Informe Técnico – Technical Report
DPTOIA-IT-2004-003
enero, 20054

METODOLOGÍA DE REINGENIERÍA DEL SOFTWARE PARA LA REMODELACIÓN DE APLICACIONES CIENTÍFICAS HEREDADAS

Juan Carlos Álvarez García
Montserrat Mateos Sánchez
María N. Moreno García

Departamento de Informática y Automática

Universidad de Salamanca

Revisado por:

Dr. Francisco José García Peñalvo

Dra. Vivian F. López Batista

Aprobado en el Consejo de Departamento de 15 de julio de 2004

Información de los autores:

Juan Carlos Álvarez García

Área de Lenguajes y Sistemas Informáticos

Departamento de Informática y Automática

Facultad de Ciencias – Universidad de Salamanca

Plaza de la Merced S/N – 37008 – Salamanca

jcag@usal.es

Montserrat Mateos Sánchez

Escuela Universitaria de Informática

Universidad Pontificia de Salamanca

C/ Compañía, 5 – 37002 – Salamanca

m.mateos@upsa.es

María N. Moreno García

Área de Lenguajes y Sistemas Informáticos

Departamento de Informática y Automática

Facultad de Ciencias – Universidad de Salamanca

Plaza de la Merced S/N – 37008 – Salamanca

mmg@usal.es

Resumen

La reingeniería de sistemas heredados tiene por finalidad reestructurar o transformar viejos sistemas en aplicaciones más fáciles de mantener, con entornos más agradables e integradas en nuevas plataformas de hardware/software. Tratamos de dar una visión general de lo que es la reingeniería de software y cuáles son las actitudes que se pueden tomar a la hora de modernizar un sistema legado que se ha quedado obsoleto en cuanto a operatividad, aspecto y software de base sobre el que se ejecuta, pero de probada eficiencia y que mantiene su funcionalidad. Mostramos las características de un proceso de desarrollo que se adapta a este tipo de aplicaciones, verificado, mediante el caso de estudio, la transformación de una aplicación escrita en un lenguaje imperativo, no estructurado, a un nuevo lenguaje visual y orientado a objetos, describiendo las diversas fases de la metodología aplicadas a un caso concreto.

Abstract

Legacy Systems Reengineering has as primary goal to restructure and transform old systems in easier to maintain applications with nicer and more integrated environments that they run in new platforms of hardware/software. We try to give a general view of what is the software reengineering and which are the approaches that can be taken when modernizing a legacy system that has been obsolete as for operability, but of proved efficiency and that maintains its functionality. We show the characteristics of a development process that adapts itself to this type of systems, verified by means of the case of study, the transformation of an application written in an imperative language, no structured, to a new visual and object oriented language, describing the different stages of the methodology applied to a particular case.

Tabla de Contenidos

Tabla de Figuras

1 Introducción

Este trabajo presenta una nueva metodología de reingeniería del software para remodelar aplicaciones de tipo científico (con unas características concretas), que por diversos motivos presentan la necesidad de tener que actualizarse. La metodología surge como consecuencia de la aplicación de procesos de reingeniería del software tradicionales, sobre una aplicación científica de considerables dimensiones y por la no consecución de buenos resultados mediante esas técnicas, que en la mayoría de los casos cuenta con una probada validez, pero que aplicadas al tipo de aplicación del caso de estudio no fueron satisfactorias.

Se han establecido las pautas a seguir para actualizar aplicaciones que por distintos factores se han quedado obsoletas y por tanto se desean reconvertir a entornos de trabajo actuales. La metodología se ha utilizado para transformar una aplicación del área química para el tratamiento de datos cinéticos de absorbancia en mezclas. Las causas que motivan una actualización en el software pueden ser muy variadas, en este caso venían derivadas por una actualización de los equipos sobre los que trabajaba la aplicación, tanto a nivel hardware como software. Sobre todo a nivel sistema operativo, dejando de lado MS-DOS, que era el sistema sobre el que estaba trabajando. Al tratarse de una aplicación en modo texto presentaba muchas limitaciones sobre todo relacionadas con la entrada y salida de datos. Así, la introducción de datos se realizaba de forma secuencial, por lo que si se producía algún error en la introducción de los datos (valores no adecuados, anomalías en los ficheros manejados, errores de formato, ausencia de datos), el programa abortaba.

Se trataba por tanto de desarrollar una aplicación multiplataforma, implementada en un entorno gráfico, que incorporara todas las ventajas de estos sistemas: menús, verificación de datos, ayudas, etc. Pero además se buscaba que el sistema fuera más mantenible, estuviera documentado y fuera fácil de ampliar. Se trataba pues, de un sistema legado, en el que era necesario reescribir o reestructurar parte o todo el sistema sin cambiar su funcionalidad.

Cuando se trata de una aplicaciones de tamaño considerable y de complejidad algorítmica, se intenta mantener la mayoría de los elementos del software que sean posibles. En nuestro caso no disponíamos de muchos de estos elementos, pero siempre que ha sido posible se han utilizado. No contábamos ni con planes de proyecto, ni estimaciones de coste (tampoco en este caso eran necesarias), ni arquitectura, ni especificaciones y modelos de requisitos ni diseños. Contábamos con el código fuente en Fortran, con documentación de usuario y técnica, con interfaces humanas, con datos y con casos de prueba. Se ha reutilizado parte del código fuente, se han usado ambos tipos de documentación que han sido muy importantes. Se han descartado las interfaces humanas ya que era uno de los puntos que había que mejorar, si bien han servido para comprender la aplicación y determinar aspectos de funcionamiento. Se han mantenido los datos, ya que era necesario hacer compatible la nueva versión con los ficheros existentes, para poder seguir utilizando muchos de los datos recogidos. Se han utilizado los casos de uso que estaban recogidos junto con la documentación.

Este documento se ha organizado de la siguiente forma: La sección 2, describe los principales aspectos y fases clásicas de la reingeniería del software. La sección 3 expone las características del proceso de reingeniería que hemos desarrollado para sistemas heredados. La sección 4 analiza el caso de estudio en el que la metodología ha sido probada, pormenorizando cada una de las fases. Finalmente, en la sección 5 se presentan las conclusiones.

2 Reingeniería del software

2.1 Proceso de reingeniería de software

Son muchas y variadas las referencias que se pueden encontrar del concepto de reingeniería. Algunos, como Arnold [1], la definen como una actividad que mejora la comprensión del software, o bien, lo prepara o mejora para incrementar su facilidad de mantenimiento, reutilización o evolución. Para otros [4], es el examen y alteración de un sistema para reconstruirlo en una nueva forma y la subsiguiente implementación de esa forma. Otros lo ven como el proceso de ingeniería inversa seguida de una ingeniería directa. El concepto de reingeniería esta muy relacionado con el concepto reutilización, y así se puede comprobar en [3], donde Biggerstoff, se refiere a la reutilización como la reaplicación de una variedad de tipos de conocimientos de un sistema a otro para reducir el esfuerzo de desarrollo y mantenimiento de ese otro sistema; es decir, la reutilización está enfocada a mejorar la calidad y reducir el esfuerzo haciendo uso de parte de un sistema en un nuevo contexto. En definitiva, el concepto de reingeniería de software se refiere a la reutilización de sistemas heredados pero transformándolos para hacerlos más mantenibles. Se trata pues de cualquier procedimiento que produce un sistema mediante la reutilización de algo procedente de algún esfuerzo anterior. Estos sistemas suelen tener algunos problemas como son los expuestos en [14], debido a que normalmente han sido desarrollados y mantenidos por muchas personas, y en muchas ocasiones, utilizando técnicas y estilos de programación propios; además, con el tiempo normalmente las especificaciones han cambiado y el diseño (si es que lo había) se ha perdido. Las dos ventajas fundamentales que presenta la reingeniería son: la reducción del riesgo, ya que si hay una aplicación que funciona se conocen sus resultados y, por tanto, ya se dispone de una especificación del sistema y la reducción del coste; se han realizado estudios [21] que muestran que la reducción del coste puede ser de un 75 por ciento.

Pero, por otra parte, también presenta una serie de dificultades, como las indicadas en [15]: falta de planificación exhaustiva para la reutilización del software, no utilización por parte de los desarrolladores de software de herramientas o componentes diseñados específicamente para ayudar e impulsar la reutilización, falta de entrenamiento para ayudar a ingenieros de software y administradores a comprender y explicar la reutilización, resistencia del personal especializado contra el concepto de reutilización, propugnar metodologías que no facilitan la reutilización, como puede ser la descomposición funcional en detrimento de enfoques orientados a objetos, falta de incentivos en las compañías para producir componentes reutilizables.

No en todos los sistemas es adecuado realizar un proceso de reingeniería. Antes de tomar esa decisión se puede valorar utilizando, por ejemplo, la matriz de decisión de Jacobson [7], para determinar si el sistema tiene un gran valor de negocio y por tanto es conveniente que se aplique reingeniería.

Según [8], se pueden definir diez elementos del software que pueden reutilizarse: planes de proyecto, estimaciones de coste, arquitectura, especificaciones y modelos de requisitos, diseños, código fuente, documentación de usuario y técnica, interfaces humanas, datos, y casos de prueba.

2.2 Fases en la Reingeniería del Software

A pesar de que no hay una definición de proceso de reingeniería, sí que hay trabajos importantes en este área, como el de Rugaber y Wills [17], en el que proponen una serie de pasos para fomentar y avanzar en la investigación en el área de la reingeniería. Además, en más de una ocasión, se ha intentado definir un ciclo de vida para el proceso de reingeniería. Algunos autores [4] lo definen como el proceso de ingeniería inversa seguido de un proceso de ingeniería, es

decir, el proceso de recuperar el diseño del sistema a partir del código fuente para luego volver a aplicar un ciclo de vida de software tradicional. Otros autores lo ven como un proceso de dos pasos, véase la figura 1. El primer paso es comprender el software existente, donde el diseño del sistema se recupera desde su código fuente con actividades como análisis de dependencias, comprensión del programa, detección, extracción y almacenamiento del diseño. El segundo paso incluye todas las actividades que se realizan para transformar el software existente en un software diferente, más fácil de mantener, entre ellas están: descomposición, reestructuración, remodularización, redocumentación, etc.

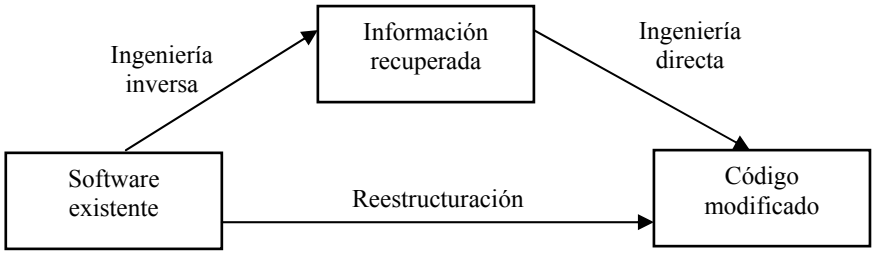

Figura 1. Proceso básico de reingeniería.

Algunos otros trabajos se centran en la reingeniería y la reutilización, como [18], donde se expone como construir o retocar el proceso de reingeniería para reutilizar componentes software existentes; otros autores [10], describen una metodología de reingeniería para reutilización, en la cual, integran técnicas especificas de reutilización dentro del proceso de reingeniería haciendo énfasis en componentes.

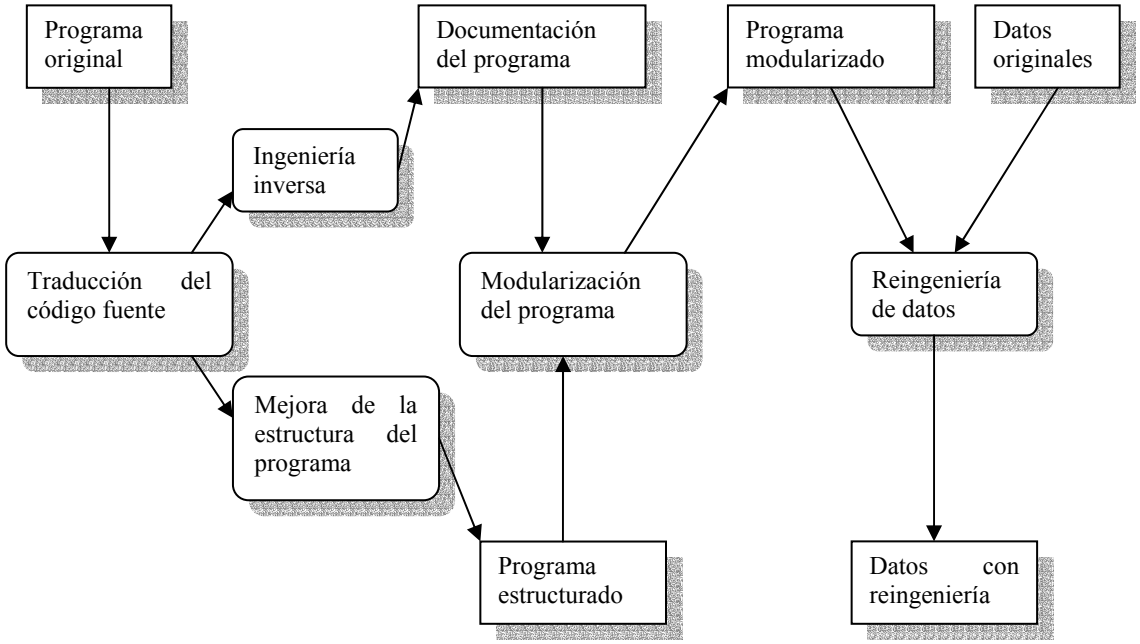

Figura 2. Proceso de reingeniería de Sommerville.

Según [20], la reingeniería de software comprende: la redocumentación del sistema, la reorganización y reestructura del sistema, la traducción del sistema a un lenguaje de programación más moderno, la modificación y actualización de la estructura y los valores de los

datos del sistema. Un posible proceso de reingeniería, más completo, dado por Sommerville, sería el que se recoge en la figura 2.

En el proceso de reingeniería se podrían distinguir las siguientes fases:

- **Traducción del código fuente**. El programa se convierte a una versión más moderna del lenguaje en que estaba codificado o a un lenguaje diferente. Los motivos que llevan a una traducción pueden ser muy diversos: falta de conocimientos del personal en ese lenguaje, falta de soporte en los compiladores, actualización de la plataforma de hardware o de software, políticas de empresa, necesidad de cambio en las interfaces de usuario, etc. El proceso será económicamente rentable, si se dispone de alguna herramienta que realice el grueso de la traducción. En muchos casos, el código que se obtiene tiene que ser modificado de forma manual.
- **Ingeniería inversa**. Se analiza el programa y se extrae información de él, la cual ayuda a documentar su organización y funcionalidad. Es el proceso de analizar el software con el objetivo de recuperar su diseño y especificación. Lo normal es que la entrada a este proceso sea el código fuente si se dispone de él. Se alterna el análisis utilizando herramientas automatizadas con el trabajo manual en el código fuente para obtener el diseño del sistema. La información obtenida suele almacenarse como grafo dirigido, que se va modificando y completando. A partir del grafo se generarán otros documentos como diagramas de estructura de programas, diagramas de estructura de datos y matrices de trazabilidad. Las herramientas que se utilizan para comprender el programa suelen ser de tipo navegadores, que permiten moverse por el código, definir unos datos y rastrearlos por el programa. Suelen ser necesarias anotaciones manuales. En [12] se hace un estudio de 5 herramientas de este tipo (cflow, CIA, field, mkfuncmap, rigiparse).
- **Mejora de la estructura del programa**. Se analiza y modifica la estructura de control del programa para hacerlo más fácil de leer y comprender. Los programas pueden presentar lógica de control no intuitiva lo que puede hacer que no se entiendan fácilmente. El principal factor a tener en cuenta es que el control sea estructurado.
- **Modularización del programa**. Es el proceso de reorganizar un programa de forma que partes relacionadas se integren de forma conjunta. Esto facilita eliminar componentes y mejorar la comprensión. Se pueden considerar diferentes tipos de módulos: abstracciones de datos, módulos de hardware, módulos funcionales, módulos de apoyo al proceso, etc.
- **Reingeniería de datos**. Se trata de analizar y reorganizar las estructuras, e incluso a veces, los valores de los datos de un sistema para hacerlos más comprensibles. Si la funcionalidad del sistema no cambia, la reingeniería de datos no es necesaria.

No son fases que tengan que desarrollarse todas necesariamente, sino que dependiendo de los casos podrán figurar unas u otras.

Otros autores [19] desarrollan un modelo de reingeniería del Software que sitúa al usuario como colaborador principal en la tarea de especificar los requisitos del sistema. Las fases, en este caso serían:

- **Definición del problema.** Se identifican objetivos, límites, beneficios, riesgos, estimaciones de tiempos, etc., estableciendo una imagen real de lo que existe realmente ahora y lo que se quiere obtener en el futuro.
- **Estudio del código antiguo**. Partiendo del código fuente en un lenguaje de tercera generación, se obtiene un conjunto de documentos que ayudan a posteriores fases de la metodología.

- **Viabilidad del proyecto**. Consiste en detectar posibles errores en las especificaciones.
- **Rediseño de especificaciones**. Se busca conseguir, que las especificaciones representen de forma real la visión futura deseada del sistema.
- **Creación de prototipos**. De aquellas partes que puedan dar problemas, o solamente de aquellas que vayan a cambiar sustancialmente de la original.
- **Planificación de la implementación**. Consiste en diseñar la forma y modo en que se va a migrar de una herramienta a otra.
- **Perfeccionamiento**. Realizar cambios en la nueva aplicación que aumenten la calidad del sistema.

2.3 La traducción del código fuente

Se trata de la forma más simple de reingeniería, pero en casi ningún caso un proceso de reingeniería se reduce a una traducción de código. Los motivos que llevan a una traducción pueden ser muy diversos: falta de conocimientos del personal en ese lenguaje, falta de soporte en los compiladores, actualización de la plataforma de hardware o de software, políticas de empresa, necesidad de cambio en las interfaces de usuario, etc.

Para llevar a cabo la traducción no es necesario comprender la operación de software en detalle, basta con conocer las equivalencias de las estructuras de control del programa. Será económicamente rentable, si se dispone de alguna herramienta que realice el grueso de la traducción, puede ser un programa específico escrito para un caso, una herramienta comercial o un sistema de comparación de patrones. En muchos casos, el código que se obtiene tiene que ser modificado de forma manual, bien porque no existe equivalencia en todas las estructuras o bien porque existan instrucciones de compilación que no son soportadas por el lenguaje destino.

2.4 Ingeniería inversa

Es el proceso de analizar el software con el objetivo de recuperar su diseño y especificación. Lo normal es que la entrada a este proceso sea el código fuente si se dispone de él. No hay que confundirla con la reingeniería. El objetivo de la ingeniería inversa es obtener el diseño o la especificación de un sistema a partir del código fuente, mientras que el objetivo de la reingeniería es obtener un nuevo sistema más mantenible. A menudo la ingeniería inversa es una parte de la reingeniería. Esto permite que el diseño recuperado sirva para comprender un programa antes de reorganizar su estructura.

Para llevar a cabo la ingeniería inversa se sigue el proceso que aparece en la figura 3.

Se alterna el análisis utilizando herramientas automatizadas con el trabajo manual en el código fuente para obtener el diseño del sistema. La información obtenida suele almacenarse como grafo dirigido, que se va modificando y completando. A partir del grafo se generarán otros documentos como diagramas de estructura de programas, diagramas de estructura de datos y matrices de rastreabilidad. Las herramientas que se utilizan para comprender el programa suelen ser de tipo navegadores, que permite moverse por el código, definir unos datos y rastrearlos por el programa. Siempre son necesarias anotaciones manuales, no se puede deducir automáticamente la especificación a partir del modelo del sistema.

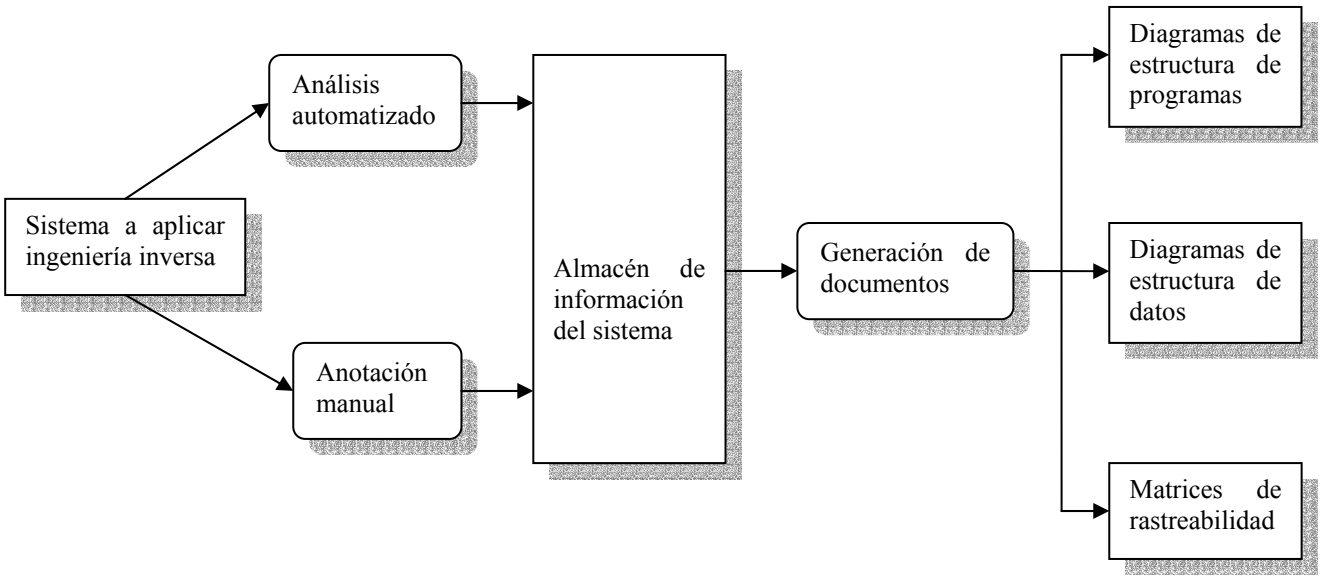

Figura 3. Proceso de ingeniería inversa.

2.5 Mejora de la estructura del programa

Es habitual, por distintos motivos, que la lógica de los programas heredados no sea la más adecuada, pueden presentar lógica de control no intuitiva lo que puede hacer que los programas no se entiendan fácilmente. Un ejemplo habitual es el caso de los programas Fortran. El principal factor a tener en cuenta es que el control sea estructurado, desde 1966 se conoce que cualquier programa puede reestructurarse con sentencias condicionales sencillas eliminando la instrucción incondicional goto, pero además se pueden simplificar las condiciones complejas.

Se puede llevar a cabo reestructuración automática de los programas, convirtiéndolos a un grafo dirigido, después se genera el programa estructurado equivalente sin instrucciones goto. Esta reestructuración automática presenta los siguientes problemas: pérdida de comentarios, pérdida de documentación y necesidad de un hardware sofisticado. En algunos casos el costo, no aconseja reestructurar algún programa, por lo que se utilizan métricas para identificar los candidatos a ser reestructurados.

2.6 Modularización del programa

Es el proceso de reorganizar un programa de forma que partes relacionadas se integren de forma conjunta. Esto facilita eliminar componentes y mejorar la comprensión. Se pueden considerar diferentes tipos de módulos: abstracciones de datos, módulos de hardware, módulos funcionales, módulos de apoyo al proceso, etc. La modularización es una tarea que se suele realizar de forma manual, examinando el código.

2.7 Reingeniería de datos

Se trata de analizar y reorganizar las estructuras, e incluso a veces los valores, de los datos de un sistema para hacerlo más comprensible. Si la funcionalidad del sistema no cambia, la reingeniería de datos no es necesaria. Sin embargo hay razones que la justifican, como son: la degradación de los datos, los límites propios del programa (suelen ser limitaciones que los

programadores hicieron en su día y que con el transcurso del tiempo se han ido alcanzando esos límites), o la evolución arquitectónica.

Antes de llevar a cabo la reingeniería de datos es necesario analizar los programas que utilizan los datos. Con este análisis se pretende descubrir la función de los identificadores en el programa, encontrar los valores de los literales a reemplazar por constantes con nombre, descubrir reglas internas de validación de datos y conversiones de representaciones de datos.

2.8 Reingeniería en el mantenimiento

La reingeniería y la fase de mantenimiento del ciclo de vida de un sistema están muy relacionados. El mantenimiento se llevará a cabo una vez que la implementación del sistema software ha concluido y se ha entregado al cliente, y puede ser provocado por nuevas necesidades del usuario o bien por la detección de errores.

Según ANSI-IEEE, el mantenimiento software es la modificación de un producto software, después de su entrega al cliente, para corregir defectos, para mejorar el rendimiento u otras propiedades deseables, o para adaptarlo a un cambio de entorno. Por lo tanto, el mantenimiento software se puede definir como el conjunto de medidas que hay que tomar para que el sistema siga trabajando correctamente. Existen 4 tipos de mantenimiento [13]:

- **Correctivo**, el cual tiene como objetivo localizar y eliminar los posibles defectos de los programas.
- **Adaptativo,** cuyo objetivo es modificar un programa para adaptarlo a los cambios hardware o software en el entorno en el que se ejecuta. Puede ser, desde un pequeño cambio, hasta una reescritura de todo el código, y es cada vez más habitual debido a la actualización frecuente los sistemas operativos.
- **Perfectivo**, consistente en el conjunto de actividades para mejorar o añadir nueva funcionalidades requeridas por el cliente.
- **Preventivo o reingeniería**, que consiste en la modificación del software para mejorarlo en cuanto a la calidad y mantenibilidad, sin alterar sus especificaciones funcionales. Algunas tareas de este mantenimiento serían: incluir sentencias que comprueben la validez de los datos de entrada, reestructuración de los programas para aumentar su legibilidad o incluir nuevos comentarios, etc. Utilizará técnicas de reingeniería e ingeniería inversa. También puede ser el mantenimiento para la reutilización especializado en mejorar la reusabilidad.

Los recursos necesarios para el mantenimiento se incrementan a medida que se genera más software, debido a la gran cantidad de software antiguo cuya creación se produjo con restricciones de tamaño y espacio de almacenamiento y con herramientas desfasadas, migraciones continuas de plataformas o SSOO, o las modificaciones, correcciones, mejoras y adaptaciones que el software experimenta con el tiempo, debido a las nuevas necesidades de los usuarios. Además, estos cambios, generalmente se han realizado sin técnicas de reingeniería o ingeniería inversa, dando como resultado sistemas con las estructuras de datos mal diseñadas, mala codificación, lógica defectuosa y escasa documentación.

Existen diferentes dificultades a la hora de realizar el mantenimiento. La primera, y más importante, es que la mayor parte del software está formado por código antiguo heredado, desarrollado hace tiempo con técnicas y herramientas en desuso, y ha sufrido varias actividades de mantenimiento pero por personas no especializadas en mantenimiento utilizando un estilo libre del programador debido principalmente a que el programador no conoce, o incluso no existen métodos, técnicas y herramientas, que proporcionan soluciones globales al problema del mantenimiento. Después de estos cambios, los programas son menos estructurados, lo que

provoca un incremento en el tiempo de compresión de los programas y una documentación desfasada. Los sistemas mantenidos son cada vez más difíciles de cambiar. Debido a estos problemas, se producen una serie de efectos secundarios sobre el código, los datos y la documentación, como por ejemplo: modificación o eliminación de subprogramas, etiquetas o identificadores, modificación de los formatos de registros o archivos, modificación de definición de variable globales, nuevos mensajes de error no documentados o modificación de las interfaces de usuario.

Las soluciones al problema del mantenimiento se pueden dividir en dos grupos: soluciones de gestión y soluciones técnicas (éstas últimas son de dos tipos: herramientas y métodos). Las herramientas sirven para soportar de forma efectiva los métodos, algunas de ellas son: formateador, analizador sintáctico, estructurador, documentador, depurador interactivo, generador de datos de prueba y comparador.

Los principales métodos serían:

- **Reingeniería**, examen y modificación del sistema para reconstruirlo en una nueva forma.
- **Ingeniería inversa**, análisis de un sistema para identificar sus componentes y las relaciones entre ellos, así como para crear representaciones del sistema en otra forma o en un nivel de abstracción más elevado.
- **Reestructuración de software**, consiste en la modificación del software para hacerlo más fácil de entender y cambiar o menos susceptible de incluir errores en cambios posteriores.
- **Transformación de programas**, técnica formal de transformación de programas.

3 Metodología para la reingeniería de sistemas heredados

Se ha desarrollado un proceso de reingeniería del software que persigue como objetivo prioritario la reconversión de aplicaciones antiguas que siguen siendo útiles pero que se desea sean más fáciles de mantener y se adapten a soportes software más evolucionados y modernos. Se trata pues, de una metodología que permita modificar sistemas legados, en los que es necesario reescribir o reestructurar parte, o todo el sistema, sin cambiar su funcionalidad.

El abordar el desarrollo de un nuevo proceso de reingeniería viene dado por la falta de buenos resultados obtenidos con otras metodologías en proyectos reales. Se probó la metodología sugerida en [20], realizando una conversión de código automática, de Fortran a C, pero los resultados no fueron los esperados. El código se multiplicó, por lo que las siguientes fases se hacían mucho más complicadas. Además se constató que la generación de código que se obtenía era literal, es decir, manteniendo las mismas estructuras del fichero de código fuente original, por tanto con instrucciones de salto incondicional. Además, existían peculiaridades del caso de estudio que hacían el trabajo diferente dado que se necesitaba mejorar la interfaz de usuario, había que descartar la introducción de datos secuenciales en una ventana DOS, y había que hacerlo más intuitivo y agradable. Hubo que hacer un replanteamiento del proceso, pero ya se había conseguido una aproximación del tipo de resultados que se podían conseguir, con lo que se pudo conocer la viabilidad del proyecto.

En el desarrollo de la metodología se apreciaron fases que se podían acometer en paralelo, partiendo del código fuente. Por una parte estaría la ingeniería inversa y el diseño de la nueva interfaz de usuario y por otra parte estarían todos los procesos relacionados con la mejora del antiguo código. Esto se puede apreciar claramente en la figura 4 que representa las fases de la metodología desarrollada.

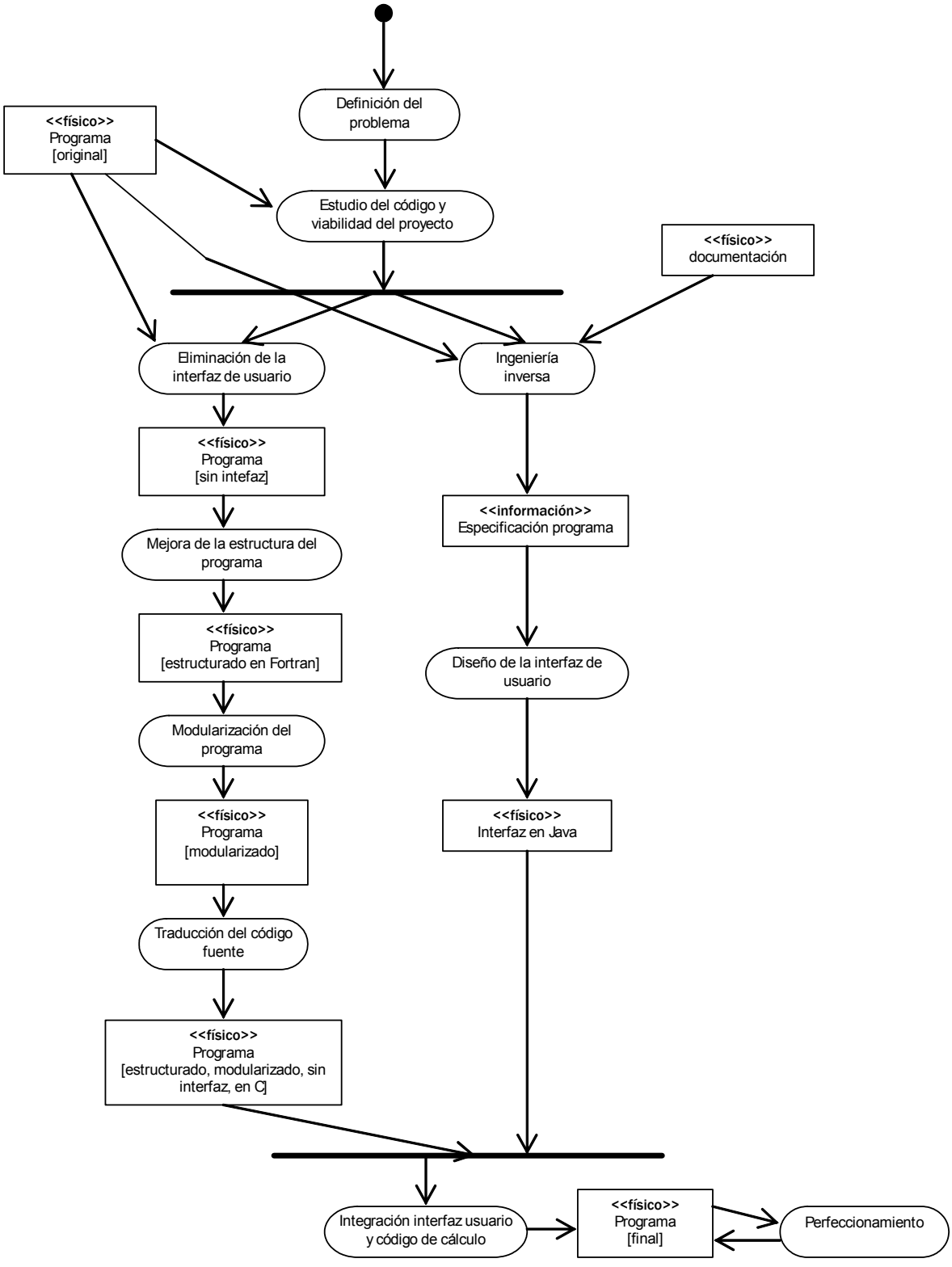

Figura 4. Proceso de reingeniería del software.

El siguiente paso sería la integración de la interfaz de usuario en Java con las funciones que soportan el cálculo matemático en C. Aunque otra posibilidad aceptable habría sido realizar

todo el desarrollo en C o C++. Por último, vendría una fase de perfeccionamiento del programa para depurarlo y comprobar con resultados reales su validez.

4 Caso de estudio

El proceso de reingeniería se ha llevado a cabo sobre una aplicación científica que realiza la computación de datos cinéticos de absorbancia total de una mezcla monitorizada a multilongitud de onda. Permite la determinación de forma individual o conjunta de la concentración de una mezcla multicomponente y de las constantes de velocidad. Además, puede discriminar entre diferentes mecanismos de reacción, realizando la elección del que mejor represente los datos experimentales. Realiza de forma automática el tratamiento de una serie de modelos de reacción, los más frecuentes, si bien el usuario puede proporcionar nuevos modelos, facilitando las características de los nuevos modelos. La aplicación estaba desarrollada para MS-DOS, escrita en lenguaje de programación Fortran 77, compilada en su día con SVS C^3 (Silicon Valley Software) y consistía en un programa principal, junto con una serie de subrutinas encargadas de llevar a cabo los diferentes tratamientos y cálculos necesarios para realizar el proceso de optimización mediante el algoritmo AGDC [11]. Contaba, en total, con unas 10100 líneas de código.

La aplicación final obtenida mediante el proceso de reingeniería, mantiene su funcionalidad original. Cuenta con una interfaz de usuario escrita en Java, y una serie de subprogramas escritos en lenguaje C, a los que se accede mediante la Interfaz de programación JNI (*Java Native Interface*).

4.1 Definición del problema

Al tratarse de una aplicación para DOS, se trabaja en modo texto directamente. Se desea una nueva aplicación en modo gráfico, que permita la introducción de los datos de forma más intuitiva, de manera que a un usuario no experto le sea más fácil la utilización de la aplicación. Se deben mantener por otra parte los ficheros de datos originales porque algunos de ellos son necesarios para que funcionen las bibliotecas matemáticas y otros porque permiten facilitar parte de los datos de entrada a la aplicación.

La reingeniería de datos ha sido muy importante porque fue necesario analizar las estructuras, ya que al mantenerse condicionaron en gran medida el proceso de reingeniería. Estas estructuras definían una serie de modelos, cuyas características podrían cambiar, sin tener por ello que verse afectada la aplicación. También fue necesario mantener los valores de los datos, ya que, al disponer de una gran cantidad de ficheros de entrada obtenidos experimentalmente que tienen que seguir utilizándose, la aplicación tiene que ser compatible con todos ellos. En cuanto a los datos de salida, se debían obtener resultados en el mismo formato que se venía haciendo, para poder comparar con muchos resultados experimentales válidos disponibles.

4.2 Estudio del código y viabilidad del producto

El código de la aplicación estaba escrito en Fortran 77, una versión, aunque extendida, bastante limitada en cuanto a sentencias estructuradas. Dispone de sentencias GO TO calculadas e incondicionales. Tres tipos de sentencias IF: IF aritmético, IF lógico y bloques IF. En cuanto a bucles son del tipo DO utilizando etiquetas numéricas para indicar el final del bucle. El programa utilizaba sentencias GO TO incondicionales, pero no tenía sentencias GO TO calculadas. De los tres IF, sólo utilizaba IF lógico; al no utilizar bloques IF, no contenía partes ELSE, ni sentencias IF encadenadas, lo que multiplicaba el número de líneas de código. Los

bucles eran del único tipo posible DO. Al intentar traducir de forma automática una sentencia GO TO incondicional, se obtenía una sentencia del mismo tipo pero en C, lo que no se podía permitir, véanse las figuras 5 y 6.

```
      IF(IEL.EQ.2) THEN
      WRITE(*,502)
      WRITE(*,511)
511   FORMAT(/10X" ** OPTIMIZACION DE CONCENTRACIONES INICIALES A0 **")
      WRITE(*,502)
      GO TO 2600
      END IF
```

Figura 5. Ejemplo de sentencia GO TO incondicional.

El caso de los bucles DO que utilizan una etiqueta final para marcar la salida, eran transformados correctamente en un bucle FOR en C, eliminando la etiqueta, véase figura 7.

Con la primera traducción realizada se pudo verificar que el código generado no era válido, su tamaño incluso se disparaba, pero los resultados de los cálculos que se obtenían se aproximaban bastante a los de la versión Fortran. Esto nos hizo tener que replantear las fases del desarrollo, debíamos primero simplificar y estructurar la aplicación, incluso separando la parte correspondiente a la interfaz de usuario, para más tarde obtener el código C.

```
if (iel == 2) {
  s_wsfe(&io___268);
  e_wsfe();
  s_wsfe(&io___269);
  e_wsfe();
  s_wsfe(&io___270);
  e_wsfe();
  goto L2600;
 }
```

Figura 6. Ejemplo de código C obtenido.

```
DO 2268 I=NC+1,NCOMP
        READ(*,268) AI(I)
2268  CONTINUE
```

```
i__1 = ncomp;
for (i__ = nc + 1; i__ <= i__1; ++i__) {
      s_rsfe(&io___317);
      do_fio(&c__1,(char *)&ai[i__ - 1], ftnlen)sizeof(doublereal));
      e_rsfe();
      }
```

Figura 7. Ejemplo de transformación de bucle DO.

Otra particularidad que tenía el código era la de la utilización de gran cantidad de códigos de formato de entrada/salida específicos de Fortran, del tipo:

250 FORMAT(G20.8)

280 FORMAT(10G10.8)

Que indicaban el tipo de los datos a leer o escribir; así el primero de ellos significaría que los valores iban a ser real General, 20 sería la anchura total del campo, que correspondería al signo

de la mantisa, el dígito que precede al punto decimal, el punto decimal, la mantisa y el exponente; 8 correspondería al número de dígitos que tiene la mantisa. En el segundo caso el 10 indicaría que se va a repetir 10 veces el código G.

Al chequear el programa con otros compiladores de Fortran también se detectó que había una serie de variables declaradas que no se utilizaban, posiblemente utilizadas como pruebas antes de la versión definitiva.

Con la primera traducción realizada se pudo verificar que el código generado no era válido, su tamaño incluso se disparaba, pero los resultados de los cálculos que se obtenían se aproximaban bastante a los de la versión Fortran. Esto hizo tener que replantear las fases del desarrollo, debiendo primero simplificar y estructurar la aplicación, incluso separando la parte correspondiente a la interfaz de usuario, para más tarde obtener el código C.

Lo importante fue constatar que el proyecto era viable.

4.3 Eliminación de la interfaz de usuario

El siguiente paso fue eliminar la entrada de datos por pantalla, ya que se pretendía que la nueva aplicación se ejecutara en un entorno gráfico con ventanas. Se decidió, por su portabilidad, elegir Java, como lenguaje para implementar la interfaz de usuario, además permite la utilización de código nativo en otros lenguajes. De igual forma se podría haber elegido cualquier otro lenguaje que facilitara el desarrollo de la interfaz siempre que también permitiera el uso de código nativo. La parte correspondiente a la interfaz de usuario debía ser reescrita completamente. Ese código también contenía sentencias incondicionales de salto que había que modificar, esto hizo que esta fase precediera la siguiente, porque de esta forma se reducía el código que tenía que ser tratado en la siguiente fase.

4.4 Mejora de la estructura del programa

El principal objetivo de esta fase era mejorar la estructuración del código y, por lo tanto, la principal medida era eliminar las sentencias GO TO incondicionales, para ello se rescribió el programa en Fortran 77 pero eliminando todas las sentencias GO TO incondicionales. En algunos casos se trataba de sentencias innecesarias, ya que estaban en condiciones de tipo IF excluyentes, es decir, que si entraba en un IF no podría entrar en el siguiente, porque el valor de la variable de la condición no se modifica dentro del IF. En otras ocasiones lo que tienen de fondo los saltos es un bucle. El programa principal constaba de 2146 líneas, al hacer el programa estructurado, y al transformar el programa principal sin la interfaz de usuario en una función para ser convertida a lenguaje C, constaba de 587 líneas.

4.5 Modularización del programa y eliminación de redundancias (Refactorización)

Se realizaron varias acciones para reducir partes de código al hacerlo más estructurado, modificando condiciones y reestructurando sentencias. En otros casos se hicieron otras modificaciones como utilizar IF encadenados, o estructurar bucles encadenados.

Otra de las acciones llevadas a cabo fue eliminar variables redundantes o no utilizadas. Además del programa principal, también se eliminaron redundancias de datos en los otros ficheros Fortran. Aparecían variables no utilizadas, y formatos de entrada/salida declarados varias veces con distintos nombres. Se eliminó todo el código que mostraba datos por pantalla, ya que en la nueva aplicación se siguen generando los ficheros de resultados y se incluyen opciones para su visualización e impresión desde la propia aplicación.

4.6 Traducción del código mejorado a C

Para la generación del código C a partir de los ficheros en Fortran se estudiaron las principales posibilidades existentes. Era muy importante la exactitud de la transformación, ya que cualquier modificación en el código llevaría a obtener resultados inadecuados, por ello se analizaron las herramientas de traducción de código de las que se podía disponer. Para muchos autores, entre ellos Sommerville [20], una conversión de código de una aplicación sólo puede realizarse si hay un traductor automático disponible, éste es el caso. Los traductores más importantes y utilizados, disponibles a través de la web, son:

- F2C. Se trata de una herramienta de ATT & Bell Laboratories. Desarrollada en un principio para sistemas Unix e incluida en la mayoría de sistemas operativos Unix y Linux. Convierte código fuente de Fortran 77 en ficheros de código fuente de C o C++. El inconveniente de este tipo de programas es que realizan una traducción literal, por lo que si en el código fuente inicial aparecen saltos incondicionales, también aparecerán en el código traducido, por lo que es más apropiado depurar el código fuente antes de ser sometido al proceso de traducción.
- LCC-WIN32. Se trata de un entorno de desarrollo para Windows. Incluye generador de código (compilador, ensamblador, enlazador, compilador de recursos, y bibliotecas), entorno integrado de desarrollo (con editor, depurador, generación de ficheros make, editor de recursos, etc.), manual de usuario y documentación técnica.

4.7 Ingeniería inversa

No se disponía de documentos de especificación de la versión anterior, por lo que a partir de la documentación facilitada y del código fuente del que disponíamos se llevó a cabo un análisis de parte del sistema implementado. Esta fase es necesaria debido a que no es posible una traducción directa de código desde un lenguaje orientado a procedimientos a un lenguaje orientado a objetos [6]. Concretamente se analizó el código correspondiente a la entrada de datos al sistema por parte del usuario, ya que era la parte que se iba a volver a programar, por lo que era necesario crear una representación [4]. Para llevar a cabo esta tarea no sólo existen herramientas que analizando el código fuente generan los grafos ASTS (*Abstract Syntax Trees*), sino que existen métodos como *Program Slicing* [22], y su versión mejorada [2], para descomponer automáticamente programas analizando el flujo de datos y el flujo de control. En este caso todo el análisis se ha llevado a cabo de forma manual, aunque utilizando una herramienta automatizada para realizar los diagramas necesarios. El tipo de diagramas seleccionado para representar el sistema antiguo fueron diagramas de flujo u organigramas, aunque podrían haberse utilizado otras alternativas, como por ejemplo [9, 16]. Esta decisión fue tomada debido a la gran cantidad de estructuras de decisión (expresadas mediante cláusulas "goto") y repetitivas que tenía.

Una vez generados los diagramas y después de analizar la descripción de los procesos se observó que hay sentencias y código redundante. Este código no fue mejorado, sino que solamente se obtuvo la especificación para después volver a desarrollar la interfaz de usuario en un nuevo lenguaje de programación orientado a objetos y visual.

4.8 Diseño de la nueva interfaz de usuario

Se optó por utilizar un entorno de desarrollo visual (concretamente Oracle Jdeveloper), que permite desarrollar aplicaciones de forma rápida, ya que disponen de numerosas clases Java prediseñadas. Una vez que disponíamos de las especificaciones de la interfaz de usuario, se pasó a realizar el diseño de la nueva interfaz. Debido a las variaciones existentes dependiendo del

modelo y tipo de optimización seleccionado, el diseño de la misma en un lenguaje de programación orientado a objetos no fue sencillo [5]; Al implementar una nueva interfaz de usuario se solucionaron problemas que tenía la antigua interfaz como dejar introducir valores no válidos, u opciones no factibles en momentos determinados.

4.9 Perfeccionamiento de la aplicación resultante

Una vez completada la aplicación se somete a un conjunto de pruebas con casos de prueba consistentes en casos reales que se habían ejecutado en la antigua aplicación y de los que se disponía de los resultados obtenidos; de esta forma se podían contrastar con los nuevos resultados. Se añaden algunas mejoras principalmente en la interfaz de usuario (activación de botones al completarse opciones, comentarios subyacentes en los botones, o la ayuda de la aplicación).

4.10 Integración de la interfaz de usuario con el código de cálculo

La interfaz de usuario desarrollada muestra por pantalla diferentes cuadros de diálogo y ventanas, que se van sucediendo en tiempo de ejecución, sobre los que se van recogiendo los datos de entrada. Estos datos obtenidos, es necesario pasarlos a las funciones escritas en C encargadas de los cálculos. Para poder utilizar código nativo de un lenguaje como C desde una aplicación Java, es necesario utilizar JNI (*Java Native Interface*). JNI fue diseñada como una interfaz entre métodos nativos escritos sólo en C o en C++. Era necesario utilizarlo ya que el código C utiliza bibliotecas matemáticas y era ineficaz rescribir todo ese código en Java.

5 Conclusiones

La reingeniería del software ha sido y es un tema muy importante para muchas empresas y organismos que tienen que seguir manteniendo sus aplicaciones porque sus desarrollos han sido costosos y adaptados a sus necesidades, lo que en muchos casos hace que no existan aplicaciones comerciales similares. El inconveniente es que estos sistemas con el paso de los años presentan un aspecto obsoleto, mostrando pantallas y diseños ya descartados, en definitiva interfaces de usuario que no son operativas y que se han superado ampliamente. Por otra parte, en muchos casos no pueden adaptarse a los avances del hardware, porque estos nuevos equipos incorporan sistemas operativos para los que no fueron pensados los sistemas legados.

Existen diferentes metodologías de desarrollo que abordan esta problemática, algunas de ellas específicas para determinados aspectos, como recuperar el diseño, desarrollar la documentación perdida, o convertir un código a un lenguaje orientado a objetos. Nuestro proceso de desarrollo trata de mantener la funcionalidad del sistema, manteniendo los datos, pero utilizando un nuevo código en un lenguaje orientado a objetos, por tanto, ya estructurado, con una interfaz de usuario totalmente nueva, que dote a las aplicaciones de un aire de modernidad y que, por tanto, facilite su utilización por parte del usuario final. Además, añade nuevas especificaciones con su correspondiente documentación, lo que permitirá la ampliación del sistema.

Aunque la tendencia en la mayoría de las metodologías para la reingeniería del software es buscar una completa automatización de los procesos, es muy difícil que todas las fases puedan abordarse sin la intervención humana. El conocimiento de expertos, hoy día se hace aún imprescindible, aunque sólo sea en determinados momentos de las metodologías o para aportar sus ideas en el desarrollo de determinadas herramientas. En nuestro proceso de desarrollo hemos utilizado herramientas cuando ha sido posible, porque facilitan la labor, podría ser

interesante el desarrollo de nuevas herramientas que se adapten mejor a las necesidades y una formalización del proceso integrando las herramientas.

Una ventaja de esta metodología es que algunas de sus fases pueden llevarse a cabo en paralelo, como se aprecia fácilmente en la figura 1. Esto hace, por una parte, que puedan ser abordadas por diferentes equipos humanos, incluso pertenecientes a distintas áreas de especialización y, por otra parte, al solaparse las fases, que se reduzca el tiempo de desarrollo del proyecto. Otra ventaja es que se adapta a dominios científicos, en los que el tiempo de procesamiento es importante y está claramente diferenciada la interfaz de usuario del procesamiento y la obtención de resultados.

Referencias

1. Arnold, R.S.: Software Reengineering. IEEE Computer Society Press, 1993.
2. Beck, J. and Eichmann, D.: Program and interface slicing for reverse engineering. In R.C. Waters and E.J. Chikosfsky, editors, Working Conference on Reverse Engineering, IEEE Computer Society Press. 1993, 54-63.
3. Biggerstaff T. and Perlis A.: Software Reusability. Addison-Wesley. November, 1990.
4. Chikofsky, E.J. and Cross, J.H.: Reverse engineering and design recovery: A taxonomy. IEEE Software. 7(1), 1990, 13-17.
5. Gall, H. and Klösch. R.: Finding Objects in Procedural Programs: An Alternative Approach. In Proceeding of the Second Working Conference on Reverse Engineering, Toronto, Canada. IEEE Computer Society Press, July 1995. 208-216.
6. Gall, H., Klösch, R., Mittermeir, R.: Object-Oriented re-architecturing. 5th European Software Engineering Conference (ESEC'95), September 1995, Sitges, Spain,. Lecture Notes in Computer Science 989, 1995, 499-519.
7. Jacobson, I., Lindström, F.: Re-engineering of old system to an object-oriented architecture. ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, Phoenix, Arizona, October 1991, 340-350.
8. Jones, C.: The economics of Object-Oriented Software, American Programmer, vol. 7, nº 10, October 1994, 28–35.
9. Landis L.D.; Hyland, P.M.; Gilbert, A.L. and Fine, A.J.: Documentation in a software maintenance environment. In Proceeding of the IEEE Conference on Software Maintenance. IEEE Computer Society 1998, 66-73.
10. Linos, P.K., Molterer S., Paech B. and Salzmann C.: Re-engineering for Reuse: Integrating reuse techniques into the reengineering process. Technical report TUM-INFO-11-I9824-100 Intitut Für Informatik, Technische Universität München, 1998.
11. Moreno, M.N., González, J.L., Arco M.A. and Casado, J.: Determination of macroscopic thermodynamic ionization constants at variable ionic strenght by an optimization algorithm. Computers and Chemistry, 14, 1990, 165-168.
12. Murphy G. C., Notkin D. and Lan E.S.: An empirical study of static call graph extractors. Technical Report 95-08-01, Department of Computer Science and Engineering, University of Washington. 1995.
13. Piattini, M., Villalba, J., Ruiz, F., Bastanchury, T., Polo, M., Martínez, M.A. y Nistal, C.: Mantenimiento del software: Modelos, técnicas y métodos para la gestión del cambio. Rama, noviembre de 2000.
14. Postema M. and Schimidt, H.W.: Reverse Engineering and Abstraction of Legacy Systems. Informatica: An International Journal of Computing and Informatics. Vol 22 nº 3, 1998, 359-371.
15. Pressman, R.S.: Software Engineering: A Practitioner's Approach. Fifth Edition. McGraw-Hill. 2001.

16. Rugaber, S. and Clayton, R.: The representation problem in reverse engineering. In Proceedings of the working Conference in Reverse Engineering, Baltimore, Maryland. IEEE Computer Society, May 1993. 8-16.
17. Rugaber, S. and Will L.M.: Creating a research infrastructure for reengineering. In 3rd Working Conference on Reverse Engineering. IEEE Computer Society Press, september 1996, 120-130.
18. Sametinger, J.: Software engineering with reusable components, Springer-Verlag, 1997.
19. SICUMA, Grupo. Leiva, J.: Construcción de especificaciones de interfaces en un proceso de reingeniería, en: 2da. Conferencia Iberoamericana en Sistemas, Cibernética e Informática CISCI 2003, Orlando (Florida)-EEUU.
20. Sommerville, I.: Software Engineering. Sixth edition. Addison Wesley, 2001.
21. Ulrich, W. M.: The evolutionary growth of software reengineering and the decade ahead. American Programmer, 3(10), 1990,14-20.
22. Weiser M.: Program Slicing. In IEEE Transactions on Software Engineering, IEEE Computer Society, July 1984, 352-357.