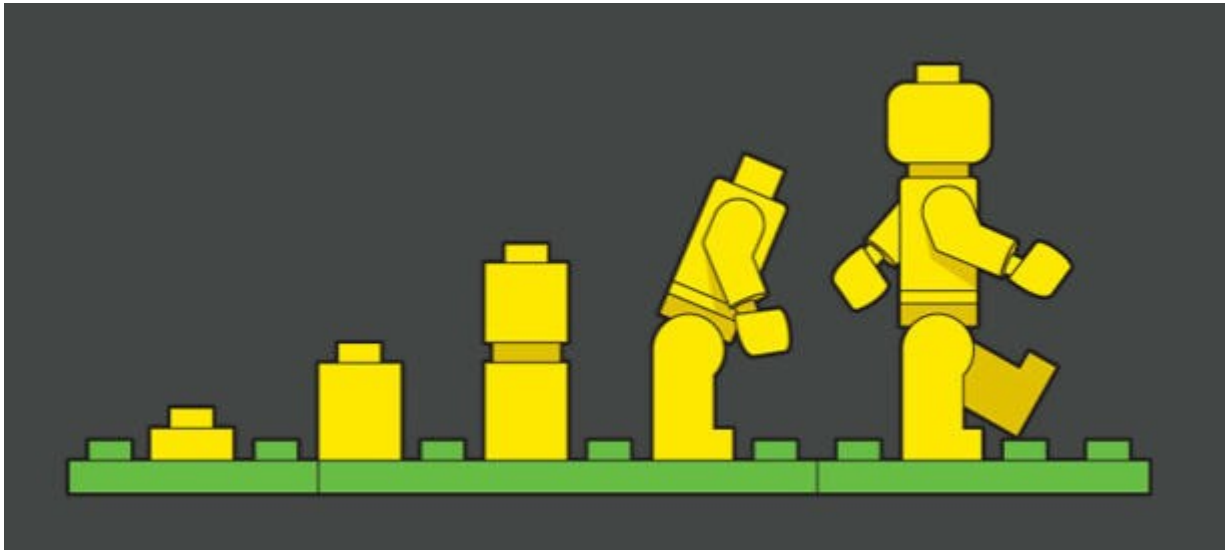
Ingeniería de Software

Evolución
Clase 1

Sommerville 10 – Cap 9, intro y sección 9.1

Introducción

- ¿Por qué evoluciona el software?



Introducción

- ¿Por qué evoluciona el software?

- Porque necesita:
  - Adaptarse al entorno operativo
  - Adaptarse al usuario
  - Mejorar en el desempeño
  - Mejorar en funcionalidades
  - Corrección de errores

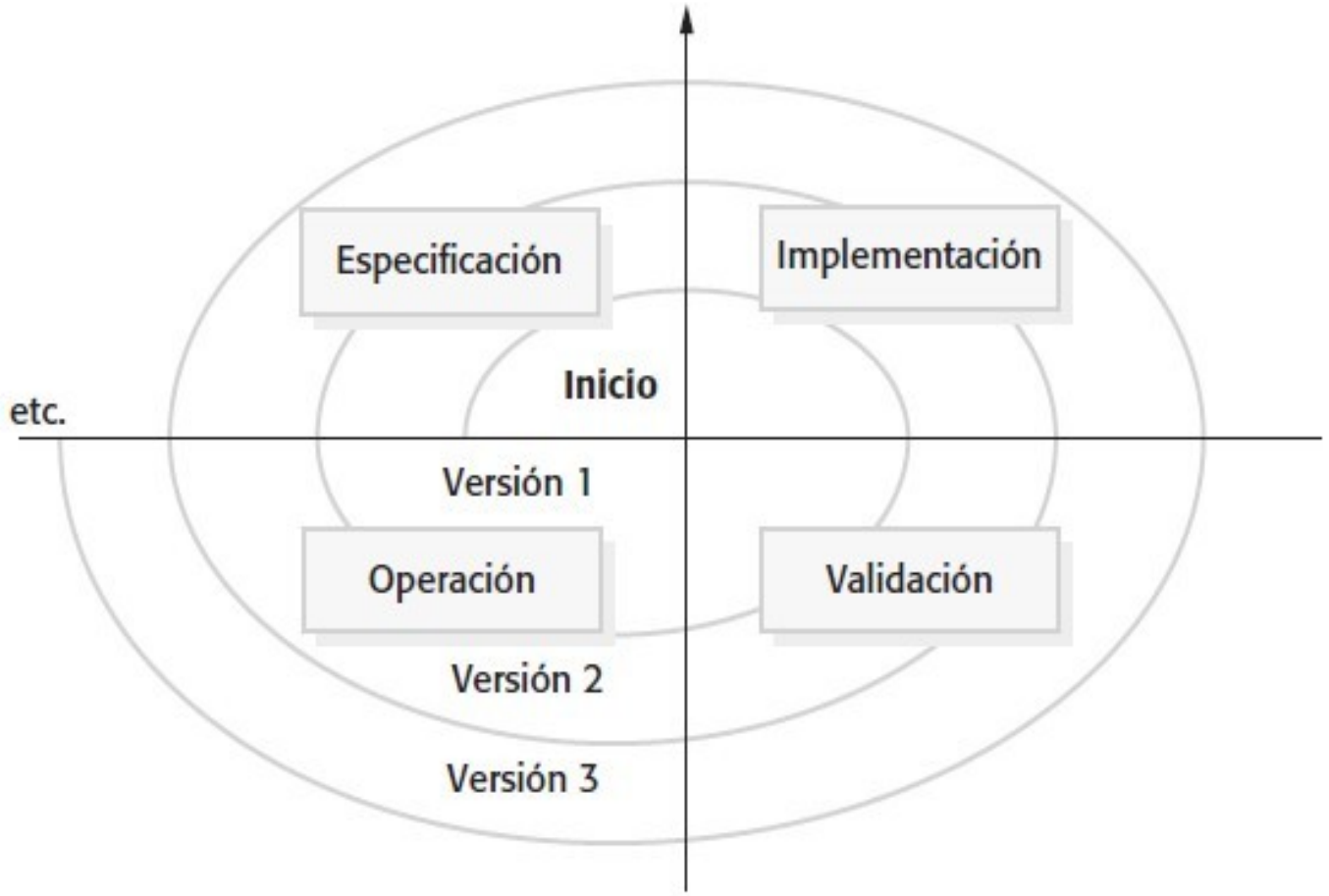
- Requiere continua adaptación, mantenimiento y mejoras para seguir siendo útil y valioso para la organización

Importancia de la evolución

- Los sistemas de software representan activos críticos del negocio de las organizaciones
- Para mantener el valor, los sistemas deben ser modificados y actualizados
- La mayor parte del presupuesto está dedicada al cambio y a la evolución del software existente

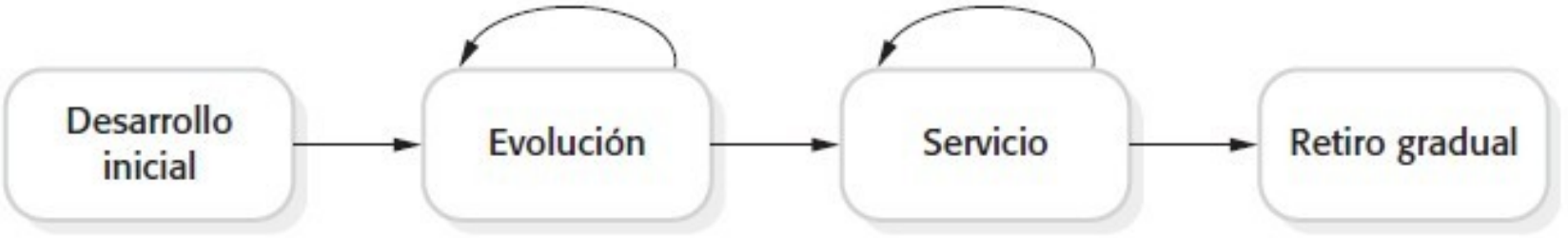
- Actividades principales de la ing de software:
  - Especificación, Diseño e Implementación, Validación y Evolución.

Modelo en espiral de desarrollo y evolución



- El tiempo entre iteraciones puede ser bastante corto.
- Cuando la transición de desarrollo a evolución está bien marcada, hablamos de mantenimiento de software.

Evolución y Servicio



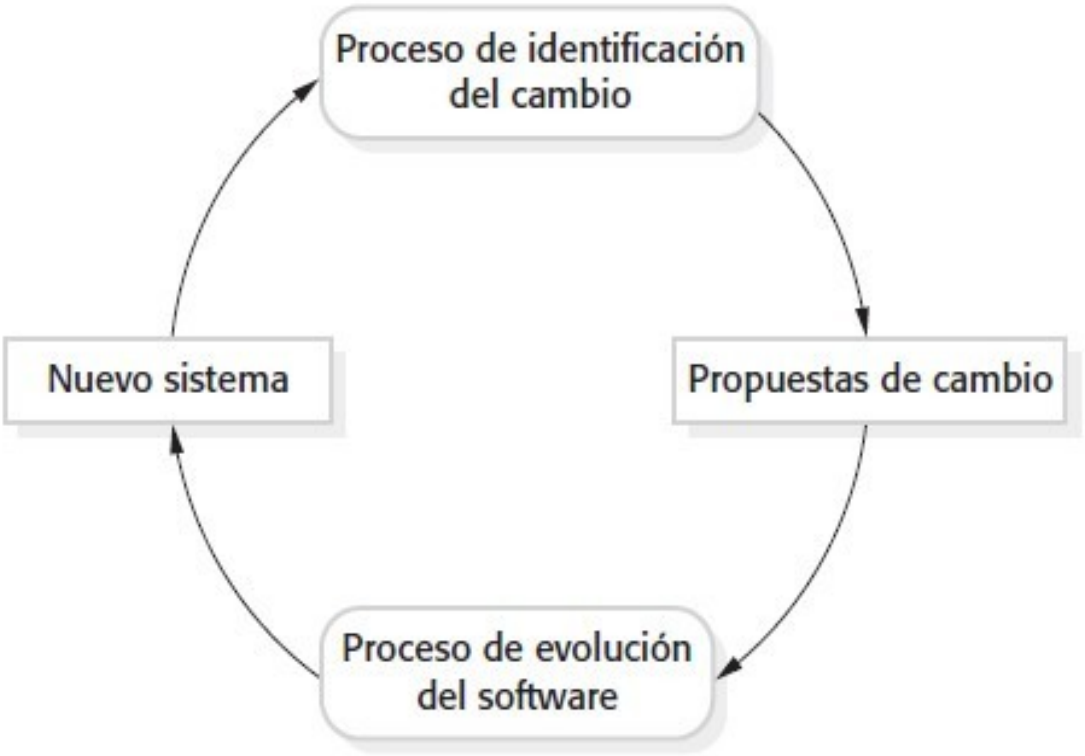
- Evolución → el sistema vive en un ciclo donde se encuentra en su uso operacional y evoluciona cuando los nuevos requisitos son propuestos e implementados en el sistema.
- Servicio → el software sigue siendo útil pero los únicos cambios realizados son para mantenerlo operativo. Por ejemplo: corrección de errores y cambios producto de modificaciones en el entorno del software. No se agregan nuevas funcionalidades.
- Retiro gradual → el software puede ser utilizado todavía pero no se realizan más cambios sobre el mismo.

Procesos de evolución

- Las propuestas de cambio guían la evolución del software.
- Sugieren cambios a un sistema existente, se basan en:
  - Requisitos preexistentes que no fueron realizados antes de la liberación
  - Nuevos requisitos
  - Errores reportados
  - Nuevas ideas del equipo de desarrollo

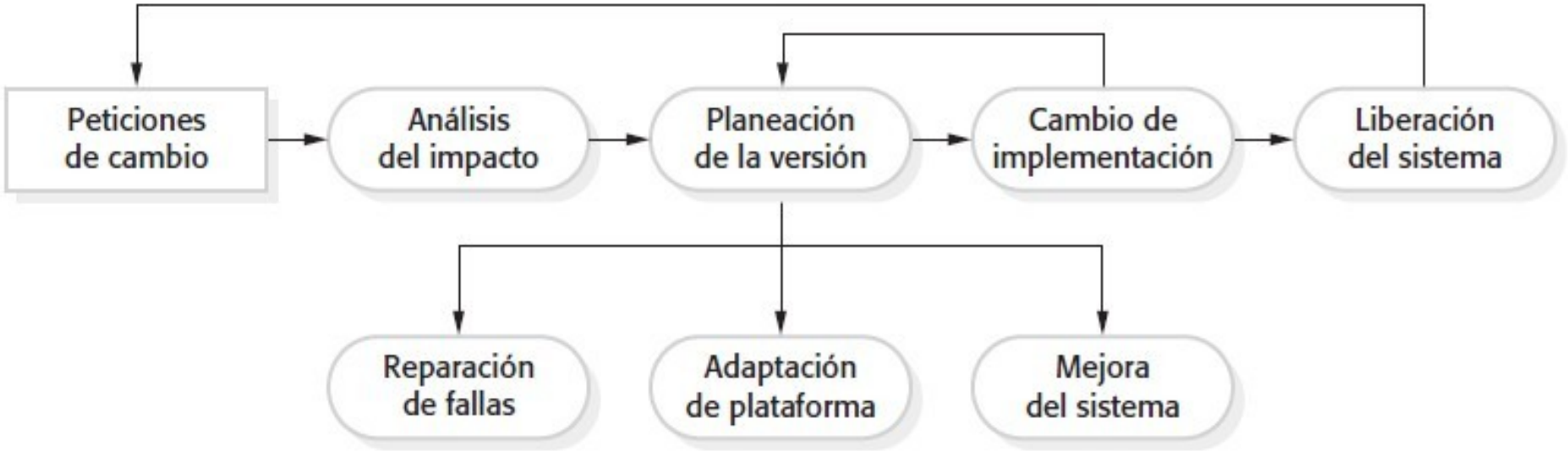
- Se deben identificar y relacionar los componentes que son afectados por un cambio, permitiendo así estimar el costo y el impacto del cambio

Procesos de evolución



- La identificación de los cambios y la evolución continuan a lo largo del tiempo de vida del sistema.

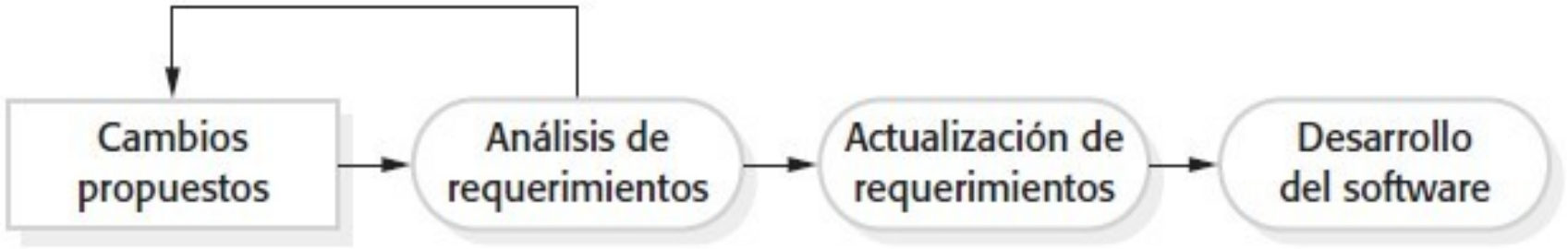
Procesos de evolución



Procesos de evolución

- Análisis de impacto
  - El costo y el impacto de los cambios se valoran para saber qué tanto resultará afectado el sistema y cuánto costaría.
  - Cuando el equipo de mantenimiento es diferente al de desarrollo, la implementación incluye una comprensión del programa.
  - Se planifica una nueva versión con los cambios aceptados.
- Planeación de la versión
  - La planificación considera todos los cambios propuestos.
  - Se decide cuáles cambios implementar en la siguiente versión.
- Liberación del sistema
  - Después de implementarse, se valida y se libera una nueva versión.
  - Luego, el proceso se repite con un conjunto nuevo de cambios propuestos para la siguiente liberación.

Implementación del cambio



- Durante el proceso, se analizan los requerimientos y pueden surgir modificaciones a los cambios propuestos.
- En la primera etapa de la implementación del cambio puede implicar entender el programa, especialmente si los desarrolladores no son responsables de la implementación del sistema desarrollado originalmente.
  - Durante esta etapa, se debe entender cómo el programa está estructurado, cuál es la forma en que ofrece funcionalidad y cómo el cambio propuesto puede afectar el programa.
- ¿y cambios urgentes?

Cambios urgentes

- Los cambios urgentes pueden tener que ser implementados sin pasar por todas las etapas del proceso de ingeniería de software.
  - Si es una falla grave, el sistema debe ser reparado para permitir que continúe la operación normal del sistema;
  - Si cambios en el entorno del sistema tienen efectos inesperados;
  - Si hay cambios en el negocio que requieren una rápida respuesta (ej.: la competencia lanza un producto).
- Para realizar los cambios de manera rápida, se elige una solución rápida y viable.
    - En general, no es la mejor solución
    - Riesgo de no actualizar la documentación
- Se puede registrar un cambio no urgente para rehacerlo.

  Por ej., Política del día después

Los métodos ágiles y la evolución

- Los métodos ágiles se basan en un desarrollo incremental, por lo cual la transición del desarrollo a la evolución es perfecta.
  - La evolución es una simple continuación del proceso de desarrollo basado en frecuentes liberaciones del sistema.
- Las pruebas de regresión automatizadas son muy valiosas cuando se realizan cambios en el sistema.
- Los cambios pueden ser expresados como historias de usuario adicionales.

Escenarios de transferencia

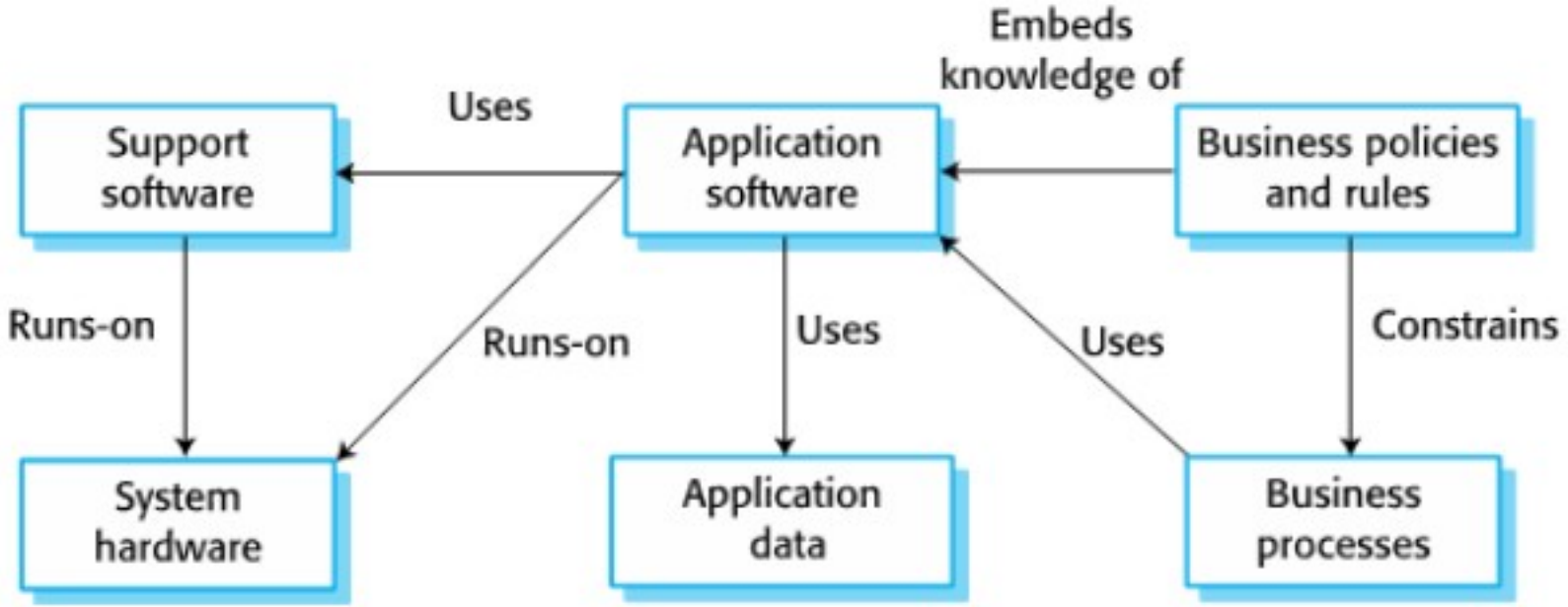
- Desarrollo con enfoque ágil, evolución con un enfoque tradicional:
  - El equipo tradicional estará esperando que se le entregue documentación detallada que sirva como soporte durante la evolución.

- Desarrollo con enfoque tradicional, evolución con un enfoque ágil:
  - Podrían no existir pruebas automatizadas ya desarrolladas
  - El código puede no haber sido refactorizado y simplificado

Sistemas heredados

- En una organización, los sistemas se remplazan a medida que el negocio cambia.
- Sin embargo, muchos viejos sistemas continúan siendo utilizados, e incluso, tienen un rol crítico en el negocio. Estos son llamados sistemas heredados.
  - No son solo sistemas de software, abarcan hardware, software, librerías, software de soporte y procesos de negocio.
  - Han tenido mantenimiento por un largo tiempo, por lo que su estructura puede estar degradada
  - Pueden depender de *hardware* antiguo
  - Es probable que no soporten nuevos procesos de negocio

Componentes de los sistemas heredados



Sistemas heredados

- El mantenimiento de estos sistemas tiene dificultades y es costoso.
  - Falta de habilidades o conocimiento de viejas tecnologías (recursos externos)
  - Dificultades para entender el código debido a que fue modificado por muchas personas con diferentes estilos
  - Sistema degradado por muchos años de mantenimiento Falta documentación o está desactualizada.
  - Vulnerabilidades de seguridad
  - Problemas para integrarse con sistemas construidos con tecnologías nuevas
  - Ausencia de soporte oficial
  - Hardware obsoleto y costoso de mantener
  - Problemas a nivel de datos:duplicación y baja calidad.

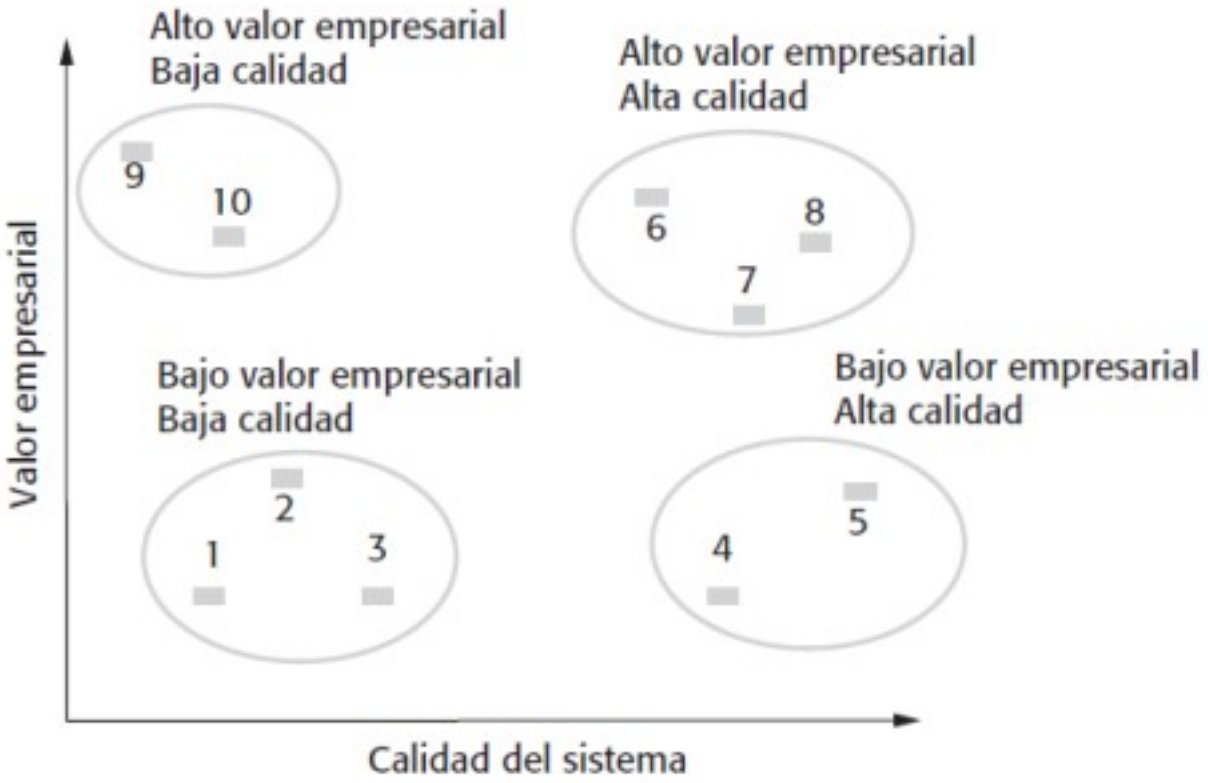
Sistemas heredados

- Tomar la decisión de remplazar un sistema heredado puede ser costoso y riesgoso.
  - No existe especificación completa del sistema
  - Los procesos de negocio seguramente tengan que ser modificados
  - Reglas de negocio pueden estar *hardcodeadas* sin documentación
  - El nuevo software trae riesgos consigo (tiempo y costo)

Gestión de sistemas heredados

- Las organizaciones que se basan en sistemas heredados deben elegir la estrategia para la evolución de ellos.
  - Desechar el sistema por completo y modificar los procesos de negocio de forma tal que no sea necesario
  - Continuar el mantenimiento del sistema
  - Transformar el sistema por medio de la reingeniería y mejorar la mantenibilidad
  - Remplazar el sistema con un sistema nuevo.
- La estrategia elegida dependerá de la calidad del sistema y del valor del sistema para el negocio.

Ejemplo de valoración de un sistema heredado



Categorías de sistemas heredados

- 1,2,3 → Estos sistemas deben ser desechados
- 4,5 → Remplazar con componentes de terceros, desechar completamente o mantener.
- 6,7,8 → Continuar con el sistema en operación y realizar un mantenimiento normal del sistema.
- 9,10 → Estos hacen una importante contribución al negocio pero son caros de mantener. Debe ser reestructurado o remplazado por otro sistema comercial que esté disponible.

Evaluación del valor para el negocio

- La evaluación debe tomar en cuenta diferentes puntos de vista
  - Usuarios finales del sistema,
  - Clientes de negocios,
  - Gerentes de línea,
  - Administradores.

- Entrevistar diferentes interesados y cotejar resultados

Aspectos básicos en la valoración para el negocio

- El uso del sistema
  - Si los sistemas son utilizados ocasionalmente o por un pequeño número de personas, el sistema puede tener un bajo valor para el negocio.
- Los procesos de negocio que son soportados
  - Un sistema puede tener un bajo valor para el negocio si fuerza a que se utilicen procesos de negocios ineficientes.
- Confianza del sistema
  - Si un sistema no es confiable y los problemas afectan directamente a los clientes del negocio, el sistema tiene un bajo valor para el negocio.
- Las salidas del sistema
  - Si el negocio depende de las salidas del sistema, entonces el sistema tiene un alto valor para el negocio.

Evaluación de la calidad del sistema

- Evaluación del proceso de negocio
  - ¿En qué medida el proceso de negocio da soporte a los objetivos actuales del negocio?
- Evaluación del entorno
  - ¿Cuán efectivo es el entorno del sistema y cuán costoso es su mantenimiento?
- Evaluación de la aplicación
  - ¿Cuál es la calidad de la aplicación del sistema de software?
    - Cantidad de solicitudes de cambios
    - Número de interfaces de usuario
    - Volumen de datos utilizados por el sistema

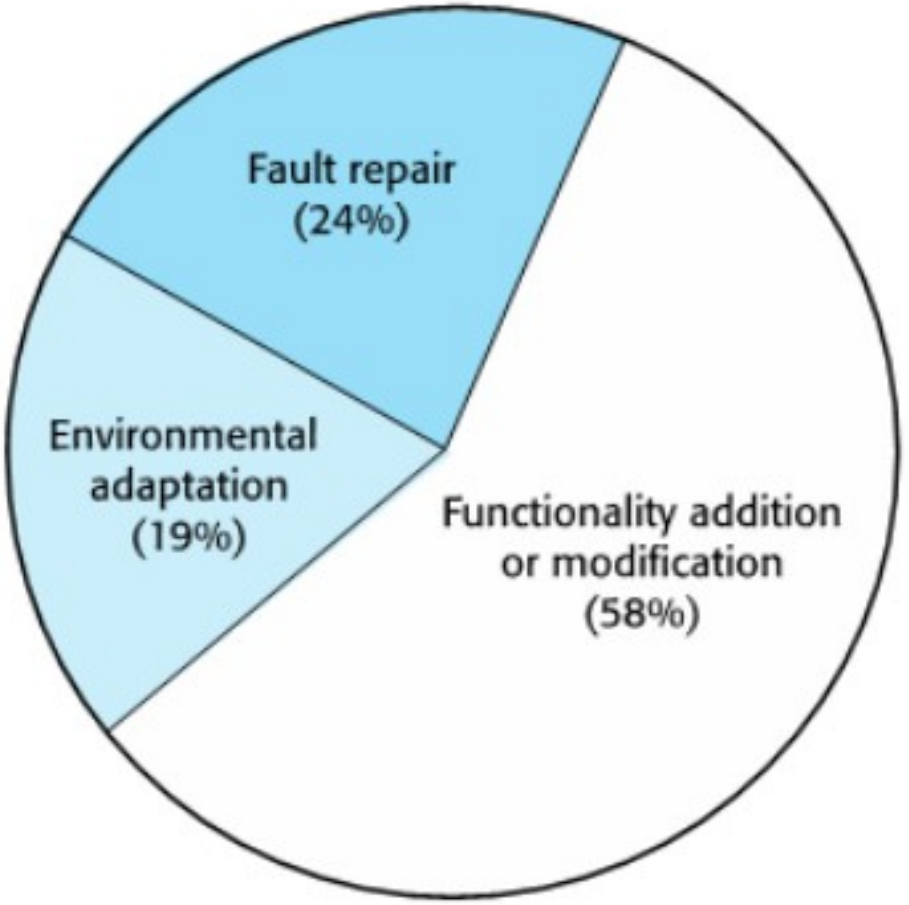
Mantenimiento del software

- Es la modificación de un programa luego de que está siendo utilizado.
- El término se utiliza, sobre todo, para realizar cambios en el software personalizado.
- Normalmente el mantenimiento no genera grandes cambios en la arquitectura del sistema.
- Los cambios implican modificar componentes existentes y agregar nuevos componentes al sistema.

Tipos de mantenimiento

- Mantenimiento para reparar defectos o vulnerabilidades del software
  - Los errores pueden haber sido introducidos en diferentes etapas, lo que influye en el costo de la corrección
- Mantenimiento para adaptar el software a un entorno operativo diferente
  - Cambios en el sistema para que este opere en entornos diferentes a los de la implementación inicial
- Mantenimiento para agregar o modificar funcionalidades al sistema
  - Modificar el sistema para satisfacer nuevos requisitos

Distribución del esfuerzo de mantenimiento



(Davidsen and Krogstie 2010).

Tipos de mantenimiento

- Otra categorización (Pfleeger, 2010)
  - **Correctivo** (21%)
    - Control del funcionamiento diario del sistema a través de la reparación de fallas
  - **Adaptativo** (25%)
    - El sistema de modifica para adaptarse a cambios en el entorno
  - **Perfectivo** (50%)
    - Mejorar funcionalidades existentes
  - **Preventivo** (4%)
    - Prevenir que el desempeño del software se degrade

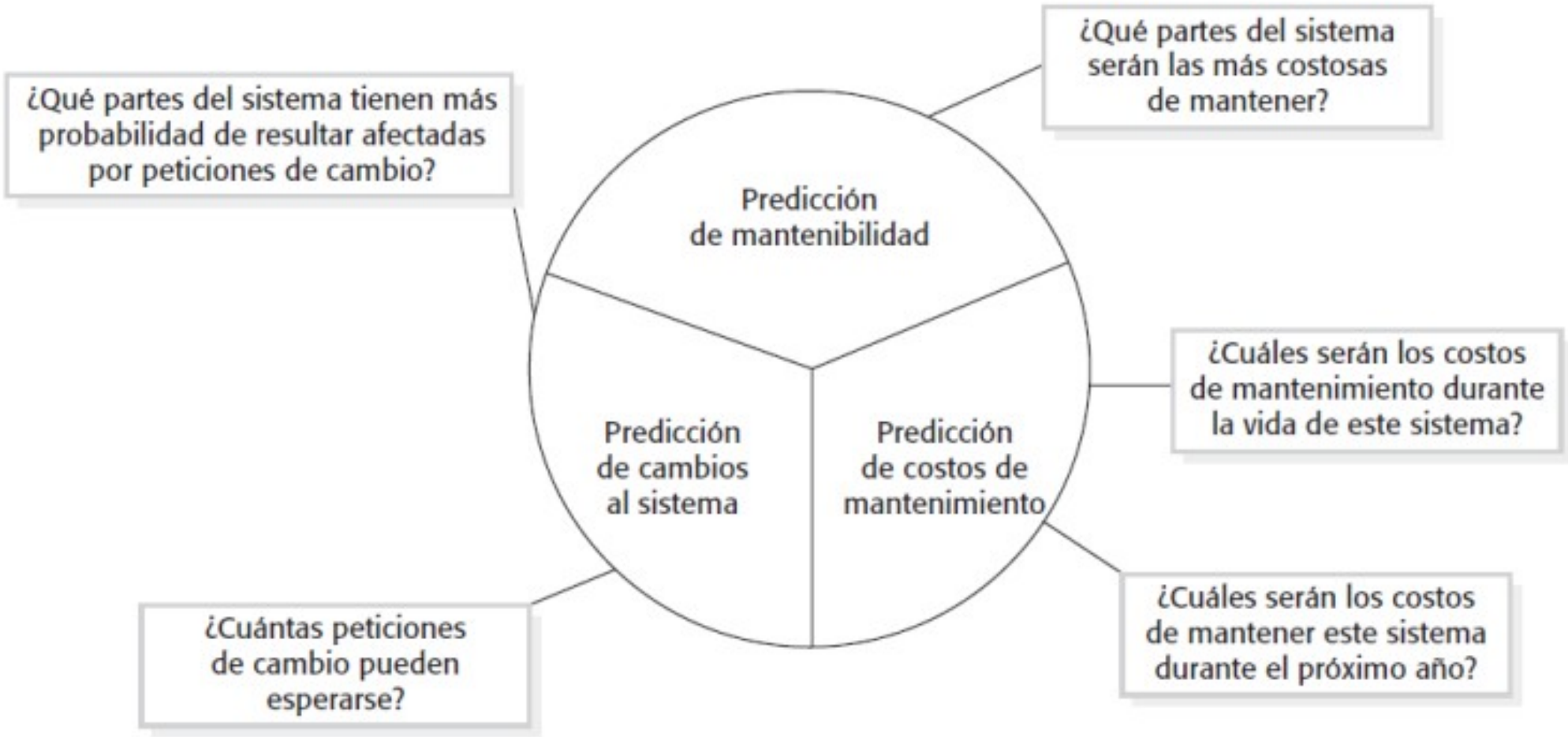
Costos de mantenimiento

- Por lo general, mayores que los costos de desarrollo.
  - Agregar funcionalidades nuevas es más costoso que haberlo hecho durante el desarrollo inicial
- El equipo de mantenimiento tiene que entender el sistema
- Falta de motivación para que el equipo de desarrollo escriba software mantenible
- La tarea de mantenimiento del software no es popular
- El mantenimiento corrompe la estructura del software y así se hace aún más difícil el mantenimiento.

Predicción del mantenimiento

- La predicción del mantenimiento se ocupa de evaluar qué partes del sistema pueden causar problemas y cuál serán los costos de mantenimiento.
  - La aceptación de los cambios en el mantenimiento depende de la capacidad de cambio de los componentes afectados por dicho cambio;
  - La implementación de los cambios degrada el sistema y reduce su capacidad de mantenimiento;
  - Los costos de mantenimiento dependen del **número de cambios** y el **costo del cambio** depende de la **capacidad de mantenimiento**.

Predicción de mantenimiento



Predicción del cambio

- Es la predicción de la cantidad de cambios requeridos y la comprensión de las relaciones entre el sistema y su entorno.
- Los sistemas fuertemente acoplados requieren cambios cada vez que cambia su entorno.
- Los factores que influencian esta relación son:
  - Cantidad y complejidad de las interfaces del sistema,
  - Cantidad de requisitos del sistema que son volátiles de forma inherente,
  - Los procesos de negocio donde el sistema es utilizado.

Métricas de complejidad

- Las predicciones de mantenimiento se pueden realizar mediante la evaluación de la complejidad de los componentes del sistema.
- Estudios muestran que el mayor esfuerzo de mantenimiento se gasta en un número relativamente pequeño de componentes.
- La complejidad depende de
  - La complejidad de la estructura de control;
  - La complejidad de la estructura de datos;
  - Objetos, métodos y el tamaño de los módulos.

Métricas de proceso

- Las métricas de proceso pueden ser utilizadas para evaluar la mantenibilidad.
  - Cantidad de solicitudes de mantenimiento correctivo;
  - Promedio de tiempo para el análisis del impacto;
  - Promedio de tiempo para implementar la solicitud de cambio;
  - Cantidad de solicitudes de cambios excepcionales.
- Si alguno se incrementa puede indicar que está disminuyendo la mantenibilidad.

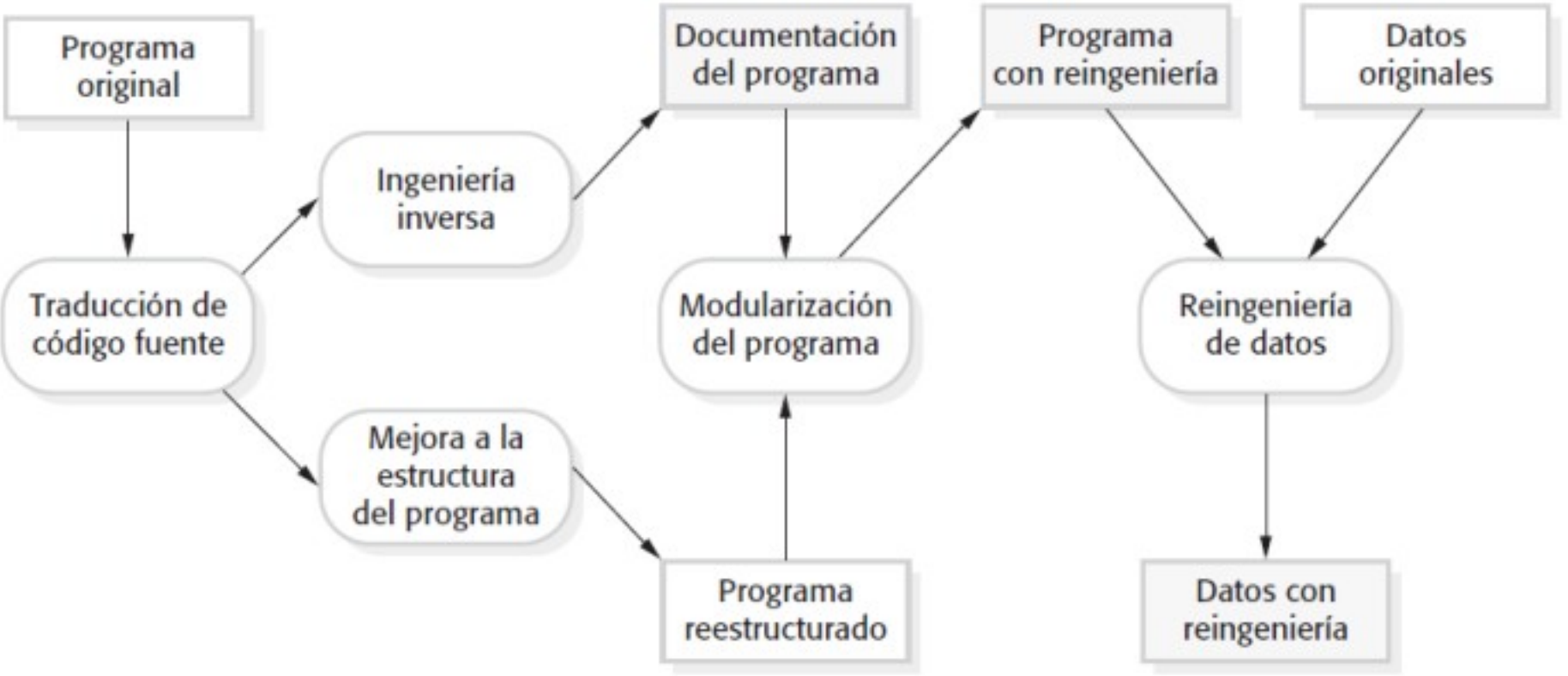
Reingeniería del software

- Es la reestructuración o la reescritura de parte o todo un sistema heredado sin cambiar su funcionalidad.
- Es aplicable cuando los sistemas necesitan un mantenimiento frecuente.
- La reingeniería involucra agregar esfuerzo para hacer que sea más fácil mantener el sistema.
  - El sistema puede ser reestructurado y redocumentado.

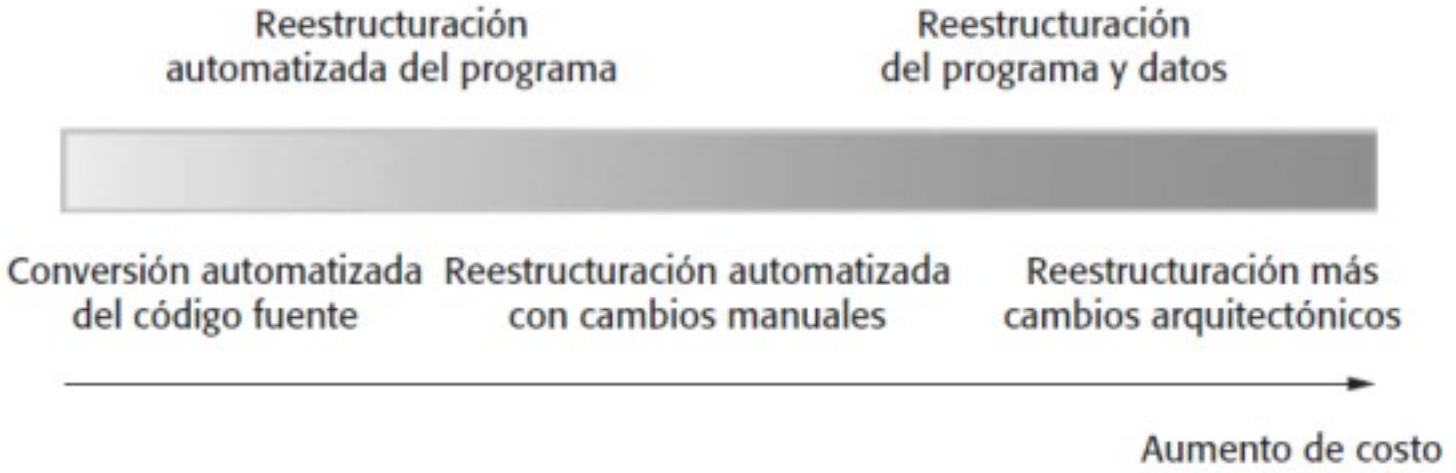
Ventajas de la reingeniería

- Reducción del riesgo.
  - Hay un alto riesgo en el desarrollo de un nuevo software. Pueden haber problemas de desarrollo, problemas de equipo y problemas de especificación.
- Reducción del costo.
  - Generalmente los costos de la reingeniería son mucho menos significantes que los costos de desarrollar nuevo software.

El proceso de reingeniería



Enfoques de la reingeniería



Factores de costo

- La calidad del software a ser rediseñado.
- Las herramientas de soporte disponibles para la reingeniería
- El grado de conversión de datos que se requiere.
- La disponibilidad de personal experto para la reingeniería.
  - Esto puede ser un problema cuando los sistemas están basados en tecnología vieja u obsoleta.

Refactorización

- La refactorización es el proceso de hacer mejoras a un programa para reducir la degradación del sistema al realizar cambios.
- Se puede pensar en la refactorización como un “mantenimiento preventivo” que reduce los problemas en cambios futuros.
- La refactorización trata de modificar un programa para mejorar su estructura, reducir su complejidad o hacerlo más fácil de entender.
- Cuando se refactoriza un programa no se deben agregar funcionalidades.
- Es inherente a las metodologías ágiles.

La refactorización y la reingeniería

- La reingeniería toma lugar luego de que el sistema ha sido mantenido por algún tiempo y los costos se incrementan.
  - Utiliza las herramientas automatizadas para procesar y rediseñar sistemas heredados para crear un nuevo sistema más mantenible.
- La refactorización es un proceso continuo de mejora en todo el proceso de desarrollo y en la evolución.
  - Se tiene la intención de evitar que se incremente la degradación de la estructura y el código para que así no se dificulte el mantenimiento del sistema.

"*Bad smells*" en el código

Señales de que el código puede ser mejorado.

¿Qué "bad smell" puede haber en el código?

"*Bad smells*" en el código

- Código duplicado.
  - El mismo código o muy similar es incluido en diferentes lugares del sistema.
  - Estos pueden ser removidos e implementados como una única función o método que puede invocarse cuando se requiere.
- Métodos largos.
  - Si el método es demasiado largo, debe ser rediseñado en métodos mas cortos.
- Enunciados Switch (case)
  - Generalmente implica duplicación. El cambio (switch) depende del tipo de un valor. Estos enunciados pueden estar dispersos en el programa.
  - Normalmente en los lenguajes orientados a objetos se puede utilizar el polimorfismo para lograr el mismo comportamiento.

"*Bad smells*" en el código

- Aglomeración de datos.
  - La aglomeración de datos ocurre cuando el mismo grupo de ítems de datos (campos en clases, parámetros en métodos) aparecen en varios lugares del programa.
  - Generalmente puede ser remplazado con un objeto que encapsule todos estos datos.
- Generalidad especulativa.
  - Esto ocurre cuando los desarrolladores incluyen generalidad al programa en caso de que sea necesario en el futuro. Por lo general, puede ser removido de forma simple.