos *elementos de datos* iguales. Las herramientas de reestructuración de datos delinean la trayectoria de un dato en especial a lo largo del sistema, permitiendo así identificar las variables que contienen los mismos valores para poder estandarizar sus nombres o eliminar alguna variable. La herramienta de reestructuración se encargará de extender los nuevos nombres de las variables a lo largo del sistema o de actualizar la variable eliminada por su variable equivalente.
Después de realizar los cambios se tendrá que probar que los sistemas funcionan correctamente y que no se han introducido nuevos errores al modificarlo.

Actualización a un diccionario de datos

En el proceso de análisis y actualización de las variables o de rastreo y actualización de un elemento de dato se realizan cambios en el código fuente al estandarizar los nombres y los tipos. Los nuevos nombres pueden ser incorporados en un diccionario de datos para poder ser utilizados en otros programas y aumentar así la estandarización de todos los sistemas existentes.

Reportes de análisis

Ya que las herramientas han leído y mecanizado los nombres de las variables y de los tipos, la información generada puede ser utilizada para producir reportes de análisis. Estos reportes proveen de un inventario completo de todas las variables y tipos del sistema; identifican las instrucciones de entrada y de salida; las llamadas a programas individuales; identifican variables que nunca se utilizan; y proveen referencias cruzadas de manera que los encargados de mantenimiento puedan predecir los impactos de cambios a un componente en particular antes de realizarlos, por ejemplo, si se convierte un archivo de COBOL a una base de datos, este reporte puede identificar todos los trabajos o procedimientos que utilicen ese archivo.

CAPÍTULO 5

INGENIERÍA INVERSA

Definición

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.

El código fuente

La principal referencia con que se cuenta para entender un programa es su código fuente. Aunque se puede tener la documentación que acompaña al sistema, esta normalmente no tiene una correspondencia directa porque normalmente es confeccionada con descuido y fuera de tiempo, además que es infrecuente encontrarla actualizada con las últimas acciones de mantenimiento.

Sin embargo, llevar a cabo la tarea de examinar miles de líneas de código fuente puede resultar imposible sin la ayuda de herramientas automáticas que documenten el software, pero que también elaboren descripciones de alto nivel donde los detalles específicos de implementación no intervienen ni abruman la tarea que se pretende

llevar a cabo. 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 que sepa contextualizar la documentación generada y pueda enriquecerla con elementos derivados de su experiencia, por lo que se debe concebir la ingeniería inversa como una práctica interactiva que requiere intervención humana y autómata.

Ingeniería inversa e ingeniería progresiva

La ingeniería inversa normalmente es utilizada para obtener la información necesaria para un nuevo diseño de un sistema, y va acompañada de un proceso de ingeniería de programación que efectúa los cambios en el diseño y los lleva acabo para obtener el nuevo producto. Durante el proceso de ingeniería inversa se examina la descripción a nivel físico para producir representaciones de nivel superior. El siguiente paso es utilizar herramientas CASE para realizar la tarea sin preocuparnos por la confección del nuevo sistema.
Esto significa que en el futuro, el mantenimiento dejará de hacerse a nivel código fuente para verificarse a nivel de especificación de diseño, lo cual reducirá sustancialmente la actividad y los costos del mantenimiento, llevándola a un nivel de abstracción más alto.
Por otra parte, se tiene como resultado adicional un acervo de información que se puede organizar dentro de un ***almacén***. El almacén es un programa que ofrece implementos para guardar y automatizar la información, además de una representación estandarizada que permite intercambiar la información entre diferentes equipos de programación y herramientas CASE, para utilizarla en nuevas aplicaciones.
La ingeniería inversa realiza labores que permiten establecer dos grandes áreas: ingeniería inversa de datos e ingeniería inversa de lógica, de las cuales, la ingeniería inversa de datos ofrece servicios más amplios, en la medida en que su materia prima es más fácil de describir y modificar.

Ingeniería inversa de datos

Es una técnica utilizada normalmente para documentar y modificar bases de datos, así como para migrarlas hacia nuevas tecnologías. El objetivo es construir modelos de las bases de datos que existen en una organización e integrarlos en un modelo coherente que coincida con los lineamientos estratégicos de la gerencia responsable.

El procedimiento se describe en la Figura 23, y consiste en extraer entidades, relaciones atributos y especificaciones de diseño a partir del código fuente, los diccionarios de datos y los DDL's. 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 (Figura 23).

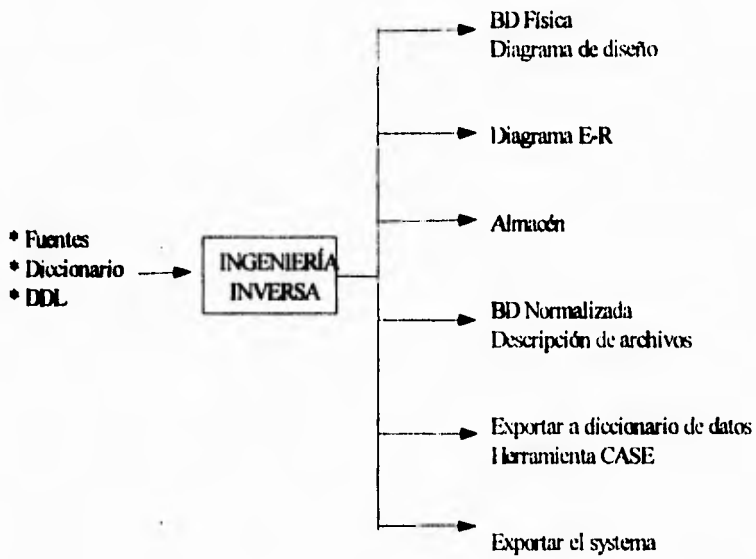

Figura 23. Proceso de la ingeniería inversa de datos

El método principia por transformar el código fuente, las descripciones de datos y los DDL's en diagramas de especificaciones físicas, que son depositados en el almacén (Figura 24). 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 ingresados 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.

La fase de ingeniería inversa consulta directamente las fuentes para obtener información inherente al diseño, como la longitud de los

Dichos diagramas serán entonces manipulados a través de un editor, instrumento básico para integrar diagramas de diferentes bases de datos, en una técnica bottom-up, como un modelo integral de la información utilizada por los programas actuales. El principal dividendo de esta herramienta es ofrecer una visión abstracta de las bases de datos existentes y efectuar las modificaciones a ese nivel, reduciendo tiempos y equivocaciones.

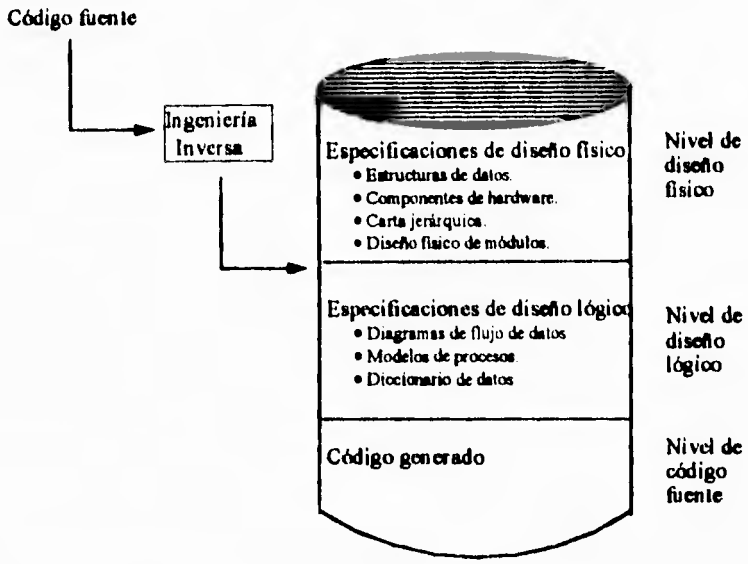

Figura 24. Almacén de la ingeniería inversa de datos

Durante la manipulación del diagrama, las herramientas proveen al desarrollo o al mantenimiento un juego completo de reglas expertas, basadas en inteligencia artificial.. Para la modelación lógica de las entidades-relaciones, las reglas demandan al usuario a seguir principios sanos de modelación de datos. Para el modelado físico de la base de datos, los sistemas expertos dan una guía basada en reglas físicas específicas de diseño de la base de datos para asegurarse de un buen diseño y una optimización de desempeño.

Finalmente, la información modificada y verificada es sometida a un procedimiento de ingeniería progresiva para producir los nuevos diagramas de la base de datos, el nuevo DDL e incluso algunas sentencias en IMS, DB2 u ORACLE. Las herramientas también pueden realizar acciones de normalización, en las que genera archivos de datos en la tercera forma normal (3NF) a partir de la información del almacén. También puede

en IMS, DB2 u ORACLE. Las herramientas también pueden realizar acciones de normalización, en las que genera archivos de datos en la tercera forma normal (3NF) a partir de la información del almacén. También puede revisar las bases de datos existentes para comprobar y, en su caso, proponer correciones para aquellas que no se encuentren en 3NF.

Otro esquema para efectuar estas labores es analizar las definiciones del código fuente para crear un modelo lógico de los datos. A continuación se refina el modelo y se importa a un almacén, donde se le aplica una fase progresiva para elaborar el nuevo código fuente.

El análisis consiste en descomponer las fuentes en objetos tales como archivos fuente, registros fuente y datos fuente. Entonces se efectúa una referencia cruzada entre estos objetos y las porciones de los programas que los utilizan, para aplicar un análisis semántico y generar el modelo lógico que consiste de los objetos y sus interrelaciones con el código. Este modelo es creado con nombres significativos, extraídos de un *vocabulario* que reside en el almacén y está constituido por nombres de aplicaciones previamente tratadas.

El modelo resultante puede ser enriquecido y manipulado por el analista, quien puede también agregar la documentación correspondiente. Todo quedará en un almacén que puede ser utilizado para generar las nuevas aplicaciones.

Estos dos esquemas ilustran la divergencia que existe entre los enfoques que pueden utilizarse para efectuar la ingeniería inversa de datos, y la necesidad del criterio al momento de aplicar sus herramientas. A riesgo de ser redundantes, se desea hacer énfasis en el carácter interactivo de estos procedimientos, que precisan la intervención y supervisión del factor humano en sus diferentes etapas.

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 las mismas a partir de el 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. Además, estas herramientas normalmente utilizan tres almacenes: el de diseño físico, el de diseño lógico y el de código generado (Figura 25).

Almacén de diseño físico

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 lo siguiente:

- **Las estructuras de datos.** La descripción en el lenguaje nativo indica los elementos que componen cada registro, arreglo, conjunto, etc., incluyendo el tipo de dato, su longitud y la forma en que están vinculados.
- **La carta jerárquica.** La modularización de los programas exhibe una jerarquía en la que ciertas subrutinas están sujetas al control de otras, y puede ser descrita en un diagrama dispuesto en forma de árbol, que incluye el nombre de cada subrutina.
- **Los diagramas HIPO.** Cada módulo debe tener un diagrama con la narrativa, los parámetros y las entradas y salidas.
- **La lógica de control.** Explica mediante un diagrama de flujo o un pseudocódigo la forma en que el módulo procesa las entradas para producir las salidas.
- **Los componentes de hardware.** Son aquellos dispositivos necesarios para completar la función de cada módulo.

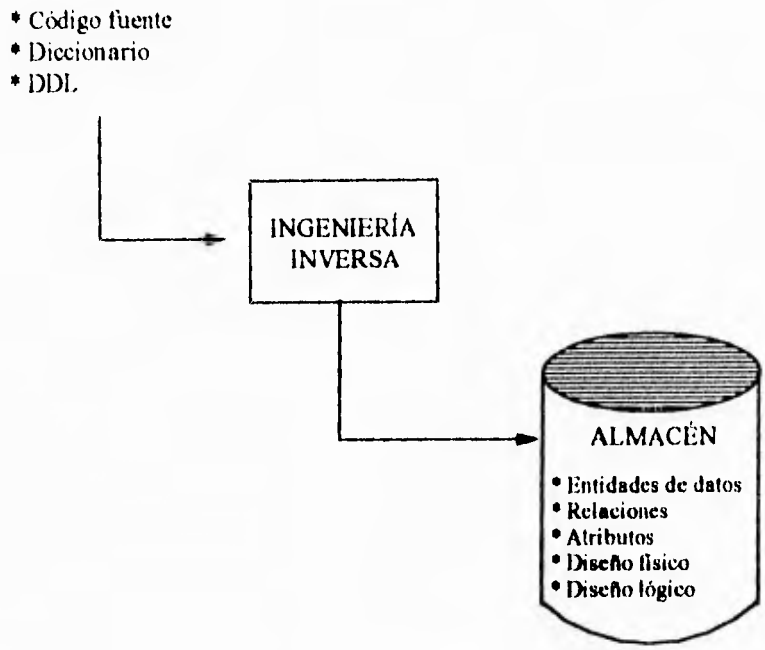

Figura 25. Proceso de la ingeniería inversa de lógica

Este tipo de almacenes normalmente tiene las siguientes prestaciones:

- Lenguaje de diseño
- Modificación de diseño
- Revisión de errores
- Documentación
- Generación de código (ingeniería progresiva)

El lenguaje de diseño es un mecanismo formal para representar esa información en el almacén físico. Dichos lenguajes pueden estar basados en objetos, como un diagrama entidad relación, o basados en pseudocódigo, el cual puede ser dependiente o independiente del lenguaje de programación.
La modificación del diseño se realiza en el almacén a través de un editor gráfico o uno de textos.
La revisión de errores es muy útil para modelar el impacto de las modificaciones durante el mantenimiento. Una vez realizadas las modificaciones en el diseño, el verificador de errores se ejecuta para comprobar la corrección del cambio y buscar efectos laterales inesperados. Después, una vez evaluada la dimensión de la modificación, se puede hacer una estimación de los recursos necesarios

para consumarlos. Este tipo de herramientas pueden trabajar en diferentes materias, como son:

- Sintáxis
- Consistencia de interfaces
- Consistencia de tipos de datos
- Consistencia en el flujo del control
- Alta cohesión
- Bajo acoplamiento
- Estancamientos

La documentación es una de las prestaciones más comunes de las herramientas de ingeniería inversa lógica, y pueden incluir los que se enlistan a continuación:

- Especificaciones de diseño formal
- Reportes de la jerarquía de los módulos
- Diagramas específicos de la metodología
- Cartas de estructura
- Reportes del modelo de datos
- Diagramas de control de flujo
- Reportes de referencias cruzadas
- Reportes de auditorías y requerimientos

La generación del nuevo código es un proceso asociado con la ingeniería progresiva y consiste en generar un nuevo código fuente a partir del diseño físico actualizado. El código generado está conformado por dos cosas: el nivel de correspondencia entre los elementos del almacén físico y las sentencias preparadas en lenguaje fuente; y el detalle de lógica y datos en dicho almacén. Algunas herrmientas muy bien integradas pueden llegar a generar entre un 80% y un 100% del nuevo código fuente, sobre todo en ambientes CASE o en almacenes con pseudocódigo específico del lenguaje. En otro caso se genera un cascarón de los programas que refleja la jerarquía, las definiciones de datos y las entradas y salidas de los mismos. Estos cascarones se generan normalmente con lenguajes estructurados como C, Pascal y Ada.

Almacén de diseño lógico

Este almacén normalmente está 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 normalmente tienen 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.

Almacén de código fuente

Las herramientas de ingeniería inversa lógica normalmente proveen un almacén para el código fuente, que 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.

Aunque no siempre se puede realizar una ingeniería progresiva completa con estas herramientas, siguen siendo una asistencia muy importante para el diagnóstico, la documentación, el modelado y la migración de los sistemas durante el mantenimiento. Entre los usos prácticos de la ingeniería inversa lógica se pueden mencionar:

- Verificación de programas recién codificados para asegurar que correspondan con sus especificaciones.
- Análisis, documentación y diagnóstico de los sistemas existentes para facilitar las tareas de mantenimiento.
- Análisis de impacto de las modificaciones, previo a la verificación de las mismas, para garantizar que están comprendidos y controlados.
- Migración a otro lenguaje o ambiente.
- Integración de varios sistemas en uno solo.

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 mas complicada que hacerlo de 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 embrollada. 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 hacia involucrar nuevas tecnologías, como la inteligencia artificial que permitan manipular la lógica de un programa a 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 (El.94).

Beneficios

La información resultante del proceso de ingeniería inversa es utilizada como un apoyo durante el mantenimiento del software y eventualmente puede favorecer estas tareas significativamente. Además, auxilia la migración y la conversión de los sistemas y permite identificar porciones de software que pueden ser aprovechadas e integradas en bibiliotecas.
La ingeniería inversa facilita el mantenimiento del software documentando los sistemas existentes, presentándole al encargado de esta tarea diagramas jerárquicos y diagramas de entidad-relación que muestran la estructura del software y las relaciones entre los diferentes componentes. Además, combinando el uso de herramientas CASE, el apoyo no se queda en la revisión de la documentación, sino que es posible hacer modificaciones en las especificaciones y generar el nuevo código de manera automática.
La migración y la conversión de los sistemas se beneficia al contar con una documentación adecuada de los sistemas de software que facilita la adopción de nuevas tecnologías y nuevas plataformas.
Asimismo, es más fácil desarrollar los sistemas que los reemplazarán, contando con la información de los sistemas existentes como prototipo. Una enorme porción del código de los programas existentes puede ser utilizada para los nuevos sistemas, ofreciendo una programación y una interface más uniforme, además de reducir el volumen de trabajo en las aplicaciones desarrolladas, reduciendo costos y riesgos de errores en los productos.

El desarrollo de nuevos programas también puede aprovechar la ingeniería inversa al recopilar información sobre sistemas previos con los que se tiene que interactuar, información que puede guiar y acelerar los nuevos esfuerzos. Por otra parte, estas metodologías pueden ser utilizadas para probar que el producto final corresponde con las especificaciones originales. Esto se logra aplicando la ingeniería inversa a dicho producto, y luego confirmar que el diseño obtenido cumple enteramente con los requerimientos del diseño inicial.

La cuestión legal

Las cortes estadounidenses han definido la ingeniería inversa como *un método honesto y honrado de iniciar con un producto conocido y trabajarlo hacia atrás para discernir el proceso que ayudó a desarrollarlo y manufacturarlo*. Para la ingeniería inversa de software dicen *el proceso de empezar con un producto terminado y trabajarlo hacia atrás para analizar cómo opera y cómo fué hecho.*

Aunque esta actividad está protegida por las leyes del derecho de autor, la ingeniería inversa está sufriendo severos ataques. De hecho, las presiones de poderosas compañías de software son hacia limitar su aplicación a los desarrollos propios, siendo un delito en cualquier otro caso.

Gran parte de la irritación que causa la ingeniería inversa proviene de las diferencias entre las legislaciones internacionales sobre protección de propiedad intelectual. Cambios recientes en las leyes de la Comunidad Económica Europea permiten ingeniería inversa sobre un programa cuando se quiere crear otro que interactúe con él, sin intenciones de reemplazarlo. A raiz de estos cambios, Japón está reexaminando sus leyes de propiedad intelectual, y las compañías norteamericanas temen que los japoneses sean igualmente indiferentes a la problemática de la ingeniería inversa. Apremiados por IBM, Microsoft, Apple y otros, los representantes del comercio estadounidenses han expresado una grave preocupación por la posibilidad de que relajen su vigilancia en la ingeniería inversa, sobre todo porque los japoneses se han caracterizado por aprovecharla con gran éxito.

Aunque no se puede negar que se ha abusado de la ingeniería inversa, su aplicación es una actividad demasiado compleja, costosa y dilatada para los piratas de software, quienes prefieren simplemente copiar los programas.

Además, existen buenas y honestas razones para utilizar la ingeniería inversa: reciclar el software viejo en nuevos sistemas, depurar defectos codificados por programadores que tal vez ya no estén en la compañía, y asegurar la compatibilidad entre el software existente. Acaso en estos días de litigios de propiedad intelectual, la ingeniería inversa provea una

manera de identificar tecnología patentada de tal suerte que se pueda evitar que la vulneren. Al final, puede ser dañino el desalentar estas prácticas, particularmente si son un recurso legal en cualquier otra parte del mundo.

CAPÍTULO 6

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 en el mantenimiento del software. Belady y Lehman demostraron que los grandes productos de software tienden a evolucionar dentro de familias de versiones, en donde cada versión es similar, pero diferente, a otros miembros de la familia. Belady y Lehman desarrollaron cinco leyes de la evolución de los programas basadas en sus estudios de la dinámica de grandes sistemas. Sus cinco leyes se presentan en la Tabla 8 (LEH80). Las primeras dos señalan la necesidad de un esfuerzo continuo para conservar el mantenimiento de un producto de software.

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 numero de revisión en

curso, el estado en curso y la historia de las solicitudes de cambios para cada versión del producto.

CINCO LEYES DE LA EVOLUCIÓN DE UN PROGRAMA

1. **Cambio continuo.** Un programa pasa por un cambio continúo o llega a ser progresivamente menos útil. El proceso de cambio continúa hasta que llega a ser costeable reemplazar el programa con una versión creada de nueva cuenta.
2. **Complejidad creciente.** A medida que se modifica un programa que evoluciona, su complejidad, que refleja una estructura deteriorada, se incrementa a menos que se trabaje en él para reducirla.
3. **La ley fundamental de la evolución del programa.** La evolución de un programa esta sujeta a una dinámica que constituye el proceso de programación y, por tanto, las mediciones del proyecto global y los atributos del sistema, autorregulado con tendencias e invariaciones determinadas estadísticamente.
4. **Conservación de la estabilidad de la organización.** La tasa de actividad global en un proyecto que apoya un programa que evoluciona es estadísticamente invariante.
5. **Conservación de la familiaridad.** El contenido liberado (cambios, adiciones, supresiones) de las versiones sucesivas de un programa que evoluciona es estadísticamente invariante.

Tabla 8

La siguiente lista contiene las preguntas que cualquier base de datos para la administración de la configuración debe poder contestar.

1. ¿Cuántas versiones de cada producto existen?
2. ¿En qué difieren las versiones?
3. ¿Qué versiones de cuáles productos estan distribuidas en que instalaciones?
4. ¿Qué documentos estan disponibles para cada versión de cada producto?
5. ¿Cuál es la historia de revision de cada componente de cada versión de cada producto?
6. ¿Cuándo estara disponible la siguiente revisión de un componente dado de una version dada de un producto?
7. ¿Qué configuracion de hardware se requiere para operar una versión específica de un producto?
8. ¿Cuáles revisiones constituyen una versión específica de un producto?
9. ¿Qué versiones se afectan con la revisión de un componente dado?
10. ¿Cuáles versiones se afectan con un informe de error específico?
11. ¿Qué errores se corrigieron en una revisión específica?

12. ¿Cuántos errores se notificaron y corrigieron en el mes pasado?
13. ¿Cuáles son las causas de las fallas del producto que han sido informadas?
14. ¿Qué componentes son funcionalmente similares en cuáles versiones?
15. ¿Algún usuario emplea todavía una versión antigua dada?
16. ¿Cuál era la configuración de una versión dada en una fecha dada?

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 y no contiene la información requerida para responder preguntas como las antes formuladas. 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 numero 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 (FA87).

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.

Administración de bibliotecas

Entre los componentes de un programa, se pueden enunciar:

- Documentos de diseño del programa
- Módulos de fuente de código
- Módulos de objetivo de código
- Bibliotecas de respaldo
- Descripciones de archivos

- Definiciones de pantalla
- JCL
- Manuales del usuario

La administración de bibliotecas provee de clasificación, almacenamiento, y acceso a estos componentes, 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
- Representación lógica
- Control de revisión
- Control de versión
- Desarrollo paralelo (ramificación)

Identificación de componentes

Los componentes del sistema se encuentran almacenados en diferentes librerías y formatos, así que la primer tarea de la administración de bibliotecas es identificar todos esos componentes físicos y clasificarlos en tipos de componentes (fuente, módulo, JCL, etc.). Los nombres físicos del conjunto de datos de los componentes y su tipo se guarda en el directorio interno de la herramienta. Independientemente del tipo de componente, la herramienta opera de la misma manera el almacenamiento, los cambios y las modificaciones.

Representación lógica

Después de identificar los componentes físicos de un sistema, la administración de bibliotecas permite al usuario definirlos. Los componentes físicos se definen de la siguiente manera:

- Agrupados en programas/ subsistemas/ sistemas/ aplicaciones
- Asignados a individuos/ trabajo a grupos
- Asociados con requisiciones/mejoras/proyectos

Al agrupar los componentes de manera lógica se obtienen dos cosas:

1. Se evita en los usuarios la necesidad de aprenderse cientos de nombres de conjuntos de datos físicos, ya que la herramienta convierte automaticamente un identificador lógico al nombre del conjunto de datos físicos correcto; y
2. Se colocan todos los componentes juntos en un sistema de jerarquías que es utilizado después, en el elaborador de sistemas.

Control de revisión

Cada programa en un sistema puede cambiar cientos o miles de veces durante la vida de dicho sistema. Es función del control de revisión administrar estos cambios. El control de revisión maneja la denominación y el almacenamiento de cada componente original y de todas sus revisiones. Tipicamente 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, método que optimiza el espacio requerido de la siguiente manera: la herramienta compara la nueva versión del componente con la versión antigua, identifica los cambios, y guarda solamente los cambios identificados. Para cada componente, entonces, la herramienta almacena una copia completa y una serie de cambios delta, cada cual con su número de revisión correspondiente. Cuando es requerida una revisión en particular, la herramienta recurre a la copia completa y efectúa todos los cambios delta hasta la revisión que se requirió. Algunas veces la copia completa es la primerísima versión y se aplican los cambios con un estilo de avance. Lo más frecuente es que la copia completa sea la versión más reciente y los cambios sean respaldos. En cada caso, la herramienta puede volver a crear cualquier revisión de cualquier componente, según sea la solicitud.

Cada vez que se hace una revisión, la herramienta captura la información relevante, así como el autor del cambio, la razón, la fecha y la hora.

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. No solamente se identifican todas las revisiones y todos los componentes, sino una revisión específica de un juego de componentes debe de ser identificada como parte de una versión en particular. Esta identificación 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)

Durante el desarrollo y el mantenimiento, es necesario hacer revisiones múltiples a los componentes del software, así como reunir las revisiones específicas en versiones. El control de revisiones y versiones cubren esta necesidad. Sin embargo, algunos usuarios pueden requerir de una versión completa, y mantenerla en operación independientemente del

avance en que se encuentren los programas que la componen. El desarrollo paralelo se encarga de ésto.
En realidad, 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 revisión principales. Si, por ejemplo. un usuario comienza una derivación de un programa particular en una revisión particular para dotar de ciertos artificios a un cliente especial y luego decide brindar estos artificios a todos los clientes, la herramienta va a poner la última derivacion en el camino principal y va a identificar cualquier conflicto que se pudiera presentar.

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
- Registro de entradas y salidas
- Elaboración de sistemas
- Reportes

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 Asimismo limita el acceso a ciertas etapas del proyecto. La mayoría de las herramientas de manejo de configuración incorporan y extienden paquetes de seguridad,
como RACF, ACF2, y Top Secret, de tal suerte que los accesos no entren en conflicto con los nieveles de seguridad existentes. La seguridad también registra todas las actividades de los usuarios por proyecto, fecha y hora.

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. Si alguien más intenta registrar la salida el mismo programa, la herramienta avisa al primer usuario, y no permite la salida a menos que se cuente con autorización especial.

Una vez modificado el programa, se registra su entrada. En ese momento, la herramienta recolecta información del usuario describiendo la naturaleza del cambio, mientras la administración de bibliotecas compara la copia nueva con la anterior, crea y almacena el delta, asigna un número de revisión, e incorpora la nueva revisión como la más reciente.

Algunas veces es necesario registrar la entrada de dos copias diferentes del mismo programa. El registro de entrada compara las dos copias, se integran los diferente cambios, y se identifican los conflictos. Cuando están resueltos los conflictos, se registra la entrada una nueva versión ya revisada, reflejando los cambios de las dos copias.

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 necesita 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. Esta lista, junto con el número de revisión, controla lo que es compilado, ligado, respaldado o almacenado.

Cuando una versión o una mejora es creada el elaborador de sistemas revisa automaticamente las etiquetas de día y hora en los componentes afectados para asegurarse de que se encuentran actualizados. Por ejemplo, cuando se translada un sistema de prueba a uno de producción, el elaborador revisa que cada módulo objeto en el sistema posea una estampa de día y hora que corresponda a la estampa de día y hora del fuente. Si se cambió el fuente, la herramienta va a recompilarlo de manera que los cambios sean actualizados. También va a compilar o ligar cualquier otro componente que dependa del componente cambiado.

El elaborador de sistemas controla el traslado de los sistemas entre las bibliotecas. Basándose en la lista de elaboración y en la solicitud de acción, trasladará un sistema a producción, lo respaldará o recuperará una versión previa si aparece cualquier problema.

Varias herramientas de manejo de configuración manejan *sites* remotos distribuyendo los componentes a las bibliotecas apropiadas a través de

las redes instaladas. Operan desarrollos de microcomputadoras y/o medios ambientes de operaciones.
El elaborador de sistemas documenta las interdependencias de los programas y permite reconstruir y desplazar el sistema de manera rápida y segura. Refuerza el monitoreo y la aprobación de todos los movimientos, y mantiene un registro de las listas de elaboración y su evolución en el tiempo.

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. Muestran el trabajo en progreso listando todos los componentes activos, quién tiene acceso a ellos, y para cuál proyecto.
- Reporte de versión. Muestra a todos los componentes incluidos en una versión particular o en una ramificación, junto con sus números de revisión. También lista todas las versiones o ramificaciones donde aparece alguna revisión de componente.
- Reporte de interdependencia. Basado en la lista de elaboración, muestra la jerarquía completa de los componentes de un sistema, es decir, los componentes que conforman un programa, los programas que conforman un sistema, y sus interconexiones. También muestra todos los programas donde se utiliza un componente individual.
- Reporte del análisis del impacto del cambio. Identifica por medio de referencias cruzadas todos los componentes en el sistema que pueden ser afectados por el cambio en un componente en particular.
- Reporte de cambio en la actividad. Lista los cambios, indexados por varios atributos como componentes, fecha y hora, analista y proyectos.
- Reporte del cambio en la historia. Muestra los cambios a través del tiempo. Un reporte puede mostrar la descripción del cambio hecho por el usuario, así como las diferencias entre las revisiones; otro puede mostrar los cambios hechos a los componentes entre una versión y otra. Este reporte es útil para dar seguimiento a los cambios, analizar las tendencias en las modificaciones del sistema, creando una historia de auditorías, e identificar lugares donde se pudieron introducir defectos.

Beneficios

En principio, el uso de estas herramientas hace más sistematica la labor. A continuación se enumera una secuencia que ilustra el proceso de administrar la configuración:

1. El administrador del proyecto utiliza la seguridad del control de asignando un identificador de usuario con permiso de cambios, seleccionando los componentes de software relacionados con el cambio, definiendo niveles de acceso y requerimientos de aprobación. El usuario (programador) ejecutará un reporte de análisis del impacto del cambio para identificar cualquier efecto del cambio e incorporarlo a el proyecto si es necesario.
2. Conforme el usuario registra la salida, modifica, y registra la entrada de los programas afectados, el administrador puede revisar y rastrear todas las actividades. Cuando es momento de efectuar las pruebas, el usuario utiliza el sistema en elaboración y el control de versión para compilar y ligar en forma automática la revisión apropiada de todos los programas afectados por el cambio.
3. Después de probar y obtener el visto bueno, el usuario traslada los componentes cambiados a la versión de prueba y a las bibliotecas de producción, al tiempo que respalda los cambios según sea necesario. Teniendo el proyecto, el usuario y el administrador utilizan el reporte de versión, el de interdependencia y el de cambios para operar el cambio.

Las heramientas 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 mantenimeinto 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. De cualquier manera, en este entorno, la noción del manejo de la configuración debe incluir no solo el manejo a nivel físico de los componentes del software sino también el manejo del nivel lógico de los componentes.

Los usuarios de herramientas CASE aprendieron que sin una configuración apropiada el manejo de todos los componentes del software guardados en el almacén es imposible, aún con esas herramientas. 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.

CONCLUSIONES

Existe una crisis de software identificada desde los años 70, y no está resuelta. Hasta la fecha, la mayoría de las soluciones propuestas están enfocadas a mejorar las actividades envueltas en el desarrollo, y sus resultados han sido pobres.
Por otra parte, diversos estudios manifiestan que dentro del ciclo de vida de un programa, el mantenimiento es una labor predominante. Entre las causas que generan actividades de mantenimiento se pueden mencionar:

1. Es necesario corregir los defectos que son introducidos a los programas durante la etapa de desarrollo.
2. Las actividades para las cuales se desarrollan los programas son actividades que están cambiando contantemente, por lo que el software debe adaptarse a dicho entorno. Además, existe un vasto avance tecnológico en los campos de hardware y software, y los programas existentes deben ser modificados para aprovechar estas nuevas tecnologías.
3. Algunas aplicaciones cumplen con los requerimientos originales, pero lo hacen de una manera ineficiente, y es necesario realizar perfeccionamientos para mejorar el desempeño de la aplicación. Asimismo, con frecuencia resulta necesario afinar la documentación, proporcionar nuevas capacidades funcionales y hacer más amigables las interfaces.

De lo anterior se identifican tres tipos diferentes de mantenimiento: correctivo, adaptivo y perfectivo. Frecuentemente el mantenimiento se relaciona con la correción de errores y por lo tanto se presupone que el mantenimiento correctivo es la actividad que más se realiza. No obstante, los estudios nos demuestran que el mantenimiento adaptivo y perfectivo son las actividades primordiales.

Ya que el mantenimiento del software es la actividad más común para la mayoría de las organizaciones, Esta debe tratarse como una labor bien planeada y organizada, dentro del ciclo de vida del software, no como una sorpresa. Si se considera que el mantenimiento adaptivo y perfectivo no son beneficiados por los avances en las técnicas de desarrollo, se puede afirmar que la mejor forma de enfrentar la crisis del software es concentrar los esfuerzos en mejorar las actividades de mantenimiento en todos sus tipos.

Además, se deberá de contar con una estrategia para el mantenimiento a largo plazo, en lugar de ejecutar los cambios en los sistemas como estos se van presentando.

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.

Los elementos aplicados a la reingeniería incorporan varias de las mejores ideas del pasado y del presente de la ingeniería de software (reestructuración, herramientas de medición, CASE, etc.), lo que permite que se puedan aplicar a una gran variedad de empresas, no importando la metodología de desarrollo, el hardware, ni mucho menos el lenguaje utilizado en sus sistemas.

Existen tres tipos de reingeniería: reestructuración, ingeniería inversa y administración de configuración y cambios.

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.

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.

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.

La reingeniería no es un procedimiento rígido y lineal, es un conjunto de estrategias que pueden ser aplicadas total o parcialmente a un problema dado, de acuerdo a las características del mismo y al criterio del grupo de trabajo que esté encargado de realizar la tarea de mantenimiento.

Cabe destacar que resulta indispensable seleccionar a los programas candidatos para el proceso de reingeniería, por lo que es necesario realizar actividades de análisis y medición antes de emplear alguno de los tipos de reingeniería mencionados.

La reingeniería es una actividad que se encuentra en continuo refinamiento. Debe existir una relación armoniosa entre hardware, software y la gente –todos estos integrados en un ambiente de políticas y procedimientos y dirigidos por los objetivos de la organización–.

Sin embargo no podemos hablar de reingeniería si no se utilizan herramientas automáticas, las cuales pueden ser clasificadas de la siguiente manera:

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

La reingeniería brinda varios beneficios a los equipos de programación, permite a los profesionales del área resolver problemas al colocarlos en una posición ventajosa para el futuro. El legado que conforman los programas desarrollados puede ser aprovechado al convertir las aplicaciones existentes en aplicaciones que incorporen los avances tecnológicos. Pero lo mejor de todo es que no es necesario un gran esfuerzo para incorporar la reingeniería a los métodos de trabajo actulaes por que los departamentos de desarrollo ya estan familiarizados con la metodología básica y con las herramientas utilizadas en el desarrollo, mismas que pueden ser utilizadas para el mantenimiento.
Cuando empezamos a utilizar la reingeniería debemos tener presente que un enfoque basado exclusivamente en herramientas automáticas casi siempre garantiza un fracaso. No es necesario, ni siquiera aconsejable, hacer uso intenso de las herramientas. Tomar pasos pequeños y medibles, con metas bien definidas, es una excelente forma de empezar. El proyecto generará una inercia con los éxitos iniciales y permitirá delimitar correctamente los objetivos de la organización en lo que se refiere a redesarrollo.

APÉNDICE

A continuación se ofrece un listado de 57 productos clasificados 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

PAGINA DUPLICADA

92
93

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-Pan/LCM 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				

Productos listados en orden alfabético	Analizadores de programas	Métricas	Reestructuración	Ingeniería Inversa	Pruebas	Convertidores	Manejo de configuración y cambios	Herramientas de redocumentación
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		
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						

Productos listados en orden alfabético	Analizadores de programas	Métricas	Reestructuración	Ingeniería inversa	Pruebas	Convertidores	Manejo de configuración y cambios	Herramientas de redocumentación
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

FALTA PAGINA

No 96 a la 97

Aide-De-Camp Software Configuration Management System (V.9.0)

Software Maintenance and Development Systems, Inc.
200 Baker Ave., Ste. 300, PO Box 555
Concord, MA 01742
508-369-7398
FAX: 508-369-8272
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Paquetes vendidos: 2,000
Fecha de liberación: 1994
Compatible con: Sun SPARCstation/SunOS; DEC VAX, DECstation/VMS, ULTRIX, OSF/1; HP 9000, Apollo/HP-UX; IBM RS/6000/AIX; Silicon Graphics
Memoria RAM mínima: 4 MB
Espacio en disco requerido: 5 MB
Hardware/Software adicional requerido: X-Windows; Motif
Lenguaje fuente: C
Atención a usuarios: Disponible.
Soporte en sitio: Sí

Descripción

Sistema de manejo de configuración que maneja cambios y control de versión de código fuente y archivos no texto en complejos ambientes de desarrollo. Almacena el código fuente en una base de datos.

AISLE/QualGen

Software Systems Design, Inc.
3627 Padua Ave.
Claremont, CA 91711
909-625-6147
FAX: 909-626-9667
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$5,000-$17,000
Paquetes vendidos: 3
Fecha de liberación: 1990
Compatible con: DEC VAX, MicroVAX, DECstation, VAXSTATION/VMS,ULTRIX; DG; PC-MS/DOS; Sun-3, 4, SPARCstation/SunOS; HP 9000,Apollo; Gould/SEL; Harris; Sequent; Alliant; Concurrent;

Memoria RAM mínima: 4 MB
Espacio en disco requerido: 4 MB
Lenguaje fuente: C

Descripción

Analiza código en Ada en cualquier fase del desarrollo para extraer hasta 200 métricas predefinidas como Halstead, McCabe, y otras específicamente diseñadas para Ada. Ayuda a identificar las porciones de código que pueden estar causando problemas o que los pueden causar después. Incluye hasta 30 reportes predefinidos que permiten al usuario ver presentaciones gráficas o tabulares de métricas de calidad para cada unidad de programas o métricas separadas para cada programador. Permite al usuario la definición de reportes.

AIX Configuration Management and Version Control (CMVC) (V.2.2)

IBM (International Business Machines)
Old Orchard Rd.
Armonk, NY 10504
"800-426-3333; 914-765-1900"
Ventas: 800-426-7695 (IBM PC Direct)
Soporte técnico: 800-237-5511
Soporte técnico BBS: 919-517-0001; 800-847-7211 (OS2)

Especificaciones:

Precio de lista: US$695 (cliente); $4,960-$19,840 (server)
Fecha de liberación: 1993
Compatible con: IBM RS/6000/AIX; HP/HP-UX; Sun/SunOS

Descprición

Permiten a los desarrolladores de software el manejo de procesos de desarrollo y reduce la complejidad. Integra el manejo de diseño/defecto con el manejo de configuración y control de versión. Ofrece facilidad para realizar reportes.
Subdivide proyectos de acuerdo a las necesidedes de desarrollo y de mantemiento. Escalable al tamaño del proyecto y número de usuarios de la herramienta. Acepta como máximo mil usuarios.
Permite a un cliente manejar cambios. Ofrece la notificación automática del término o comienzo de ciertos eventos predeterminados. Importa datos de SCCS

Answer:Comparex/CDF

Sterling Software, Inc. (Applications Management Division)
5900 Canoga Ave.
Woodland Hills, CA 91367-4237
"800-267-9972; 818-716-1616"
Ventas: 800-587-1001
FAX: 818-716-5998
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$15,000
Fecha de liberación: 1993
Compatible con: IBM/VSE, VM, MVS/XA, MVS/ESA
Soporte en sitio: No

Descripción

Herramienta de comparación de datos que automatiza la integración de multiples versiones de código fuente. Automáticamente compara más de ocho versiones de código fuente y realiza una nueva versión que señala e identifica el código original y las diferencias con las versiones anteriores.

C-DOC Documentation Tools for C and C++ (V.5.0)

Software Blacksmiths, Inc.
6064 St. Ives Way
Mississauga, ON, CD L5N 4M1
905-858-4466
Soporte técnico: Utilizar número telefónico principal.
Soporte técnico BBS: 416-858-1916

Especificaciones

Precio de lista: US$199
Paquetes vendidos: 2,500
Fecha de liberación: 1992
Compatible con: PC-MS/DOS
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 256 KB
Soporte en sitio: Sí

Descripción

Herramientas de documentación automatizadas para programadores en C y C++. Incluye C-Metric, C-Call, C-CMT, C-List, y C-Ref. Incluye

diagramas de árbol gráficos de jerarquías llamado/llamante para grupos de programas. Analiza y documenta el uso de identificadores de parámetros globales y locales y muestra la estructura de flujo interna de funciones. Genera, inserta y actualiza bloques de comentarios en funciones. Calcula complejidad de vías ciclomáticas y cuenta código.

C-Metric

Software Blacksmiths, Inc.
6064 St. Ives Way
Mississauga, ON, CD L5N 4M1
905-858-4466
Soporte técnico: Utilizar número telefónico principal.
Soporte técnico BBS: 416-858-1916

Especificaciones

Precio de lista: US$59
Compatible con: PC-MS/DOS
Lenguaje fuente: C; C++

Descripción

Complemento de software documental para C y C++. Cuenta líneas con comentarios, código y oraciones de C y calcula la complejidad de flujo ciclomático para todas las funciones.

CA-Pan/Icm Configuration Manager

Computer Associates International, Inc.
One Computer Associates Plaza
Islandia, NY 11788-7011
"800-225-5224; 516-342-5224"
FAX: 516-342-5734
Soporte técnico: Soportado vía telefónica.

Especificaciones

Fecha de liberación: 1993
Compatible con: IBM/MVS/SP, MVS/XA, MVS/ESA, VSE/ESA, VSE
Memoria RAM mínima: 1 MB
Espacio en disco requerido: 3.5 MB
Lenguaje fuente: C
Atención a usuarios: Disponible.
Soporte en sitio: No

Descripción

Extenso sistema de manejo de configuración. Determina la dependencia entre los componentes de la aplicación y automátiza la recompilación, reensamblamiento y religamiento de las aplicaciones de software de sistemas.

CCC/Manager (Change and Configuration Control) (V.2.1)

Softool Corp.
340 S. Kellogg Ave.
Goleta, CA 93117
"800-723-0696; 805-683-5777"
FAX: 805-683-4105
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$495
Paquetes vendidos: 1,000
Fecha de liberación: 1992
Compatible con: Sun/SunOS, Solaris; IBM/MVS/SP, MVS/XA, MVS/ESA, AIX; DEC/ULTRIX, VMS, OSF/1; HP/HP-UX; Silicon Graphics
Memoria RAM mínima: 1 MB
Lenguaje fuente: FORTRAN; C
Atención a usuarios: Soportado vía telefónica
Soporte en sitio: Sí

Descripción

Provee un ambiente para cambios de software y manejo de configuración. Incluye auditoría de cambios y recuperación de versión para todos los componentes del ciclo de vida del software. Soporte automatizado para cualquier número de etapas del ciclo de vida.

CISLE/CDADL

Software Systems Design, Inc.
3627 Padua Ave.
Claremont, CA 91711
909-625-6147
FAX: 909-626-9667
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$1,000-$9,000
Paquetes vendidos: 10
Fecha de liberación: 1991

Compatible con: DEC VAX, MicroVAX, DECstation, VAXstation/VMS,ULTRIX; DG; PC-MS/DOS; Sun-3, 4, SPARCstation/SunOS; HP 9000,Apollo; Gould/SEL; Harris; Sequent; Alliant; Concurrent; Pyramid
Memoria RAM mínima: 4 MB
Espacio en disco requerido: 2 MB
Lenguaje fuente: C
Precio de renta: Renta 10% del precio de lista por mes.
Soporte en sitio: Sí

Descripción

Lenguaje de diseño de programas en C. Analiza tanto el pseudo código como el código ejecutable en C para detectar errores lógicos y produce reportes impresos para el entendimiento del diseño. Contiene cerca de 25 reportes que describen el diseño en cualquier fase del desarrollo incluyendo referencias cruzadas del uso de tipos, objetos y funciones, arboles de invocación y estructuras de declaraciones. Los usuarios pueden crear hasta 10 reportes de manejo de proyectos personalizados

COBOL ANALYST (V.3.0)

SEEC, Inc.
5001 Baum Blvd.
Pittsburgh, PA 15213
412-682-4991
FAX: 412-682-4958
Soporte técnico: 800-682-7332

Especificaciones

Precio de lista: US$1,200 por usuario (producto base); $2,500 por usuario (paquete completo)
Paquetes vendidos: 500
Fecha de liberación: 1994
Compatible con: Windows 3.X
Memoria RAM mínima: 8 MB
Espacio en disco requerido: 3.5 MB
Hardware/Software adicional requerido: Windows 3.X
Compatibilidad con redes: Novell
Lenguaje fuente: C
Soporte en sitio: Sí

Descripción

Ayuda al mantenimiento de aplicaciones en COBOL. Dirige el mantenimiento desde la perspectiva de una aplicación entera en lugar de

un programa a la vez. Almacena aplicaciones en un diccionario global de aplicaciones que incluye programas, "copybooks", IMS, DB2, y modelos de datos VSAM y MFS, así como definición de pantallas CICS. Utiliza ambiente para redes Windows GUI.
Presenta un formato de pantalla dividida que permite la visualización tanto de la definición como del uso de una entidad, e invoca la presentación en pantalla de su definición en otra ventana al presionar dos veces el botón del ratón sobre su nombre. Provee un modo configurable por el usuario de presentación selectiva de oraciones que permite la identificación y aislamiento de entidades. Implementa reglas para el uso de teclas incluyendo un doble clic en separadores que muestra líneas ocultas.
Identifica el flujo de ejecución del programa y puede presentar invocaciones condicionales en ramas. Las cartas gráficas de estructuras muestran a nivel programa y párrafo el trazado de jerarquía de control de rutinas llamadas y llamantes. El muestreo gráfico de modelos de datos IMS y DB2, estructuras de archivos VSAM y definiciones de registros de COBOL permiten el entendimiento visual de las jerarquías de datos, y relaciones físicas y lógicas. El Procesador de Sinónimos permite la detección de alias de elementos de datos y la asociación de estructuras de bases de datos, definición de archivos, y pantallas con sus correspondientes definiciones de datos en COBOL.

COBOL Structuring Facility (V.2.0)

IBM (International Business Machines)
Old Orchard Rd.
Armonk, NY 10504
"800-426-3333; 914-765-1900"
Ventas: 800-426-7695 (IBM PC Direct)
Soporte técnico: 800-237-5511
Soporte técnico BBS: 919-517-0001; 800-847-7211 (OS2)

Especificaciones

Compatible con: IBM/MVS/SP, VM/SP, VM/SP HPO
Precio de renta: Renta $596-$3,290 por mes.

Descripción

IBM No. 5688-045. Transforma programas en COBOL no estructurados en programas estructurados. Utiliza ciencias computacionales y técnicas de inteligencia artificial para desenredar programas no estructurados y transformarlos en programas estructurados.
Define y forza a seguir una norma para la práctica de programación estructurada. Permite a los usuarios transformar complejos programas no

estructurados en programas estructurados, genera reportes para guiar al programador en desarrollar nuevas mejoras en la estructura del programa, e identificar partes de programas aprovechables para su potencial reutilización. Llamando a la Ayuda de Conversión a Nivel Comando de COBOL y SICS, este programa puede asistir la migración de viejos programas de OS/VS COBOL a VS COBOL II o COBOL 370. Provee reestructura de código para OS/2, LAN, VM o COBOL basado en MVS. Incluye la capacidad para personalizar, salvar y restaurar opciones pre-salvadas que pueden ser compartidas para obligar normas de codificación. Se integra con SAA AD/Cycle Workstation Platform/2.

CodeBreaker (V.4.1)

McCabe & Associates, Inc.
5501 Twin Knolls Rd., Ste. 111, Twin Knolls Professional Park
Columbia, MD 21045
"800-638-6316; 410-995-1075"
FAX: 410-995-1528
Soporte técnico: 800-343-1891

Especificaciones

Fecha de liberación: 1993
Compatible con: Sun SPARCstation/SunOS; DEC VAX, MicroVAX/VMS,ULTRIX; PC-MS/DOS; HP 3000, 9000, Apollo; IBM RS/6000/AIX;Silicon Graphics; SCO UNIX"
Memoria RAM mínima: 640 KB (PC)
Espacio en disco requerido: 20 MB
Lenguaje fuente: Ada; C; FORTRAN; COBOL; PDL; Assembler; Pascal;C++
Precio de renta: Existen varios precios de renta disponibles.
Soporte en sitio: No

Descripción

Herramientas de ingeniería inversa que identifica los códigos redundantes y reutilizables. Compara el diseño de dos módulos así como el diseño de dos programas. Verifica que las versiones reestructuradas de módulo conserven la estructura de decisión al llamar la estructura del módulo original.

Codecheck/2 (V.5.0)

Abraxas Software, Inc.
5530 S.W. Kelly Ave.
Portland, OR 97201
"800-347-5214; 503-244-5253"

FAX: 503-244-8375
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$995
Paquetes vendidos: 1,500
Fecha de liberación: 1993
Compatible con: OS/2
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 1 MB
Lenguaje fuente: C; C++
Soporte en sitio: Sí

Descripción

Analizador de códigos fuente C/C++. Identifica códigos que no son portables a/de algún ambiente y que contengan estilo o uso inaceptablede programación (programación estructurada). Cuantifica la capacidad de mantenimiento de código con medidas definidas del usuario.

Coverage

Integrated Development Corp.
190 Main St., PO Box 592
Hampstead, NH 03841
"800-333-3429; 603-329-5522"
FAX: 603-329-4841
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Compatible con: PC-MS/DOS

Descripción

Registra cada llamada en tiempo de ejecución a líneas especificas de código fuente en Clipper. Permite al usuario eliminar código muerto, y ver que partes de código son las mas usadas y a través de las cuales hay mayor flujo de lógica, para moverlas a la parte alta de manera que el código corra mas rápido.

CrossView Debugger for Windows

BSO/Tasking
333 Elm St., Norfolk Place
Dedham, MA 02026-4530
"800-458-8276; 617-320-9400"

FAX: 617-320-9212

Especificaciones

Precio de lista: US$1,600
Paquetes vendidos: 500
Fecha de liberación: 1993
Compatible con: Windows 3.X
Memoria RAM mínima: 4 MB
Espacio en disco requerido: 6 MB
Hardware/Software adicional requerido: Windows 3.X
Compatibilidad con redes: Novell; Ethernet
Lenguaje fuente: C
Precio de renta: Existen varios precios de renta disponibles.
Soporte en sitio: Sí

Descripción

Supervisa y controla la ejecución de código fuente escrito en C o lenguaje ensamblador.
Incluye despliegue orientado a ventanas (window-oriented display), se permiten elaborar macros para comandos y puntos de quiebre (breakpoints).

DCD-PC

MARBLE Computer, Inc.
101 S. Spring St., PO Box 2088
Martinsburg, WV 25401-7088
"800-252-1400; 304-267-2941"
FAX: 304-267-3046
Soporte técnico: Soportado via telefónica.

Especificaciones

Paquetes vendidos: 10
Fecha de liberación: 1991
Compatible con: PC-MS/DOS
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 6 MB
Compatibilidad con redes: Novell
Lenguaje fuente: COBOL
Soporte en sitio: Sí

Descripción

Provee reportes comprehensivos, detallando la actividad de los programas y sistemas en COBOL. La Facilidad de Listados de Compilación Alternativa produce un listado fuente que documenta el uso

de elementos de datos y el flujo de la lógica a través del programa, permitiendo delinear y seguir el código en COBOL. Todos los datos correlacionados, información detallada para cada archivo, registro, dato simple y rama de la lógica es impresa a un lado de las líneas de código. La otra Facilidad de Reportes en COBOL provee análisis de múltiples programas dentro de un sistema. Once archivos de trabajo CASE y una interface con diccionario de datos están disponibles para proyectos involucrados con la reingeniería de programas o sistemas en COBOL.

DOC Professional (V.5.0)

Software Blacksmiths, Inc.
6064 St. Ives Way
Mississauga, ON, CD L5N 4M1
905-858-4466
Soporte técnico: Utilizar número telefónico principal.
Soporte técnico BBS: 416-858-1916

Especificaciones

Precio de lista: US$299
Fecha de liberación: 1992
Compatible con: Windows 3.X; PC-MS/DOS; OS/2
Hardware/Software adicional requerido: Presentation Manager;Windows 3.X

Descripción

Contiene todas las características de C-DOC y provee un modo de texto tipo DOS, Windows y OS/2 GUI's. Maneja mas de 150,000 líneas.

ETi Source Compare

ETI Solutions, Inc.
4755 Oceanside Blvd., Ste. 130
Oceanside, CA 92056
"800-231-5280; 619-631-5280"
FAX: 619-631-5285
Soporte técnico: Soportado via telefónica.

Especificaciones

Precio de lista: US$2,995-$6,995
Compatible con: IBM AS/400/OS/400
Soporte en sitio: Sí

Descripción

Analiza y reporta las diferencias y similitudes de código fuente y librerías de aplicaciones. Genera reportes de resultados.

For_Struct (V.2.0)

Cobalt Blue, Inc.
875 Old Roswell Rd., Ste. D-500
Roswell, GA 30076
404-518-1116
FAX: 404-640-1182
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$825 (DOS); $1,250-$1,475 (XENIX; UNIX;HP; DEC); $9,000 (Cray)"
Fecha de liberación: 1992
Compatible con: PC-MS/DOS; IBM RS/6000/AIX; Sun,SPARCstation/SunOS; DEC VAX/VMS; XENIX; HP/HP-UX; Cray
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 640 KB
Lenguaje fuente: C
Soporte en sitio: Sí

Descripción

Herramienta de estructuración para transformar código desordenado en FORTRAN en código limpio y mantenible.

Language Translators

Computer Task Group, Inc.
700 Delaware Ave.
Buffalo, NY 14209
"800-367-2687; 716-881-3000"
FAX: 716-888-3583
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Compatible con: IBM/MVS, DOS/VSE; OS/2; PC-MS/DOS
Soporte en sitio: No

Descripción

Programa de conversión de lenguajes de software. Incluye IBM Assembler a COBOL/COBOL II, PL/I a COBOL/COBOL II, RPG/RPG II a PL/I, RPG a COBOL/COBOL II, 1400 SPS/Autocoder a COBOL/COBOL II, Easycoder/Easytran a COBOL/COBOL II y COBOL a COBOL/COBOL II.

Legacy Workbench (V.5.1)

KnowledgeWare, Inc.
3340 Peachtree Rd., NE, Ste. 1100
Atlanta, GA 30326-1050
"800-444-8575; 404-231-8575"
FAX: 404-364-0267
Soporte técnico: 800-344-2662

Especificaciones

Precio de lista: US$100,000 (5-usuarios)
Fecha de liberación: 1994
Compatible con: IBM/MVS; OS/2
Memoria RAM mínima: 16 MB
Espacio en disco requerido: 10 MB
Lenguaje fuente: C; Assembler

Descripción

Paquete de herramientas, para mantenimiento y mejoramiento de sistemas existentes. La aplicación de estimación es una herramienta de apoyo de decisión que proporciona cinco categorías de medición, medidas y exposición gráfica de la calidad de una aplicación basada en la sintaxis, complejidad, estructura y errores. El usuario puede disponer gráficamente de los resultados de la aplicación de estimación usando Lotus 1-2-3 o Microsoft Excel. Su documentación de programas ayuda al programador a entender la complejidad de sus programas y a detectar potenciales zonas problemáticas. Produce listados de notas que explican lo que el programa está haciendo párrafo por párrafo. La reestructuración de programas transforma códigos "espagueti" o muertos en códigos estructurados y detalla la funcionalidad completa de programas legados. Su herramienta de mantenimiento gráfico presenta tres vistas sincronizadas que apoya el avance del programador y su comprensión de la arquitectura de programas. Las herramientas de análisis de sistemas abiertos basados en PC permite al analista tomar un inventario visual de una aplicación compleja completa o de un solo programa. Su representación gráfica facilita el análisis interactivo de todos los componentes del sistema reduciendo el tiempo requerido para realizar el mantenimiento.

Legacy Workbench for Windows (V.5.1)

KnowledgeWare, Inc.
3340 Peachtree Rd., NE, Ste. 1100
Atlanta, GA 30326-1050
"800-444-8575; 404-231-8575"

FAX: 404-364-0267
Soporte técnico: 800-344-2662

Especificaciones

Precio de lista: US$100,000 (5-usuario)
Fecha de liberación: 1994
Compatible con: Windows 3.X
Memoria RAM mínima: 16 MB
Espacio en disco requerido: 10 MB
Hardware/Software adicional requerido: Windows 3.X
Lenguaje fuente: C; Assembler

Descripción

Conjunto de herramientas que aceleran el mantenimiento, rediseño y migración de sistemas existentes basados en computadoras *mainframe*. Incluye aplicaciones de estimación, documentación de programas, reestructuración de programas que interactua con análisis de sistemas de circulación completa y mantenimiento gráfico vistas dinámicamente elaboradas de los patrones de estructura de programas de alto nivel, patrones de flujo de secuencias de ejecución y códigos fuente.

LISP to C Translator (Rel.3.2)

Chestnut Software, Inc.
2 Park Plaza, Ste. 205
Boston, MA 02116
617-542-9222
FAX: 617-542-9220
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Fecha de liberación: 1993
Compatible con: Sun-3, SPARCstation/SunOS; TI Explorer; PC-MS/DOS
Memoria RAM mínima: 300 KB

Descripción

Translada código LISP a código estándar ANSI C. Soporta el manejo de memoria común de LISP y su tabla de símbolos.

Logiscope

Logiscope Technologies, Inc.
3010 LBJ Frwy., Ste. 900
Dallas, TX 75234
214-241-6595

FAX: 214-241-6594

Especificaciones

Precio de lista: US$8,000 mínimo
Paquetes vendidos: 10,000
Fecha de liberación: 1985
Compatible con: HP 9000 Series 300, 700, Apollo Series 3000,4000/HP-UX; Sun-3, 4/SunOS; DEC VAXstation, VAX 8600/VMS;ULTRIX; IBM/VM, MVS/XA
Memoria RAM mínima: 10 MB
Espacio en disco requerido: 30 MB
Lenguaje fuente: Pascal
Soporte en sitio: Sí

Descripción

Provee verificación de calidad del código, análisis de chequeo y cubierta de chequeo a través de un juego de herramientas de ingeniería reversa. Provee análisis de complejidad, resultados gráficos, aceptación de criterio y documentación, revisa ejecución casual, y reestructuración de código.

MASM to C Translator

Micro-Processor Services, Inc.
92 Stone Hurst Lane
Dix Hills, NY 11746
516-499-4461
FAX: 516-499-4727
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$575-$875
Paquetes vendidos: 200
Fecha de liberación: 1992
Compatible con: PC-MS/DOS
Memoria RAM mínima: 2 MB
Espacio en disco requerido: 350 MB
Lenguaje fuente: MASM
Soporte en sitio: Sí

Descripción

Translada automáticamente código en MASM hacia C. Los errores encontrados en el código fuente en MASM se reportan en un archivo de errores y en la pantalla. Si no existen errores, el programa genera código fuente equivalente en C.

McCabe Slice Tool (V.4.1)

McCabe & Associates, Inc.
5501 Twin Knolls Rd., Ste. 111, Twin Knolls Professional Park
Columbia, MD 21045
"800-638-6316; 410-995-1075"
FAX: 410-995-1528
Soporte técnico: 800-343-1891

Especificaciones

Fecha de liberación: 1993
Compatible con: Sun SPARCstation/SunOS; DEC VAX, MicroVAX/VMS,ULTRIX; PC-MS/DOS; HP 3000, 9000, Apollo; IBM RS/6000/AIX;Silicon Graphics; SCO UNIX"
Memoria RAM mínima: 640 KB (PC)
Espacio en disco requerido: 20 MB
Precio de renta: Existen varios precios de renta disponibles.
Soporte en sitio: No

Descripción

Herramienta de visualización que rastrea datos a través de la arquitectura de sistemas. Es usado en la eliminación de errores, optimización del tamaño, necesidades de trazabilidad e identificación de códigos redundantes y reutulizables.

Microsoft Source Profiler (V.1.0)

Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
"800-426-9400; 206-882-8080"
Ventas: 800-MSPRESS
FAX: 206-883-8101
"Soporte técnico: 206-454-2030; 206-637-7098 (Windows)"
Soporte técnico BBS: 206-936-6735

Especificaciones

Precio de lista: US$79
Fecha de liberación: 1991
Compatible con: PC-MS/DOS; OS/2
Memoria RAM mínima: 150 KB
Espacio en disco requerido: 2 MB

Descripción

Genera información estadística para programas escritos en DOS, OS/2 y Windows. El delineador (profiler) trabaja con cualquier tamaño de programa. El delineador opera con código generado en cualquier compilador de Microsoft que proporciones información CodeView. Ejecuta código Microsoft. Genera estadísticas del número de veces que una porción de código es ejecutada, el tiempo de ejecución de una lo varias líneas. Descubre que parte del código es la que se ejecuta un número de veces mayor. Identifica que parte del código puede ser optimizada en tiempo al usar un algoritmo mas efectivo, más apropiado, o un lenguaje mucho más rápido como C o ensamblador.

Microsoft Source Profiler for Windows (V.1.0)

Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
"800-426-9400; 206-882-8080"
Ventas: 800-MSPRESS
FAX: 206-883-8101
"Soporte técnico: 206-454-2030; 206-637-7098 (Windows)"
Soporte técnico BBS: 206-936-6735

Especificaciones

Precio de lista: US$79
Fecha de liberación: 1991
Compatible con: Windows 3.X
Memoria RAM mínima: 150 KB
Espacio en disco requerido: 2 MB
Hardware/Software adicional requerido: Windows 3.X

Descripción

Ayuda a programadores a provar y optimizar aplicaciones escritas en los compiladores de Microsoft como: C, FORTRAN, BASIC, Macro Assembler, COBOL, Microsoft Pascal, Microsoft QuickBASIC y QuickC con QuickAssembler. Indica cuando una función es llamada y el tiempo de ejecución de ésta.

MultiScope Debuggers for Windows (V.2.01)

Symantec Corp.
10201 Torre Ave.
Cupertino, CA 95014-2132
"800-441-7234; 408-253-9600"
FAX: 408-253-3968
Soporte técnico: 415-892-1424
Soporte técnico BBS: 503-484-6669

Especificaciones

Precio de lista: US$379
Fecha de liberación: 1992
Compatible con: Windows 3.X
Memoria RAM mínima: 2 MB
Espacio en disco requerido: 4 MB
Hardware/Software adicional requerido: Windows 3.X
Compatibilidad con redes: NetBIOS
Soporte en sitio: Sí

Descripción

Prueba 100% de las aplicaciones escritas para Windows. Incluye interfaces en modo gráfico y modo caracter. Soporta pruebas en uno o dos monitores, pruebas remotas, pruebas a archivos DLL, visualización de datos y evaluación de expresiones.

Non-IBM COBOL to IBM COBOL/COBOL II Translator

Computer Task Group, Inc.
700 Delaware Ave.
Buffalo, NY 14209
"800-367-2687; 716-881-3000"
FAX: 716-888-3583
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Fecha de liberación: 1989
Compatible con: IBM/DOS/VSE, VSE/ESA, MVS; OS/2; PC-MS/DOS
Memoria RAM mínima: 256 KB
Lenguaje fuente: COBOL
Soporte en sitio: No

Descripción

Código no compatible con IBM lo convierte a código COBOL/COBOL II. Convierte programas de una versión o nivel de COBOL a otra. Convierte código en COBOL de un sistema operativo a otro y de una compañía a otra. Genera una hoja de cálculo con las especificaciones derivadas de la conversión.

PC/MC/UX/VX-Metric (V.4.0)

Set Laboratories, Inc.
PO Box 868
Mulino, OR 97042
503-829-7123
FAX: 503-829-7220
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$199 (Mac); $399 (DOS); $525 (UNIX);$650 (VAX)
Paquetes vendidos: 5,000
Fecha de liberación: 1988
Compatible con: PC-MS/DOS; Apple Macintosh; SunSPARCstation/SunOS; IBM RS/6000/AIX; SCO UNIX; DEC VAX/VMS;HP/HP-UX
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 2 MB
Lenguaje fuente: C; Pascal
Precio del código fuente: $8,500-$14,000
Soporte en sitio: Sí

Descripción

Analizador de código fuente. Analiza el código fuente de programas en C, C++, COBOL, FORTRAN, Pascal, Modula-2, Basic, Ada y dBase y produce medidas para determinar la complejidad del código. Provee una característica distintiva de referencia cruzada que enlista en cada una variable que se usa en cada función o procedimiento.
Provee un sistema de análisis integrado que ofrece una variedad de reportes predefinidos, gráficas y un reporte de escritura personal del usuario. Identifica sobre códigos complejos las funciones y procedimientos que exceden niveles de complejidad especificados. La complejidad de una pieza de software se puede comparar antes y después de realizar un cambio.
Provee un conjunto selecto de medidas complejas incluyendo Medidas científicas de Software (esfuerzo, volumen, tiempo estimado y defectos), complejidad ciclomática, medidas estándar IEEE (informe de la fuente física, informe de la fuente lógica, informes no ejecutables, directivas de

compilación, líneas vacías y comentarios). Cada medida es compilada para todas las funciones y/o procedimientos así como para el sistema como uno solo. Las medidas se disponen en un formato tabular en el reporte de complejidad y las funciones y/o procedimientos que exceden estándares predefinidos de complejidad en un reporte excepcional. La información puede ser almacenada en un archivo para importar hacia hojas de cálculo. Las facilidades de escritura de reportes provee más de 30 reportes predefinidos, incluyendo una variedad de presentaciones gráficas. El escritor de reportes del usuario permite al usuario crear sus propios medidas usando campos calculados. Los gráficos incluyen histográmas, gráficas pie, barras y diagramas Kiviat.

PL/1 to C Translator

Micro-Processor Services, Inc.
92 Stone Hurst Lane
Dix Hills, NY 11746
516-499-4461
FAX: 516-499-4727
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$575 - $1,875
Paquetes vendidos: 100
Fecha de liberación: 1991
Compatible con: PC-MS/DOS
Memoria RAM mínima: 2 MB
Espacio en disco requerido: 400 KB
Lenguaje fuente: PL/1
Soporte en sitio: Sí

Descripción

Permite la conversión directa de programas en PL/1 a código en C. Soporta las sintaxis más comunes. Verifica la sintaxis en PL/1, los errores de banderas, y genera un listado con las operaciones realizadas junto con el archivo final en C. El sistema translada 11 diferentes dialectos de PL/1, incluyendo IBM, DRI, Stratus y Prim.

PL/1 to COBOL/COBOL II Translator

Computer Task Group, Inc.
700 Delaware Ave.
Buffalo, NY 14209
"800-367-2687; 716-881-3000"
FAX: 716-888-3583
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Fecha de liberación: 1989
Compatible con: IBM/DOS/VSE, VSE/ESA, MVS; OS/2; PC-MS/DOS
Memoria RAM mínima: 256 KB
Lenguaje fuente: COBOL
Soporte en sitio: No

Descripción

Convierte PL/I a COBOL/COBOL II. Analiza el programa en PL/I antes de generar la salida en COBOL. Control por medio de tablas. Genera listados de las operaciones realizadas, indica las instrucciones en PL/I que fueron cambiadas a instrucciones en COBOL. Produce un diagnostico de ayuda.

PL/M to C Translator

Micro-Processor Services, Inc.
92 Stone Hurst Lane
Dix Hills, NY 11746
516-499-4461
FAX: 516-499-4727
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$575-$875
Paquetes vendidos: 600
Fecha de liberación: 1986
Compatible con: PC-MS/DOS
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 200 KB
Lenguaje fuente: PL/M
Soporte en sitio: Sí

Descripción

Permite la conversión directa de programas en PL/I a código en C. Soporta las sintaxis mas comunes. Verifica la sintaxis en PL/I, los

errores de banderas, y genera un listado con las operaciones realizadas junto con el archivo final en C. El sistema soporta PL/M 51, 80, 86, 286, 96 y 386. El programa funciona en modo protegido de DOS.

Pro_Struct

Cobalt Blue, Inc.
875 Old Roswell Rd., Ste. D-500
Roswell, GA 30076
404-518-1116
FAX: 404-640-1182
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$375
Fecha de liberación: 1990
Compatible con: PC-MS/DOS
Memoria RAM mínima: 640 KB
Espacio en disco requerido: 640 KB
Lenguaje fuente: C
Soporte en sitio: No

Descripción

Herramienta de reestructuración para FORTRAN la cual transforma y reestructura código fuente de versiones viejas de FORTRAN al FORTRAN-77 estándar. Preserva la lógica original del código, elimina código muerto y ofrece varias opciones de reformateo.

Product Configuration Management System (V.4.0.5)

Digital Equipment Corp. (DEC)
146 Main St.
Maynard, MA 01754-2571
"800-344-4825; 508-493-5111"
Ventas: 800-722-9332 (DEC Direct/Digital PC)
FAX: 508-493-8780
Soporte técnico: 800-354-9000
Soporte técnico BBS: 508-496-8800

Especificaciones

Fecha de liberación: 1993
Compatible con: DEC VAX, MicroVAX, VAXstation, VAXserver,DECstation/OpenVMS, ULTRIX; Sun SPARCstation/SunOS; HP 9000Series 700/HP-UX
Hardware/Software adicional requerido: Oracle

Soporte en sitio: No
Combina IPSE con control de manejo de configuración. Soporta diferentes tipos de software y hardware. Utiliza bases de datos relacionales para coordinar información de todas las personas involucradas en un proyecto incluyendo diseñadores y desarrolladores.

Profile Code (V.3.20)

Implant Sciences Corp.
107 Audubon Rd., Ste. 5
Wakefield, MA 01880-1246
617-246-0700
FAX: 617-246-1167
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$995
Paquetes vendidos: 150
Fecha de liberación: 1993
Compatible con: PC-MS/DOS
Memoria RAM mínima: 640 KB
Lenguaje fuente: BASIC
Soporte en sitio: Sí

Descripción

Proporciona información estadistica del tiempo de ejecución de un programa o unas líneas seleccionadas en particular, incluye la capacidad de delinear diferentes tipos de código, los resultados pueden ser escritos en varios formatos de archivos disponibles.

PVCS Configuration Builder (V.5.1)

Intersolv, Inc.
3200 Tower Oaks Blvd.
Rockville, MD 20852
"800-547-4000; 301-230-3200"
Ventas: 800-777-8858
FAX: 301-231-7813
Soporte técnico: 800-443-1601
Soporte técnico BBS: 301-816-9803

Especificaciones

Precio de lista: US$299
Paquetes vendidos: 20,000
Fecha de liberación: 1993

Compatible con: OS/2; PC-MS/DOS; IBM RS/6000/AIX; Sun/SunOS, Solaris; SCO UNIX; 386/ix; QNX; HP 9000/HP-UX; OSF/1,Memoria RAM mínima: 256 KB (DOS); 4 MB (OS/2)
Espacio en disco requerido: 250 KB
Compatibilidad con redes: Novell; LAN Manager; LAN Server; NetBIOS
Lenguaje fuente: C
Soporte en sitio: No

Descripción

Documenta las modificaciones y las necesidades para reconstruir un programa/sistema. Puede ser usado en ambiente de programación multilenguaje. Estandariza el ambiente de trabajo en la modificación del pograma. Almacena y repite las operaciones comunes.

PVCS for Software Configuration Management

Intersolv, Inc.
3200 Tower Oaks Blvd.
Rockville, MD 20852
"800-547-4000; 301-230-3200"
Ventas: 800-777-8858
FAX: 301-231-7813
Soporte técnico: 800-443-1601
Soporte técnico BBS: 301-816-9803

Especificaciones

Precio de lista: US$200-$1,500
Paquetes vendidos: 140,000
Fecha de liberación: 1993
Compatible con: OS/2; PC-MS/DOS; IBM RS/6000/AIX; Sun/SunOS, Solaris; SCO UNIX; 386/ix; HP/HP-UX
Memoria RAM mínima: 256 KB
Hardware/Software adicional requerido: Presentation Manager
Compatibilidad con redes: Novell; 3Com; NetBIOS; Token-Ring; LAN Manager
Lenguaje fuente: C
Soporte en sitio: No

Descripción

Soporta ambientes para desarrolladores que trabajan en workstations, en un mainframe o en un equipo LAN-basados en mainframe y PC. Opera através de PC-DOS, OS/2, plataformas de LAN's y UNIX. Con PVCS Production Gateway se sincroniza el desarrollo en workstation.

PVCS Version Manager (V.5.1)

Intersolv, Inc.
3200 Tower Oaks Blvd.
Rockville, MD 20852
"800-547-4000; 301-230-3200"
Ventas: 800-777-8858
FAX: 301-231-7813
Soporte técnico: 800-443-1601
Soporte técnico BBS: 301-816-9803

Especificaciones

Precio de lista: US$499 (DOS, OS/2); $599 (UNIX)
Paquetes vendidos: 140,000
Fecha de liberación: 1993
Compatible con: OS/2; PC-MS/DOS; IBM RS/6000/AIX; Sun/SunOS,Solaris; SCO UNIX; 386/ix; QNX; Open Software OSF/1
Memoria RAM mínima requerida: 300 KB (DOS); 4 MB (OS/2)
Espacio en disco requerido: 4.8 MB (DOS); 2.5 MB (OS/2)
Compatibilidad con redes: Novell; NetBIOS; LAN Manager; LAN Server
Lenguaje fuente: C
Soporte en sitio: Sí

Descripción

Ayuda a los usuarios a manejar las revisones de cualquier código fuente, ejecutable, gráfica o cualquier otro texto o archivo binario. Ayuda al usuario a seguir la pista de las versiones del sistema. Asigna revisiones específicas de los archivos individuales

Q/Auditor COBOL (V.2.4)

Eden Systems Corp.
9302 N. Meridian St., Ste. 350
Indianapolis, IN 46260-1820
"800-288-9510; 317-848-9600"
FAX: 317-843-2271
Soporte técnico: Soportado via telefónica.

Especificaciones

Precio de lista: US$9,995 mínimo
Fecha de liberación: 1994
Compatible con: IBM/MVS; PC-MS/DOS; OS/2; Tandem/Guardian
Memoria RAM mínima: 65 KB (IBM; Tandem); 2 MB (DOS); 8 MB (OS/2)
Espacio en disco requerido: 1-5 MB
Lenguaje fuente: COBOL

Precio de renta: Renta 7% del precio de lista por mes.
Soporte en sitio: Sí

Descripción

Herramienta para mantener la calidad para programas en COBOL. Las compañías seleccionan su factor critico de éxito para alta calidad de programas en COBOL dentro de un inventario de 500 elementos de medición. Mide un rango de condiciones de programación abarcando desde las normas establecidas de programación mas simples hasta la métrica de volumen de Halstead, Complejidad escencial de McCabe y complejidad ciclomática. Encuentra las estructuras no validas (GO TO'S hacia etiquetas inexistentes), código colocado erróneamente como comentario, párrafos no referenciados, aislamiento de entradas o salidas, y anidacion excesivamente compleja.

Q/Auditor COBOL for Windows (V.2.4)

Eden Systems Corp.
9302 N. Meridian St., Ste. 350
Indianapolis, IN 46260-1820
"800-288-9510; 317-848-9600"
FAX: 317-843-2271
Soporte técnico: Soportado via telefónica.

Especificaciones

Precio de lista: US$9,995 mínimo
Fecha de liberación: 1994
Compatible con: Windows 3.X
Memoria RAM mínima: 8 MB
Espacio en disco requerido: 1-5 MB
Hardware/Software adicional requerido: Windows 3.X
Lenguaje fuente: COBOL
Precio de renta: Renta 7% del precio de lista por mes.
Soporte en sitio: Sí

Descripción

Herramienta para mantener la calidad para programas en COBOL. Las compañías seleccionan su factor critico de éxito para alta calidad de programas en COBOL dentro de un inventario de 500 elementos de medición. Mide un rango de condiciones de programación abarcando desde las normas establecidas de programación mas simples hasta la métrica de volumen de Halstead, Complejidad escencial de McCabe y complejidad ciclomática. Encuentra las estructuras no validas (GO TO'S hacia etiquetas inexistentes), código colocado erróneamente como

comentario, párrafos no referenciados, aislamiento de entradas o salidas, y anidacion excesivamente compleja.

QA FORTRAN (V.6.0)

ASTA, Inc.
1 Chestnut St., Ste. 205/206
Nashua, NH 03060
"800-350-2782; 603-889-2230"
FAX: 603-881-3740
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$6,000 mínimo
Paquetes vendidos: 250
Fecha de liberación: 1992
Compatible con: Sun-3, 4, SPARCstation/SunOS, DEC/VMS, ULTRIX; HP9000 series 300, 800, 700/HP-UX; IBM RS/6000/AIX; Cray/UNICOS
Memoria RAM mínima: 2 MB
Espacio en disco requerido: 10 MB
Lenguaje fuente: C
Soporte en sitio: Sí

Descripción

Sistema completo que proporciona pruebas de aseguramiento de calidad extensiva para aplicaciones FORTRAN. Examina el código fuente. Compara la calidad del código con métricas de calidad compiladas de entre más de 10 millones de líneas de FORTRAN. Provee características (métricas) de calidad incluyendo cuenta nodos y reparación, complejidad ciclomática, intervalos de McCabe's y extensión de cerrado. Despliega medidas de calidad del software gráficamente en una escala porcentual o en forma numérica cruda.
Compila complejidades jerárquicas y estructurales de un paquete. Estima tiempos de traslado y desarrollo. Abandera variables no inicializadas o innecesariamente inicializadas. Indica las características distintivas de conformación ANSI.
Transpone caracteres en nombres variables y aquellos con ambigüedades l-l y O-O. Examina arriba de 30,000 líneas por minuto en estaciones de trabajo RISC. Provee análisis global de uso common-block y abandera globalmente variables de bloque común inusuales o no inicializadas. Realiza análisis fan-in/fan-out para identificar rutinas llave estructurales en paquetes extensos. Incluye una herramienta que suma las estructuras y utilerías de expansión/contracción de archivos. Permite al usuario incorporar estándares de programación. El sistema de mensajes provee

10 niveles de advertencia y más de 500 mensajes. Verifica la compilación de interfaces con rutinas de librería del usuario.

Software Configuration Management Supervisor/2 (SCMS/2)

Progressive Software Technologies, Inc.
PO Box 620339
Littleton, CO 80162-0339
303-932-2051
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$595
Fecha de liberación: 1992
Compatible con: OS/2
Memoria RAM mínima: 8 MB
Espacio en disco requerido: 4 MB
Compatibilidad con redes: TOPS
Lenguaje fuente: C
Atención a usuarios: 30-day free Soportado vía telefónica
Soporte en sitio: Sí

Descripción

Ayuda en el manejo de configuración en el control de cambios de software. Se puede utilizar desde desarrollos iniciales de baja escala hasta desarrollos completos de alta escala. Proporciona un inventario de software y hardware. Permite rastrear las entradas de software que requieren cambios vía propuestas de ingeniería detallando costo, programa e impacto técnico de un cambio propuesto y solicitud para desviaciones/suspensiones.

Software Profile Management Facility

IBM (International Business Machines)
Old Orchard Rd.
Armonk, NY 10504
"800-426-3333; 914-765-1900"
Ventas: 800-426-7695 (IBM PC Direct)
Soporte técnico: 800-237-5511
Soporte técnico BBS: 919-517-0001; 800-847-7211 (OS2)

Especificaciones

Precio de lista: US$900
Compatible con: IBM 370/MVS, MVS/XA, MVS/SP, VM/SP, VM/XA;PC-MS/DOS; OS/2"

Hardware/Software adicional requerido: IBM's NetView Distribution
Compatibilidad con redes: IBM NetView

Descripción

IBM no. 5799-EPA. provee la funcionalidad que extiende a IBM's NetView Distribution Manager.
Utiliza programas generados por estaciones de trabajo dependiendo de la plataforma y puede reportar los resultados a bases de datos de DB2. Proporciona selección dinámica (puede trabajar en varias estaciones distintas a la vez) de estaciones de trabajo. Automáticamente construye NetView DM

Software Refinery (V.4.0)

Reasoning Systems
3260 Hillview Ave.
Palo Alto, CA 94304
415-494-6201
FAX: 415-494-8053
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$19,600
Paquetes vendidos: 300
Fecha de liberación: 1994
Compatible con: Sun-3, 4, SPARCstation/SunOS, Solaris; IBM,RS/6000/AIX; HP 9000 series 700"
Memoria RAM mínima: 28 MB
Espacio en disco requerido: 70 MB
Hardware/Software adicional requerido: X-Windows; OpenWindows
Lenguaje fuente: Refine
Soporte en sitio: No

Descripción

Provee una base de datos de orientación al objeto para modelar el código fuente y la información derivada.
Incluye un patrón para encontrar todas las secciones de código que satisfagan una propiedad particular de semántica o de sintaxis
Apoya al usuario con una capacidad de transformación de programas para hacer sistemática y automáticamente los cambios definidos por el usuario en un completo sistema de programas. Incluye un generador de impresión y apoyo para la construcción de frases para herramientas CASE.

Utiliza una herramienta basada en el sistema X Window para construir GUI'S. Puede usarse para analizar y redocumentar viejos sistemas, convertir programas a nuevos lenguajes, transformar bases de datos y migrar a otras plataformas de equipo, compara códigos fuente con estándares de codificación y genera situaciones de revisión desde el código fuente.
Incluye DIALECT para producir lenguajes de programación de impresión o parsers, REFINE para analizar y modificar la representación de el objeto base del programa, INTERVISTA para construir los requerimientos de interfaces del usuario en herramientas de reingeniería de programas y WORKBENCH que provee un grupo de componentes reusables de lenguaje neutral para la construcción de herramientas de programación.

Source Program Compare

MacKinney Systems, Inc.
2740 S. Glenstone, Ste. 103
Springfield, MO 65804
417-882-8012
FAX: 417-882-7569
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$50 (por PC); $495 (Mainframe, AS/400)
Paquetes vendidos: 360
Fecha de liberación: 1994
Compatible con: IBM mainframe, AS/400/OS, DOS/VSE, MVS, OS/400;PC-MS/DOS
Memoria RAM mínima: 96 KB
Lenguaje fuente: COBOL
Soporte en sitio: Sí

Descripción

Compara dos versiones de programas o JCL's e imprime las diferencias. Facilidad de imprimir solamente las diferencias, los cambios de contexto o el programa entero, señalando los cambios. Permite al usuario imprimir un reporte de la comparación efectuada.

STW/Advisor

Software Research, Inc. (CA)
625 Third St.
San Francisco, CA 94107-1997
"800-942-SOFT; 415-957-1441"
FAX: 415-957-0730

Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$5,000
Paquetes vendidos: 1,500
Fecha de liberación: 1993
Compatible con: Sun SPARCstation/SunOS, Solaris; IBM RS/6000/AIX;Silicon Graphics Iris/IRIX; 386/ix; AT&T UNIX System V; SCO;UNIX; SCO Open Desktop; HP 9000/HP-UX; DEC DECstation,Alpha/ULTRIX; PC-MS/DOS.
Memoria RAM mínima: 8 MB (UNIX)
Espacio en disco requerido: 4-10 MB (UNIX)
Hardware/Software adicional requerido: SCO Open Desktop
Lenguaje fuente: C
Precio de renta: Existen varios precios de renta disponibles.
Soporte en sitio: Sí

Descripción

Como producto aislado o parte de el software de revisión de trabajos, estas herramientas de programa establecen y miden la calidad con mediciones, analiza el código fuente en busca de anomalías con análisis estáticos y genera automáticamente una variedad de exámenes de datos. Usa 17 métricas para medir los datos del programa, la complejidad lógica y de extensión para añadirla en la toma de decisiones de una manera objetiva y cuantitativa. Incluye METRIC, que analiza la fuente C, C++, Ada ó FORTRAN y calcula las métricas científicas de software Halstead para medir la complejidad de datos, las métricas de complejidad ciclomática estiman la complejidad lógica, y métricas de tamaño básicas, como el número de líneas, comentarios y declaraciones ejecutables. Las métricas son reportadas en un formato tabular y en listas gráficas Kiviat.
El STATIC provee información detallada de semántica y sintaxis para programas en C. Procesa un programa entero o archivos individuales y genera un reporte que indica cualquier anomalía en el código que puedan tener errores.
El resultado del análisis se presenta en un reporte. Incluye TDGEN, un sistema generador de revisión de datos que crea exámenes adicionales para substituir cualquier valor de datos secuenciales o de forma variable en un guión de exámenes existente.

STW/Advisor for Windows

Software Research, Inc. (CA)
625 Third St.

San Francisco, CA 94107-1997
"800-942-SOFT; 415-957-1441"
FAX: 415-957-0730
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$5,000
Paquetes vendidos: 1,500
Fecha de liberación: 1993
Compatible con: Windows 3.X
Memoria RAM mínima: 8 MB (UNIX)
Espacio en disco requerido: 4-10 MB (UNIX)
Hardware/Software adicional requerido: Windows 3.X; SCO Open Desktop
Lenguaje fuente: C
Precio de renta: Existen varios precios de renta disponibles.
Soporte en sitio: Sí

Descripción

Conjunto de herramientas para pruebas de software. Parte de un conjunto integrado de productos de prueba de programas. Provee funciones esenciales y básicas para el desarrollo del cielo de vida de programas críticos. Consiste en TDGEN independiente, Static y Metric.

Turbo Debugger and Tools (V.3.0)

Borland International, Inc.
100 Borland Way
Scotts Valley, CA 95066-3249
"800-233-2444; 408-431-1000"
Ventas: 800-331-0877
FAX: 408-431-4122
Soporte técnico: 408-461-9155
Soporte técnico BBS: 408-431-5096

Especificaciones

Precio de lista: US$150
Fecha de liberación: 1993
Compatible con: PC-MS/DOS
Memoria RAM mínima: 384 KB

Descripción

El conjunto de productos consiste en una actualización de las herramientas de programación: Turbo Debugger, Turbo Assembler y Turbo Profiler. Turbo Debugger contiene como característica adicional

el poder ejecutar el código en forma reversa, es decir, puede eliminar el efecto de la última instrucción y posicionarse en la instrucción anterior. Ayuda a encontrar errores en la programación.
Turbo Profiler indica como un programa puede ser mejorado con respecto a su tiempo de ejecución. Proporciona estadísticas críticas como son: el numero de veces que una rutina es llamada, el tiempo de ejecución de una rutina en especial, que archivos son utilizados por una rutina y que tan eficientes son los archivos y programas temporales (overlays).
Turbo Assembler utiliza NOP. Ensamblador de "múltiples pasadas" que genera código optimizado para los microprocesadores 8086, 80286, 80386 y 80486. Compatible con Microsoft Assembler 4.0, 5.0 y 5.1.

VIA/Insight2

Viasoft, Inc.
3033 North 44th St.
Phoenix, AZ 85018
"800-525-7775; 602-952-0050"
FAX: 602-840-4068
Soporte técnico: 800-622-6682

Especificaciones

Fecha de liberación: 1994
Compatible con: OS/2

Descripción

Sistema interactivo que automatiza el proceso de examinar programas escritos en COBOL.
Permite al usuario visualizar la estructura del programa, crear múltiples *vistas*, permite delinear la lógica de un programa y revela anomalias en los programas.

VIA/Recap

Viasoft, Inc.
3033 North 44th St.
Phoenix, AZ 85018
"800-525-7775; 602-952-0050"
FAX: 602-840-4068
Soporte técnico: 800-622-6682

Especificaciones

Fecha de liberación: 1994
Compatible con: IBM/MVS/XA, MVS/ESA, TSP/ISPF

Lenguaje fuente: Pascal
Soporte en sitio: Sí

Descripción

Provee manejadores IS con información cuantificable concerniente a la calidad, maleabilidad y evalúo de aplicaciones COBOL complejas. Este programa hace una análisis costo-beneficio, establece prioridades para el desarrollo, fija planes para el mantenimiento, renueva y migra a nuevas plataformas, realiza auditorías en la calidad del programa. Produce un conjunto de métricas para los estándares industriales e información resumida en reportes enlistados. Las métricas de estándares industriales que se generan incluyen puntos de función, volumen de programas científicos, complejidad esencial, complejidad ciclomática, y cuenta nodos.

Visual Debugger

BYTWARE, Inc.
621 Todd Court, PO Box 1112
Grass Valley, CA 95945-1112
"800-932-5557; 916-273-4595"
FAX: 916-273-4593

Especificaciones

Precio de lista: US$950-$3,400
Fecha de liberación: 1994
Compatible con: IBM AS/400

Descripción

Debugger diseñado específicamente para mantenimiento de programas de AS/400.Soporta trabajos remotos y trabajos en lote, proporciona al programador la posibilidad de probar más de 10 programas simultáneamente.

C Set++ Debugger

IBM (International Business Machines)
Old Orchard Rd.
Armonk, NY 10504
"800-426-3333; 914-765-1900"
Ventas: 800-426-7695 (IBM PC Direct)
Soporte técnico: 800-237-5511
Soporte técnico BBS: 919-517-0001; 800-847-7211 (OS2)

Especificaciones

Compatible con: OS/2
Hardware/Software adicional requerido: Presentation Manager
Soporte en sitio: No

Descripción

Permite detectar y diagnosticar errores en código desarrollado con compiladores de C/C++ de 32-bits. Permite al usuario provar Presentation Manager, prueba código *sencillo* y *multithread*.
Contiene una ventana que muestra los trabajos adicionales que el programa puede ejecutar cuando se encuentra trabajando en *multithread*.

Xinotech COBOL Composer (V.2.0)

Xinotech Research, Inc.
1313 Fifth St., SE, Technology Center, Ste. 213
Minneapolis, MN 55414
"800-379-3844; 612-379-3844"
FAX: 612-379-3833
Soporte técnico: Soportado via telefónica.

Especificaciones

Precio de lista: US$495 (OS/2); $795 (UNIX)
Fecha de liberación: 1993
Compatible con: OS/2; Sun SPARCstation/SunOS
Memoria RAM mínima: 8 MB
Espacio en disco requerido: 5 MB
Hardware/Software adicional requerido: Motif
Soporte en sitio: No

Descripción

Herramienta CASE para ayudar al desarrollo, mantenimiento, análisis, y estadarización de programas escritos en COBOL. Previene errores de sintaxis, genera reportes de la estructura del programa.

XperCASE (V.3.5)

Boston Technical Distribution Corp.
3 Center Plaza, Ste. 440, PO Box 1340
Boston, MA 02108
617-248-8989
FAX: 617-248-8986
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Precio de lista: US$495-$1,994 (por módulo); $4,495 (integrado)
Fecha de liberación: 1994
Compatible con: Windows 3.X
Hardware/Software adicional requerido: Windows 3.X

Descripción

Consiste de XperCASE (SD) System Design, XperCASE (SP) Structured Programming, XperCASE (C), XperCASE (COB) y XperCASE (SPX).
Los programas usan técnicas de programación estructurada. XpercASE (SD) asegura el análisis y diseño de sistemas de proceso de información. Permite al usuario analizar y describir el estado actual y el proyectado en el desarrollo de un sistema de programas al crear editar e imprimir planes de comunicación. Estos proveen una representación gráfica de un sistema de proceso de información.
XperCASE (SP) Structured Programming asegura la ingeniería de software y permite a los usuarios crear, editar e imprimir diagramas estructurados Nassi-Shneiderman, así como crear un ambiente de ingeniería integrado que permite enlazar diagramas de estructuras con otros documentos, transferencia de archivos entre PC y Mainframes, y activación y enlace con cualesquiera otras herramientas.
XperCASE (C) y XperCASE (COB) permite a los usuarios diseñar nuevos programas y mantenerlos, documentar y desarrollar programas existentes en C o COBOL. El usuario puede analizar archivos de código fuente y crear diagramas de estructuras para ellos. Los errores estructurales en un archivo fuente existente son identificados y se da oportunidad a corregirlos. Los archivos existentes pueden ser objeto de reingeniería y documentación. (C) y (COB) pueden llamar un compilador directamente para convertir código fuente creado a partir de un diagrama de estructuras.
XperCASE (SPX) provee ingeniería y reingeniería de software. Permite a los usuarios crear, editar e imprimir programas en virtualmente cualquier lenguaje de programación incluyendo Ada, Basic, Clipper, dBase IV, Fortran, Modula 2, Natural y PL/I en forma de diagramas estructurados de Nassi-Shneiderman. Genera código fuente en cualquier lenguaje de programación que pueda ser configurado a partir de un diagrama de estructuras. Incluye archivos de configuración para lenguajes de programación comunes y sus dialectos. Los archivos de configuración pueden ser adaptados a requerimientos individuales o usados como base para otros lenguajes de programación.

XPONENT Configuration Management (XCM) for Windows

Software Solutions Unlimited, Inc.

5600 Wyoming Blvd., NE, One Sycamore Plaza, Ste. 10
Albuquerque, NM 87109
505-828-9000
FAX: 505-823-1841
Soporte técnico: Utilizar número telefónico principal.

Especificaciones

Fecha de liberación: 1994
Compatible con: Windows 3.X
Hardware/Software adicional requerido: Windows 3.X

Descripción

Trabaja con otros productos XPONENT para proporcionar ambientes de desarrollo integrados. Proporciona para el control de código fuente que usuario hizo cambios en XPONENT. Da seguimiento a cada una de las revisiones.

BIBLIOGRAFÍA

(BO92) Borland International, *Turbo Profiler Ver. 2.0 User's Guide*, 1992.

(ES88) Esquer Duarte Enrique y Martínez Ramos Mario, *Métodos para el desarrollo de software y su aplicación a la solución de problemas de ingeniería civil*, Tesis inédita para obtener el título de ingeniero civil, Universidad Autonoma de México,1988.

(FA87) Richard Fairley, *Ingeniería de Software*, McWraw-Hill, 1987.

(HE91) Ricardo Hernández Jiménez, *Administración de centros de cómputo*, Trillas, 1991.

(MA93) Steve Maguire, *Código sin errores*, McGraw Hill, 1993.

(MC92) Carma McClure, *The Three R's of Software Automation*, Prentice Hall, 1992.

(MU88) Robert G. Murdick , John C. Munson, *Sistemas de Información Administrativa*, Segunda Edición, Prentice Hall, 1988.

(SC81) G. Michael Schneider, Steven C. Bruell, *Advanced Programming and problem solving with Pascal*, John Wiley & Sons, 1981.

(SC83) Martin L. Shooman, *Software Engineering*, McGraw Hill, 1983.

Hemerografía

(AL94) Chuck Allison, *Control structures*, C Users Journal, junio de 1994, v12 n6, p.81-95.

(BA94) Melinda-Carol Ballou, *Market for re-engineering tools burgeons*, Computer World, 7 de febrero de 1994, v28 n6, p.95.

(BE93) Mitch Betts, *Making the Mexican connection*, Computerworld, diciembre de 1993,v27 n50, p.91-93.

(BE94) Kent Beck, *Patterns and software development*, Dr. Dobb's Journal, febrero de1994, v19 n2, p.18-21.

(BR94) Doug Brown, *Software Maintenance*, The Computer Conference Analisys Newsletter, 11 de marzo de 1994, n337, p.6.

(CM93) *Complexity*, The Computer Conference Analysis Newsletter, 9 de diciembre de 1993, n333, p.7.

(CO93) Peter Coffee, *Software development on front burner*, PCWEEK, 13 de diciembre de 1993, v10 n49 p.103-104.

(CO94) Larry Constantine, *Essentially speaking*, Software Development, noviembre de 1994, v2 n11, p.96.

(CR94) Michael Croxton, *Maintenance still on it resume*, Software Magazine, Junio de 1994, v14 n6, p.4.

(DE94) EDGE, *2000 AD: seminar takes aim at $75 billion computer glitch*, EDGE: Work - Group Computing Report, 13 de junio de 1994, v5 n212, p.9.

(DL89) John M. Dlugosz, *DOS profilers*, Computer Language, octubre de 1989, p.81.

(DR93) Diane Drotos, *Tools by themselves not the answer*, Computing Canada, 21 de junio de 1993,v19 n13, p.29

(GI94) Lauren Gibbons Paul, *Finding the right tools depends on your goals a variety of metrics tools complicates the decision*, PCWEEK, 28 de noviembre de 1994, v11 n47, p.26.

(HA93) Tom Halligan, *Allways bet the underdog*, LAN Computing, diciembre de 1993,v4 n12, p.18.

(HO93) Nevin Howard, *7 worthy goals for your re-engineering program*, Government Computer News, 25 de octubre de 1993, v12 n23, p.19.

(KA94) Siyan Karanjit S., *Coping with complex programs (Quality control using software metrics)*, Dr. Dobb's Journal.

(KE94) Warren Keuffel, *Result metrics: use and abuse*, Software Development, Junio de 1994, v2 n6, p.27-31

(KR93) Alan Krull, *Defects & reengineering (20th Annual Computer Security Conference and Exhibition, session on quality control)*, The Computer Conference Analysis Newsletter, 18 de noviembre de 1993, n332, p.9.

(LE80) Lehman, M., *On Understanding Laws, Evolution and Conservation in the Large-Program Life Cycle* , The Journal of Systems and Software, 1980, v1 n3. Citado en (FA87)

(LE94) Moisey Lerner, *Software maintenance crisis resolution : the new IEEE standard*, Software Development, agosto de 1994, v2 n8, p.65-69.

(LY94) Tin Lynch, *Avoiding another tower of Babel*, Computer World, 7 de marzo de 1994, v28 n10, p.73.

(MA93) *Companies race to reengineer computers, work habits (Market Monitor)*, Digital News & Review, 3 de mayo de 1993, v10 n9, p.62.
(MA94) Kristin Marks, *Cover your assets*, LAN Magazine, febrero de 1994, v9 n2, p.139-144.
(MN94) Emma Mansell Lewis, *Softly softly*, Computer Weekly, 25 agosto de 1994, p.24-26.
(MC93) Lee Mcnamar , Barbara Von Halle, *Life in the spider's* , Database Programming & Design, diciembre de 1993, v6 n13,p.13-15.
(MI94) Carol Minton, *Don't look now, but your code is no good (need to rewrite applications for the downing communications era)*, MIDRANGE Systems, 15 de abril de 1994, v7 n7, p.28.
(MO94) Terry Moriarty, *CASE metamorphosis*, Database Programming & Design, v7 n5, p.26-29.
(OC93) Tom Ochs, *Cleaning Out the Leftovers: Software Reengineering*, Software Development, diciembre de 1993, p.59-66.
(PA93) Michael F. Parry, *Workflow re-engineering*, The Computer Conference Analysis Newsletter, 1 de septiembre de 1993, n323, p.10.
(PU94) Alan Pursell, *Is Cobol dead?*, Computerword, 25 de abril de 1994, v28 n17, p.112-113.
(QU93) *Quatity*, The Computer Conference Analysis Newsletter, 9 de diciembre de 1993 n333, p.9.
(RA92) Steve Rabin, *Reengineering Opportunities*, Computer Language, abril 1992.
(RE94) Glenn Reid, *Effort saved by reuse can be quantified*, Computing Canada, 14 de septiembre de1994, v20 n19, p.20.
(RO94) Robert, Troy, *Industrialization of software*, Troy Robert, Electronic Engineering Times, 3 de enero de 1994 n778 p.20.
(SC94) Robert L. Scheier, *Put your system to the test - early*, PCWEEK, 27 de junio, 1994, v11 n25, p.25-26.
(SI94) Dr. Waheed Siddigee, *Couting costs (costs estimation of software projects) (quality week)*, The Computer Conference Analysis Newsletter, 30 de mayo de 1994, n343, p.7.
(SW89) E. Swanson, *The Dimensions of Maintenance*, 1989, p. 492-496
(TE94) Mark Tebbe, *Put your apps to the test; don't make the mistake of cutting testing from your development process; you'll regret it*, PCWEEK, 27 de junio de 1994, v11 n25, p.33.
(TR94) Robert Troy, *Industrialization of software (a revolution is required in software engineering)*, Electronic Engineering Times, 3 de junio de 1994, n778, p.20.
(WE92) Kevin Weeks, *Is your code done yet?*, Computer Language, Abril 1992, p.63-68.
(WH94) Sam, Whitmore, *Clock is ticking to fix code before 2000 time bomb*, PC Week, 7 de noviembre de 1994, v11 n44, p.93.

(WI94) Karl wiegers, *Lessons from software work effort metrics*, Software Development, octubre de1994, v2 n10, p.37.

(WN93) Connie Winkler, *Software tools aid re-engineering (Business Process Re-Engineering supplement)*, Federal Computer Week, 20 de septiembre de 1993, v7 n28 p.S22 -S24.

(WR94) Paul Winsberg, Daniel Richards, *Build, evaluate, revise*, Computer World, 9 de mayo de 1994 v28 n19, p11.