Desarrollo de Software

8º Semestre

Programa de la unidad didáctica:
Pruebas y mantenimiento de sistemas de software

Unidad 3.

Mantenimiento de sistemas de software

Clave:

15144842

Ciudad de México, mayo del 2024

Universidad Abierta y a Distancia de México

DCEIT

Índice

Unidad 3. Mantenimiento de sistemas de software 3
Presentación de la unidad 3
Logros.....4
Competencia específica..... 4
3.1. Mantenimiento del software 4
3.1.1. Tipos de mantenimiento..... 9
3.1.2. Pronóstico del mantenimiento 11
3.2. Procesos de evolución del software..... 19
3.2.1. Evolución del software 20
3.2.2. Reingeniería de sistemas..... 23
3.2.3. Mantenimiento de sistemas existentes..... 36
3.3. Gestión del mantenimiento 42
3.3.1. Costos de un plan de mantenimiento de sistemas de software 43
3.3.2. Viabilidad y análisis de riesgo 44
Cierre de la unidad 73
Para saber más 75
Fuentes de consulta 76

Unidad 3. Mantenimiento de sistemas de software

Presentación de la unidad

Después de establecer los fundamentos del aseguramiento de la calidad en la unidad 1, y de someter al sistema a un proceso de pruebas para lograr un grado de aceptación por parte del cliente en la unidad 2, donde mediante el diseño de un plan de pruebas se realizó el análisis de los requerimientos del software, éste pasa a una etapa de implementación; es decir, que el sistema de software se libera.

El sistema, puesto ya en producción, implica su evolución. Lo que es funcional ahora, mañana ya no lo será, porque surgen nuevos requerimientos o son necesarios cambios estructurales en el sistema para adaptarse rápidamente a nuevas necesidades, de tal forma que se requieren acciones.

No es menos complejo que desarrollar un sistema de software. Se trata ni más ni menos que del mantenimiento, donde antes de la liberación del sistema se establecen las fechas del mantenimiento periódico, para conservar al sistema funcional. Durante la vida en producción es posible que broten errores ocultos que hay que arreglar, o nuevos requerimientos en el proceso de negocio que necesiten cambios en el sistema y que deben implementarse si no se quiere perder ventaja competitiva.

La fase del mantenimiento se ubica como la última dentro del proceso de desarrollo de software y, generalmente, no es un proceso económico; por lo que, para comprender su importancia y peso en el desarrollo de la presente unidad, se abordarán los siguientes temas:

El 3.1. Mantenimiento del software es una introducción a la fase del mantenimiento, se identifica el concepto, los tipos de mantenimiento según la norma y los pronósticos del mismo, sienta las bases necesarias para la comprensión del siguiente, el 3.2. Procesos de evolución del software, donde se expone el proceso de evolución, el papel de la reingeniería como herramienta para su evolución lógica y el mantenimiento de un software en operación. Para darle sentido a los temas anteriores, es necesaria una adecuada gestión, que es el último tema de la Unidad 3.3. Gestión del mantenimiento, que engloba en una metodología los pasos necesarios para asegurar el mantenimiento en tiempo, sin abusar del presupuesto, y con el objetivo de conservar la funcionalidad del software.

Logros

Al término de esta unidad lograrás:

- Identificar los tipos de mantenimiento de sistemas de software.
- Relacionar el proceso de evolución del software con sus requerimientos de mantenimiento.
- Analizar la viabilidad y los riesgos de implementar procesos de mantenimiento de software.
- Seleccionar el tipo de mantenimiento para un producto de software.

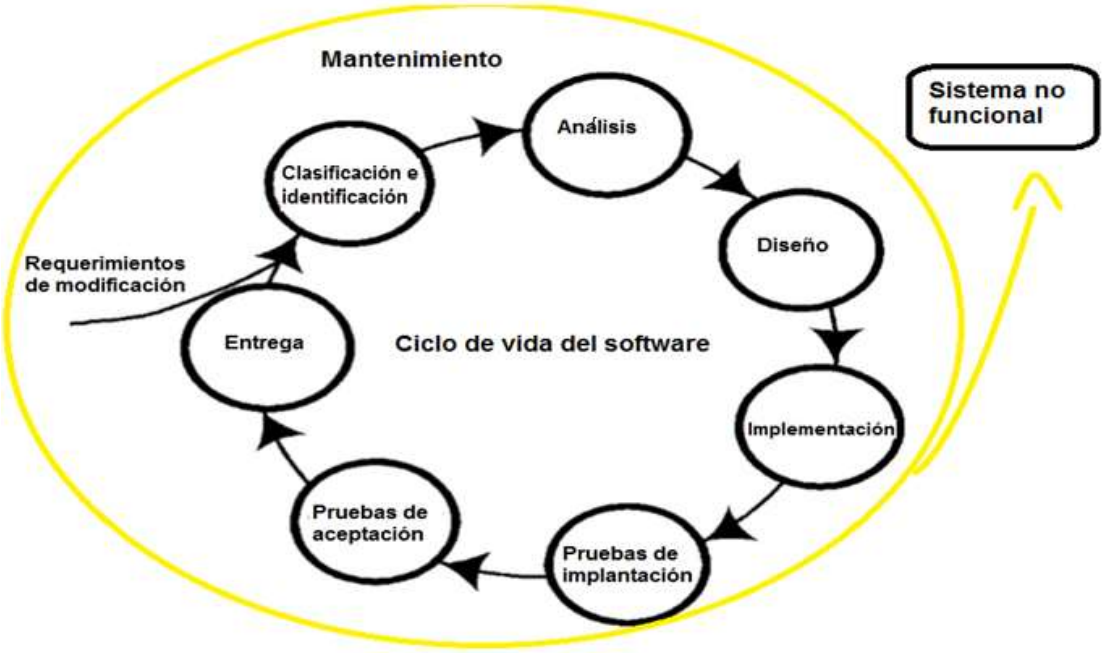
Competencia específica

- Diseñar el proyecto de mantenimiento de un producto software para asegurar el funcionamiento, de acuerdo con los estándares de calidad, mediante un estudio de viabilidad, riesgos y costos.

3.1. Mantenimiento del software

Una vez entregado un producto de software para su puesta en marcha, y después de verificar que cumple con los requerimientos de los usuarios, así como con los requerimientos no funcionales propios del sistema, uno pensaría que con esto se termina la responsabilidad del equipo de desarrollo, sin embargo, de acuerdo con el ciclo de vida del software, no es así. Es posible que durante el uso cotidiano del sistema de software surjan errores que hayan pasado desapercibidos, los ambientes operativos cambien o el proceso de negocio de la empresa se modifique y, por ende, surjan nuevos requerimientos de usuario que se necesiten integrar en el producto de software, es en estos casos cuando cobra sentido la fase de mantenimiento de software.

De acuerdo con el ciclo de vida del software, se observa que el mantenimiento es una de sus fases. Su ciclo propio se repetirá hasta que haya un cambio tecnológico y el sistema ya no sea soportado, o simplemente deje de ser funcional y entre en una situación de obsolescencia. En la imagen siguiente se ilustra la fase de mantenimiento como una más del ciclo de vida del software.



El mantenimiento en el ciclo de vida del software. Basado en Sicilia, 2008.

Como se observa en la imagen anterior, el mantenimiento comienza cuando se identifican nuevos requerimientos después de un periodo de garantía o soporte post-entrega; sin embargo, también se realizan actividades de mantenimiento propias del sistema, como la depuración de archivos temporales durante su operación.

Hace unas décadas, la fase de mantenimiento no tenía la importancia que tiene hoy en día. Actualmente, se considera que un sistema de software no es una inversión cualquiera y se trata de aprovechar al máximo. Esto es posible gracias al mantenimiento, mediante el cual se alarga la vida operativa de un sistema de software.

Bourque y Fairley (2014) en la guía para la ingeniería de software SWEBOK®, *Software engineering body of knowledge,* (que es un proyecto del IEEEComputer Society), definen al mantenimiento del software como “el conjunto de actividades necesarias para hacer que los programas sean rentables” (p. 5-1). Dichas actividades son:

- La que se realiza **durante** la etapa **previa** a la **entrega** es la **planificación** de las operaciones posteriores a la entrega, tales como el mantenimiento y la logística para las actividades de transición entre las versiones del sistema de software.
- Las **posteriores** a la entrega del sistema de software, incluyen la **clasificación** e **identificación** de **requerimientos**, **análisis**, **diseño**, **implementación**, **pruebas de implantación**, **de aceptación** de las modificaciones realizadas al sistema (que

es el mantenimiento mismo), y **entrega** de la nueva versión del sistema de software.

Para el estándar ISO/IEC 14764 (2006), el mantenimiento es la “modificación de un producto de software después de la entrega para corregir los fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un nuevo entorno”.

El mantenimiento se realiza con el fin de:

• **Corregir** fallas detectadas.

• **Mejorar** el diseño.

• **Implementar** mejoras.

• **Mantener** la interfaz con otros sistemas (ejemplo: tienda virtual con los portales bancarios).

• **Adaptar** los programas de manera que las características del sistema y de las telecomunicaciones se puedan seguir utilizando en diferentes plataformas de hardware y software.

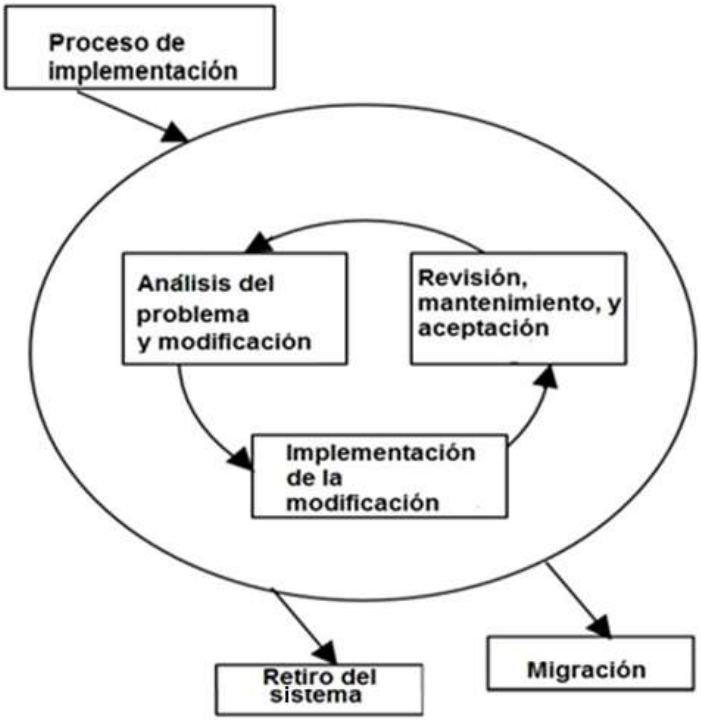
• **Migrar** el legado del software.

• **Identificar** cuándo es necesario retirar el software de su vida productiva.

De acuerdo con el **tipo** de mantenimiento, se persiguen distintos objetivos tales como:

- Mantener el control sobre las funciones del software del día a día. Se realiza una serie de modificaciones para permitir la funcionalidad del sistema a través del tiempo (**correctivo**).
- Mantener el control sobre modificación de software. Al ir adaptando el sistema de software a los distintos requerimientos, es necesario tener una vigilancia para que tales modificaciones no lo degeneren (**adaptativo**).
- El perfeccionamiento de las funciones existentes. Tales modificaciones buscan el mejoramiento de los procesos del sistema (**perfectivo**).
- Prevenir el rendimiento del software de degradar a niveles inaceptables. Son acciones de limpieza de basura informática, de procesos incensarios que hacen lento el sistema, operaciones que provoquen desbordamiento de la memoria, etcétera (**preventivo**).

Las actividades del proceso de mantenimiento, como se observa en la siguiente imagen, es un **proceso continuo** hasta que el sistema se retira y se reemplaza por otro, o hay una **migración** de una **versión** a otra. Las actividades observadas corresponden al ciclo de vida del software.



Actividades del proceso del proceso de mantenimiento. Fuente: ISO/EC 14764, 2006.

Las actividades del mantenimiento de software, según ISO/IEC 14764 (2006), se clasifican de la siguiente manera:

Tareas de procesos de mantenimiento

- Desarrollar planes y procedimientos de mantenimiento
- Establecer procedimientos para la petición de modificación
- Implementar el proceso de gestión de la configuración.

Tareas de nuevos requerimientos y modificaciones

- Realizar el análisis inicial
- Verificar el problema
- Desarrollar opciones para aplicar la modificación
- Documentar los resultados

- Obtener la aprobación para la modificación

Tareas de implementación de la modificación

- Realizar un análisis detallado
- Desarrollar el código y probar la modificación

Tareas de revisión, mantenimiento y aceptación

- Obtener la aprobación para su modificación
- Realizar las pruebas pertinentes

Tareas de migración

- Asegurar de que la migración esté de acuerdo con la norma ISO/IEC 12207, y desarrollar un plan de migración
- Notificar a los usuarios de los planes de migración
- Llevar a cabo operaciones paralelas de migración y pruebas de compatibilidad de datos
- Notificar al usuario de que la migración se ha iniciado
- Llevar a cabo una revisión posterior a la operación
- Asegurar de que los datos de la versión anterior estén accesibles

Tareas de retiro de software

- Desarrollar un plan de retiro del sistema de software
- Notificar a los usuarios de los planes de retiro del sistema de software
- Llevar a cabo operaciones paralelas del retiro del sistema de software, así como la implantación del nuevo
- Notificar al usuario que el proceso de retiro se ha iniciado
- Asegurar de que los datos del sistema en retiro estén accesibles para el nuevo sistema de software

Una estrategia de mantenimiento, según ISO/IEC 14764 (2006), se conforma por "todas las actividades para preparar los recursos humanos y materiales necesarios para el mantenimiento de software para productos de software" (p. 31).

Un *plan de mantenimiento* es un documento que indica cuáles son las prácticas específicas, los recursos y la secuencia de actividades relevantes para el mantenimiento, según ISO/IEC 14764 (2006).

Todo lo anterior debe estar documentado en un plan de mantenimiento del software (FHWA CA División, 2009). Es necesario que revises los elementos que conforman este plan en el documento *Plantilla del plan de mantenimiento FHWA CA División* (2009), apartado 8.4.12., con los elementos básicos a considerar.

Las diversas causas (mencionadas previamente) por las cuales la fase de mantenimiento se inicia, da lugar a los diferentes tipos de mantenimiento, como se detallará en el subtema siguiente.

3.1.1. Tipos de mantenimiento

ISO/IEC 14764 (2006) es un estándar internacional que proporciona los requerimientos para el proceso de mantenimiento del software, y establece tres tipos basados en la clasificación de Lientz y Swanson (1978).

- **Mantenimiento correctivo**: son modificaciones **reactivas** a un producto software, hechas después de la entrega, para **corregir** defectos descubiertos.
- **Mantenimiento adaptativo**: es una modificación de un producto software realizada **después** de la entrega, con el fin de que siga siendo productivo en un ambiente diferente.
- **Mantenimiento perfectivo**: es la **modificación** de un producto software **después** de la entrega para mejorar, o seguir **manteniendo** su rendimiento.

La siguiente tabla es la propuesta del estándar ISO/IEC 14764 (2006), donde se observan las características de los tipos de mantenimiento. Así se tiene que el preventivo (tipo que agrega el estándar ISO/IEC 14764) es de corrección y es proactivo, se adelanta a los hechos. El perfectivo también se adelanta a los hechos, pero es de mejora. El adaptativo es de mejora, pero de naturaleza reactiva, reacciona a los hechos. El correctivo, como su nombre lo dice, es de corrección y de naturaleza reactiva.

Naturaleza	Correctiva	Mejora
Proactiva	Preventivo	Perfectivo
Reactiva	Correctivo	Adaptativo

Características de los tipos de mantenimiento. Fuente: ISO/IEC 14764, 2006.

Tipos de mantenimiento, según ISO/IEC 14764 (2006)

- **Mantenimiento preventivo:** son **modificaciones** del producto software tras la entrega para detectar y corregir fallos latentes antes de que se conviertan en efectivos. El mantenimiento preventivo trata de **mejorar** en puntos específicos, detectados sin cambiar los requerimientos originales, tales como la validación de datos de entrada, mejora de la legibilidad de los programas o la documentación, y la detección de módulos reutilizables para otros sistemas.
- **Mantenimiento correctivo**: **identifica** y **elimina** los defectos o fallos en el sistema de software. Los fallos son provocados en el procesamiento de salidas incorrectas del programa en el rendimiento del sistema, como lentitud en el tiempo de respuesta de búsqueda, en la programación al encontrarse inconsistencias en el diseño del programa con lo programado, en la documentación de la funcionalidad de un programa que indica para qué sirve mientras el manual del usuario dice otra cosa.
- Mantenimiento **adaptativo**: es la **adaptación** del sistema a un nuevo entorno de operación, tales cambios incluyen el del sistema operativo, una computadora con diferentes características a la usual, una arquitectura de red a otra, o nuevos elementos al entorno de operación del sistema, tales como herramientas administrativas de orígenes de datos ODBC[1].

Los cambios en el entorno de ejecución del software se realizan en dos aspectos:

- Cambio en el **entorno de los datos**, de un archivo de datos a tablas de base de datos.
- Cambio en el **entorno de los procesos**, de una plataforma de desarrollo a otra, como de C++ a C#.

[1]Open Data Base Connectivity

El mantenimiento perfectivo se hace necesario cuando surgen nuevos requerimientos, por ejemplo, cambios en el formato de la factura, nuevos procedimientos de cálculo o hasta un nuevo componente.

Se dice que se tiene un mantenimiento **perfectivo de aplicación** cuando al sistema se le incorporan nuevas funcionabilidades, o un mantenimiento **perfectivo de eficiencia** cuando sólo se busca mejorar un proceso existente.

Una vez identificados los tipos de mantenimiento y las características que los determinan, la actividad siguiente es estipular cada cuándo hacer un mantenimiento y el impacto económico que representa dentro del costo del sistema.

3.1.2. Pronóstico del mantenimiento

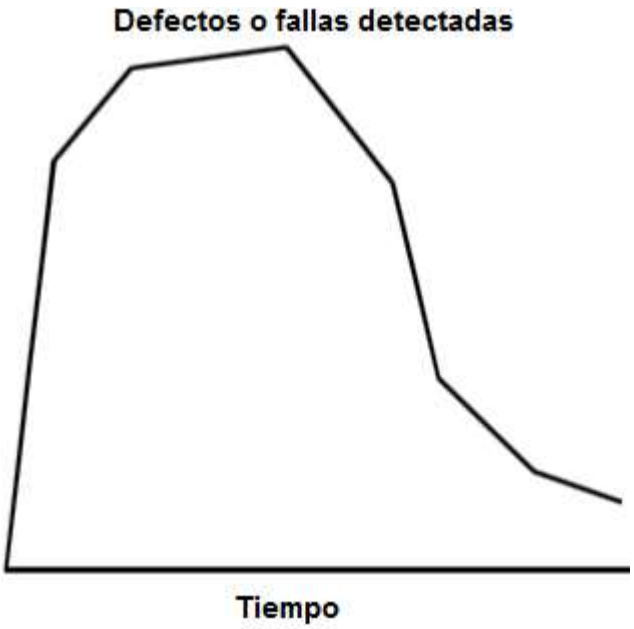
Para no caer en la situación de que el mantenimiento sobrepase el costo del sistema, es necesario hacer un pronóstico del mantenimiento.

Un pronóstico no es más que una predicción de la evolución de un proceso o de un hecho futuro a partir de criterios lógicos o científicos, según la Real Academia Española (2014a), concepto aplicado al mantenimiento del software en pronosticar el costo de realizar los ajustes necesarios para tener el control.

Una aplicación sencilla de un pronóstico sería, por ejemplo, el costo anual de mantener un sitio web estático después de que se puso en marcha; esto es más o menos equivalente a lo del desarrollo. Si el desarrollo costó $150,000.00, se puede esperar que el costo de mantenimiento sea de $112,500.00 (Foster, 1993) hasta los $150,000.00 en un periodo de 4 a 5 años (s.a, 2008).
El ejemplo anterior es sencillo, pero se complica cuando los sistemas son dinámicos y tienden a crecer exponencialmente; por ejemplo, una plataforma de comercio electrónico o una universidad cuya matrícula se incremente semestre tras semestre.

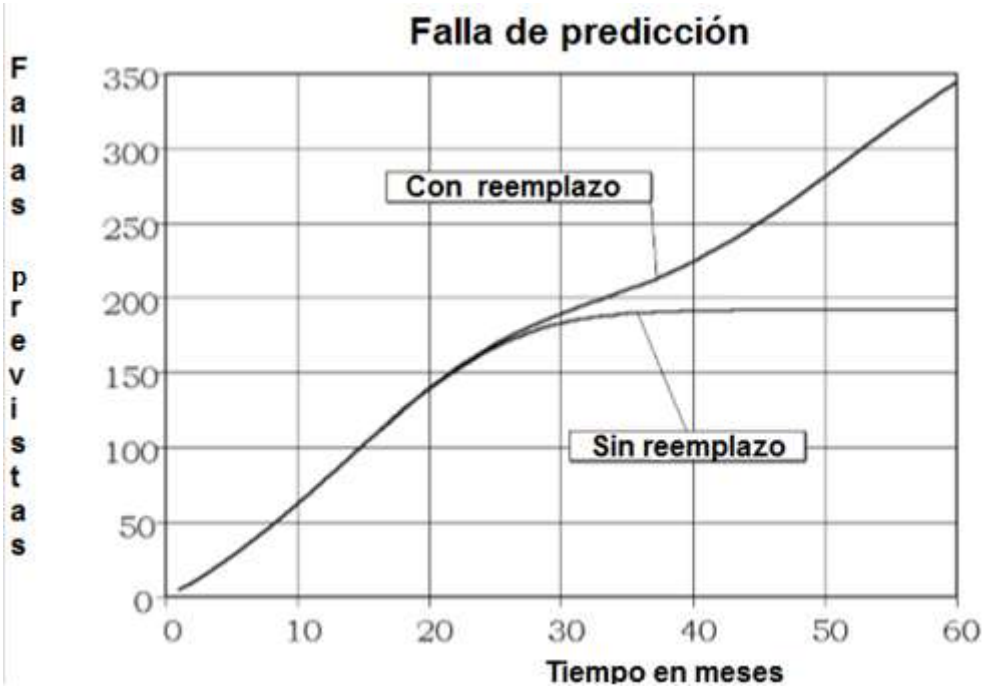
Hay modelos en el mercado para predecir los costos del mantenimiento, basados en la densidad de los defectos tales como:

La curva de Raleigh. Este modelo predice la detección de fallos durante la vida del esfuerzo de desarrollo de software; por lo tanto, también del mantenimiento. Se puede utilizar este perfil para medir el estado de defecto. El modelo supone que, durante la vida del proyecto, los fallos detectados por mes se asemejarán a una curva de Raleigh, según Lakey (2014).



Curva de Raleigh relación fallas-tiempo. Basado en Lakey, 2014, p. 10.

Análisis Weibull. Es la técnica mayormente elegida para estimar una probabilidad, basada en datos medidos o asumidos. La distribución de Weibull es útil, por su habilidad, para simular un amplio rango de distribuciones como la normal o la exponencial, y para que en los proyectos de mantenimiento se tenga una aproximación a un número de fallos (Abernethy, 2006).



Predicción de fallas (basado en Abernethy, 2006, p. 5)

KSLOC (por sus siglas en inglés *Source Logical Code*), "es el número de miles de líneas de código fuente que se entregarán al finalizar el proyecto del mantenimiento", según Newport (1994, p. 84).

KSLOC es utilizado para determinar el número de fallas inherentes en el proyecto de mantenimiento mediante la siguiente fórmula:

$$N = FD * KSLOC$$

Donde:

N: Es el número estimado de fallas inherentes

Para realizar esto es necesario obtener el valor de FD:

$$FD = A * B \ (si\ B\ es\ conocida)$$

Donde:

FD: errores por KSLOC

Variables:

A: Son las características del tipo de aplicación a desarrollar

B: Es el tamaño aproximado de la aplicación en líneas de código fuente y el número de unidades

C: Número de fallas por KSLOC = es A si C no es conocida

Sin embargo, estos modelos no tienen un 100% de fiabilidad, ya sea por imprecisión o complejidad, sin mencionar su costo.

Hay una metodología rescatable, basada en el modelo de estimación de proyectos de Boehm (2000), que es ampliamente aceptada en la industria del software como un modelo válido para la predicción de los costos de mantenimiento, es el COCOMO II® (Constructive Cost Model II). Es un modelo que permite estimar el costo, el esfuerzo y el tiempo en la planificación de una nueva actividad de desarrollo de software. COCOMO II® es la última ampliación del COCOMO 81®, publicado en 1981. Es relativamente fácil de entender y, más importante aún, permite refinar el pronóstico gracias a los multiplicadores de costo.

La fórmula básica de Boehm (2000) que se utiliza en el modelo COCOMO es la siguiente:

$$\text{AME} = \text{ACT X SDT}$$

Donde:

AME: es el esfuerzo de mantenimiento anual medido en meses-persona.
ACT: es el tráfico de variación anual, lo que representa una fracción de las instrucciones de código de un producto de software que sea objeto de modificaciones durante un año típico, mediante la adición o modificación.
SDT: es el tiempo de desarrollo de software en meses-persona.

Por ejemplo, un proyecto de software requiere **100 meses-persona** de esfuerzo de desarrollo y se estima que el 15% del código se modificaría en un año típico. Por tanto, la estimación del esfuerzo de mantenimiento anual básica (AME) es:

AME= 0,15 x 100 = 15 meses-persona.

Lo cual significa que se debe **planear** 15 meses-persona de esfuerzo por año para mantener este proyecto de software específico, que representa el costo base del mantenimiento.

La estimación base de costo anual de mantenimiento, obtenida anteriormente, se puede refinar por la importancia de cada **factor** que afecta el costo y la **selección** del **multiplicador** de costo adecuado.

El costo de mantenimiento base se multiplica por cada multiplicador de costo para obtener una estimación integral del costo de mantenimiento.

Los multiplicadores de costo son las siguientes variables significativas que afectan el costo del mantenimiento del software, con lo cual se logra una mayor precisión en su estimación:

CPLX: complejidad del producto.

AEXP: experiencia de los analistas.

El resultado del cálculo del AME, en el ejemplo anterior, que se traduce como el **costo base** del mantenimiento, se refina considerando el factor de la **complejidad** del producto CPLX como alto, debido al número de **componentes** y de **transacciones** que se hacen en la base de datos y el factor AEXP bajo, debido a la poca disponibilidad de personal de apoyo con experiencia en aplicaciones. En las imágenes siguientes se observa el rango de valores del factor CPLX, obtenido a través de la experiencia, según Gómez (2014).

Atributo	Valores					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
CPLX	Se usan expresiones matemáticas simples					Se usan muchos recursos de planificación
	0,7	0,85	1	1,15	1,3	1,65

Rango de valores de CPLX.

CPLX = 1,30

AEXP = 1,29

Atributo	Valores					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
AEXP	<= 4 meses	1 año	3 años	6 años	>= 12 años o reimplementación de un subsistema	-
	1,29	1,13	1	0,91	0,82	-

Rango de valores de AEXP.

La fórmula para refinar la estimación del costo del mantenimiento queda de la siguiente manera:

$$AEM = ACT\ x\ SDT\ x\ CPLX\ x\ AEXP$$

Sustituyendo los valores:

AEM = 0.15 x 100 x 1,30 x 1,29 = 25,1 meses-persona

El pronóstico de las mejoras no es un asunto trivial, ya que se requiere conocer las futuras demandas de los clientes. Estas solicitudes de cambio ya no representarán un costo para la empresa desarrolladora, porque se integrarán a los costos estimados en la planeación y se cobrarán a la empresa que se le vendió el sistema de software. Si es un producto que se creó en el departamento de sistemas, las solicitudes de cambio sí representan un costo extra para la empresa, ya que se requiere financiar los cambios.

La predicción del mantenimiento se ocupa de la evaluación de qué partes del sistema pueden causar problemas, así como sus costos asociados.

Distribución de la predicción del mantenimiento. Basado en Henry y Wake, 1988, p.14.

La predicción de cambios determina el **número** necesario para el sistema, con base en la relación con su entorno. Los sistemas fuertemente acoplados requieren cambios siempre que se modifica el medio ambiente.

Los factores que influyen en la relación entre el sistema y su entorno son:

- La **cantidad** y complejidad de las interfaces del sistema
- El **número** de los requisitos del sistema inherentemente volátiles[2]
- Los **procesos** de negocio que utiliza el sistema

La predicción del costo del mantenimiento se puede realizar mediante la **evaluación** de la **complejidad** de los componentes del sistema. Hay estudios de base que han demostrado que la mayor parte del esfuerzo se gasta en el mantenimiento de un número relativamente pequeño de los componentes del sistema, y la complejidad depende de los siguientes factores:

[2] Son requerimientos que probablemente cambian durante el proceso de desarrollo del sistema, o después de que se haya puesto en funcionamiento.

- La complejidad de las estructuras de control
- La complejidad de las estructuras de datos
- Del objeto, método (procedimiento) y el tamaño del módulo

La **predicción** de la mantenibilidad permite que los procesos sean **medidos** y **usados** para **evaluarla** a través de los siguientes casos:

- El número de solicitudes de mantenimiento correctivo
- El tiempo promedio necesario para el análisis de impacto
- El tiempo promedio necesario para poner en práctica una solicitud de cambio
- El número excepcional de solicitudes de cambio

Si cualquiera, o todos los casos anteriores, es cada vez más frecuente, indica una **disminución** en la **capacidad** de mantenimiento.

Para determinar la **frecuencia** de un mantenimiento preventivo es necesario considerar el entorno de uso, según la siguiente tabla:

Entorno de uso	Frecuencia de mantenimiento
Muy poco uso (equipo aislado, pocas aplicaciones, etcétera)	Una vez al año
Uso esporádico (consultas puntuales, ámbitos atípicos, etcétera)	Dos veces al año
Uso diario (entorno de oficina)	Tres veces al año
Uso intensivo (multiusuario, flujo de datos alto, página de comercio electrónico)	De cuatro a seis veces al año
Uso muy intensivo (multiusuario, red, alto flujo de datos, por ejemplo: algunos sitios de redes sociales)	De seis a doce veces al año

Tabla de frecuencia de mantenimiento de acuerdo con el entorno de uso. Basado en Gallego, 2010, p.10.

Los demás tipos de mantenimientos se determinan de acuerdo con los tipos de requerimientos que surjan.

En este tema se ha desarrollado la forma de **pronosticar** costos del mantenimiento, **identificar factores** que lo incrementan, así como la distribución de la predicción del mantenimiento para medir el impacto económico que la empresa absorberá, con tal de conservar funcional el sistema de software.

De manera general, se han identificado los tipos de mantenimiento y sus características para establecer la metodología adecuada para su realización, cuidando la funcionabilidad del sistema y que el costo no sobrepase el presupuesto asignado.

Para poder aplicar los conocimientos anteriores, es necesario conocer y reconocer el proceso de evolución del software, y a la vez determinar cuándo hacer reingeniería o un tipo mantenimiento, en particular a un software en producción. De eso trata el siguiente tema.

3.2. Procesos de evolución del software

Un software puesto en marcha no significa que se quede estático y al paso del tiempo todo siga funcionando de igual forma. Los cambios son inevitables y si se desea que la inversión en el sistema de software sea redituable, se tiene que invertir en su mantenimiento.

Como bien se ha expuesto al principio de la unidad, los cambios surgen cuando:

- El software se utiliza.
- Surgen modificaciones en el entorno de negocio.
- Surgen errores y deben ser arreglados.
- Se integran nuevas arquitecturas de computadora y equipos.
- El rendimiento o la confiabilidad del sistema pueden y deben ser mejorados.

Las organizaciones se enfrentan al reto de poner en práctica y gestionar los cambios en su sistema de software existente, por lo que hacen grandes inversiones y las consideran sus activos críticos del negocio. Para mantener el valor de estos activos deben ser actualizados. Por ello, la mayor parte del presupuesto de software se dedica a la evolución de los programas informáticos existentes, en lugar de desarrollar un nuevo software.

A consecuencia de lo anterior, se observa que el software cambia a lo largo de su vida útil; por lo que surge el concepto de la dinámica de evolución del programa, que es el estudio de los procesos de cambio del sistema, derivado de los estudios de Lehman y Belady (1985), quienes determinaron una serie de leyes que se aplican a todos los

sistemas a medida que evolucionan; también hay observaciones aplicables a los grandes sistemas que se detallan en el siguiente subtema.

3.2.1. Evolución del software

En la ingeniería de software, la evolución de software se estableció en una serie de leyes que Lehman y Belady (1985) formularon entre 1974 y 1996. Éstas describen un equilibrio entre las fuerzas impulsoras de nuevos desarrollos, por un lado, y las que frenan el progreso, por otro.

Al observar que la mayoría del software está sujeto a cambios en el curso de su existencia, los autores se propusieron determinar las leyes que suelen obedecer, o deben obedecer para que el software pueda sobrevivir.

Lehman y Belady (1985) publicaron, en su artículo de 1980, la aplicación de las leyes mediante la distinción de tres categorías de software:

1. **Un programa S** se escribe de acuerdo con una especificación exacta de lo que el programa puede hacer.
2. **Un programa P** se escribe para implementar ciertos procedimientos que determinan lo que el programa puede hacer.
3. **Un programa E** está escrito para llevar a cabo algún tipo de actividad en el mundo real; su comportamiento está relacionado con el entorno en el que se ejecuta. Un programa de este tipo tiene que adaptarse a las diferentes necesidades y circunstancias del medio ambiente.

Las leyes siguientes se aplican a la última categoría de sistemas, mencionada anteriormente.

Ley	Descripción
Cambio continuo	Un programa que se utiliza en un entorno real necesariamente debe cambiar, o será cada vez menos útil en ese entorno.
El aumento de la complejidad	Es una consecuencia de la evolución de los cambios del programa, su estructura tiende a ser más compleja y se requieren recursos adicionales, que se aplican para la preservación y la simplificación de la estructura.

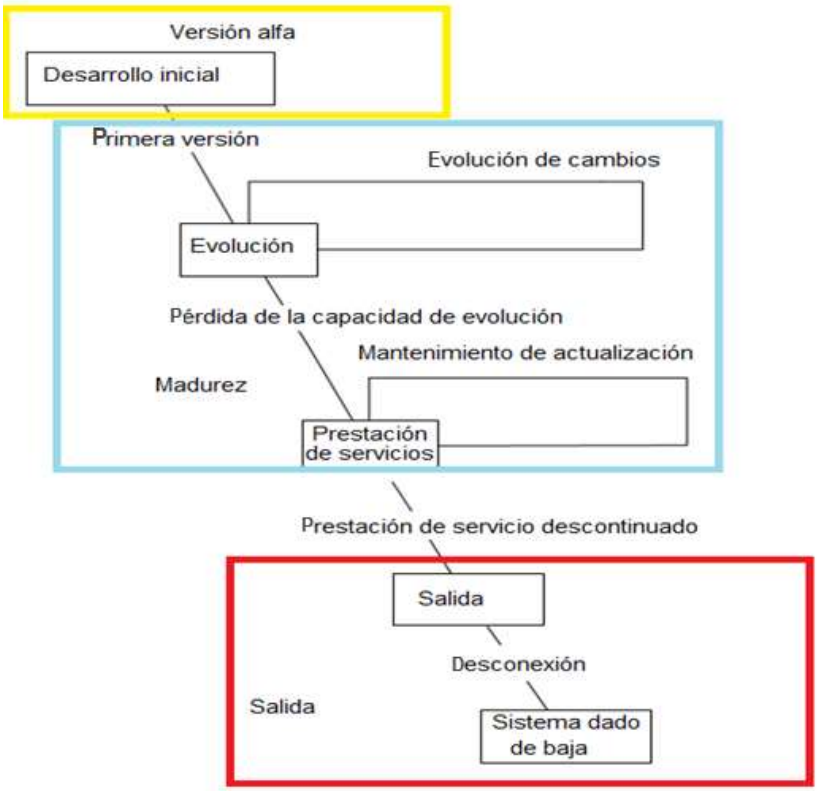
Evolución de sistemas grandes	El programa es un proceso de auto-regulación, los atributos del sistema, tales como el tamaño, el tiempo entre las versiones y el número de errores notificados, es aproximadamente invariable para cada versión.
Estabilidad organizacional	Durante la vida del programa, su ritmo de desarrollo es aproximadamente constante e independiente de los recursos dedicados al desarrollo del sistema.
Conservación de familiaridad	Durante la vida útil de un sistema, el cambio incremental en cada versión es aproximadamente constante.
El continuo crecimiento	La funcionalidad ofrecida por los sistemas tiene que aumentar continuamente para mantener la satisfacción del usuario.
Disminución de la calidad	La calidad de los sistemas suele decaer, a menos que se adapte a los cambios en su entorno operativo.
Sistema de retroalimentación	El proceso de evolución incorpora un multiagente[3] como un sistema de retroalimentación, y hay que tratarlos como sistemas de retroalimentación para lograr una mejora significativa del producto.

Leyes y descripciones de Lehman y Belady. Fuente: Lehman y Belady, 1985.

Las leyes se aplican a un tipo de software en particular, con ciertas características, pero ¿qué pasa con el resto de los sistemas de información que no están en la categoría E?, aquellos sistemas que realizan tareas rutinarias, administrativas, que no son influenciados por el entorno de en el que se ejecutan.

Hay una propuesta de evolución del software de Bennett y Rajlich (2000), que se muestra en la siguiente imagen, la cual considera a todos los sistemas de software, incluyendo a los de programa E, y se basa en el principio de que todos los sistemas tienen una versión alfa, una etapa de madurez y una de salida.

[3] El sistema multiagente (SMA) está compuesto por múltiples agentes inteligentes que interactúan entre sí.



Evolución del software (basado en Bennett y Rajlich, 2000, p.4).

La propuesta consiste en separar las etapas de evolución, seguidas por una de mantenimiento

Etapa 1 de la evolución del software: versión alfa

Durante la versión alfa del sistema de software, a pesar de las diferentes pruebas, es posible que se detecte la falta de algunas características, que se incorporarán durante esta etapa también conocida como de desarrollo inicial.

También es posible vislumbrar posibles cambios o modificaciones en el futuro. La mayoría de las referencias en esta etapa se basan en escenarios o casos de estudios. El desarrollo inicial genera un banco de conocimiento, tal como el de dominio de aplicación, requisitos de los usuarios, reglas de negocio, políticas, soluciones, algoritmos, etcétera. El conocimiento se determina como un factor importante para la fase siguiente de la evolución. Una vez que la versión alfa se completó con éxito, el sistema de software es puesto en marcha, lo que conlleva a la necesidad de la evolución como consecuencia del uso.

Etapa 2 de la evolución del software: madurez

Se origina por que los usuarios tienden a cambiar sus necesidades, así como su propia percepción de mejoras en el sistema.

Independientemente de lo anterior, se sabe que la industria del software se enfrenta al reto de cambios vertiginosos en el entorno, de ahí que la meta de la evolución sea la adaptación de la aplicación a las siempre cambiantes necesidades de los usuarios y el medio ambiente de trabajo. En el sistema de software ya en producción, y durante los primeros días, los usuarios pueden detectar fallas, que se pueden corregir durante la etapa de madurez a partir de requisitos más específicos y precisos, debido al estudio de casos o escenarios.

Etapa 3 de la evolución del software: salida

El software evoluciona continuamente manteniéndose estable hasta que el sistema ya no sea adaptable, entonces se llega a la etapa de salida, que se caracteriza porque ya no hay soporte técnico; sin embargo, el software todavía está en producción. Por último, el sistema es dado de baja, se apaga o se interrumpe y los usuarios son redireccionados hacia el nuevo.

De esta forma se facilita la identificación de las fases del sistema de software y se determina a tiempo la posible salida de un sistema para ir preparando el nuevo, considerando siempre que las necesidades del usuario sean satisfechas. Si se conocen las leyes de evolución del software, que, aunque se aplican a un tipo en particular de sistema, y las etapas de evolución, se tienen parámetros reales para medir la funcionalidad, determinar cuando ya no da más y sustituirlo a tiempo antes de perder ventaja competitiva. Antes de que el sistema sea obsoleto es el momento en que entra la reingeniería de sistemas para identificar el código reutilizable, y tomarlo como base para el nuevo componente, módulo o sistema.

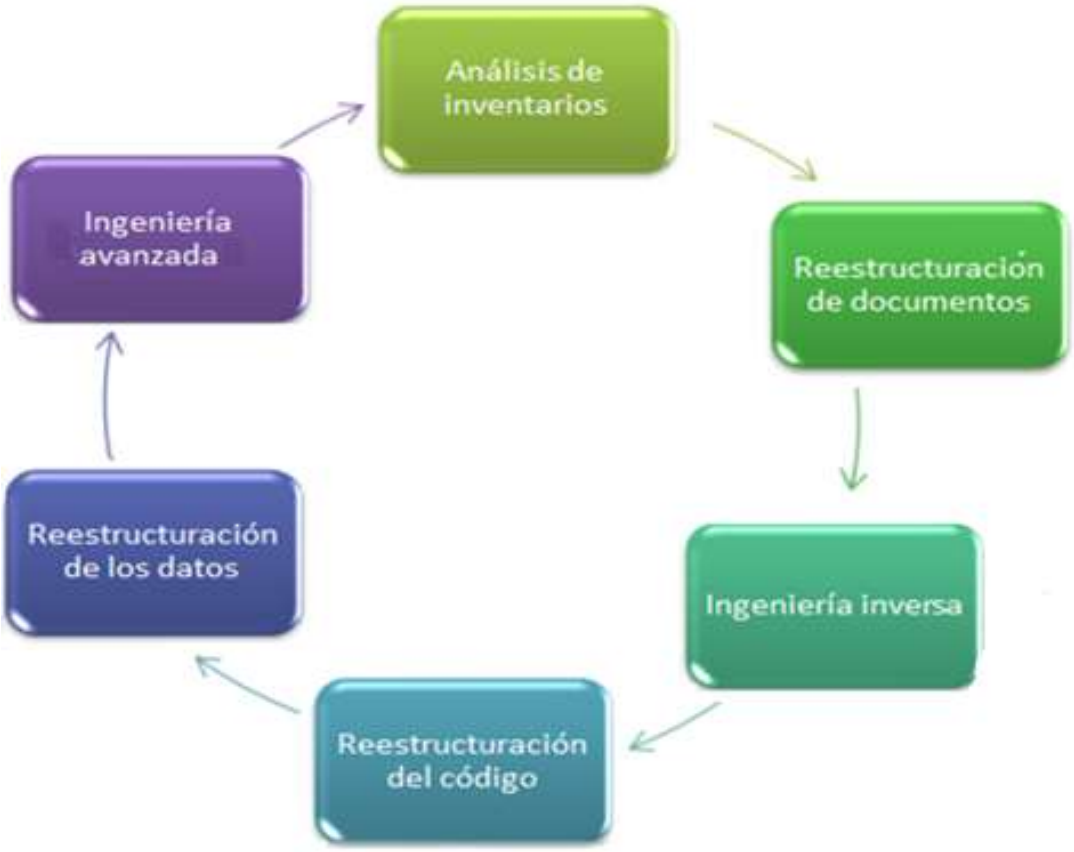
3.2.2. Reingeniería de sistemas

La ingeniería de sistemas analiza, diseña y organiza los diversos elementos de un sistema que se traduce en un producto, servicio o tecnología para la transformación de información o control de la información, según Pressman (2010). Como parte de la

ingeniería del software, hay una técnica que permiten abordar el mantenimiento, de manera que su impacto en coste dentro del ciclo de vida sea menor. Eso es la reingeniería de sistemas.

La reingeniería de sistemas se define, según Sommerville (2011, p. 248), como “la modificación de un producto software, o de ciertos componentes, usando para el análisis del sistema existente técnicas de ingeniería inversa y, para la etapa de reconstrucción, herramientas de Ingeniería Directa, de tal manera que se oriente este cambio hacia mayores niveles de facilidad en cuanto a mantenimiento, reutilización, comprensión o evaluación”.

Según las etapas de evolución del software vistas en el subtema 3.2.1. Evolución del software, se observa que después de la madurez del sistema llega un momento en que se torna inestable, debido a la serie correcciones, adaptaciones o mejoras que han podido surgir a lo largo del tiempo. Y antes de que el sistema tenga reacciones inesperadas y afecte la producción de la empresa, o comience a dar un servicio intermitente se aplica la técnica de la reingeniería al sistema con el fin de mejorarlo en cuanto a la calidad y mantenibilidad, sin alterar sus especificaciones funcionales. La imagen siguiente muestra las actividades que se desarrollan dentro de la reingeniería de sistemas:



Modelo de proceso de la reingeniería de sistemas. Fuente: Pressman, 2010.

Pressman (2010), describe cada elemento del modelo de la reingeniería de sistemas de la siguiente forma:

- **Análisis de Inventarios**: es un modelo en una hoja de cálculo que contiene información detallada del tamaño, edad, importancia para el negocio de las aplicaciones activas. Los módulos o componentes candidatos a la reingeniería se ordenan en función de su importancia para el negocio, longevidad, mantenibilidad actual y otros criterios localmente importantes. Se asignan recursos a las aplicaciones candidatas para el trabajo de reingeniería. El inventario se debe revisar con cierta regularidad, porque el estado de los componentes puede cambiar en función del tiempo y, como resultado, cambiarán las prioridades para la reingeniería.

En la tabla siguiente, la letra C indica que el sistema en cuestión es candidato de mantenimiento próximo.

Sistemas de software	Funcionalidad	Fecha de producción	Tamaño en SLOC	Importancia	Fecha última de mantenimiento	Fecha del próximo mantenimiento	Nivel
SIA Misional SIREL	Función territorial estatal	01/01/2010	350, 000	Muy importante	01/06/2013	01/06/2014	C
SIA Contraloría	Función de control estatal	01/01/2013	175,000	Muy importante	01/12/2013	01/06/2014	B
SIRECI	Función territorial municipal	01/01/2011	285,000	Muy importante	01/01/2013	01/01/2014	A
SIA Misional PGA	Función municipal	01/01/2013	155,000	Importante	01/01/2014	01/12/2014	A
PNA	Función municipal	01/01/2009	255, 000	Importante	01/01/2010	01/01/2014	A
SICA	Función municipal	01/01/2010	223,568	Importante	01/01/2011	01/01/2013	C

Ejemplo de inventarios de aplicaciones

- **Reestructuración de documentos:** se debe crear una documentación breve y comprensible que soporte el código del componente que permita una actualización ágil y eficaz.

```
/*-------------------------------------------------------------------------
                        Instituto de Investigaciones
                Departamento de Sistemas y Computación

Módulo     :
Archivo    :
objetivo   :

>>  Variantes  <<
(archivos *.h)
            Definición de prototipos, variables simbólicas, macros
            estructuras y tipos definidos para << aplicación >>
(archivos *.c)
            Código y variables globales para << aplicación>>
(archivos *.hpp)
            Definición de las clases:
(archivos *.cpp)
            Implementación de las funciones método para la clase

diseñó     :
Implementó:
Versión    :

Copyright:
       Copyright (c) 2014 , ..................
       Todos los derechos reservados.
-------------------------------------------------------------------------*/
```

Ejemplo de prólogo del programa como documentación.

Ingeniería inversa: es el proceso de análisis de un programa con el fin de crear una representación con un nivel de abstracción más elevado que el código fuente. Recupera el diseño original arquitectónico, de proceso, y la información de los datos.

A continuación, se expone un ejemplo basado en García (2010), donde en un sistema de software se detecta un error de protección básica y se busca reforzarlo en pos de la seguridad del sistema.
Los pasos de la ingeniería inversa aplicados en este caso son:

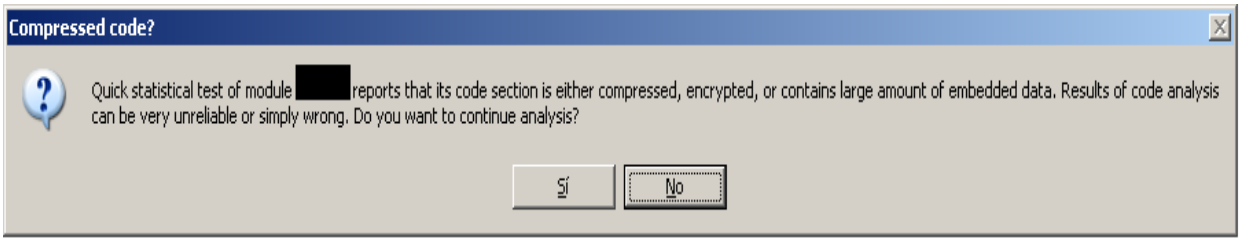
Paso 1. Valorar el tipo de programa y la forma en la que está protegido. La aplicación a estudiar tiene una protección excesivamente básica, ante ello se prueba buscar un archivo para comprobar el hecho.

A continuación, se observa una imagen que representa la visualización de su ejecución normal:

Ventana de error

Se comprueba que el programa busca el archivo archivo.opl. Si no existe, el programa no se ejecuta y envía el siguiente mensaje: File archivo.opl not found. Es una protección bastante burda, puesto que existen multitud de llamadas conocidas para buscar un archivo en el sistema.

Paso 2. Se ejecuta el programa Olly Dbg para depurar el programa y se genera el siguiente cuadro de diálogo:

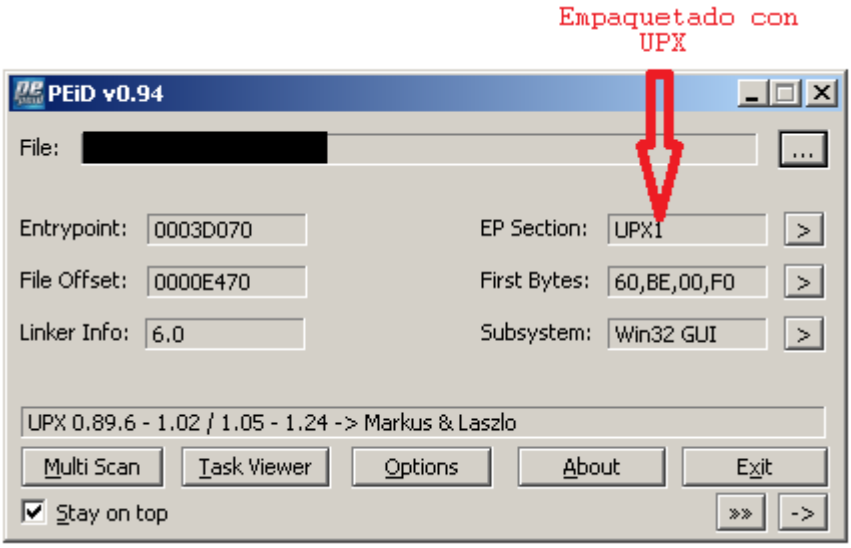


Ventana de OllyDbg que indica que el programa está comprimido.

Olly Dbg indica que este programa está empaquetado. El *empacamiento* es una protección frecuente y útil. Sin embargo, existen herramientas que facilitan el trabajo a cualquier usuario, malicioso o no, y que permite conocer detalles sobre las cabeceras PE.[4]

Paso 3. Se abre el programa con PeId y se presenta la siguiente información:

[4] Portable ejecutable



Ventana de PEID que indica la herramienta de empaquetado.

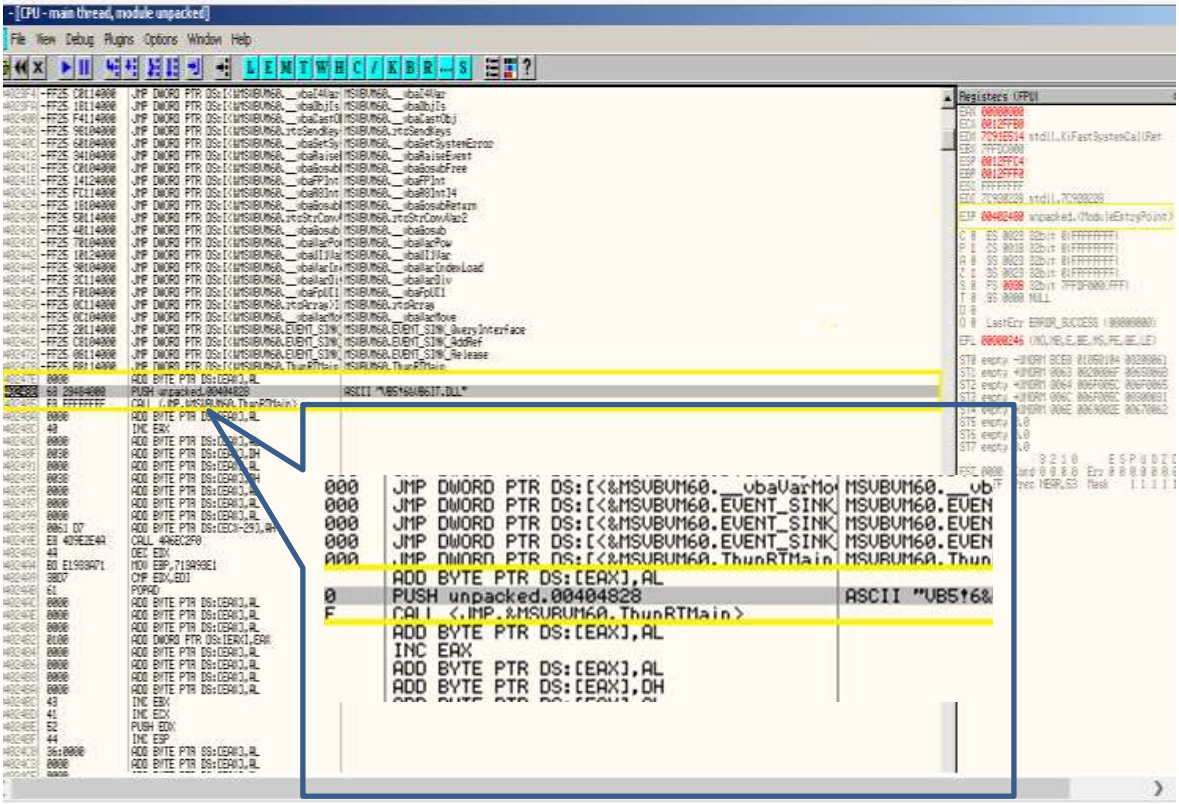
Esta ventana ofrece información crucial y además indica la mala protección del sistema. Lo primero que se observa es que, efectivamente, el ejecutable está empacado con una protección comercial UPX que, incluso, su utilería trae un desempacador.

Lo ideal hubiera sido **proteger** el programa con un **empacador propio** o, al menos, editar las cabeceras para que PeId no pudiera mostrar la versión del utilizado.

Paso 4. Se utiliza la herramienta PeId para **desempacar** el programa gracias a un plugin que se ejecuta. El resultado es dicho programa desempacado y con sus instrucciones listas para ser leídas por el cracker.[5]

Paso 5. Se **ejecuta** el programa otra vez con Olly Dbg y se presenta la siguiente ventana:

[5] Persona que diseña o programa los crack informáticos, que sirven para modificar el comportamiento o ampliar la funcionalidad del software o hardware original al que se aplican.



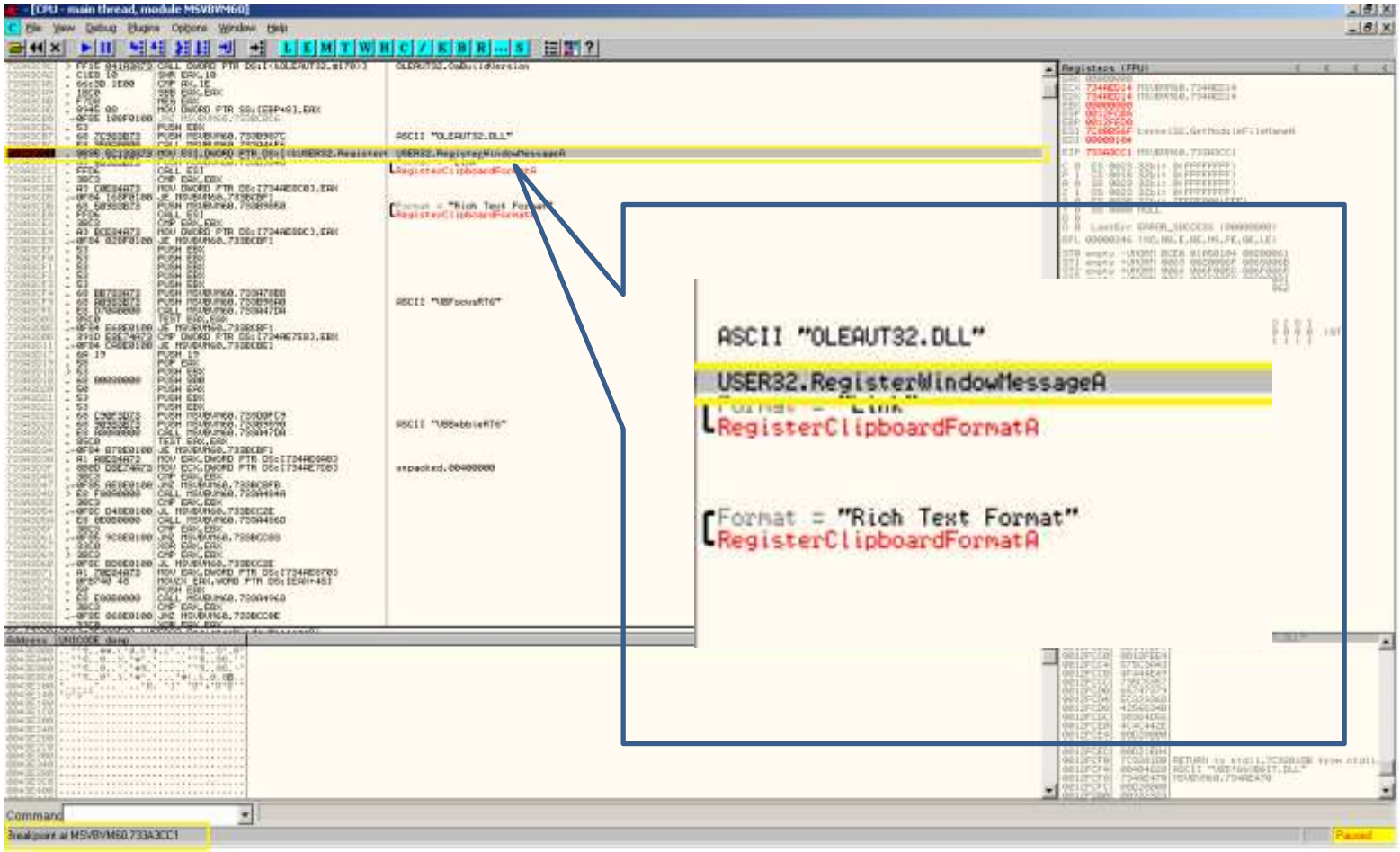
Ventana de OllyDbg con el programa desempaquetado.

La advertencia anterior no aparece, lo que indica que el programa estaba, efectivamente, empacado.

Se **depura** con el OllyDbg hasta que salte la ventana. Es decir, se depura de atrás hacia adelante, en busca del punto exacto en el que el programa se rompe, por decirlo de alguna manera.

Paso 6. Se sabe que una ventana se muestra en Windows con un **register Window Message**, se busca ese comando y se inserta un punto de ruptura.

Se obtiene el siguiente resultado:



Ventana de Olly Dbg con inserción de punto de ruptura.

Está claro que si se llega a ese punto de ruptura depurando y la ventana no ha salido, es muy posible que esa llamada sea la ventana de error; así que se continúa hacia atrás para ver dónde se bifurca el programa (si es que lo hace), para terminar en esa ventana.

Paso 7. Se **sigue depurando** hacia atrás después de establecer el punto de ruptura, y se encuentra una **bifurcación** muy simple: un salto que se realiza en función de una comparación. Si se cambian las banderas de las instrucciones, se descubre que el comando de esa ventana se salta y se puede continuar sin mostrarla. Parece que se ha avanzado en la dirección correcta. Sin embargo, si se sigue hacia adelante, el flujo muestra este error:

Ventana del sistema con mensaje de error

El programa analizado no sólo valida que **exista** el archivo que se busca, sino que también verifica su **integridad**. Esto es completamente lógico y es un buen síntoma de que, al menos, no se han equivocado al decidir la protección.

Se observa que, efectivamente, se llama al mismo comando que antes para **mostrar** otra ventana. Se sigue depurando y se encuentra con, exactamente, lo mismo que antes: una bifurcación simple y llana en la que se pasa o no por la ventana de error, dependiendo de una bandera de verificación. Se realiza el cambio, idéntico al anterior.

Se continúa depurando el programa y se comprueba que se muestra la ventana de inicio de la aplicación; de modo que se ha logrado corregir el ejecutable y saltar la protección.

Paso 8. Después de verificar que se ha corregido el código para mejorar la validación de archivos, se **guardan** los cambios y se **crea** un ejecutable con Olly Dbg.

Con base en la ingeniería inversa, se ha llegado al código y se ha modificado para mejorar un comportamiento del programa,

Reestructuración de código: se aplica a módulos cuya codificación no permite comprenderlos, probarlos ni mantenerlos.

El siguiente ejemplo presenta un método que realiza una verificación de días y meses.

```
public boolean verifica () { //Método
if (dia < 1 || dia > 31)
return false;
if (mes < 1 || mes > 12)
return false;
// determina la cantidad de días del mes:
int diasMes = 0;
switch (mes) {
case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12: diasMes = 31; break;
case 4:
case 6:
case 9:
case 11 : diasMes = 30; break;
case 2 : // verificación de año bisiesto
 if ( (anio % 400 == 0) ||( (anio % 4 == 0) && (anio % 100 != 0) ) )
diasMes = 29;
else diasMes = 28;
break;
}
if (dia > diasMes)
return false;
else return true;
}
```

El método **verifica** es poco legible, poco cohesivo y difícil de mantener.

La solución a la problemática con el código es la siguiente, en la separación de las operaciones de verificación para mejorar su legibilidad, la cohesión y su mantenimiento:

```
private int diasMes() { // Método que verifica día y mes
int diasMes = 0;
switch (mes) {
case 1:
case 3:
```

```
case 5:
case 7:
case 8:
case 10:
case 12: diasMes = 31; break;
case 4:
case 6:
case 9:
case 11 : diasMes = 30; break;
case 2 : if ( bisiesto() )//Método que verifica si el año es bisiesto
diasMes = 29;
else diasMes = 28;
break;}
return diasMes;}
```

```
private boolean bisiesto() {//método que verifica si el mes es bisiesto
if ( (anio % 400 == 0) || ( (anio % 4 == 0) && (anio % 100 != 0) ) )
return true;
else return false; }
```

Y el método **verifica** queda de la siguiente forma:

```
public boolean verifica () {
if (dia < 1 || dia > 31)
return false;
if (mes < 1 || mes > 12)
return false;
if (dia > diasMes())
return false;
else return true;
}
```

El método **verifica** quedo pequeño, sencillo, fácil de depurar y mantener, sobre todo fácil de entender, según los criterios del código de calidad.

Reestructuración de datos, es una actividad de reingeniería a gran escala y comienza con una actividad de ingeniería inversa. Se analiza la arquitectura de datos actual con minuciosidad y se definen los modelos de datos necesarios, se identifican los objetivos de datos y los atributos, y después se revisa la calidad de las estructuras de datos existentes. Se verifican los cambios obtenidos de la revisión, y se actualiza ya sea la arquitectura o el código mismo.

A continuación, se expone un ejemplo basado en Date (2001, p. 294):

Es necesario dividir la base de datos **Varrel** por VNC y VT, y realizar la separación, para un mejor control y protección de los datos.

```
VAR VNC BASE RELATION { V# V#, PROVEEDOR NOMBRE, CIUDAD CHAR }
    PRIMARY KEY { V# } ;

VAR VT BASE RELATION { V# V#, STATUS INTEGER }
    PRIMARY KEY { V# } ;
```

Código de división de la base de datos Varrel.

Se crea una vista de ambas bases para consultas:

```
VAR V VIEW
    VNC JOIN VT ;
```

Vista de VNC y VT
Ahora se podrán hacer consultas sin poner en riesgos las tablas originales, gracias a esta reestructuración.

Ingeniería avanzada, recupera la información de diseño a partir del software existente. Utiliza esta información para alterar o reconstruir el sistema con la finalidad de mejorar su calidad global. En la mayoría de los casos, el software sometido a reingeniería vuelve a implementar la función del sistema existente y también añade nuevas funciones o mejoras.

El procedimiento es parecido al de la ingeniería inversa, pero busca hacer crecer al módulo o al sistema.

Las herramientas de apoyo para el proceso de reingeniería son las siguientes:

- ***IBM Rational Software Architect,*** es considerada una herramienta de modelado y diseño avanzado, la cual tiene las características de realizar modelado de UML extensivo, generación de código, simulación y validaciones entre otras herramientas https://www.ibm.com/products/rational-software-architect-designer.

- ***Softwarenaut*****, del grupo *Software Composition*** (Instituto de Ciencias de la Computación y Matemática Aplicada de la Universidad de Berna), apoya el análisis

estático y evolutivo del código fuente, a través de la visualización y exploración. Es una herramienta libre de derechos que se puede utilizar y experimentar en el sitio oficial SCG. Esta herramienta es utilizada tanto en el ciclo de vida como en el proceso de mantenimiento del software y, por ende, en la reingeniería de sistema y el análisis evolutivo del código.

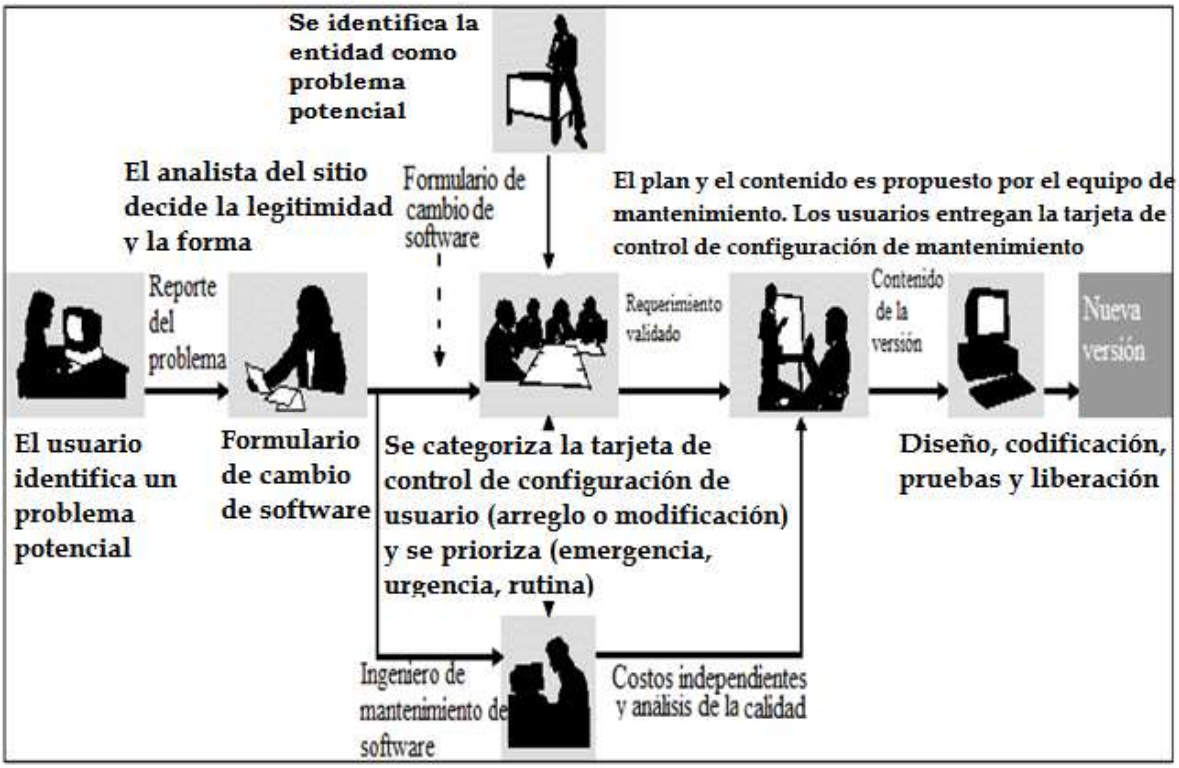
Las actividades anteriores, como parte de la reingeniería, tratan de rescatar lo bueno de un sistema que comienza a ser intermitente en su servicio, lo renuevan con la funcionalidad original y otros agregados que le dan un valor. Unos autores dicen que ese proceso no es barato, y otros que ahorra costos. Lo cierto es que el viejo sistema es renovado por un porcentaje aceptable de código nuevo.

La reingeniería de sistemas es una técnica de mantenimiento pero, ¿cómo empezar?, ¿dónde se observa la necesidad del mantenimiento?, ¿y cuál es el proceso? Las respuestas se desarrollan en el siguiente tema.

3.2.3. Mantenimiento de sistemas existentes

En el tema anterior, se detalló la técnica de la reingeniería de sistemas como un conjunto de actividades que renuevan a un sistema que comienza a ser decadente, como forma de mantenimiento para renovarlo y que dure unos años más.

A continuación, se exponen las actividades que integran el proceso de mantenimiento:



Proceso del mantenimiento del software. Basado en Stark, 1996.

El proceso comienza cuando un usuario identifica un cambio deseado, o una entidad externa se identifica como un problema potencial; por ejemplo, un cambio en los requisitos o necesidad de arreglar algún funcionamiento inadecuado, o un módulo sin el nivel de seguridad necesaria. Tal situación se documenta en el reporte del problema.

Un analista en sitio, junto con el usuario **revisan** el informe de problemas, **verifican** que efectivamente sea un problema para la integridad del sistema, **comprueban** comparando los informes de problemas conocidos para eliminar la duplicación, y para **determinar** si el usuario entiende correctamente el sistema.

Si la dificultad es nueva y se identifica como tal, el analista la clasifica en el informe como un problema del equipo de software, hardware o de comunicaciones.

En caso que sea de software, se genera un formulario de cambio de software (FCS). El analista documenta con un número el FCS con la siguiente información:
Formulario de cambio de software 1

- Fechas de presentación y aprobación

- Método actual
- Cambio propuesto
- Justificación
- Estimaciones de recursos

El analista envía el FCS a un comité de usuarios para su validación y al ingeniero de mantenimiento de software para la evaluación independiente del requerimiento. El ingeniero de mantenimiento evalúa el FCS con base en tres criterios:

- El esfuerzo necesario para realizar el cambio
- Los recursos informáticos necesarios
- El impacto sobre la calidad del servicio

El ingeniero hace la estimación del esfuerzo sobre la base de la taxonomía de los tipos de cambio, que es la siguiente:

Computacionales
- Operandos incorrectos en ecuación
- Uso incorrecto de paréntesis
- Ecuación incorrecta o inexacta
- Error de redondeo o truncamiento

Lógicos
- Operandos incorrectos en una expresión lógica
- Lógica fuera de secuencia.
- Variable incorrecta
- Falta de prueba lógica o condición
- Número de iteraciones incorrectas en un ciclo

De entrada
- Formato incorrecto
- Lectura de entrada desde ubicación incorrecta
- Fin de archivo no encontrado o encontrado prematuramente

Manejo de datos
- Archivo de datos no disponible
- Datos de referencia fuera de límites
- Datos de inicialización variables usados como bandera, o índice no ajustado apropiadamente
- Los datos no están apropiadamente definidos ni dimensionados
- Error de subíndices

Salida
- Datos escritos en otra ubicación
- Formato incorrecto
- Salida incompleta o perdida
- Salida confusa

Interfaz
- Interfaz de software/hardware
- Interfaz de usuario software
- Interfaz de base de datos de software

Operaciones
- Cambio de software
- Configuración del control

Mejora
- Mejora de funciones existentes
- Mejora de interfaz

Rendimiento
- Tiempo límite excedido
- Límite de almacenamiento excedido
- Código o diseño ineficiente
- Eficiencia de la red

Especificación
- Interfaz del sistema
- Especificación incorrecta o inadecuada
- Especificación de requerimientos incorrecta o inadecuada
- Entrenamiento / manual de usuario inadecuado

Taxonomía de cambios de software. Basado en Stark, 1996.

Si el análisis es 20 % mayor o menor que las estimaciones del usuario original, el usuario y el ingeniero se reúnen para resolver la diferencia. Lo anterior se realiza con el fin de aclarar el requisito.

El equipo de mantenimiento genera los FCS, que afectan a los sistemas de los que no son directamente responsables, pero con los que deben interactuar. Estos pueden ser el resultado de las modificaciones técnicas. Por ejemplo, si el sistema A está actualizando su protocolo de comunicación y se intercambia información con el sistema B, los usuarios del sistema A pueden generar un FCS para el sistema B con el fin de actualizar su protocolo de comunicaciones.

El comité de usuario clasifica el FCS ya sea como una modificación o una corrección:

- Una modificación es un cambio que no es una parte de los requisitos actuales del sistema.
- Una corrección es la mejora de fallos.

El equipo de mantenimiento asigna también una prioridad al FCS:

- **Emergencia,** si el FCS es necesario para evitar tiempo de inactividad del sistema, o si los requisitos son de alta prioridad.
- **Urgente,** si se necesita el FCS en la próxima entrega de software para un proceso en particular, o para solucionar un problema que surgió como resultado de un cambio en las operaciones.
- **Rutina** para el resto de los FCS, por ejemplo, la conversión de unidades, cambios en el valor por defecto y para arreglar la impresión.

A continuación, se pone en una cola de FCS para su incorporación en la siguiente versión del sistema que se libere. Los usuarios, el equipo de mantenimiento, y el equipo de desarrollo de sistemas negocian el contenido de la versión.

Este proceso de planificación y aprobación incluye una revisión de varias métricas: complejidad, fiabilidad, mantenibilidad, y la utilización de recursos informáticos.

El equipo de mantenimiento revisa el plan de lanzamiento y programa la liberación. El equipo de desarrollo de software termina el diseño, la programación del código, las pruebas, la instalación y garantiza la calidad de la liberación con los requerimientos del usuario.

En la siguiente tabla se muestran las métricas que reflejan los cuatro productos de información de la imagen del proceso de mantenimiento, de los que el equipo tiene control:

- Validación de los requerimientos
- Costo independiente
- Definición de contenido de entrega
- Liberación de implementación

Meta	Cuestión	Métricas
Máxima satisfacción del cliente	¿Cuántos problemas afectan al cliente?	Atrasos en el cambio actual
		Confiabilidad de software
	¿Qué tiempo tomará arreglar un problema catalogado como de emergencia o urgente?	Cambio del ciclo del tiempo y plazos aprobados, con fecha de aplicación
Minimizar costo	¿Cuál será el coste de la entrega del mantenimiento?	Coso por entrega
	¿Cómo se asignan los costos?	Costo por actividad
	¿Qué tipo de cambios se están haciendo?	Número de cambios por tipo
	¿Cuánto esfuerzo es necesario por tipo de cambio?	Días de *staff* utilizados por tipo de cambios
	¿Cuántos cambios no válidos se evalúan?	Porcentaje de cambios inválidos Solicitudes cerradas cada tres meses
Minimizar la agenda de actividades	¿Qué tan difícil es la entrega de la nueva versión?	Evaluación de la complejidad
		Mantenibilidad del software
		Utilización de recursos informáticos
	¿Cuántos cambios se hacen para la entrega de contenido planeada?	Porcentaje de contenidos de cambios por entrega
	¿Se cumple con la entrega programada?	Porcentaje de entrega en tiempo

Metas, cuestiones y métricas del mantenimiento del software. Basado en Stark, 1996.

En el presente tema, se ha desarrollado el proceso de mantenimiento del software, desde que surgen los requerimientos hasta que están satisfechos en una nueva versión del software que los contiene. Cabe destacar que, en este proceso, muchos son desechados por que no representan un problema real, tal como el de configuración que con los

parámetros adecuados funcione adecuadamente, o la falta de capacitación del usuario al usar el modulo del sistema. Se recalca, entonces, la importancia de validar los problemas reales y darles solución.

Las actividades descritas en los temas 3.1. y 3.2., más los costos, el estudio de viabilidad y los riesgos del mantenimiento de software deben estar englobadas en una metodología de gestión de mantenimiento, que administre y documente cada proceso y actividad involucradas, con el fin de apegarse a los estándares establecidos para ello, en la norma ISO/IEC 14764 (2006). La gestión del mantenimiento se desarrolla en el próximo y último tema.

3.3. Gestión del mantenimiento

Adolfo Crespo Márquez (Crespo, 2013), autor del libro *El marco de gestión de mantenimiento*, dice que la gestión del mantenimiento "se refiere a todos los planes y actividades relacionadas con el trabajo de mantenimiento" mantenimiento" p.25 Por su parte, Ned Chapin menciona en el prólogo de *Software maintenance management,* de Alain (2008), que las mejores prácticas del mantenimiento deben estar en una gestión del mantenimiento del software

Por lo anterior, la gestión del mantenimiento del software es la administración del conjunto de actividades y procesos necesarios para llevar a cabo las modificaciones pertinentes, que permitan al sistema seguir siendo funcional sin desviarse de su propósito original.

En la guía SWEBOK, de Bourque y Fairley (2014), se establece que las actividades previas al mantenimiento son aquellas que se realizan antes de la entrega del sistema de software, que incluyen la planificación de las operaciones después de la entrega, así como la determinación de la mantenibilidad y la logística de las actividades de transición.

Las actividades posteriores a la entrega son aquellas que incluyen la modificación del software, la capacitación, operación o interfaz con una mesa de ayuda.

La guía presenta un desglose de tópicos de la gestión del mantenimiento de software, de los cuales, muchos se han desarrollado en los temas anteriores, para revisar la guía, consulta el documento Tópicos de la gestión del mantenimiento de software.

La guía contempla todas las actividades previas y posteriores a la entrega de un producto de software con respecto al mantenimiento. Debe dejarse muy claro que es necesaria una adecuada administración y control (gestión del mantenimiento de software) para asegurar que el sistema sea funcional, hasta donde sea posible, mientras tenga la característica de mantenibilidad.

A continuación, se detallarán los costos del mantenimiento, el estudio de viabilidad y el análisis de riesgos.

3.3.1. Costos de un plan de mantenimiento de sistemas de software

Los costos del mantenimiento suelen ser altos porque cada modificación, mejora o requerimiento se aplica a todo el ciclo de vida, y se requiere el personal calificado para mantener la calidad.

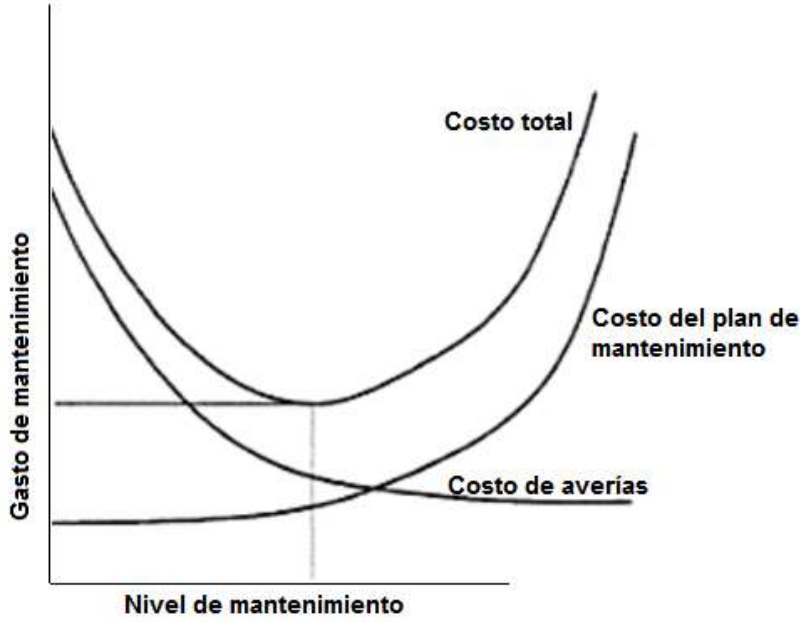
Es necesario tener claro los tipos del mantenimiento para establecer un plan adecuado. Deben considerarse los siguientes factores que influyen para ser eficientes en el manejo de los recursos:

- Entorno de funcionamiento (hardware y software necesarios)
- Entorno organizacional (políticas, competencia, proceso, producto y personal)

Respecto de la estabilidad del equipo, el costo del mantenimiento se reduce si el equipo de desarrollo es el mismo, porque ya se conoce el sistema, se ahorra tiempo, dinero y esfuerzo.

Con respecto a la responsabilidad contractual, el costo se reduce si se establece con los desarrolladores del sistema, ya que se generan incentivos para participar en el diseño de cambios futuros.

En los sistemas con un considerable tiempo de vida se observa una degradación de su estructura original, se vuelven difíciles de entender y cambiar.



Costo del mantenimiento. Fuente: Gómez de León, 1998, p.36.

Sin embargo, hay una serie de herramientas para estimar tanto el costo y el riesgo con el fin de minimizarlos, sin sacrificar calidad, por supuesto. Lo siguiente es apegarse al presupuesto lo más que se pueda y hacer los ajustes si se observan desviaciones.

Para establecer el costo del mantenimiento, la norma IEEE 14764 (ISO/IEC 14764, 2006). establece que lo ideal es la combinación de un modelo paramétrico y la experiencia. Procedimiento que se desarrollará en el siguiente subtema.

3.3.2. Viabilidad y análisis de riesgo

¿Cuántas veces se cree que se tiene una gran idea y no se le ve ningún inconveniente? Sin embargo, para cualquier proyecto que implique inversión, y para cuestiones de mantenimiento de software, es necesario realizar un estudio de viabilidad y análisis de riesgos para verificar si las modificaciones generadas de los nuevos requerimientos están al alcance en cuestiones técnicas, operativas y económicas. La información que muestran los estudios son el costo, los riesgos inherentes, el tiempo aproximado y el número de personas necesarias para el proyecto de mantenimiento.

El estudio debe hacerse, cuando menos, para una segunda opción y, al final, para seleccionar una de las alternativas; es parte de la gestión del mantenimiento que se debe documentar y presentar.

Viabilidad

Según el Diccionario de la Real Academia Española (2014c), la palabra viabilidad significa:

- Cualidad de viable
- Condición del camino o vía por donde se puede transitar

La viabilidad, en el mantenimiento del software, es el resultado de un estudio de factibilidad que determina si un proyecto es viable o no, con respecto a una serie de factores que influyen en el tipo de mantenimiento, para tratar de minimizar los costos y riesgos que implica. Los factores de análisis de viabilidad que se mencionan a continuación están basados en la taxonomía de riesgos de Kuna (2008, p. 17).

Análisis de riesgos

Según el Diccionario de la Real Academia Española (2014b), la palabra riesgo significa:

- Contingencia o proximidad de un daño
- Cada una de las contingencias que pueden ser objeto de un contrato de seguro
- Dicho de acometer una empresa o de celebrar un contrato, sometiéndose al influjo de suerte o evento, sin poder reclamar por la acción de éstos
- Estar expuesto a perderse o a no verificarse

Los riesgos en el mantenimiento del software son que el:

- costo del mantenimiento sobrepase el del sistema mismo
- mantenimiento haga perder mantenibilidad al sistema
- mantenimiento acelere el retiro del sistema
- mantenimiento afecte el funcionamiento general de los componentes del sistema

El análisis de riesgos consiste en estudiar los peligros y el impacto que podrían tener sobre el sistema de software, crear las estrategias para minimizarlas, apoyadas en el estudio de viabilidad, el cual presenta cuando menos dos alternativas de solución.

El riesgo siempre va a existir, no se puede evitar, pero sí minimizar. Para eso hay técnicas de riesgos que se pueden aplicar, considerando siempre la experiencia del experto. Las técnicas incluyen:

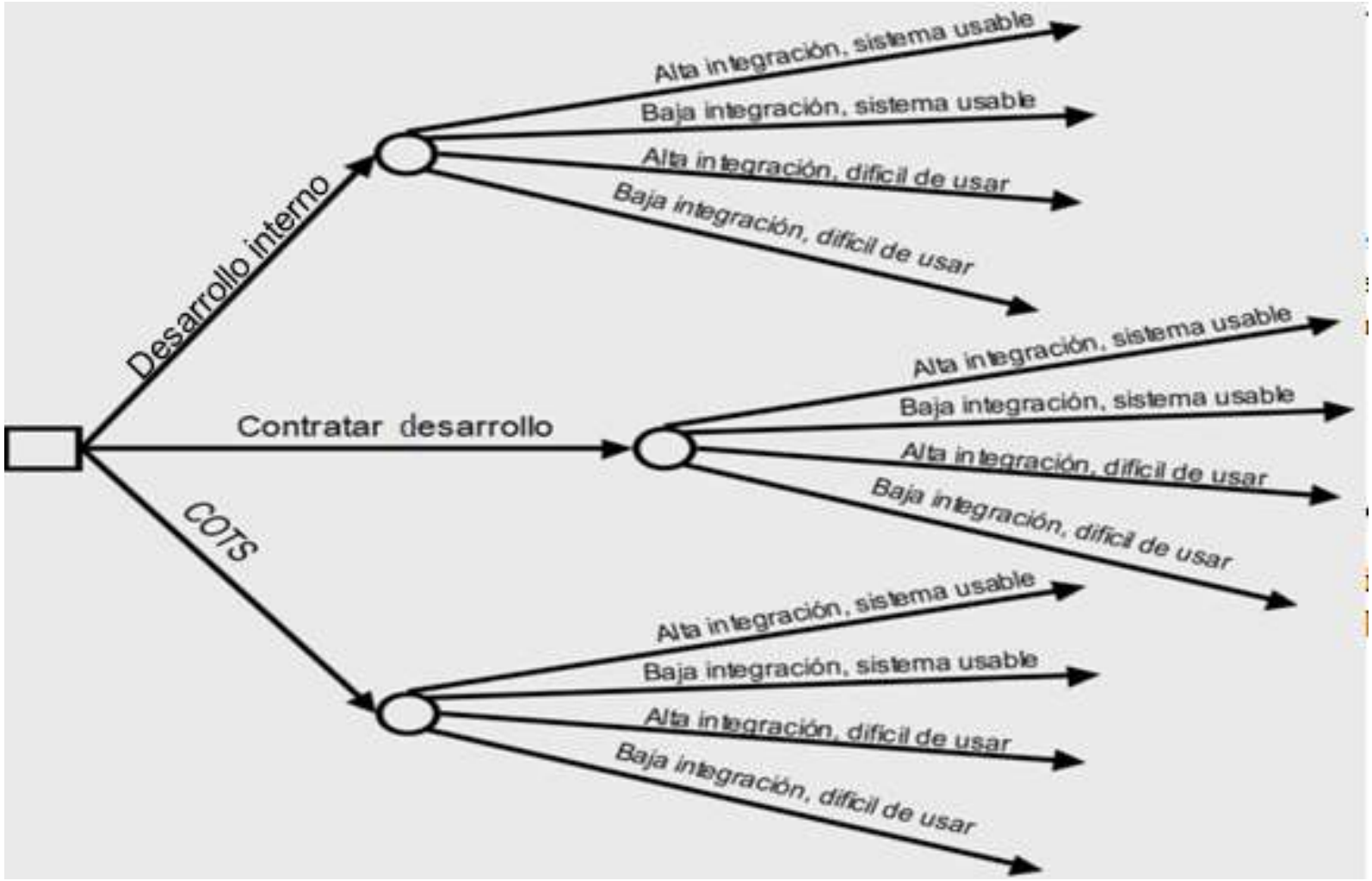
- **Toma de decisiones.** Proceso de análisis y escogencia entre diversas alternativas para determinar un curso a seguir (Chiavenato, 2001).
- **El análisis de Monte Carlo.** Es una simulación para tomar decisiones, en la cual, las distribuciones de probabilidad describen ciertos elementos económicos. Este método utiliza las distribuciones, que pueden ser empíricas o teóricas, para generar resultados aleatorios que, a su vez, se combinan con los resultados técnico-económicos de un estudio de factibilidad para tomar decisiones con respecto al proyecto. Mientras más simulaciones se efectúen, se espera que el resultado sea más confiable (Baca, 2010).
- **Los árboles de decisión**. Es un enfoque por medio del cual es posible realizar un análisis de cómo las decisiones tomadas en el presente pueden afectar las futuras, ya que muchas de las primeras no consideran las consecuencias a largo plazo. Este enfoque se utiliza cuando es importante considerar las secuencias de decisión y se conocen las probabilidades de que sucedan en el futuro los eventos bajo análisis (Baca, 2010).

A continuación, se expone un ejemplo de aplicación de los árboles de decisión (Morales, 2008):

La gerencia de sistemas de información de una cadena de tiendas de zapatos se ha planteado sustituir el software de punto de venta de las 14 cajas (8 tiendas) en las que está instalado. Esto, a pesar de que recientemente se ha renovado la licencia de uso del software por un año más.

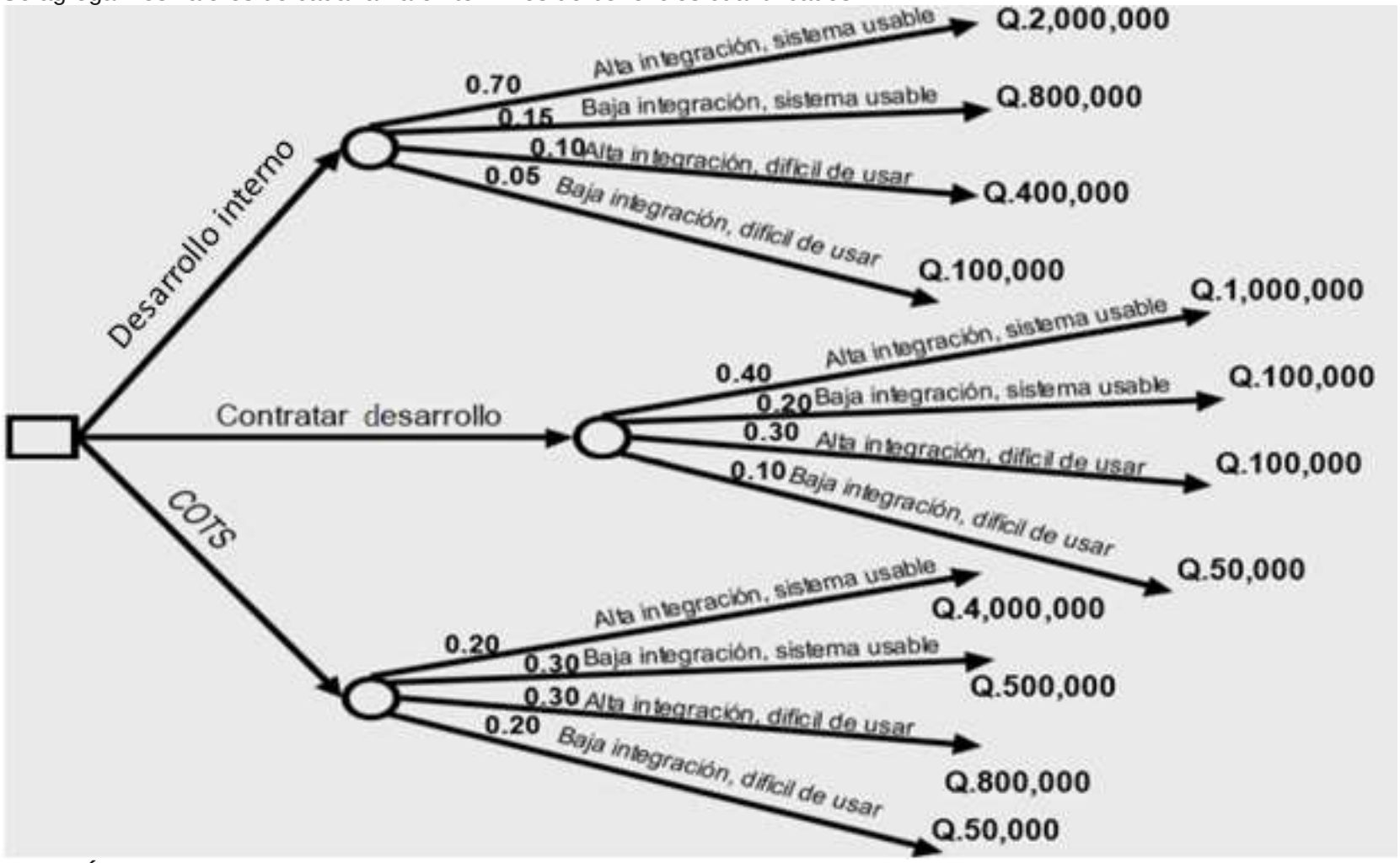
Las razones son variadas, van desde la escasa integración que el software actual permite con el sistema ERP de la empresa, hasta las dificultades para usarlo y para entrenar al personal de nuevo ingreso.

El gerente no sólo debe decidir si recurre a un sistema COTS (soluciones listas para usar que abundan en el mercado), si contrata un desarrollo externo (outsourcing) o si lleva a cabo todo el desarrollo internamente, para lo cual necesitaría ampliar su plantilla de desarrolladores y, probablemente, contratar de manera temporal a un administrador del proyecto. También debe justificar su decisión tomando en cuenta que ya se hizo la inversión en licencias por un año más. El árbol de decisión siguiente plantea estas alternativas:



Árbol de decisión con tres alternativas. Fuente: Morales, 2008.

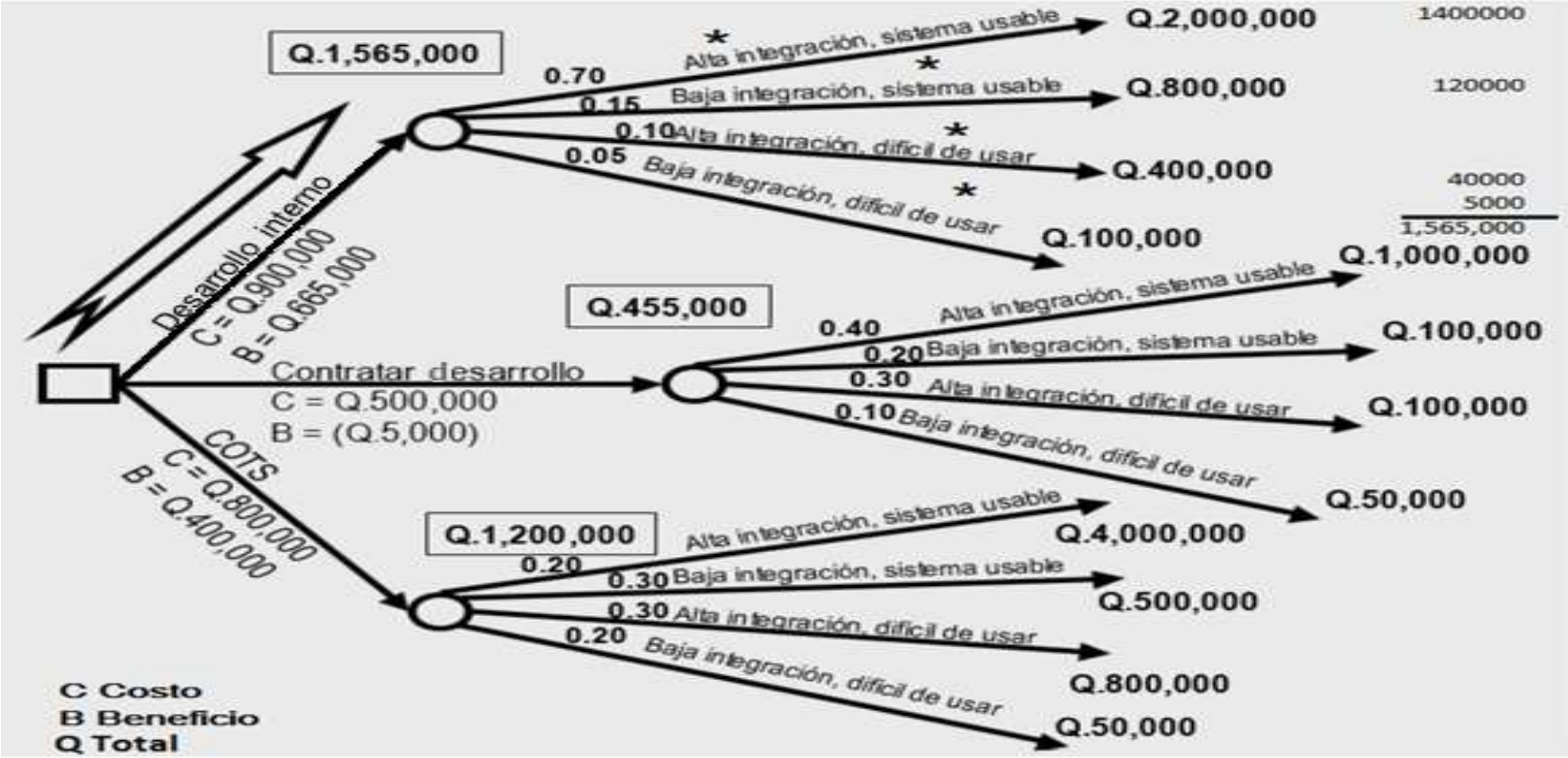
Se agregan los valores de cada rama en términos de beneficios cuantificados.



Árbol de decisión con valores y porcentaje de probabilidad de tres alternativas. Fuente: Morales, 2008.

Se determina el costo/beneficio de cada alternativa, se observa que el desarrollo interno es la mejor opción, se realiza previamente el análisis de costos de cada una, probablemente, mediante cotizaciones de productos y servicios, además del

análisis de personal a contratar, modalidad de contratación, etcétera. La mejor opción es la que presenta los mayores beneficios, como se observa en la siguiente imagen:



Árbol de decisión con costo/beneficio de tres alternativas. Fuente: Morales, 2008.

Valor esperado de la información perfecta. Es la cantidad o comportamiento deseado que se puede mejorar, cuando se sabe de antemano qué evento va a ocurrir (Krajewski, 2000).

Cuando se tiene una serie de alternativas de solución a un requerimiento, para seleccionar la mejor alternativa, se realiza un proceso de evaluación, se minimiza el impacto negativo que pudiera surgir en un momento dado, se analizan los procedimientos, los pros y los contras mediante un diagrama de flujo como el siguiente:

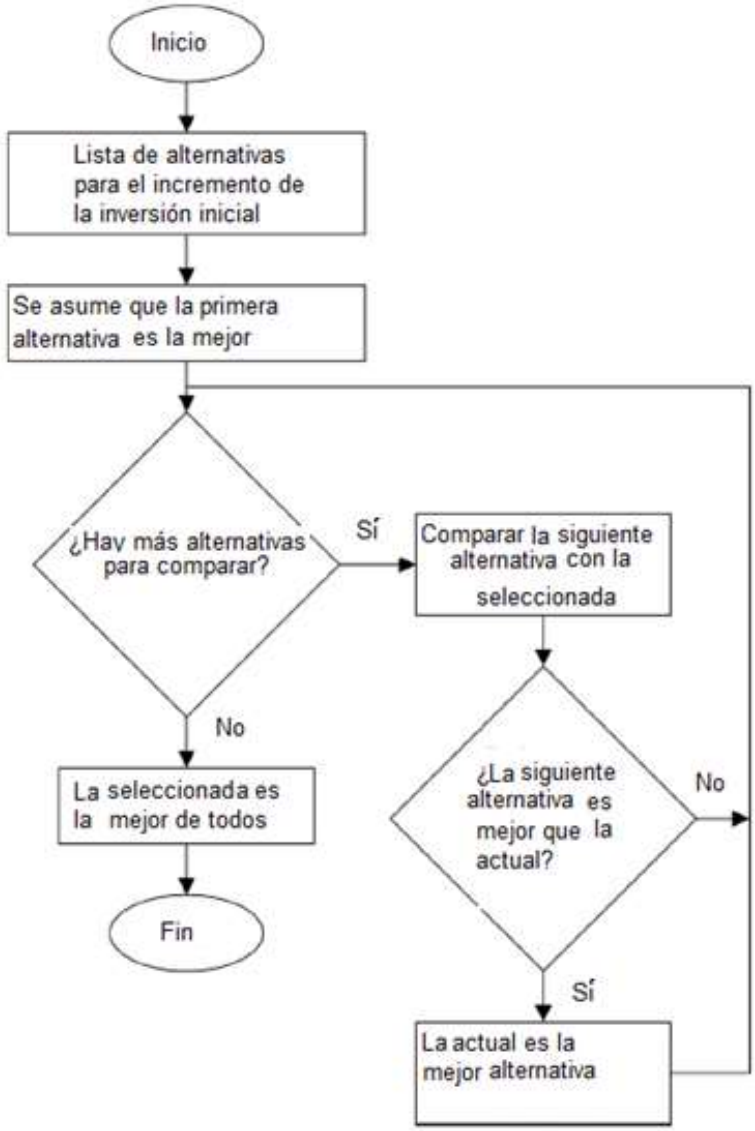


Diagrama de flujo de selección de la mejor alternativa.

Los pasos para desarrollar un estudio de viabilidad y análisis de riesgos son los siguientes:

1. Identificar los requerimientos

2. Identificar el tipo de mantenimiento
3. Identificar la solución a tales requerimientos
4. Identificación de riesgos y medidas de prevención
5. Estudio de factibilidad operativa
6. Estudio de factibilidad técnica
7. Estudio de factibilidad económica
8. Estudio de factibilidad de tiempo
9. Estudio de factibilidad de derechos de autor
10. Selección de alternativas
11. Conclusiones del estudio de viabilidad y análisis de riesgos

Los siguientes son documentos que se generan con base en los resultados de los estudios de factibilidad, y respaldan la decisión final por una alternativa en particular:

Anexo A. Análisis costo/beneficio
Anexo B. Punto de equilibrio
Anexo C. Análisis de amortización
Anexo D. Cronograma de actividades
Anexo E. Métrica aplicada

A continuación, se expone el siguiente ejemplo en relación con un sistema de control de activos fijos (SCAF) de la Secretaría de Hacienda (SH) que tiene un año de operación, pero, por el cambio de administración se incorporó nuevo personal, y el que se encontraba laborando se distribuyó en otras secretarías.

1. Identificar los requerimientos. Los nuevos usuarios **detectan** que al sistema de control de activos le hace falta una serie de funcionalidades como:

- El sistema debe reportar los activos fijos para su donación al 100% de su periodo de vida útil.
- Realizar la bitácora de cada activo fijo, desde su ingreso hasta que ya no sea utilizado por el personal de la SH.

2. Identificar el tipo de mantenimiento. Según los requerimientos generados por los usuarios del sistema, el tipo de mantenimiento es perfectivo, ya que busca obtener **nuevas** funcionalidades o características no contempladas en el momento de la

implementación del software. El mantenimiento perfectivo adapta la aplicación a los requerimientos.

3. Identificar la solución a tales requerimientos. Se ha **analizado** la situación y se ha determinado una solución a las nuevas necesidades de los usuarios.

El sistema debe permitir:

- Generar reportes de los activos fijos mediante un criterio de selección, que pueden basarse en el número de AF (activo fijo), descripción, ubicación, etcétera.
- Presentar una bitácora que refleje el historial de vida de cada activo fijo.
- Registrar los movimientos de cada activo fijo (cuando se realiza un traslado o cambio de usuario).
- Generar un respaldo de los datos.

Se requiere integrar algunas restricciones del usuario como:

Debe de estar orientado a trabajar sobre una arquitectura de red centralizada y de dominio público para los usuarios de departamento de sistemas de la Secretaría de Hacienda.

- Adaptarlo a la plataforma Windows 8
- Utilizar SQL Server 2012 como administrador de base de datos y Visual Basic 2012 para el diseño de la interfaz.

Requisitos del sistema. Para el desarrollo de los módulos y para aprovechar las capacidades de información contenida dentro de su registro de base de datos, se determinaron los siguientes módulos como necesarios para satisfacer los nuevos requerimientos, las condiciones técnicas del servidor y el software necesario para mantener la funcionabilidad.

Módulos del sistema:

- Actualización de datos
- Generación de reportes

Equipo de hardware:

- Número de procesador E3-1125C
- Cantidad de núcleos 4
- Cantidad de subprocesos 8
- Velocidad del reloj 2 GHz
- Memoria caché L 38 MB
- Conjunto de instrucciones 64-bit
- Extensiones de conjunto de instrucciones AVX, AES-NI
- Disco duro de 1 TB
- Un quemador para respaldar la información

Software:

- SQL Server y Visual Basic 2012
- Sistema operativo Windows 8

Las condiciones del local actual son:

Cuenta con un sistema eléctrico polarizado y ambiente entre 20º C y 22º C para garantizar un mejor aprovechamiento y seguridad, tanto del hardware como de los usuarios.

4. Identificación de riesgos y medidas de prevención. Como parte del **análisis de riesgos**, se identificaron los peligros. Después de analizarlos, se establecieron las diferentes medidas para su control en el proyecto de mantenimiento del software del SCAF, que se observan en la siguiente tabla:

Tipo de riesgo	Riesgo	Medidas preventivas	Medidas correctivas
Del proyecto	Siniestros	Ninguna	Restablecer la funcionalidad de los equipos dañados a la mayor brevedad posible

	Planteamiento de requerimientos adicionales por parte del usuario	Establecer una buena comunicación con el usuario de manera que se instituyan todos los parámetros antes de concluir el estudio de factibilidad	Replantear el estudio de factibilidad y hacerle notar al usuario las consecuencias de esta medida
Técnico	Presencia de virus	Instalar antivirus y actualizarlo periódicamente para proteger las computadoras	Eliminar virus
	Equipos dañados durante el desarrollo del proyecto	Realizar periódicamente mantenimiento preventivo para los equipos	Realizar mantenimiento correctivo
	Pérdida de datos	Configurar SQL Server para que cree respaldos periódicamente Crear respaldos del sistema (base de datos y programa), en cualquier dispositivo de almacenamiento de datos	Crear o ingresar nuevamente los datos, en caso de ser posible
Tecnológico	Necesidad de una interfaz propia para el administrador del sistema informático	Establecer permisos para cada usuario del sistema	Desarrollar la interfaz
Asociado con el tamaño y experiencia de la plantilla del personal	Pérdida de un desarrollador de software	Contratar personal confiable	Buscar un nuevo personal Redistribuir las actividades
	Falta de experiencia por parte de los desarrolladores en el uso del software	Garantizar que todos los desarrolladores tengan los conocimientos necesarios para el desarrollo del sistema	Dar capacitación intensiva al personal

Análisis de riesgos, medidas preventivas y correctivas de mantenimiento del sistema de activos fijos.

Como se observa, los riesgos son claros pero también las medidas preventivas y correctivas. Si bien los riesgos no se podrán evitar, sí se podrán minimizar para no impactar de forma negativa en el proyecto y la gestión del mantenimiento del software (reingeniería).

5. Estudio de factibilidad operativa. Este análisis da inicio al estudio de viabilidad del proyecto de mantenimiento para el SCAF.

La factibilidad operativa es la determinación de la probabilidad de que un nuevo sistema se use como se supone. Para eso es necesario el desarrollo de dos módulos más:

- Actualización de datos
- Generación de reportes

Debe seguirse la misma interfaz amigable que los módulos existentes en el SCAF de tal forma que, sin mucha dificultad, el usuario pueda adaptarse y aprovechar al máximo las facilidades que brinda, con lo que se ahorra gran parte de su tiempo y permite la realización de otras actividades.

A continuación, se expone el ejemplo de un sistema que funcionará en red, al cual se accederá a través de la página oficial de la Secretaría de Hacienda. Los usuarios podrán visualizar la información que soliciten; sin embargo, no se les permitirá alterarla si no cuentan con los permisos necesarios.

Al implantar este sistema en la Secretaría de Hacienda, se facilitará el trabajo del encargado de llevar el control de los activos fijos y se reducirá el tiempo que se invierte en la elaboración de los informes de inventario solicitado.

La Secretaría de Hacienda y sus directivos se encuentran anuentes a aceptar los cambios y mejoras que el sistema ofrezca dentro del entorno de su organización. Se llega a la conclusión de que el sistema es factible operativamente, ya que se cuenta con una base de desarrollo (sistema existente SCAF) y la aceptación de los directivos.

6. Estudio de factibilidad técnica. Determina si el equipo y el software están disponibles (o si este último puede desarrollarse), y si existen las capacidades técnicas requeridas por cada alternativa del diseño que se esté considerando.

A continuación, se realiza el estudio de factibilidad técnica para el proyecto de mantenimiento del SCAF.

Actualmente, la Secretaría de Hacienda cuenta con 15 computadoras, de las cuales tres serán asignadas a los desarrolladores del sistema. Éstas se utilizan para ejecutar trámites internos y se encuentran conectadas a la red local a la que se puede tener acceso a través de la Web, lo que permite compartir e intercambiar información.

A continuación, se detalla una descripción del equipo con que se cuenta:

Hardware

15 computadoras personales marca DELL, modelo Opti Plex GX200 con las siguientes características:

Procesador: Intel Pentium 4 de 2.4 GHz
Memoria RAM: 512 MB DDR
Disco duro: 1HDD de 40 GB
Monitor: 15”
Teclado: Standard PS–2
Mouse: OMEGA PS–2
DVD-RW: LG
Unidad de disco flexible 3½”
Estabilizador ProNet de1000 W
2 Impresoras:
HP color LaserJet 8550-PS (compartida para todos los usuarios de la red local)
Canon Bubblet -Jet S450
Matricial EPSON LQ 570-e (ubicada en recepción)

Plataforma de software actual

Sistema operativo: MS Windows XP Pro
MS Windows 2000 Pro + Service Pack 4

Programas instalados:

MS Office XP
DoNet
SQL Server EnterPrice Manager + Service Pack 4
Visual Basic 6.0
WinZip 9.0

WinRar 3.3
Adobe Acrobat 6.0
Nero Burning ROM 6.0.0.9

Para cumplir con los requerimientos del SCAF, es necesario hacer las siguientes adquisiciones:

- El servidor
- El sistema operativo Windows 8
- Visual Basic 2012
- SQL Server 2012

Se debe hacer el presupuesto de las nuevas adquisiciones; sin embargo, se considera que es un paso lógico en la actualización del equipo de cómputo y software, y que se tendrá un periodo de vida útil de cinco años antes de pensar en cambiar nuevamente la tecnología.

7. Estudio de factibilidad económica. Incluye el análisis de costos y beneficios asociados con cada alternativa del proyecto. Todos los costos y beneficios de adquirir y operar cada alternativa se identifican y se comparan.

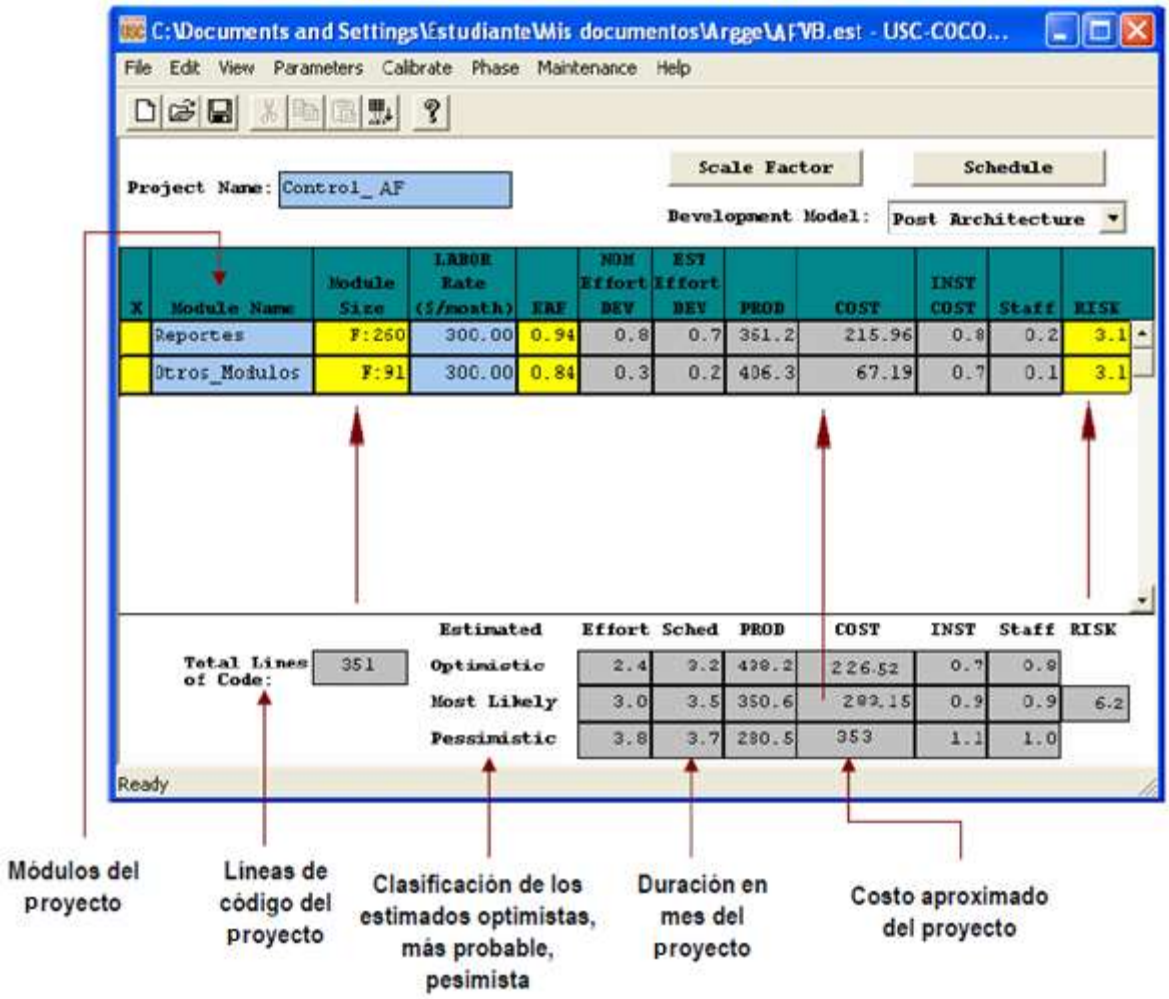
Ejemplo de alternativa 1: SQL Server y Visual Basic 2012

Costo de desarrollo del sistema

El personal de desarrollo ha sido contratado con los conocimientos necesarios en programación y administración de servidores.

2 desarrolladores	750.00 al mes
1 analista	750.00 al mes

Para el desarrollo de la factibilidad económica se utilizará la herramienta libre COCOMO® II.2000.4, creada por alumnos de la Universidad del Sur de California USC (University of Southern California), disponible en el sitio oficial, para diferentes plataformas.



Módulos para el proyecto alternativa 1

Fórmula del costo real y el tiempo real:

$$\text{Real} = \frac{\text{Optimista} + 4 * \text{Más probable} + \text{Pesimista}}{6}$$

Costo real = 226.52+4*283.15+353/6

Costo real = 285.35

Tiempo real = 3.2+4*3.5+3.7/6

Tiempo real= 3.48

Equipo de cómputo y software:

El servidor	726.00
El sistema operativo Windows 8	120.00

Visual Basic 2012	438.00
SQL Server 2012	859.00

Insumos:

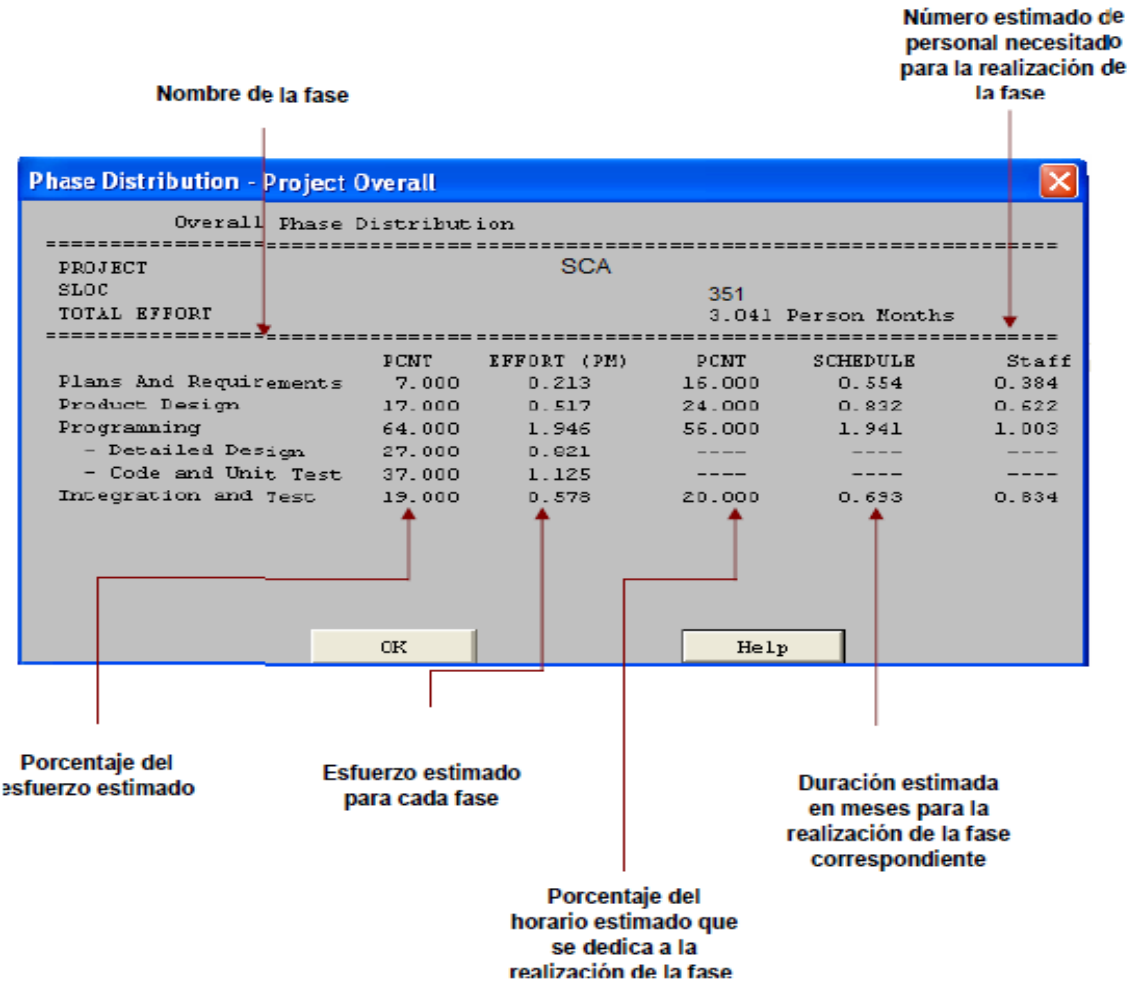
2 Paquetes de papel	$ 6.00
1 Cajas de lapiceros	$ 3.00
6 Lápiz mecánicos	$ 8.00
10 Cajas de minas 0.5	$ 6.00
Llamadas telefónicas	$ 40.00
Impresión	$ 20.00

Costo mensual de operación del sistema

Costos variables

1Paquete de papel	$ 3.00
1 Cartucho de tinta negra	$ 20.00
1 Cartucho de tinta color	$ 35.00

La programación de las fases quedó de la siguiente forma en la ventana *Phase distribution Project Overall* del programa COCOMO II:



Ventana de distribución de fases del proyecto en general para la alternativa 1

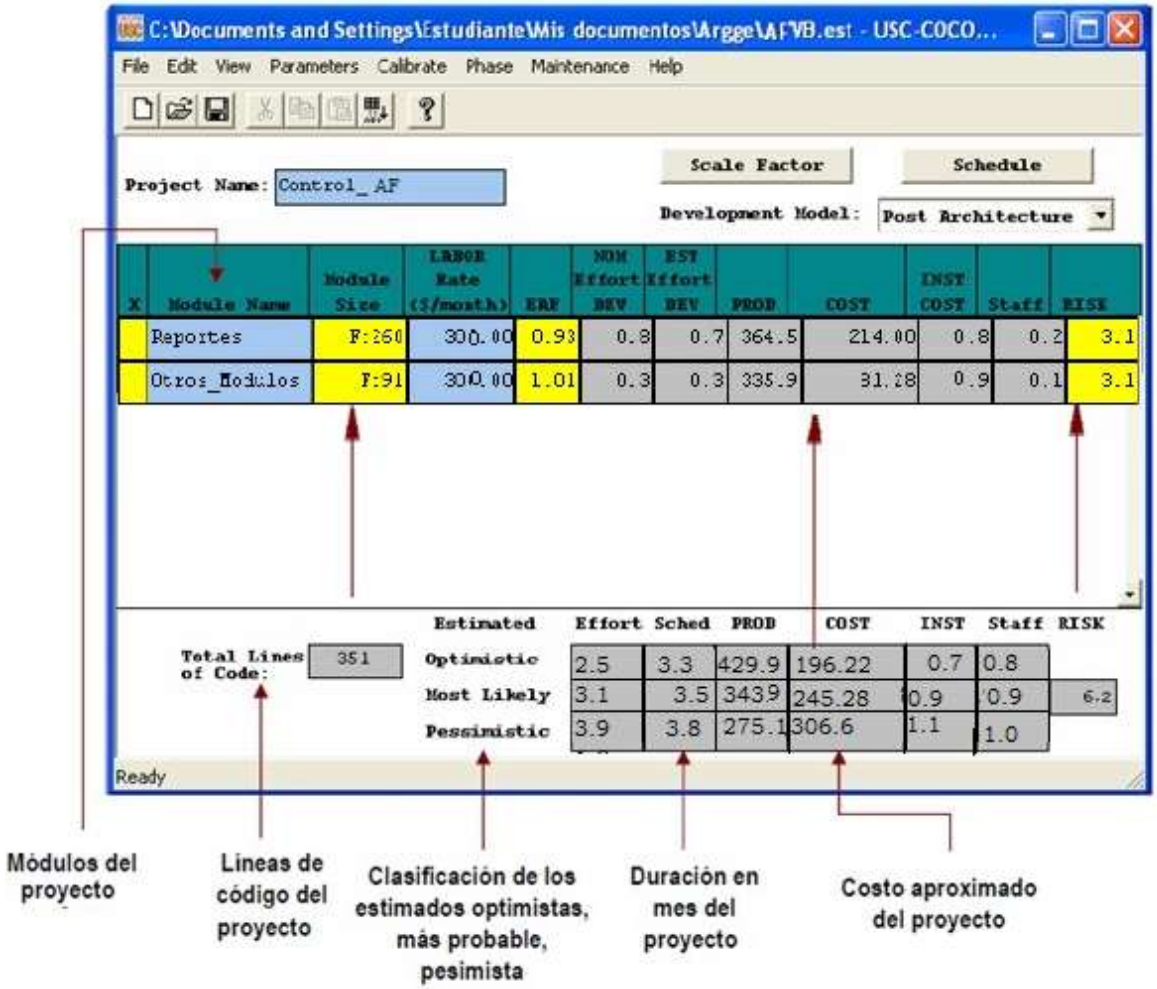
Ejemplo de alternativa 2: SQL Server 2012 y .Net

Costo de desarrollo del sistema

Construcción

El personal de desarrollo ha sido contratado con los conocimientos necesarios en programación y administración de servidores.

2 desarrolladores	790.00 al mes
1 analista	790.00 al mes



Módulos para el proyecto alternativa 2.

Fórmula del costo real y tiempo real

$$\text{Real} = \frac{\text{Optimista} + 4 * \text{Más probable} + \text{Pesimista}}{6}$$

Costo real = 196.22+4*245.28+306.6/6
Costo real = 247.32
Tiempo real = 3.3+4*3.5+3.8/6
Tiempo real= 3.51

Insumos:

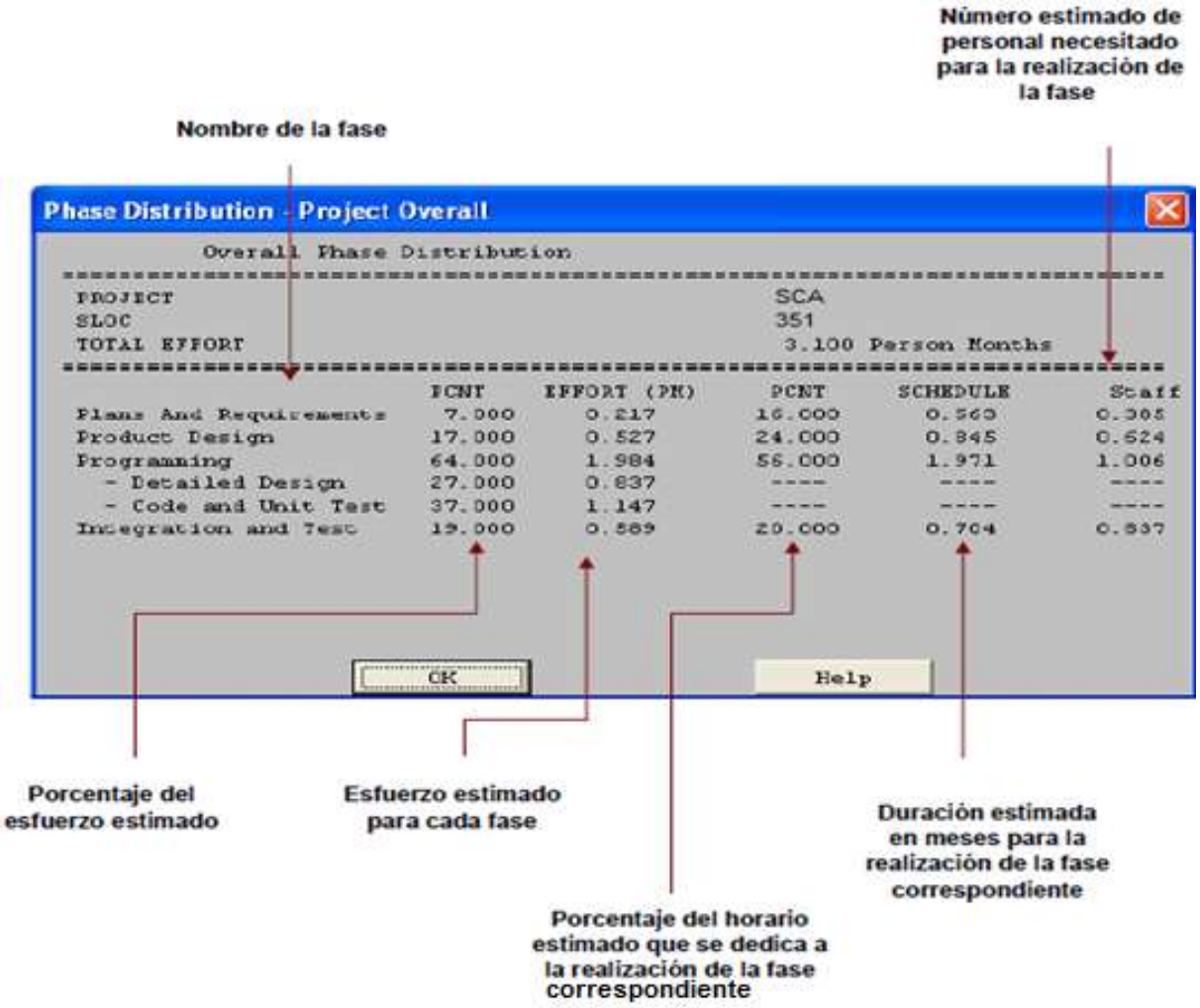
2 Paquetes de papel	$ 6.00
1 Cajas de lapiceros	$ 3.00
6 Lápiz mecánicos	$ 8.00
10 Cajas de minas 0.5	$ 6.00
Llamadas telefónicas	$ 40.00
Impresión	$ 20.00

Costo mensual de operación del sistema

Costos variables

1 Paquetes de papel	$ 3.00
1 Cartucho de tinta negro	$ 20.00
1 Cartucho de tinta color	$ 35.00

La programación de las fases se integró de la siguiente forma:



Ventana de distribución de fases del proyecto en general para la alternativa 2.

Los beneficios de la integración de dos módulos más al sistema de control de activos son:

Tangibles

- Disminución de errores
- Mejor utilización del tiempo disponible
- Reducción de los costos variables
- Manejo adecuado de la información

Intangibles

- Satisfacción del cliente
- Mejora en la toma de decisión

Los costos están a la vista; sin embargo, la decisión de una alternativa se maneja como un paso que se explica más adelante.

8. Estudio de factibilidad de tiempo. Se refiere a la estimación del tiempo necesario para cada fase del proyecto de mantenimiento del software, contiene la planificación de cada una de las tareas que deben llevarse a cabo para conseguir los objetivos del proyecto, que se contabilizó en un aproximado de 3.5 meses de trabajo (ver alternativa 2, tiempo real), desde el inicio del proyecto hasta su conclusión.

A los desarrolladores del proyecto les interesa tener una visión amplia de lo que es todo el entorno del problema y sus posibles soluciones.

9. Estudio de factibilidad, derechos de autor. Sirve para verificar que no se violen los derechos de autor y ni se cometa una infracción por omisión. En el presente proyecto se respeta y se hace cumplir la ley de los derechos de autor, junto con todas las prerrogativas que dicha ley establece. El objetivo es evitar multas o demandas a la hora de implementar el sistema. El departamento correspondiente adquirirá los permisos de derechos de autor de cada software mencionado en los requerimientos, a través de la compra de las licencias.

Una vez aprobado el proyecto SCAF, se tendrá derecho de establecer las cláusulas de contratación de los desarrolladores del sistema.

10. Selección de alternativa. Es una acción del análisis de riesgos. Es parte del estudio de viabilidad, que seleccionará precisamente la mejor alternativa, después de analizar concienzudamente los reportes de factibilidad y los análisis de costo-beneficio, ya que infiere directamente en que el proyecto se logre, de forma satisfactoria, en tiempo, costo y funcionabilidad. Las acciones anteriores son parte de la gestión del mantenimiento.

Para realizar la selección de la alternativa se tomó en cuenta, tanto los requerimientos del usuario como los del sistema. Por tal motivo, se ha seleccionado la alternativa 1, ya que ofrece seguridad; además, se puede desarrollar una interfaz gráfica sencilla y agradable para el usuario, e incluye mayor capacidad de almacenamiento de datos.

Alternativa	Descripción	Costo de la inversión (en términos monetarios)
1: SQL Server con Visual Basic	Esta alternativa pretende utilizar como motor de base de datos SQL Server, que es una herramienta diseñada para el manejo de bases de datos y para el diseño de interfaz Visual Basic.	$ 3, 220.00
2: SQL Server con .NET	Esta alternativa pretende utilizar como motor de base de datos SQL Server, y .Net como diseñador de interfaz gráfica.	$ 3, 373.00

Análisis de alternativas. Basado en PAE, 2013.

Alternativa	Ventajas	Desventajas
1SQL Server con Visual Basic	• Precio • Presenta interfaces que hacen más fácil la labor de implementación y desarrollo. • Genera archivo ejecutable para que los usuarios no manipulen el código del programa.	• Limitación en el enlace de bases de datos que no incluyan su versión. Tendrá que realizarse la conversión manualmente. • Se pueden desarrollar aplicaciones que funcionen en red, pero no cuenta con los suficientes soportes.
2SQL Server con .NET	• Mayor seguridad • Mucha flexibilidad en la manipulación de bases de datos • DoNet permite la creación de un archivo ejecutable. • Tanto Microsoft SQL Server como DoNet son lenguajes diseñados especialmente para trabajar en red. Ofrecen suficientes soportes	• Precio, puesto que se pagará más a los desarrolladores que trabajen en esta aplicación. • DoNet presenta un grado de complejidad mayor que Visual Basic.

	para su implementación.	

Comparación de alternativas. Basado en PAE, 2013.

Anexo A. Análisis del costo/beneficio

El análisis del costo/beneficio es un comparativo de las factibilidades donde se debe observar siempre que el beneficio es mayor que el costo, ya que si es al contrario el esfuerzo del proyecto no vale pena y se evita un gasto innecesario.

Es importante hacer un comparativo de cuáles son los gastos y costes actuales, y cuáles serían después de implementarse los dos módulos propuestos.

El análisis de costo/beneficio proporciona una medida de los gastos que se tienen en la realización de un proyecto, y compara dicha previsión de costes con los beneficios esperados de la realización de dicho proyecto. Según la siguiente tabla, se observa una disminución de gastos y costes y un aumento considerable de los beneficios.

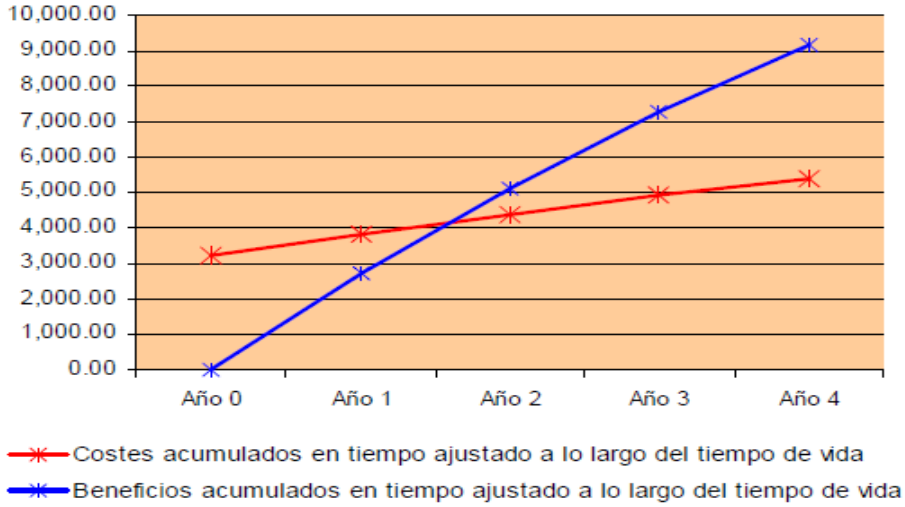
Detalles	Actual operación del SCAF	Operación del SCAF con dos módulos más
Horas extras	$180.00	$0.00
Gastos de oficina	$130.00	$58.00
Total	$310.00	$58.00
Beneficio	$0.00	$252.00

Análisis costo-beneficio. Basado en PAE, 2013.

El resultado plasmado en la tabla anterior, señala que la implementación de los dos módulos propuestos como parte del mantenimiento del sistema de software es beneficiosa.

Anexo B. Punto de equilibrio

El gráfico de punto de equilibrio muestra el lugar en que los costes y los beneficios son iguales, para, de ahí en adelante, observar un crecimiento de los segundos a lo largo de un periodo de tiempo.



Punto de equilibrio costes/beneficios

Anexo C. Análisis de amortización

La fórmula del valor presente es la siguiente:

$$VP_n = 1/(1+i)^n$$

$i = 12\%$

Con una tasa de amortización del 12%, la tasa de amortización es el factor en el cual el valor del sistema se deprecia lo largo de un periodo determinado.

Costo del sistema 3220	Fórmula	1/(1+.12)^1	1/(1+.12)^2	1/(1+.12)^3	1/(1+.12)^4
Costo operación y mantenimiento-696	Aplicación de la fórmula →Factor de amortización	0.893	0.797	0.712	0.636
	Aplicación del factor	-696*0.893	-696*0.797	-696*.712	-696*.636
Costo, operación y mantenimiento real		-621.429	-554.847	-495.399	-442.321
	Costo acumulado	-3220-621.43	-3841.43-554.84	-4396.28-495.40	-4891.67-442.32
	-3220	-3841.43	-4396.28	-4891.67	-5334.00

Tabla de referencia del análisis de amortización

En los datos de la siguiente tabla se observa el comportamiento de dos variables importantes a considerar: los costes acumulados en tiempo ajustado, que va disminuyendo año con año, y los beneficios acumulados en tiempo, que van aumentando:

Descripción del flujo de efectivo	Año 0	Año 1	Año 2	Año 3	Año 4
Costo de análisis, diseño e implementación	-3,220.00				
Costo de operación y mantenimiento		-696.00	-696.00	-696.00	-696.00
Factor de descuento al 12%	1.00	0.893	0.797	0.712	0.636
Coste en tiempo ajustado (ajustado al valor actual)	-3,220	-621.528	-554.712	-495.552	-442.656
Costes acumulados en tiempo ajustados a lo largo del tiempo de vida	-3,220	-3,841.52	-4,396.24	-4,950.95	-5,393.60
Beneficios obtenidos del funcionamiento del nuevo sistema	0	3,024.00	3,024.00	3,024.00	3,024.00
Factor de descuento al 12%	1.000	0.893	0.797	0.712	0.636
Beneficios en tiempo ajustado (valor real o actual)	0	2,700.432	2,410.128	2,153.088	1,923.264
Beneficios acumulados en tiempo ajustado a lo largo del tiempo de vida	0	2,700.432	5,110.56	7,263.648	9,186.912
Costes + beneficios acumulados en tiempo ajustado a lo largo del tiempo de vida	-3,220	-1,141.088	714.32	2,312.698	3,793.312
Periodo de amortización en tiempo ajustado|----------|				2 años	

Análisis de amortización. Basado en Bassi, 2003, p. 3.

Anexo D. Cronograma de actividades

La grafica muestra la planeación en tiempos de cada una de las fases que comprende la realización del proyecto de mantenimiento del software.

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Plantear los objetivos	1 día	vie 18/03/14	vie 18/03/14
2	Establecer los requerimientos del usuario	2 días	lun 21/03/14	mar 22/03/14
3	Establecer los requisitos del sistema	2 días	mar 22/03/14	mié 23/03/14
4	Realizar el estudio de factibilidad	7 días	mié 23/03/14	jue 31/03/14
5	Revisión del estudio de factibilidad	2 días	vie 01/04/14	lun 04/04/14
6	Presentación y entrega del documento	1 día	mar 05/04/14	mar 05/04/14
7	Realizar el análisis orientado a datos	8 días	mié 06/04/14	vie 15/04/14
8	Elaborar el diseño orientado a datos	8 días	lun 18/04/14	mié 27/04/14
9	Revisión del adelanto del AOD y DOD	1 día	lun 25/04/14	lun 25/04/14
10	Presentación del análisis y diseño orientado a datos	1 día	vie 29/04/14	vie 29/04/14
11	Implantación del proyecto informático	20 días	lun 02/05/14	vie 27/05/14
12	Pruebas y correcciones del PI	5 días	lun 30/05/14	vie 03/06/14
13	Elaboración del documento técnico	5 días	mié 01/06/14	mar 07/06/14
14	Revisión del documento técnico	2 días	mar 07/06/14	mié 08/06/14
15	Entrega del proyecto	1 día	vie 10/06/14	vie 10/06/14
16	Defensa del proyecto	1 día	vie 17/06/14	vie 17/06/14

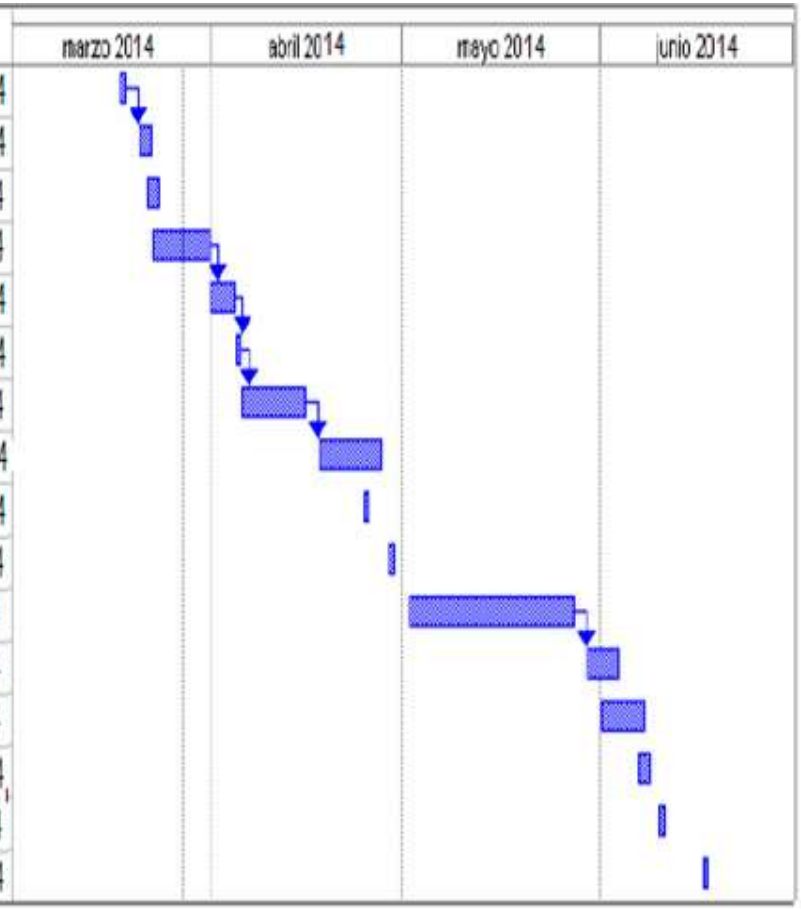


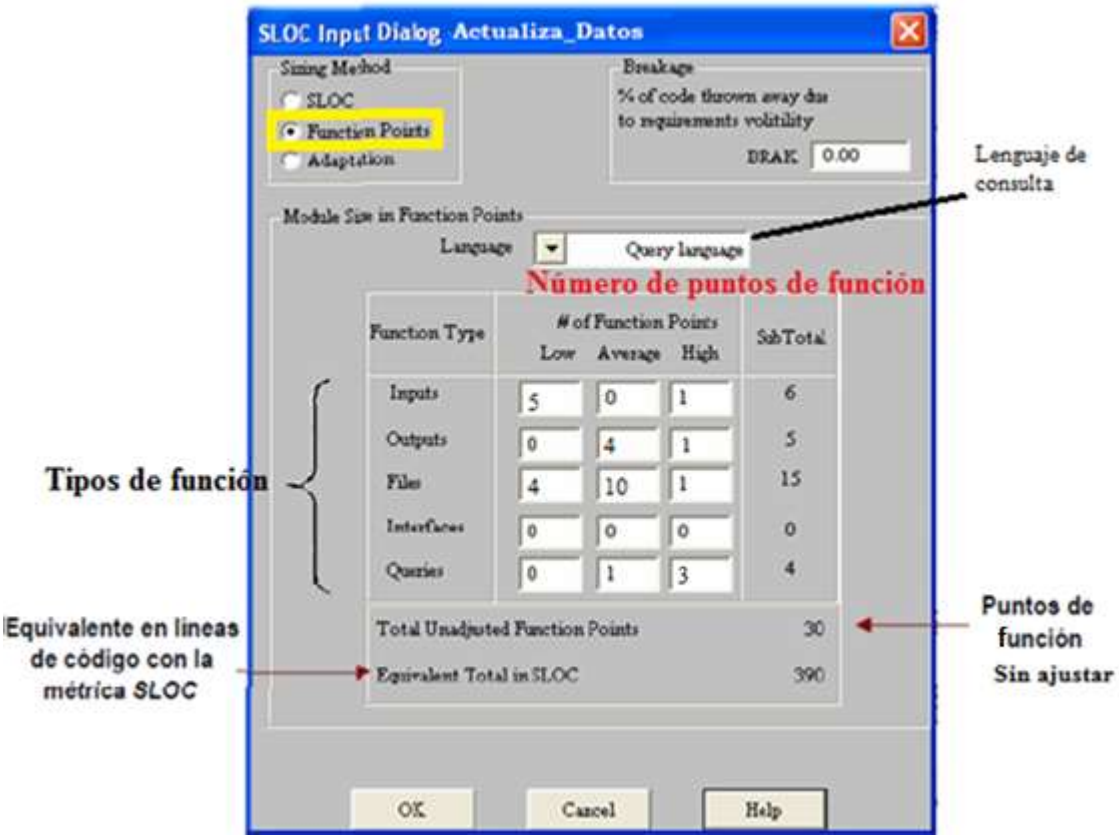
Diagrama de Gantt de actividades

Anexo E. Métrica utilizada. Mide la funcionalidad entregada al usuario, independientemente de la tecnología utilizada para la construcción y explotación del software. Es un método utilizado para medir el tamaño del software. Definido por Allan Albrecht (1979), se aplica en cualquiera de las fases de vida del software, desde el diseño inicial hasta la implementación y mantenimiento.

La determinación del tamaño del software es un parámetro necesario en el estudio de viabilidad y análisis de riesgo para la obtención de los reportes de factibilidad correspondientes, como parte de la gestión del mantenimiento.

Métrica punto de fusión. Se calculan los puntos de función para los dos módulos a agregar. En el módulo *Actualización_Datos*, se tienen los siguientes puntos de fusión:

- Entradas
- Salidas
- Archivos
- Interfaces
- Consultas



Cálculo de los puntos de función, módulo *Actualización_datos*

Como se observa, un estudio de factibilidad es una exposición detallada y profunda, tanto de factores, elementos, esfuerzos y costos, para determinar si una alternativa es viable o no, y escoger la mejor con la intensión de minimizar posibles impactos negativos en el proyecto de mantenimiento del software.

Conclusiones del estudio de factibilidad

Con este estudio de factibilidad, se observó que el capital invertido por el SCAF para el manejo de la información de sus activos fijos es rentable y factible, ya que la recuperación es prácticamente a corto plazo y sus beneficios son amplios.

Al seleccionar la mejor de las dos alternativas analizadas, se garantiza que el proyecto se realizará en tiempo y forma, de acuerdo con un presupuesto. Lo anterior, como parte fundamental del análisis de viabilidad y riesgos está como parte esencial de la gestión del mantenimiento del software.

Al implementar este proyecto se conseguirá lo siguiente:

- Información actualizada disponible en todo momento
- Registros altamente organizados
- Reducir el tiempo de respuesta de las diferentes instancias a las que el SCA envía el reporte general de todos sus activos fijos
- Desvíos de gastos monetarios hacia otras áreas en las cuales se requiera más presupuesto

En conclusión, el análisis de viabilidad y riesgos en la etapa de mantenimiento de un sistema de software determina si el proyecto se realiza o no. Ahí radica su importancia, en no correr riesgos innecesarios, y si se corren que sean mínimos; y son mínimos que sus posibles efectos negativos sean controlables, de tal forma que el proyecto de mantenimiento se realice en tiempo, forma y presupuesto.

Cierre de la unidad

Durante la presente unidad se han abordado las nociones básicas del mantenimiento, identificando los tipos y realizando pronósticos de los próximos mantenimientos del software. Es importante reconocer la evolución del software para aplicar la técnica de mantenimiento de reingeniería de sistemas o el proceso de mantenimiento a un sistema existente, concordando que todas las actividades inherentes deben estar englobadas en una gestión como tal, para determinar los costos a través de un estudio de viabilidad y análisis de riesgos que determine si el proyecto es viable o no. Si es viable se seguirá el proceso de la metodología de desarrollo de software para la modificación solicitada y si no tal vez ya se ha llegado al fin de la vida útil del sistema.

Grandes empresas dedicadas al desarrollo de software tienen clara la importancia del seguimiento de las normas de calidad, para asegurar la misma en los productos desarrollados. Las pruebas no son un tema a tomar a la ligera, ya que, si se aplican en los momentos correspondientes, se garantiza un producto con un porcentaje mínimo de errores que impacten en la funcionalidad del sistema de software. El mantenimiento fue considerado por mucho tiempo como un proceso necesario, pero no con la importancia

que cobra hoy en día, donde una buena planeación garantiza una buena evolución del sistema de software, conservándolo funcional por mucho más tiempo.

Sin embargo, hay muchas otras que por falta de especialistas ocupan a los programadores para las diferentes actividades antes descritas, programación, pruebas y mantenimiento, y eso resulta en un trabajo a medias, clientes inconformes, sistemas inestables.

Es un reto el que se tiene como profesionista, que va desde especializarse hasta hacer una carrera profesional en la empresa. Por lo anterior, esta materia ha presentado un panorama de dichos tópicos, que tienen un amplio horizonte de desarrollo.

Es cierto que muchos tienen la experiencia, pero es necesario certificarla para asegurar trabajos de alta calidad y de acuerdo con las exigencias actuales.

Recuerda que es importante retomar los aspectos de actualizaciones de los estándares señalados en el cuerpo del documento, toda vez que son actualizados y es necesario consultar los sitios web oficiales https://www.iso.org/obp/ui/es/#home y https://standards.ieee.org/, para conocer más sobre las versiones públicas de los estándares.

Lo anterior genera las siguientes interrogantes:

¿Es necesario certificarse? ¿Valdrá la pena certificarse? ¿Qué empresas tienen la capacidad de certificar? ¿La certificación efectivamente garantizará un trabajo de calidad?

Las respuestas las obtendrás en el desarrollo de tu vida profesional.

En México, hay mucho camino por recorrer aún, tanto en investigación como en la profesionalización; ya sea que te inclines por una opción o la otra, es importante desempeñar siempre el mejor papel.

Para saber más

Hoy en día, en México, en una industria creciente que ocupa tanto maquinaria como equipo de cómputo y software, el mantenimiento ha cobrado gran importancia. Se cuenta con varias empresas que ofrecen el servicio *outsourcing* y la venta de sistemas de gestión del mantenimiento, como PSL. Para saber más sobre este sistema de gestión visitita los siguientes sitios:

- *PSL México.* (s. f.).
- *Milenium*. (s. f.). Disponible en http://www.informaticamilenium.com.mx
- *MP* Software. (s. f.). Disponible en http://www.mpsoftware.com.mx/#

En México se cuenta con la norma mexicana NMX-I-059/02-NYCE-2011 enfocada a las organizaciones que se dedican al desarrollo y mantenimiento de software. Establece los requisitos de conformidad para el método de verificación del modelo de procesos para la industria de software (MoProSoft). Puedes consultar más sobre este modelo de procesos MoProSoft:

- *NYCE.* (s. f.). Disponible en https://www.nyce.org.mx/

Particularmente, el proceso de operación (OPE) considera para el desarrollo y mantenimiento de software (DMS) que:

- El propósito de desarrollo y mantenimiento de software es la realización sistemática de las actividades de obtención de requisitos, análisis, diseño, construcción, integración y pruebas de productos nuevos o modificados, pero siempre cumpliendo con los requisitos especificados.

- Ya hay normas establecidas en el país que están de acuerdo con las internacionales, que guían el desarrollo correcto del proceso de mantenimiento de software. Hace falta personal capacitado y certificado que dé respuesta a un mercado demandante y creciente.

Para saber más sobre el proceso de operación OPE, consulta su sitio oficial:

- *OPE.* (s. f.). Disponible en https://www.gob.mx/cre/acciones-y-programas/oficialia-de-partes-electronica-ope?idiom=es

Fuentes de consulta

Bibliografía Básica

- Abernethy, R. (2006). *The New Weibull Handbook,* 5a. ed. Florida, EE.UU: Robert B. Abernethy.

- Alain, A., Alain, A. (2008*). Software Maintenance Management: Evaluation and Continuous Improvement*. New Jersey: John Wiley & Sons.

- Albrecht, A. (1979). *Measuring Application Development Productivity*. En Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium. Monterrey, California. [En línea]: https://www.scirp.org/(S(lz5mqp453edsnp55rrgjct55))/reference/ReferencesPapers.aspx?ReferenceID=2012816

- Baca, G. (2010). *Evaluación de proyectos,* 6a. ed. México: McGraw-Hill.

- Bennett, K.,Rajlich, V.(2000). Software Maintenance and Evolution: A Roadmap. Future of Sofware Engineering Limerick Ireland Copyright ACM 2000.

- Boehm, B. (2000). *Software Cost Estimation with COCOCOMO II.*New Jersey:. Prentice Hall.

- Bourque, P. y Fairley, R. (2014). Guide to the Software Engineering Body of Knowledge Version 3.0. Disponible en https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf

- Crespo, A. (2013). *The Maintenance Management Framework*. (Chinese Edition). Beigjing, China. National Defense Industry Press. ISBN 978-7118083972

- Chiavenato, I. (2001). *Administración. Proceso Administrativo,* 3a. ed. Bogotá, Colombia: McGraw- Hill.

- Date, C. (2001). *Introducción a los* sistemas de bases de datos. México: Pearson Prentice Hall.

- FHWA CA División. (2009) *Guidebook for ITS v3.0*.EU. *Disponible en https://www.fhwa.dot.gov/cadiv/segb/files/segbversion3.pdf*

- Foster, J. (1993). Thesis Cost Factors in Software Maintenance. Disponible en https://core.ac.uk/download/pdf/108739.pdf

- Gallego, J C. (2010). *Mantenimiento de sistemas microinformáticos*. Madrid: Editex.

- García Martín, D. (2010). *Ingeniería inversa: modifica un programa*. [En línea]: https://www.redeszone.net/2010/11/22/ingenieria-inversa-modifica-un-programa/

- Gómez de León, F. C., (1998). *Tecnología del mantenimiento industrial*. Murcia: Universidad de Murcia.

- Gómez, J.(2014). *Guía práctica de estimación y medición de proyectos software: ¿por qué?, ¿para qué? y ¿cómo?.* Madrid, España: Edición Kindle.

- Henry, S. y Wakem, S. (1988). *Predicting Maintainability with Software Quality Metrics*. *Disponible en* http://eprints.cs.vt.edu/archive/00000131/

- Institute of Electrical and Electronics Engineers, IEEE. (2017). IEEE/ISO/IEC 12207-2017 International Standard - Systems and software engineering -- Software life cycle processes [En línea]. URL: https://standards.ieee.org/ieee/12207/5672/

- IEEE Standard. (2014). 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology. Disponible en https://n9.cl/3q2u3

- ISO/IEC 14764 (2006). *Standard for Software Engineering-Software Life CycleProcesses-Maintenance. Disponible en https://standards.ieee.org/ieee/14764/7701/*

- ISO (2001). *ISO/IEC TR 9126-1*. *Software engineering: Product quality. Part 1: Quality model*. Suiza: ISO.

- Kuna, H. (2008). *Plan de Riesgos para la implementación, desarrollo y mantenimiento de componentes de Web 2.0 en Bibliotecas, caso de estudio en una* biblioteca e*specializada. Disponible en* http://sedici.unlp.edu.ar/handle/10915/48365

- Krajewski, L. y Ritzman, L. (2000). *Administración de operaciones estratégicas y análisis,* 5a. ed. México: Pearson Educación.

- Lakey, P. *et ál.* (2014). *Software Reliability Assurance Handbook*. Disponible en https://www.cs.colostate.edu/~cs530/rh/

- Lehman, M. y Belady, L. (1985). *Program evolution: process software change*. Vol.27 of A.P.I.C. Studies in data processing. Texas, USA. Editorial Academic Press. ISBN 0124424406, 9780124424401

- Lientz, B.P. y Swanson, E. (1978). *Characteristics of Application Software Maintenance.* Volume 21 No. 6. *New York:* Magazine *Communications of the ACM.* p..466-471. *[En línea]: https://dl.acm.org/citation.cfm?id=359522*

- Llorens, J. (2005) *Gerencia de proyectos de tecnología de información. Caracas,* Venezuela. Colección Minerva. Editorial Los libros El Nacional.

- Morales, L. (2008). *Análisis por árboles de decisión para elección de un curso de acción empresarial sobre desarrollo interno, contratación o COTS. Disponible en* http://www.ingenieriasimple.com/problemas/EjemploArbolesDecision.pdf

- Newport, J.R. (1994). *Avionic Systems Design*. Florida, EE.UU: Press Inc.

- Pressman, R. S. (2010). *Software Engineering: a Practitioner's Approach*. Nueva York: McGraw-Hill Higher Education.

- RAE Real Academia Española (2014a). *Pronóstico*. [En línea] http://lema.rae.es/drae/?val=Pron%C3%B3stico

- RAE Real Academia Española (2014b). *Riesgo*. [En línea]: https://www.rae.es/?val=riesgo

- RAE Real Academia Española (2014c). *Viabilidad*. [En línea]: https://www.rae.es/?val=viabilidad

- Sommerville, I. (2011). *Ingeniería del software.* 9a.ed. Madrid: Pearson Educación.

- Stark, G. (1996). Measurements for Managing Software Maintenance. Disponible en https://www.researchgate.net/publication/221307869_Measurements_for_managing_software_maintenance

- (S.A) (2008). El verdadero costo del mantenimiento del software. [Mensaje de un blog]. Disponible en https://webcool.wordpress.com/2008/02/22/el-verdadero-costo-del-mantenimiento-del-software/

- UDELAR, Universidad de la República de Uruguay. (2003). *Casos de estudio.*

- **Fuentes de imagen:**

- PAE, Portal Administración Electrónica. (2013).*Métrica V3. Estudio de Viabilidad del Sistema (Proceso EVS)*. Disponible en https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

- Bassi, R. (2003). *Apunte de clases: estimación de costos de proyectos informáticos y TCO (*Total Cost of Ownership*).*