20

**Ingeniería de Software**

**Evolución de Software**

**Paez, Daniel M**

# 

El proceso de Desarrollo de software es un proceso constante, por lo que la entrega de este no significa el final en el trabajo del equipo de desarrollo, sino que deben estar pensando en futuras modificaciones en las funciones y requerimientos que el cliente pueda pedir en respuesta del uso de este, e incluso desde antes que se presentara.

El proceso de evolución es una adaptación del modelo en espiral, por lo que varias versiones del software surgirán para satisfacer las constantes necesidades de cambios. Llevar a cabo el mantenimiento del software y adaptarlo conlleva al mayor costo con respecto a los demás etapas.

El siguiente informe trata de explicar en qué consiste el proceso de evolución, brindar conceptos de mantenimiento y conocer que herramientas conocidas ayudan a llevar adelante este proceso.

Contenido

[Evolución de Software 3](#_Toc42457361)

[Evolución Dinámica del Programa. 6](#_Toc42457362)

[Mantenimiento del Software. 6](#_Toc42457363)

[Predicción del Mantenimiento. 8](#_Toc42457364)

[Reingeniería de Software. 9](#_Toc42457365)

[Mantenimiento Preventivo mediante Refactorización. 10](#_Toc42457366)

[Administración de Sistemas Heredados. 11](#_Toc42457367)

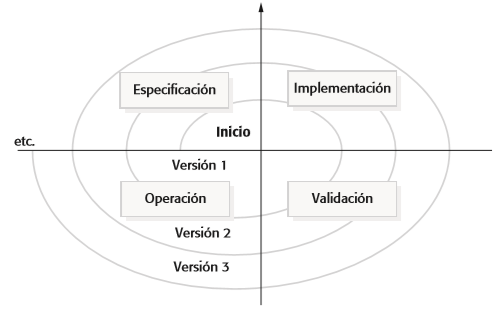
[Conclusiones. 13](#_Toc42457368)

[Bibliografía. 14](#_Toc42457369)

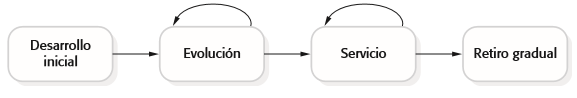
# Evolución de Software

El desarrollo del software no se detiene cuando un sistema es entregado, sino que continúa a lo largo de su vida. Si se desea que el software siga siendo útil después de entregarlo, será inevitable realizarle modificaciones para ello. Los cambios empresariales como las expectativas del cliente se verán reflejados en nuevos requisitos que harán que el software se adapte a estos. Por lo tanto, la evolución del software es importante, porque las organizaciones invierten grandes cantidades de dinero en él y en la actualidad son completamente dependientes de estos sistemas.

**L**a ingeniería de software se debe considerar como un proceso en espiral (incluyendo requerimientos, diseño, implementación y pruebas continuas) a lo largo de la vida del sistema. Se crea una 1° versión del sistema, la cual es entregada, y a partir de esta se proponen cambios. En consecuencia, se comienza con el desarrollo de la 2° versión casi al instante de la recopilación de los cambios. Es posible que la necesidad de evolución pueda volverse evidente incluso antes de que el sistema se distribuya, de manera que las futuras versiones del software estarían en desarrollo antes de que se libere la versión actual.



Rajlich y Bennett (2000) propusieron una visión alternativa del ciclo de vida de la evolución del software. En ese modelo, distinguen entre **evolución** y **servicio**. La **evolución** es la fase donde es posible hacer cambios significativos a la arquitectura y la funcionalidad del software. Durante el **servicio**, los únicos cambios que se realizan son relativamente pequeños.

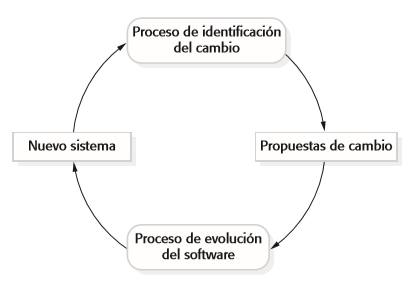


Durante la evolución, el software se usa con éxito y hay un flujo constante de propuestas de cambios a los requerimientos. Sin embargo, conforme el software se modifica, su estructura tiende a degradarse y los cambios se vuelven más y más costosos. En alguna etapa de su ciclo de vida, el software alcanza un punto de transición donde los cambios significativos que implementan nuevos requerimientos se vuelven cada vez menos rentables. En dicha fase, el software avanza de la evolución al servicio. Durante la fase de servicio, el software todavía es útil y se utiliza, pero sólo se le realizan pequeños cambios tácticos. En la fase final, de retiro gradual, el software todavía puede usarse, aunque no se implementan más cambios. Los usuarios tienen que sobrellevar cualquier problema que descubran.

**Procesos de Evolución.**

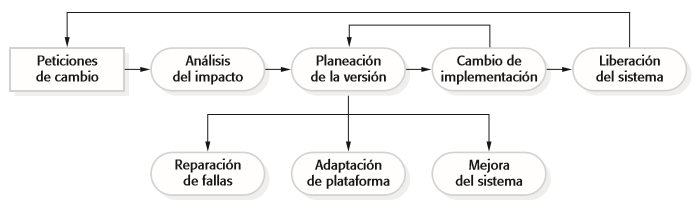
Los procesos de evolución del software varían dependiendo del tipo de software que se mantiene, de los procesos de desarrollo usados en la organización y de las habilidades de las personas que intervienen. En algunas organizaciones, la evolución es un proceso informal, donde las solicitudes de cambios provienen sobre todo de conversaciones entre los usuarios del sistema y los desarrolladores. En otras compañías, se trata de un proceso formalizado con documentación estructurada generada en cada etapa del proceso.

Los procesos de identificación de cambios y evolución del sistema son cíclicos y continúan a lo largo de la vida de un sistema.



Las propuestas de cambio deben vincularse con los componentes del sistema que se van a modificar para implementar dichas propuestas. Esto permite que el costo y el impacto del cambio logren valorarse.

El costo y el impacto de dichos cambios se valoran para saber qué tanto resultará afectado el sistema por el cambio y cuánto costaría implementarlo. Si los cambios propuestos se aceptan, se planea una nueva versión del sistema. Durante la planeación de la versión se consideran todos los cambios propuestos (reparación de fallas, adaptación y nueva funcionalidad). Entonces se toma una decisión acerca de cuáles cambios implementar en la siguiente versión del sistema. Después de implementarse, se valida y se libera una nueva versión del sistema. Luego, el proceso se repite con un conjunto nuevo de cambios propuestos para la siguiente liberación.



Esta versión de tomar la evolución del software es presentado por Arthur en 1988.

Durante el proceso de evolución, se analizan a detalle los requerimientos y surgen implicaciones de los cambios que no eran aparentes al comienzo del proceso de análisis de cambio. Esto significa que los cambios propuestos pueden modificarse y quizá se requieran más pláticas con el cliente antes de la implementación.

En ocasiones, las peticiones de cambio se relacionan con problemas del sistema que tienen que enfrentarse de manera urgente. Estos cambios urgentes surgen básicamente por tres razones:

1. Si ocurre una falla seria del sistema que se deba reparar, para la continuidad de la operación normal.
2. Si los cambios a los sistemas que operan el entorno tienen efectos inesperados que perturban la operación normal.
3. Si hay cambios no anticipados a la empresa que opera el sistema, como el surgimiento de competidores nuevos o la introducción de una nueva legislación que afecte al sistema.

En tales casos, la necesidad de realizar el cambio rápidamente significa que quizá no pueda seguir el proceso formal de análisis de cambio. En vez de modificar los requerimientos y el diseño, se puede hacer una reparación de emergencia al programa para resolver el problema de inmediato. Sin embargo, el riesgo es que los requerimientos, el diseño del software y el código se vuelvan inconsistentes. Aunque se trate de documentar el cambio en los requerimientos y el diseño, es posible que el software requiera reparaciones de emergencia adicionales, las cuales tienen prioridad sobre la documentación.

Por lo general, las reparaciones de emergencia del sistema tienen que completarse tan rápido como sea posible. Se elige una solución rápida y viable, en lugar de la mejor solución en cuanto a la estructura del sistema. Esto acelera el proceso de degeneración del software, de modo que los cambios futuros se vuelven progresivamente más difíciles y, con ello, aumenta el costo de mantenimiento.

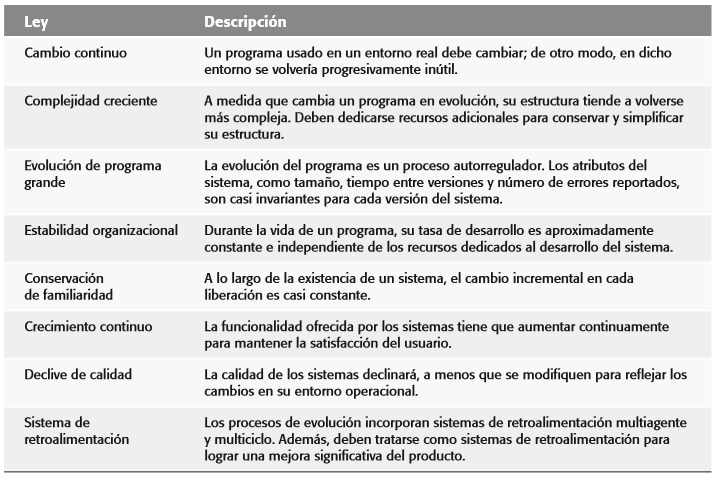
Los **Métodos y Procesos Agiles** se utilizan tanto para la evolución como para el desarrollo del programa (por el hecho de que están basados en el desarrollo incremental, no tienen complicaciones en pasar de la entrega a la evolución). Los cambios pueden expresarse como historias de usuario, y el involucramiento del cliente priorizará los cambios que se requieran en un sistema operacional. En resumen, la evolución simplemente implica la continuación del proceso de desarrollo ágil.

No obstante, es posible que surjan problemas en situaciones donde haya transferencia de un equipo de desarrollo a un equipo separado responsable de la evolución. Existen dos situaciones potencialmente problemáticas:

1. El equipo de desarrollo ha usado un enfoque ágil, pero el equipo de evolución no los conozca y use un enfoque basado en un plan (en este caso, el equipo de evolución espere documentación detallada para apoyar la evolución).
2. De forma contraria, el equipo de desarrollo realizo su labor en base a un plan, mientras que el equipo de evolución prefiere el uso de enfoques agiles (el equipo de evolución tal vez deba comenzar desde cero para desarrollar pruebas automatizadas y, además, es posible que el código en el sistema no se haya refactorizado y simplificado como se espera en el desarrollo ágil. En este caso, tal vez se requiera algo de reingeniería para mejorar el código, antes de usarlo en un proceso de desarrollo ágil.).

# Evolución Dinámica del Programa.

La dinámica de evolución del programa es el estudio del cambio al sistema. En las décadas de 1970 y 1980, Lehman y Belady (1985) realizaron varios estudios empíricos acerca del cambio al sistema, con una visión para entender más sobre las características de la evolución del software. El trabajo continuó en la década de 1990, conforme Lehman y sus colegas investigaron la importancia de la retroalimentación en los procesos de evolución. A partir de estos estudios, propusieron las “leyes de Lehman” relacionadas al cambio del sistema



Las primeras cinco leyes fueron las propuestas iniciales de Lehman; las leyes restantes se agregaron después de un trabajo posterior. Las leyes sexta y séptima son similares y, en esencia, indican que los usuarios de software se volverán cada vez más infortunados con el sistema, a menos que se le mantenga y se le agregue nueva funcionalidad. La ley final refleja el trabajo más reciente sobre los procesos de retroalimentación.

En general, las observaciones de Lehman parecen sensatas. Hay que tomarlas en cuenta cuando se planee el proceso de mantenimiento. Podría suceder que las consideraciones empresariales requieran ignorarlas en algún momento

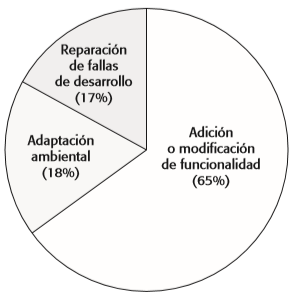
# Mantenimiento del Software.

El mantenimiento del software es el proceso general de cambiar un sistema después de que éste se entregó. En general, está más vinculado con software personalizado, en el que grupos de desarrollo separados intervienen antes y después de la entrega. Los cambios realizados al software van desde los simples para corregir errores de codificación, los más extensos para corregir errores de diseño, hasta mejorías significativas para corregir errores de especificación o incorporar nuevos requerimientos. Los cambios se implementan modificando los componentes del sistema existentes y agregándole nuevos componentes donde sea necesario.

Existen tres tipos de mantenimiento de software:

* **Reparaciones de Fallas:** encontrando errores en diferentes etapas y repararlos, siendo que, los errores de codificación son relativamente baratos de corregir; los errores de diseños son más costosos (quizás impliquen la reescritura de muchos componentes del programa) y los errores de requerimientos son los más costosos de reparar (debido a que podrían ser necesarios un extenso rediseño del sistema).
* **Adaptación Ambiental:** este tipo de mantenimiento es solicitado cuando algún aspecto del entorno del sistema, como el hardware o la plataforma operativa del sistema u otro soporte, cambia el software. Se tiene que modificar el sistema para lidiar con dichos cambios.
* **Adición de Funcionalidad:** este mantenimiento es necesario cuando varían los requerimientos del sistema, en respuesta a un cambio organizacional o empresarial. La escala de los cambios requeridos en el software suele ser mayor que en otros tipos de mantenimiento.

El mantenimiento del software toma una proporción más alta de presupuestos de TI que el nuevo desarrollo (casi dos tercios en mantenimiento y un tercio en desarrollo). También coinciden en que una mayor parte del presupuesto de mantenimiento se destina a la implementación de nuevos requerimientos, y no a la reparación de bugs. Evolucionar el sistema para enfrentar nuevos entornos y requerimientos nuevos o cambiantes consume más esfuerzo de mantenimiento.



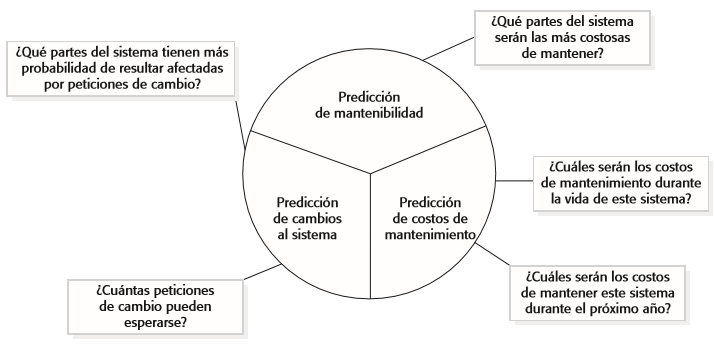
En general, resulta más costoso agregar funcionalidad después de que un sistema está en operación, que implementar la misma funcionalidad durante el desarrollo. Las razones son:

1. ***Estabilidad del Equipo:*** Después de que un sistema se entrega, es normal que el equipo de desarrollo se separe y que los individuos trabajen en nuevos proyectos. El nuevo equipo o los individuos responsables del mantenimiento del sistema no entienden el sistema o los antecedentes de las decisiones de diseño del mismo. Necesitan emplear tiempo para comprender el sistema existente, antes de implementar cambios en él.
2. ***Practica de Desarrollo Deficiente:*** El contrato para mantener un sistema por lo general está separado del contrato de desarrollo del sistema. El contrato de mantenimiento puede otorgarse a una compañía diferente, y no al desarrollador original del sistema.
3. ***Habilidades del Personal:*** El personal de mantenimiento con frecuencia es relativamente inexperto y no está familiarizado con el dominio de aplicación. El mantenimiento tiene una mala imagen entre los ingenieros de software. Lo ven como un proceso que requiere menos habilidades que el desarrollo de sistemas y se asigna a menudo al personal más novato. Incluso los sistemas pueden estar escritos en lenguajes de programación obsoletos. Es posible que el personal de mantenimiento no tenga mucha experiencia de desarrollo en estos lenguajes y debe aprenderlos para mantener el sistema.
4. ***Antigüedad y Estructura del Programa:*** Conforme se realizan cambios al programa, su estructura tiende a degradarse. En consecuencia, a medida que los programas envejecen, se vuelven más difíciles de entender y cambiar. Algunos sistemas se desarrollaron sin técnicas modernas de ingeniería de software. Es posible que nunca hayan estado bien estructurados y tal vez estuvieron optimizados para eficiencia y no para comprensibilidad. La documentación del sistema puede estar perdida o ser inconsistente. Es posible que los sistemas antiguos no se hayan sujetado a una gestión rigurosa de configuración, de modo que se desperdicia tiempo para encontrar las versiones correctas de los componentes del sistema a cambiar.

El mantenimiento se ve como una actividad de segunda clase, y no hay incentivo para gastar dinero durante el desarrollo para reducir los costos del cambio de sistema. La única solución a largo plazo a este problema es aceptar que los sistemas rara vez tienen una vida definida pero continúan en uso, en cierta forma, durante un periodo indefinido.

## Predicción del Mantenimiento.

Los gerentes aborrecen las sorpresas, sobre todo si derivan en costos inesperadamente elevados. Por consiguiente, se debe tratar de predecir qué cambios deben proponerse al sistema y qué partes del sistema es probable que sean las más difíciles de mantener. También hay que tratar de estimar los costos de mantenimiento globales para un sistema durante cierto lapso de tiempo.



Predecir el número de peticiones de cambio para un sistema requiere un entendimiento de la relación entre el sistema y su ambiente externo. Para evaluar las relaciones entre un sistema y su ambiente, se debe valorar:

* *El número y la complejidad de las interfaces del sistema.*
* *El número de requerimientos de sistema inherentemente inestables.*
* *Los procesos empresariales donde se usa el sistema.*

Las mediciones de complejidad son muy útiles para identificar componentes de programa que suelen ser costosos de mantener.

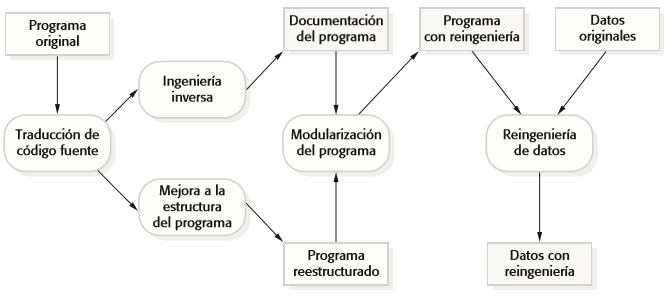
Después de poner en servicio un sistema, se deben usar datos de proceso para auxiliarse a predecir el mantenimiento. Los siguientes son ejemplos de métricas de proceso que sirven para valorar el mantenimiento:

* **Número de Peticiones para mantenimiento correctivo:** Un aumento en el número de reportes de bugs y fallas indicaría que se introdujeron más errores en el programa de los que se repararon durante el proceso de mantenimiento. Esto podría revelar un declive en el mantenimiento.
* **Tiempo Promedio Requerido para Análisis del Impacto:** Refleja el número de componentes de programa que se ven afectados por la petición de cambio. Si este tiempo aumenta, implica que más componentes resultaron afectados y que el mantenimiento decrece.
* **Tiempo Promedio Tomado para Implementar una Petición de Cambio:** Éste no es el mismo que el tiempo para el análisis del impacto, aunque puede correlacionarse con él, sino más bien es la cantidad de tiempo que se necesita para modificar el sistema y su documentación, después de valorar cuáles componentes serán afectados. Un aumento en el tiempo necesario para implementar un cambio puede indicar un declive en el mantenimiento.
* **Número de Peticiones de Cambios Pendientes:** unaumento en este número implicaría un declive en el mantenimiento.

La información predicha sobre las peticiones de cambio y las predicciones acerca del mantenimiento del sistema se usan para predecir los costos de mantenimiento. La mayoría de los gerentes combinan esta información con la intuición y la experiencia para estimar costos.

## Reingeniería de Software.

Para hacer que los sistemas de software heredados sean más sencillos de mantener, se pueden someter a reingeniería para mejorar su estructura y entendimiento. La reingeniería puede implicar volver a documentar el sistema, refactorizar su arquitectura, traducir los programas a un lenguaje de programación moderno, y modificar y actualizar la estructura y los valores de los datos del sistema. La funcionalidad del software no cambia y, normalmente, conviene tratar de evitar grandes cambios a la arquitectura de sistema.



La reingeniería de programas no necesariamente requiere de todos los pasos, por ejemplo:

* Si aún se usa el lenguaje de programación de la aplicación, no será necesario la traducción del código fuente.
* Si se logra hacer automáticamente toda la reingeniería, entonces quizá no sea necesaria la recuperación de documentación mediante ingeniería inversa.
* La reingeniería de datos sólo se requiere cuando las estructuras de datos en el programa cambian durante la reingeniería del sistema.

Este proceso trae dos beneficios importantes:

* **Reducción del Riesgo.**
* **Reducción del Costo:** El costo de la reingeniería puede ser significativamente menor que el costo de desarrollar software nuevo.

El problema con la reingeniería de software es que existen límites prácticos a cuánto mejora un sistema gracias a la reingeniería. No es posible, por ejemplo, convertir un sistema escrito con un enfoque funcional, a un sistema orientado a objetos. Los grandes cambios arquitectónicos o la reorganización radical de la gestión de los datos del sistema no pueden realizarse automáticamente, de modo que son muy costosos. Aunque la reingeniería podría mejorar la mantenibilidad, el sistema con reingeniería probablemente no será tan mantenible como un sistema nuevo desarrollado usando modernos métodos de ingeniería de software.

## Mantenimiento Preventivo mediante Refactorización.

La refactorización es el proceso de hacer mejoras a un programa para frenar la degradación mediante el cambio. Ello significa modificar un programa para mejorar su estructura, reducir su complejidad o hacerlo más fácil de entender. Mientras se refactoricé un programa, no se debe agregar funcionalidad, sino que hay que concentrarse en la mejora del programa. Por ende, se puede considerar la refactorización como el “mantenimiento preventivo” que reduce los problemas de cambios futuros.

Existen situaciones esteorotipicadas (llamadas ***“malos olores”***), en las cuales el código de un programa es susceptible de mejorarse. Los ejemplos de malos olores que pueden mejorarse mediante refactorización incluyen:

1. **Código Duplicado.**
2. **Métodos Largos.**
3. **Enunciados de Switch (case).**
4. **Aglomeración de Datos.**
5. **Generalidades especulativas.**

La refactorización, realizada durante el desarrollo del programa, es una forma efectiva de reducir los costos a largo plazo en el mantenimiento de un programa. Sin embargo, si se toma la responsabilidad del mantenimiento de un programa, cuya estructura esté significativamente degradada, entonces sería casi imposible refactorizar sólo el código. Tal vez se tenga que considerar la refactorización del diseño, que probablemente resulte ser un problema más costoso y difícil. La refactorización del diseño implica identificar patrones de diseño relevantes y sustituir el código existente con código que implementen dichos patrones de diseño.

# Administración de Sistemas Heredados.

Para los sistemas de software nuevos desarrollados con modernos procesos de ingeniería de software, como el desarrollo incremental y el CBSE, es posible planear cómo integrar el desarrollo y la evolución del sistema. Cada vez con mayor frecuencia, las compañías empiezan a entender que el proceso de desarrollo del sistema es un proceso de todo el ciclo de vida y que no es útil una separación artificial entre el desarrollo del software y su mantenimiento.

La mayoría de las organizaciones, por lo general, tienen un portafolio de sistemas heredados, que se usan con un presupuesto limitado para mantenimiento y actualización. Deben decidir cómo obtener el mejor retorno de la inversión. Esto requiere hacer una valoración realista de sus sistemas heredados y, luego, decidir acerca de la estrategia más adecuada para hacer evolucionar dichos sistemas. Existen cuatro opciones estratégicas:

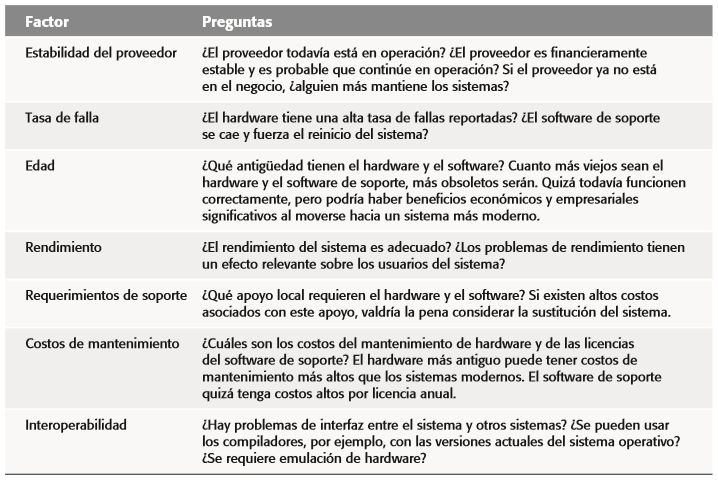
1. *Desechar Completamente el Sistema:* debe elegirse cuando el sistema no vaya a realizar una aportación efectiva a los procesos empresariales. Por lo general, esto ocurre cuando los procesos empresariales cambiaron desde la instalación del sistema y no se apoyan más en el sistema heredado.
2. *Dejar sin Cambios el Sistema y Continuar el Mantenimiento Regular:* debe elegirse cuando el sistema todavía se requiera, pero sea bastante estable y los usuarios del sistema hagan relativamente pocas peticiones de cambio.
3. *Someter el Sistema a Reingeniería para Mejorar su Mantenibilidad:* debe elegirse cuando la calidad del sistema se haya degradado por el cambio y todavía se propone un nuevo cambio al sistema. Este proceso podría incluir el desarrollo de nuevos componentes de interfaz, de modo que el sistema original logre trabajar con otros sistemas más recientes.
4. *Sustituir Todo o Parte del Sistema con un Nuevo Sistema:* tiene que elegirse cuando factores como hardware nuevo signifiquen que el viejo sistema no pueda continuar en operación o donde sistemas comerciales permitirían al nuevo sistema desarrollarse a un costo razonable. En muchos casos se opta por una estrategia de sustitución evolutiva, en la cual grandes componentes del sistema se sustituyen con sistemas comerciales, y otros componentes se reutilizan siempre que sea posible.

Para calcular el valor empresarial de un sistema, se tiene que identificar a los participantes del sistema, como sus usuarios finales y sus administradores, y plantear una serie de preguntas acerca del sistema. Existen cuatro temas básicos que se deben analizar:

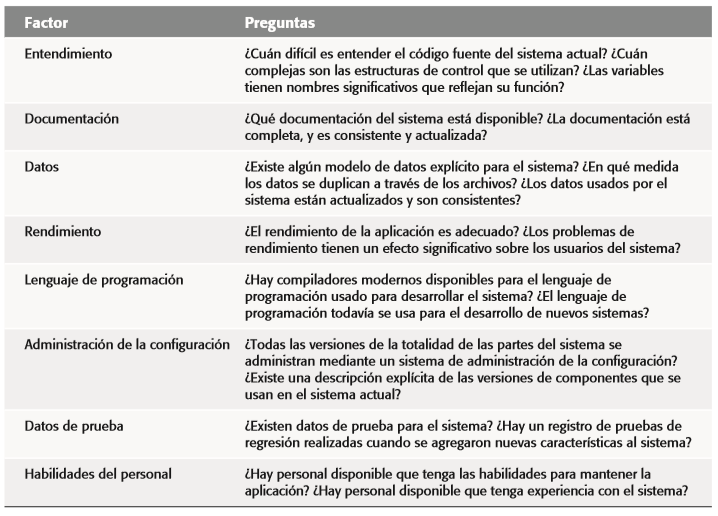
* **El uso del Sistema:** Si los sistemas sólo se usan ocasionalmente o por un número pequeño de individuos, quizá tengan un bajo valor empresarial. En ocasiones un sistema heredado se desarrolló para satisfacer una necesidad empresarial que cambió o que ahora, de manera más efectiva, se satisface de otras formas. Sin embargo, se debe tener cuidado acerca del uso ocasional pero, a la vez, importante de los sistemas.
* **Los Procesos Empresariales que se Mantienen:** Cuando se introduce un sistema, los procesos empresariales se diseñan para explotar las capacidades del sistema. Si el sistema es inflexible, sería casi imposible modificar dichos procesos empresariales. Sin embargo, conforme cambia el entorno, los procesos empresariales originales suelen volverse obsoletos. Por lo tanto, un sistema puede tener un bajo valor empresarial porque fuerza el uso de procesos empresariales ineficientes.
* **Confiabilidad del Sistema:** Si un sistema no es confiable y los problemas afectan directamente a los clientes de la empresa o hacen que los trabajadores en la empresa se distraigan de otras tareas para resolver dichos problemas, el sistema tiene un valor empresarial bajo.
* **Las Salidas del Sistema:** Si la empresa depende de dichas salidas, entonces el sistema tiene un alto valor empresarial. Por el contrario, si tales salidas pueden generarse fácilmente en alguna otra forma, o si el sistema produce salidas que rara vez se utilizan, entonces su valor empresarial suele ser bajo.

Para valorar un sistema de software desde una perspectiva técnica, se necesita considerar tanto el sistema de aplicación en sí como el entorno donde opera el sistema. El entorno incluye el hardware y todo el software de soporte asociado (compiladores, entornos de desarrollo, etcétera) que se requieran para mantener el sistema. El entorno es importante porque muchos cambios del sistema resultan de cambios al entorno, como actualizaciones al hardware o al sistema operativo.

Los factores que se deben considerar durante la valoración del entorno son:



Para valorar la calidad técnica de un sistema de aplicación, quizá se deban valorar diversos factores que se relacionan principalmente con la confiabilidad del sistema, las dificultades de mantener el sistema y su documentación.



También se puede recolectar datos que ayudarán a juzgar la calidad del sistema. Los datos que podrían ser útiles en la valoración de la calidad son:

1. **Número de Peticiones de cambios del Sistema:** Los cambios al sistema, por lo general, corrompen la estructura del sistema y dificultan cambios futuros. Cuanto más alto sea este valor acumulado, más baja será la calidad del sistema.
2. **Número de Interfaces de Usuario:** éste es un factor importante en los sistemas basados en formas, donde cada forma puede considerarse como una interfaz de usuario separada. Cuanto más interfaces haya, más probabilidad habrá de que existan inconsistencias y redundancias en dichas interfaces.
3. **Volumen de datos usados por el Sistema:** Cuanto más alto sea el volumen de datos (número de archivos, tamaño de base de datos, etcétera), más probable será que haya inconsistencias de datos que reduzcan la calidad del sistema.

# Conclusiones.

La vida del software no termina en el momento en que el software se entrega, la presentación de cambios en los requerimientos es inevitable e incluso puede presentarse desde antes de la entrega de este. Por ende los trabajos de evolución de software deben ser tomados como una parte importante del ciclo de vida del software, ya que como se dictó en lecturas anteriores es un proceso que es llevado a cabo en menor medida e incluso es llevada por gente de baja interacción con el equipo de desarrollo. Esta perspectiva conlleva a que el software no tenga una vida de uso esperada e incluso incrementar los costos en este proceso en comparación con los demás en el ciclo de vida del software. Se vio que incluso las herramientas actuales de refactorización e incluso los procesos de desarrollo orientados en las metodologías agiles son de gran ayuda para esta etapa, solo hay que avalarse de que la gente que lleve a cabo esta tarea este familiarizada de estas metodologías. En caso contrario, es buen plan para este equipo el uso de la reingeniería, aunque se deberá plantear las etapas de esta, ya que será posible que no sean necesarias.

# Bibliografía.

Ingeniería de Software – Ian Sommerville – 9na Edición.