# Tema 8: Despliegue, Evolución y Mantenimiento del Software

## BLOQUE V: MANTENIMIENTO DE LOS SISTEMAS SOFTWARE

Ingeniería del Software
Grado en Ingeniería Informática
Curso 2024/2025

**David Cáceres Gómez**

UNIVERSIDAD
D
CÓRDOBA

# Índice

**1. Introducción** **2**

**2. Proceso de Despliegue** **4**

**3. Evolución del Software** **7**

**4. Mantenimiento del Software** **9**

**5. Proceso de Reingeniería** **12**

**Referencias** **13**

# 1. Introducción

¿Y después de las pruebas?

Una vez que el software ha sido construido y sometido a pruebas exhaustivas, no se considera que el trabajo esté completamente terminado. En realidad, este es solo el punto de partida para una serie de actividades cruciales que garantizarán la utilidad y la sostenibilidad del software en el tiempo. Estas actividades, que forman parte del ciclo de vida del software, se enfocan en:

1. **La puesta a disposición para los clientes o usuarios**

   El software probado debe ser preparado para su uso real. Esto implica:

   - La instalación o despliegue en un entorno operativo adecuado.
   - La configuración inicial para satisfacer las necesidades del cliente.
   - La capacitación de los usuarios finales o administradores para su correcta utilización.

   Este proceso asegura que el software esté disponible y sea utilizable de manera efectiva.

2. **La incorporación de nuevas funcionalidades y la mejora de las existentes**

   A medida que los usuarios interactúan con el software, surgen necesidades adicionales. Estas pueden incluir:

   - Nuevas características para aumentar el alcance o el valor del sistema.
   - Mejoras en las funcionalidades existentes para optimizar su desempeño o adaptarlas a mejores prácticas.

   Estas actividades son esenciales para mantener la relevancia del software en un entorno dinámico.

3. **La corrección de errores no detectados durante las pruebas**

   Por más rigurosas que sean las pruebas realizadas, es posible que algunos defectos no se detecten. Una vez que el software se encuentra en uso, estos errores pueden emerger debido a:

   - Casos de uso imprevistos.
   - Datos específicos del entorno real.
   - Limitaciones en las pruebas realizadas previamente.

La corrección de estos errores es una tarea continua y crítica para garantizar la confiabilidad del sistema.

4. **La mejora en el modo de funcionamiento del sistema en nuevos entornos**

   El entorno operativo de un software puede cambiar con el tiempo, por ejemplo, debido a actualizaciones en hardware, sistemas operativos o infraestructuras tecnológicas. El software debe ser ajustado para:

   - Asegurar la compatibilidad con estas nuevas condiciones.
   - Aprovechar nuevas capacidades tecnológicas que mejoren su rendimiento.

5. **La readaptación de la concepción del sistema ante cambios en el negocio**

   Las empresas y organizaciones evolucionan constantemente, y el software debe adaptarse a:

   - Nuevos procesos de negocio o cambios en los existentes.
   - Regulaciones o normativas actualizadas.
   - Cambios en las expectativas del mercado o los usuarios.

   Esta readaptación asegura que el software siga alineado con los objetivos estratégicos de la organización.

Todas estas actividades forman parte del ciclo de vida del software. Estas actividades no son eventos aislados, sino que constituyen fases esenciales dentro del ciclo de vida del software. Para abordarlas de manera efectiva, se requiere un enfoque estructurado que incluye:

- **Metodologías tradicionales o ágiles que faciliten la labor de los ingenieros** y permitan abordar proyectos según sus necesidades, ya sea con iteraciones rápidas, procesos estructurados o integración continua para agilizar entregas y mejorar la colaboración.
- **Herramientas de soporte para la gestión y automatización de tareas** del sistema que optimizan el mantenimiento, reducen esfuerzos manuales y aseguran un desempeño eficiente del software.

## 2. Proceso de Despliegue

El proceso de despliegue consiste en poner el software o sus actualizaciones a disposición de los usuarios para su uso en un entorno real. Este proceso abarca una serie de operaciones clave, como la preparación y lanzamiento de versiones, configuración e instalación del sistema, y monitoreo del rendimiento tras la puesta en marcha [1]. Actualmente, las técnicas de despliegue combinan procesos manuales y automáticos, lo que facilita la distribución eficiente de actualizaciones y parches.

Las operaciones más comunes en el despliegue del software son:

- **Preparación y lanzamiento de versiones:** Compilación de los cambios aprobados en una nueva versión lista para ser distribuida.
- **Configuración e instalación:** Adaptación del software al entorno específico en el que será usado, asegurando compatibilidad y funcionamiento.
- **Monitoreo del rendimiento:** Supervisión continua para identificar posibles problemas y garantizar la estabilidad del sistema.

El despliegue suele seguir tres pasos fundamentales:

1. **Recibir peticiones:** Identificar cambios o mejoras requeridas, como *pull requests* o actualizaciones necesarias.
2. **Elegir e implementar peticiones:** Evaluar, aceptar e integrar las peticiones al sistema, considerando su impacto y viabilidad.
3. **Hacer los cambios disponibles:** Liberar una nueva versión del software que incluya los cambios implementados.

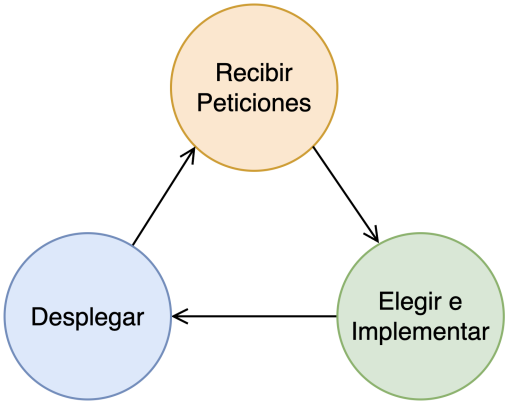


Figura 1: Pasos en el despliegue del software

El proceso de despliegue puede variar significativamente según las características del sistema y su entorno operativo. Cada organización debe definir un flujo de trabajo claro, estableciendo actividades, prácticas y herramientas adecuadas a sus necesidades.

Cuando se decide incorporar una nueva funcionalidad, el software pasa por tres estados esenciales:

1. **Preparación:** Se recopila todo el código y recursos necesarios, como configuraciones y librerías, para su integración.
2. **Prueba:** El código preparado es evaluado exhaustivamente en un entorno específico de pruebas (*staging*) para evitar la introducción de errores, utilizando casos como pruebas de regresión.
3. **Despliegue:** Una vez probado, el código es integrado en el entorno de producción, quedando disponible para los usuarios junto con las funcionalidades previas.

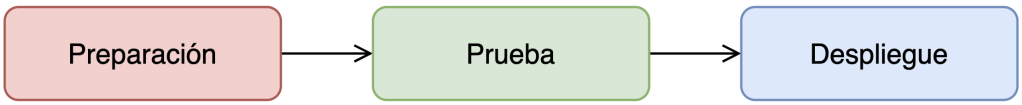


Figura 2: Etapas en el despliegue del software

Disponer de un proceso bien estructurado y riguroso para el despliegue del software trae consigo múltiples beneficios. En primer lugar, facilita la incorporación segura de nuevas funcionalidades que responden rápidamente a las necesidades del negocio, garantizando un impacto positivo en la evolución del sistema. Este enfoque también mejora las operaciones de la organización al optimizar procesos y aumentar la eficiencia general. Además, un proceso controlado permite monitorear cada etapa, recolectar datos valiosos y utilizarlos para mejorar continuamente el despliegue.

Cuando el despliegue está correctamente automatizado, se aceleran las pruebas, y los cambios se hacen transparentes y accesibles para los usuarios, lo que reduce la incertidumbre y aumenta la confianza en el sistema.

### Despliegue Continuo (CD) y su Integración con CI

En la actualidad, muchas organizaciones adoptan el enfoque de **despliegue continuo** (CD), que consiste en un sistema totalmente automatizado para desplegar nuevas funcionalidades con alta frecuencia. Este enfoque se combina comúnmente con la **integración continua** (CI), donde los cambios en el código se integran continuamente en la rama principal. El modelo CI/CD asegura una entrega fluida, rápida y confiable de nuevas versiones del software.

Para maximizar la efectividad del despliegue continuo y la integración continua, es importante implementar las siguientes prácticas recomendadas [2]:

- **Mantener un repositorio único de código:** Centralizar el código fuente facilita la colaboración y evita la duplicación de esfuerzos.
- **Automatizar la construcción (*build*):** Un sistema automatizado asegura que las versiones del software se generen de manera consistente y confiable.
- **Pruebas automáticas:** Implementar pruebas automatizadas permite detectar errores de forma temprana y mantener la calidad del código.
- ***Commits* diarios a la rama principal:** Fomenta la integración frecuente para evitar conflictos complejos y mantener un estado funcional del sistema.
- **Integración correcta de *commits*:** Cada cambio debe ser probado y verificado para garantizar su compatibilidad con la versión actual.
- ***Bugs* corregidos con casos de prueba asociados:** Cada error solucionado debe ir acompañado de pruebas que aseguren que no vuelva a ocurrir.
- **Construcciones rápidas:** Mantener tiempos cortos de construcción agiliza los ciclos de prueba y despliegue.
- **Pruebas en un entorno específico (*staging*):** Utilizar un entorno intermedio permite validar los cambios antes de liberarlos en producción.
- **Acceso rápido y fácil a nuevas funcionalidades:** Los usuarios y probadores deben poder acceder fácilmente a las nuevas características para evaluar su impacto.
- **Transparencia en errores de construcción:** Si el proceso de construcción falla, todos los involucrados deben poder identificar rápidamente el problema y tomar medidas correctivas.

# 3. Evolución del Software

La evolución del software es un proceso inevitable y necesario para que los sistemas sigan siendo útiles y relevantes en un entorno cambiante. Los cambios y la evolución forman parte integral de la ingeniería del software, y reflejan la necesidad de adaptar los sistemas a nuevas condiciones, requerimientos y expectativas.

Los cambios en el software pueden ser debidos a:

- **Nuevas situaciones en el negocio:** Cambios en procesos, modelos de negocio o estrategias que exigen ajustes en el software.
- **Expectativas de los clientes:** Nuevos requisitos o funcionalidades solicitadas para satisfacer demandas actuales.
- **Adaptación tecnológica:** Necesidad de operar en nuevos entornos de hardware o software.
- **Mejoras de rendimiento y aspectos no funcionales:** Incrementar la eficiencia, seguridad, escalabilidad u otras características técnicas del sistema.

La frecuencia de lanzamiento de versiones ha aumentado notablemente en las últimas décadas. Lo que antes ocurría cada varios años ahora puede suceder en meses, semanas, o incluso diariamente con enfoques modernos como CI/CD. Esto es especialmente relevante en sistemas empresariales, donde los cambios en un componente pueden tener un impacto significativo en otros sistemas interconectados.

El proceso de evolución del software puede variar en formalidad según el tipo de sistema. Generalmente, comienza con la identificación y propuesta de cambios por parte de los interesados, ya sean clientes, usuarios o el equipo de desarrollo, y culmina con la generación de una nueva versión del sistema que incorpora dichas modificaciones.

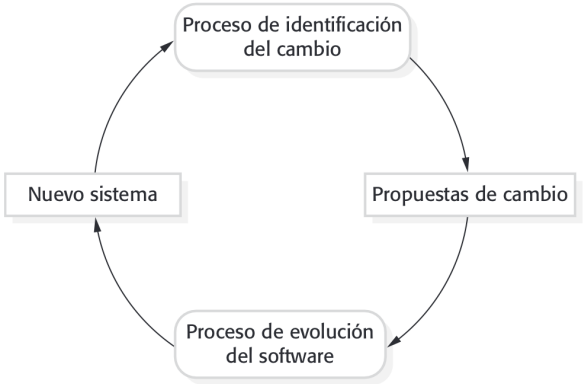


Figura 3: Identificación de los cambios y procesos de evolución [3]

## El Proceso de Evolución del Software

La evolución del software implica analizar cuidadosamente cómo un cambio afecta tanto al sistema como a su entorno operativo. Este proceso puede ser llevado a cabo por la empresa que desarrolló el software, la que lo adquirió, o incluso por terceros. La formalidad del proceso puede variar según el tipo de sistema y su criticidad. Este proceso tiene las siguientes etapas:

1. **Identificación y propuesta de cambios:** Cualquier interesado (clientes, usuarios o el propio equipo) puede proponer cambios al sistema. Estas propuestas deben ser evaluadas para determinar su viabilidad.
2. **Análisis de impacto:** Se debe identificar qué partes del sistema se verán afectadas por los cambios, evaluando tanto el coste asociado como el impacto en el funcionamiento general del software y sus dependencias.
3. **Planificación de la nueva versión:** Incluir los cambios aprobados, como la corrección de errores, adaptaciones tecnológicas o la adición de nuevas funcionalidades, y establecer un cronograma para su implementación y pruebas.
4. **Implementación y validación:** Realizar los cambios en el código y validar la nueva versión para asegurar que cumpla con los requisitos establecidos.
5. **Entrega del sistema:** Finalmente, se entrega la nueva versión del sistema a los usuarios o al entorno de producción.

En situaciones críticas, como vulnerabilidades de seguridad o inoperatividad del sistema, el proceso de evolución puede simplificarse para completar los cambios con la mayor rapidez posible.

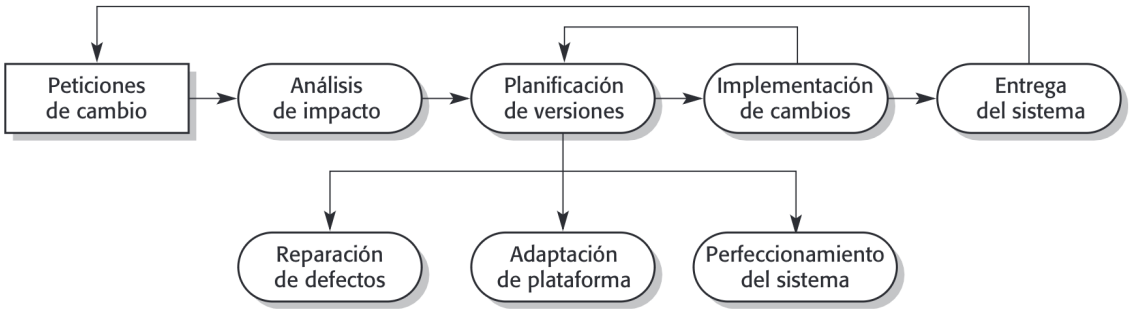


Figura 4: Proceso de evolución del software [3]

# 4. Mantenimiento del Software

El mantenimiento del software abarca todos los procesos necesarios para realizar cambios en un sistema después de su entrega (delivery). Este proceso es especialmente relevante en sistemas desarrollados a medida para empresas clientes, donde los equipos encargados del desarrollo y del mantenimiento suelen ser diferentes.

Los cambios en el software pueden variar desde simples correcciones de errores en el código hasta modificaciones más profundas que implican rediseños o la incorporación de nuevos requisitos. Entre las actividades de mantenimiento más comunes se incluyen:

- **Reparación de fallos:** Corregir errores y vulnerabilidades que afectan el funcionamiento del sistema.
- **Adaptación a nuevas plataformas o entornos de operación:** Ajustar el software para que sea compatible con nuevas tecnologías o sistemas operativos.
- **Incorporación de nuevas funcionalidades o soporte para nuevos requisitos:** Agregar nuevas características para satisfacer necesidades emergentes del cliente o mejorar el rendimiento.

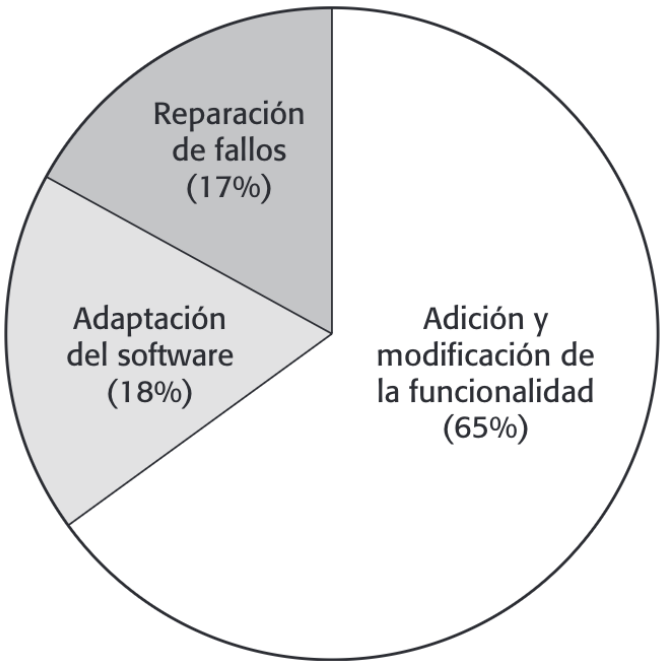


Figura 5: Distribución del esfuerzo de mantenimiento [3]

### Tipos de Mantenimiento

Existen varios tipos de mantenimiento del software, cada uno enfocado en aspectos específicos de la evolución y sostenibilidad del sistema:

- **Mantenimiento Correctivo:** Se enfoca en corregir errores o fallos en el sistema después del despliegue, como *bugs* o vulnerabilidades, que afectan su funcionamiento. Este tipo de mantenimiento es esencial para asegurar que el sistema continúe operando correctamente.
- **Mantenimiento Adaptativo:** Consiste en modificar el software para que sea compatible con nuevos entornos, plataformas o tecnologías, como la actualización a nuevas versiones de sistemas operativos o hardware. Permite que el software siga siendo útil y funcional en un entorno tecnológico cambiante.
- **Mantenimiento Perfectivo:** Se centra en mejorar el rendimiento del sistema o agregar nuevas funcionalidades, en respuesta a cambios en los requisitos del negocio o a la demanda de nuevas características. Esto puede incluir la optimización del código, la mejora de la interfaz de usuario o la incorporación de nuevas funciones.
- **Mantenimiento Preventivo:** Su objetivo es anticipar posibles problemas y evitar que ocurran errores futuros. Esto implica la refactorización del código, la mejora de la documentación y la revisión de la arquitectura del sistema para mantenerlo eficiente y sostenible a largo plazo.

### Costes del Mantenimiento

Aunque la reparación de fallos no suele ser la tarea de mantenimiento más costosa, la adición de nuevas funcionalidades representa una inversión más significativa. Estos cambios requieren un análisis detallado y a menudo, un rediseño del sistema. Además, el equipo de mantenimiento debe tener un buen entendimiento del sistema para implementar cambios sin comprometer su estabilidad. En muchos casos, el equipo de desarrollo original no estuvo tan involucrado en la creación de un código fácilmente mantenible, lo que puede hacer que las modificaciones sean más complicadas y costosas.

### Desafíos del Mantenimiento del Software

El mantenimiento del software a menudo se percibe como tareas menos creativas o relevantes, lo que puede llevar a que estas actividades sean delegadas a personal con menos experiencia. Además, trabajar con sistemas que utilizan tecnologías o lenguajes obsoletos puede complicar aún más el proceso. A medida que el sistema

crece y se agregan más funcionalidades, su estructura se vuelve más compleja, lo que hace que la implementación de nuevos cambios sea más difícil. La documentación también tiende a volverse inconsistente o desactualizada, lo que dificulta la comprensión y modificación del sistema a largo plazo.

El mantenimiento del software, aunque esencial para su sostenibilidad y evolución, presenta retos significativos que requieren un enfoque planificado y bien gestionado para garantizar que el sistema siga funcionando de manera eficiente y continúe satisfaciendo las necesidades del negocio.

### Predicción del Mantenimiento

En la gestión de software, anticiparse a problemas es clave para evitar sorpresas costosas e inesperadas. Por ello, es fundamental predecir los posibles cambios que requerirá un sistema, identificar las partes más difíciles de mantener y estimar los costes de mantenimiento a lo largo de un periodo determinado. Estas predicciones están interrelacionadas de la siguiente manera:

- La decisión de aceptar o rechazar un cambio depende, en parte, de la facilidad con que los componentes del sistema afectados puedan ser mantenidos.
- La implementación de cambios tiende a degradar la estructura del sistema, reduciendo su mantenibilidad a largo plazo.
- Los costes de mantenimiento están directamente relacionados con el número de cambios, mientras que los costes de implementación dependen de la mantenibilidad de los componentes del sistema.

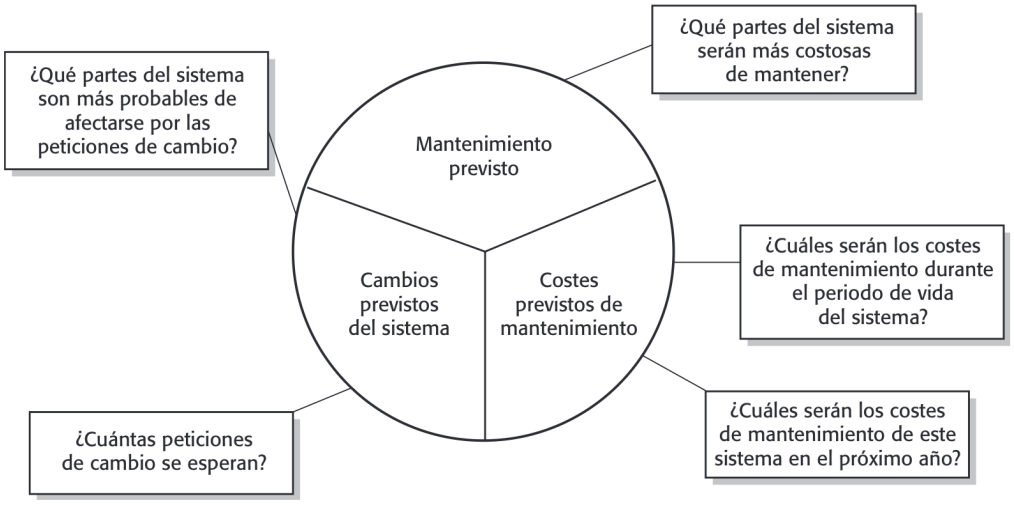


Figura 6: Predicción del mantenimiento

# 5. Proceso de Reingeniería

La reingeniería del software es un proceso aplicado a sistemas heredados (*legacy systems*), aquellos que llevan mucho tiempo en funcionamiento y cuya actualización puede ser compleja. Este enfoque es especialmente útil cuando realizar cambios directamente sobre el sistema no es viable o cuando reemplazarlo completamente por uno nuevo resulta poco práctico o demasiado arriesgado.

Optar por un proceso de reingeniería en lugar de desarrollar un nuevo sistema desde cero puede ser beneficioso por varias razones:

- **Reducción del riesgo:** En sistemas críticos para el negocio, un retraso en la implementación de un nuevo sistema podría resultar en pérdidas significativas. Modernizar un sistema existente minimiza estas incertidumbres.
- **Reducción de costes:** Al reutilizar y mejorar componentes del sistema existente, se evitan los gastos asociados con el desarrollo completo de un sistema nuevo. Por ejemplo, puede ser suficiente modernizar solo la tecnología subyacente.

El propósito de la reingeniería es obtener una versión mejorada y reestructurada de un sistema existente, optimizando su funcionamiento, adaptabilidad y sostenibilidad. Este proceso no siempre requiere aplicar todas las actividades disponibles; la selección depende de las necesidades específicas del sistema y del negocio.

Las actividades del proceso de reingeniería son:

1. **Traducción de código:** Consiste en migrar el código fuente a una versión más moderna del lenguaje de programación o incluso a un lenguaje completamente diferente, mejor adaptado a las necesidades actuales.
2. **Ingeniería inversa:** Analizar el código fuente para comprender su estructura, diseño y funcionalidades, facilitando su modificación y mejora, especialmente cuando la documentación original está desactualizada.
3. **Mejora de la estructura del programa:** Reorganizar el código mediante técnicas como la refactorización, con el objetivo de hacerlo más claro, eficiente y mantenible.
4. **Modularización del programa:** Reestructurar el sistema a nivel arquitectónico, dividiendo el software en módulos más independientes y cohesivos, lo que mejora su flexibilidad y escalabilidad.

5. **Reingeniería de datos:** Redefinir tablas, esquemas y estructuras de las bases de datos asociadas al sistema, para optimizar su desempeño y garantizar que sean compatibles con tecnologías actuales.

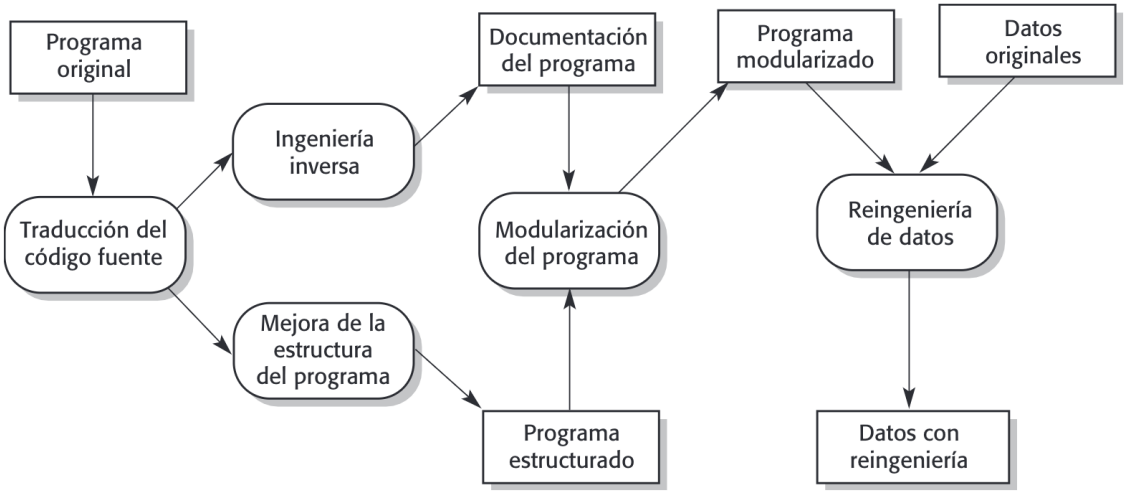


Figura 7: El proceso de reingeniería [3]

# Referencias

[1] Software Deployment, `https://www.atatus.com/glossary/software-deployment/`.

[2] Paul Duvall, Steve Matyas y Andrew Glover, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.

[3] I. Sommervile, Ingeniería del Software, 9th Edition, Pearson, 2011.