Mecánica Computacional Vol. XXII
M. B. Rosales, V. H. Cortínez y D. V. Bambill (Editores)
Bahía Blanca, Argentina, Noviembre 2003.

UNA METODOLOGÍA DE DESARROLLO DE UN *FRAMEWORK* PARA LA SIMULACIÓN DE SISTEMAS MULTIFÍSICA

Claudio E. Jouglard[1], Adriana A. Echeverria[2] y Luis A. Herrera[3]

[1]Laboratorio de Mecánica Computacional. Departamento de Física.
Facultad de Ingeniería. Universidad de Buenos Aires.
Av. Paseo Colón 850, (1063) Buenos Aires, Argentina
e-mail: cjougla@fi.uba.ar

[2]Departamento de Computación.
Facultad de Ingeniería. Universidad de Buenos Aires.
Av. Paseo Colón 850, (1063) Buenos Aires, Argentina
e-mail: aechevi@fi.uba.ar

[3]Instituto de Investigaciones Fisicoquímicas Teóricas y Aplicadas. INIFIA
Departamento de Química. Facultad de Ciencias Exactas.
Universidad Nacional de La Plata
email: lherrera@fceia.unr.edu.ar

Palabras clave: Métodos Computacionales, Frameworks, Control de Calidad.

Resumen. La simulación numérica de sistemas multifísica permite describir la interacción entre una gran variedad de fenómenos físicos de diferente naturaleza. En los proyectos científicos ó de desarrollo de aplicaciones de simulación numérica suelen trabajar un gran número de especialistas, en general sin una formación ni entrenamiento suficiente en tareas de programación ó análisis de sistemas. Los sistemas resultantes son en general de difícil mantenimiento y adecuación a nuevas tecnologías de hardware y software. Una manera de superar estas limitaciones es mediante la programación orientada a objetos donde el modelo computacional se caracteriza mediante un conjunto de objetos con atributos y comportamientos específicos. La optimización de la reusabilidad se consigue mediante la utilización de frameworks que consisten en un conjunto de bloques de software prefabricado que los programadores pueden usar, extender ó adecuar para aplicaciones específicas. Se pone particular énfasis en asegurar la calidad del proceso de desarrollo considerando este aspecto en todas las etapas de la propuesta metodológica.

1 INTRODUCCIÓN

La evolución continua del hardware y software de simulación numérica han permitido el análisis de complejos problemas de ingeniería con un alto grado de confiabilidad, complementando e incluso substituyendo costosos modelos experimentales. Podemos citar como ejemplos: la simulación de impactos de vehículos, la simulación de problemas de fluidos en la industria aeronáutica y aeroespacial, simulación de fenómenos ambientales sobre construcciones, etc.

La característica básica de los grandes sistemas de simulación numérica es que poseen un conjunto elaborado de algoritmos y la mayor parte de los trabajos de programación se realizan durante su evolución, cuando el código es adaptado a la creciente complejidad de los métodos numéricos. La calidad de tales desarrollos está dada por su mantenimiento, esto es el código debe poseer una suma de atributos como legibilidad, comprensión, extensibilidad y facilidad de corrección.

Tradicionalmente los programas de modelado numérico han sido escritos en Fortran y como la mayoría de los códigos basados en procedimientos carecen de flexibilidad para satisfacer los requerimientos básicos de mantenimiento. Históricamente, los primeros componentes reusables de software han sido procedimientos y funciones y a finales de la década de los 60 se introdujeron, con el lenguaje Simula 67, los conceptos de objetos, clases y herencia que resultaron en las librerías de clases. Tanto las librerías de procedimientos como las de clases están focalizadas en la reutilización del código, pero no en la reutilización del diseño, por lo que es necesario un gran esfuerzo de codificación para obtener una nueva aplicación basada en estas librerías.

Con el objetivo de incrementar tanto la reusabilidad del código como del diseño se introdujeron a mediados de los 80 los marcos genéricos de programación orientados a objetos (*object oriented frameworks*). Un framework contiene las partes principales de las aplicaciones a ser construidas, junto con las relaciones e interacciones entre estas partes. Para llegar a una aplicación utilizable a partir del framework es necesario agregar código solo en algunas partes especificas del framework para personalizar la aplicación. Actualmente se reconoce que la programación orientada a objetos, en general, y los marcos genéricos de programación (*software frameworks)*, en particular, proveen el medio adecuado para la construcción y el mantenimiento eficiente de grandes sistemas de software.

La introducción de técnicas orientadas a objetos en códigos de análisis por elementos finitos[1,2] de problemas de ingeniería se remonta a inicio de los 90[3-11]. Las primeras experiencias con códigos orientados a objetos consistían de construcciones de objetos dentro de la estructura básica de las implementaciones basadas en procedimientos, en particular relacionadas con el método de elementos finitos. Aunque estos esfuerzos produjeron códigos más fáciles de mantener y extender, la transformación directa de los métodos estándar no proveía un marco genérico de programación fácilmente extensible a técnicas de análisis diferentes, además tampoco disponían de formas simples de asegurar la confiabilidad de la implementación.

Actualmente existen varios frameworks orientados a métodos numéricos. Entre los

primeros paquetes para ecuaciones a derivadas parciales (EDP) tenemos al PLTMG[12] (Piecewise Linear Triangle Multigrid Package) y al MGHAT[13] (Multigrid Galerkin Hierarchical Adaptive Triangles). Estos son programas poco modulares escritos en Fortran y solamente reusables como un todo. Entre los paquetes más modernos orientados a objetos tenemos al DIFFPACK[14], que es un framework para la solución de EDPs mediante diferencias finitas ó elementos finitos y KASKADE[15] que cubre las mismas aplicaciones. Otro framework interesante es TRELLIS[16] que soporta implementaciones de varios métodos numéricos y posee interfaces directas con sistemas de CAD para obtener información geométrica.

Mientras no siempre es difícil extender estos frameworks con componentes propios, la reusabilidad de los algoritmos que lo conforman no es sencilla y depende fuertemente del ambiente provisto por cada framework. Luego la combinación de lo mejor de dos ó más de estos frameworks normalmente no es posible. Además, en la mayoría de estos sistemas prácticamente no se le presta gran atención al desarrollo e implementación de metodologías de control de calidad. Estas metodologías son cada vez más imprescindibles en todo proceso de desarrollo de software, sirviendo de guía las recomendaciones presentes en la norma ISO 9000.3[17] y en el modelo de madurez de capacidad del software (CMM)[18].

En este trabajo se describe una metodología para desarrollar un framework que posibilite la elaboración de aplicaciones de simulación de sistemas multifísica considerando los aspectos de control de calidad en todas las etapas del desarrollo.

En la sección 2 describiremos las especificaciones del framework, en la sección 3 detallaremos el proceso de desarrollo, en la sección 4 presentamos la metodología para el control de calidad y finalmente en la sección 5 se presentan algunas conclusiones.

2 ESPECIFICACIONES DEL FRAMEWORK PARA SIMULACIÓN NUMÉRICA

2.1 Descripción de sistemas físicos

Un sistema físico es una parte del medio que aislamos para su análisis. Un sistema físico está compuesto de *entidades* que poseen propiedades de interés para su estudio. Cada entidad tiene asociada una región del espacio que llamaremos *dominio* de la entidad. Las entidades interactúan entre sí y con el medio mediante *interacciones* que provocan cambios en el sistema. Las interacciones pueden ser *internas* cuando se producen entre dos entidades del sistema, ó *externas* cuando se producen entre una entidad y el medio.

Llamaremos *atributos* a las propiedades del sistema que permiten especificar completamente su comportamiento frente a determinadas interacciones. La descripción del conjunto de entidades, atributos e interacciones en un instante determinado del tiempo se denomina *estado del sistema*. Si el estado del sistema varía con el tiempo tenemos un sistema *dinámico*, caso contrario el sistema es llamado *estático*. En la figura 1 vemos una representación esquemática de un sistema físico donde E_i son las entidades del sistema, I^i_{pq} es una interacción interna entre las entidades P y Q, e I^e_p es una interacción externa sobre la entidad P.

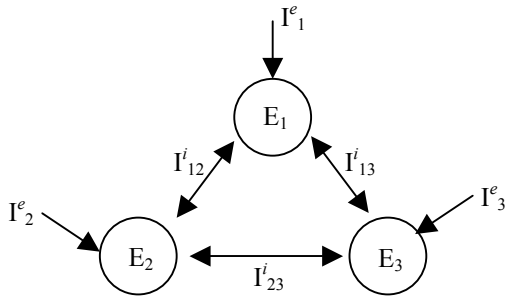

Figura 1. Interacciones internas y externas sobre un sistema físico

Las interacciones actúan sobre ciertas partes del dominio de cada entidad, que llamaremos *interfaces*. Las modificaciones que se producen en una entidad ante interacciones pueden ser descriptas completamente por *funciones de estado*, que son un conjunto de variables definidas sobre todo el dominio de cada entidad.

Las interacciones pueden ser de dos tipos: *de acción* ó *de vinculación.* Una interacción de acción es definida por *funciones de acción* que actúan sobre la interfaz de interacción. Las funciones de acción pueden depender de las funciones de estado y de los otros atributos de las entidades donde actúan. Si la interfaz de una interacción de acción ocupa total ó parcialmente el interior del dominio de la entidad diremos que se trata de una *acción de cuerpo*, sino diremos que se trata de una *acción de superficie* en cuyo caso la interfaz estará contenida en el contorno de la entidad.

Las interacciones de vinculación prescriben la variación de las funciones de estado ó imponen relaciones entre diferentes funciones de estado en su interfaz de interacción mediante *ecuaciones de vinculación*. Estas ecuaciones representan condiciones impuestas por las entidades entre sí ó por el medio sobre cada entidad.

Las variaciones de las funciones de estado pueden estar prescritas por *ecuaciones de restricción* en ciertas partes del dominio ó del contorno de cada entidad que llamaremos *regiones de restricción.* Las ecuaciones de restricción consisten, en general, en simplificaciones ó idealizaciones efectuadas al modelo físico para facilitar su análisis. Las ecuaciones de restricción son independientes de las interacciones.

El comportamiento de una entidad frente a las variaciones de las funciones de estado en su dominio es determinado por leyes físicas que dependen de las *propiedades constitutivas*. Estas propiedades constitutivas pueden variar por regiones del dominio que llamaremos *regiones constitutivas*. El análisis de un sistema físico consiste en estudiar sus cambios de estado ante variaciones de algún *parámetro de control*. Para sistemas dinámicos el parámetro de control más usual es el tiempo, y para sistemas estáticos es posible identificar el parámetro de control con la magnitud de alguna acción ó vinculación. En problemas de optimización se pueden tener, además, otros parámetros de control, en general asociados a las propiedades constitutivas, ó a la geometría del dominio.

En un sistema multifísica podemos tener acciones de naturaleza diferente actuando

simultáneamente sobre cada entidad, ó bien podemos tener diferentes tipos de acciones en cada entidad, como en los problemas de interacción fluido-estructura. Esta descripción de un sistema físico es bastante genérica permitiendo englobar una gran cantidad de fenómenos físicos de la mas variada naturaleza, y servirá de base para el modelo computacional.

2.2 Modelos matemáticos y discretos de sistemas físicos

Para determinar el comportamiento de un sistema físico es necesario aplicar las leyes de la física que se pueden expresar en forma matemática. Luego es necesario asociar al sistema físico un *modelo matemático*. La variación de las funciones de estado en un modelo matemático es generalmente gobernada por ecuaciones diferenciales definidas sobre el dominio de las entidades. Los coeficientes de estas ecuaciones diferenciales están asociados a las propiedades constitutivas del sistema y los lados derechos de estas ecuaciones están asociados a las interacciones de acción de cuerpo sobre el dominio de la entidad.

En un modelo matemático se deben definir matemáticamente la geometría de los dominios de cada entidad y de las partes de cada dominio asociadas a interfaces de interacción, regiones constitutivas y regiones de restricción. Luego es necesario definir un *modelo geométrico* que contenga toda la información necesaria para describir estas geometrías.

Si los dominios son de forma complicada es necesario utilizar métodos numéricos de discretización para hallar la solución aproximada al modelo matemático. Luego a partir del modelo matemático definimos un *modelo discreto* donde las variaciones de las funciones de estado quedan determinadas por un número finito de parámetros de estado que llamaremos *coordenadas generalizadas*. Todo modelo discreto tiene una *malla* asociada que se genera a partir de la geometría del dominio de cada entidad del sistema.

El tipo de malla depende del método numérico empleado. Por ejemplo, si usamos elementos finitos cada dominio es subdividido en elementos de formas geométricas simples, como hexaedros ó tetraedros en el espacio y cuadriláteros y triángulos en el plano. En general, las coordenadas generalizadas representan valores de las funciones de estado en puntos particulares de la malla. En general, todos los métodos conducen a un sistema de ecuaciones algebraico cuyas incógnitas son las coordenadas generalizadas, pudiendo haber incógnitas de otros tipos como, por ejemplo, multiplicadores de Lagrange, asociados a condiciones de restricción.

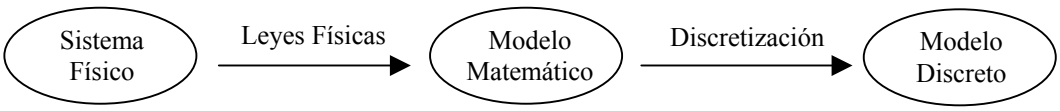

Figura 2. Aproximaciones al sistema físico.

Un simulador numérico de un sistema físico permite obtener la solución del problema matemático discreto asociado a dicho sistema mediante una implementación computacional.

2.3 Objetivos del framework

Muchos sistemas de software son construidos mediante componentes que son diseñados y desarrollados para una aplicación específica. En el desarrollo de software numérico la duplicación de código no es inusual, debido a las dificultades para reusar código existente. Estas dificultades son ocasionadas por la falta de un ambiente adecuado de desarrollo que permita la reusabilidad, la generación rápida de prototipos y la portabilidad del software. Las nuevas tecnologías de software basadas en el diseño y programación orientados a objetos tratan de remediar estas deficiencias.

El *diseño orientado a objetos* consiste en el modelado de un sistema mediante objetos que tienen una clara identificación con los componentes del sistema. Este modelo de objetos es independiente del lenguaje de programación y permite describir los componentes del sistema y sus atributos. Cada objeto tiene datos vinculados a los atributos del componente representado y procedimientos que modifican esos datos representando acciones sobre los atributos. La fase de implementación, que consiste en traducir dicho diseño en un código en un lenguaje de programación se denomina *programación orientada a objetos*.

Un marco genérico ó *framework* de programación consiste en un modelo orientado a objetos de un sistema particular que incluye la definición de los mecanismos y protocolos para la interacción entre los objetos. Este marco provee un plano de la arquitectura del programa de la aplicación dentro de la cual el programador inserta código específico de la aplicación.

El framework para simulación numérica tiene como objetivo principal desarrollar aplicaciones de simulación para sistemas multifísica, esto es, sistemas donde interactúan una gran variedad de fenómenos físicos de diversa naturaleza, sirviendo como herramienta para la implementación y verificación de nuevos algoritmos y metodologías de solución.

Usaremos como base la descripción previa de sistema físicos para identificar las diferentes clases de objetos que componen el framework, y nos concentraremos en una primera etapa en las aplicaciones de análisis y no tanto en la descripción de la geometría. Esto es, consideramos como entradas para el análisis a modelos discretos definidos por mallas estructuradas ó no estructuradas donde los elementos de la malla contienen la información suficiente para definir la geometría. Este último punto es importante para implementar técnicas adaptativas para *refinar* el modelo discreto incorporándole un mayor número de coordenadas generalizadas en las regiones de mayor error, pues es necesario tener un conocimiento de la geometría para generar la nueva malla.

Consideramos dos tipos de usuarios: 1) desarrolladores de aplicaciones y 2) usuario final de la aplicación. La documentación necesaria para cada uno de ellos es bastante diferente. El desarrollador de una aplicación debe contar con el detalle necesario para entender como interactúan las distintas partes del framework y debe poder concentrarse en el desarrollo de los nuevos componentes necesarios para su aplicación. En este punto es vital un buen sistema de documentación que permita también incorporar en forma sencilla la documentación de nuevos componentes. Por otro lado, el usuario final solo necesita la documentación suficiente para comprender el funcionamiento de una aplicación.

3 PROCESO DE DESARROLLO DEL FRAMEWORK

Un proceso de desarrollo de software es un método de organizar las actividades relacionadas con la creación, presentación y mantenimiento de los sistemas de software. Se adopta como metodología de desarrollo al *proceso unificado de desarrollo de software* elaborado por Booch et. al.[19], en el marco del paradigma de la orientación a objetos. Este proceso tiene tres características principales: está centrado en la arquitectura del futuro sistema, es iterativo e incremental y está dirigido por casos de uso, esto es, especificaciones funcionales.

Este proceso consiste en dividir en cuatro fases al desarrollo: 1) planificación, 2) elaboración, 3) construcción , 4) transición. La planificación consiste en definir la extensión del proyecto, requerimientos de los diferentes tipos de usuario, definir los casos de uso, modelo de arquitectura, etc. En la elaboración se refinan y se analizan los requerimientos, se implementan prototipos y se revisa la arquitectura. La construcción consiste en obtener un sistema de software que atienda debidamente los requerimientos, empleando sucesivos ciclos iterativos de análisis, diseño, codificación y pruebas. Finalmente, la fase de transición que consiste en la instalación e integración con el hardware, es decir, el uso fuera del ambiente de desarrollo.

Cada fase de desarrollo se compone de varios ciclos iterativos, compuestos por tareas de análisis, diseño, implementación y pruebas. En cada ciclo se aborda un conjunto relativamente pequeño de requerimientos y el sistema va creciendo con cada ciclo que concluye. A lo largo de cada fase se genera documentación necesaria para el seguimiento del desarrollo, para el control de la calidad del proceso y para los usuarios finales del sistema.

En consistencia con el proceso unificado de modelado se usa el *lenguaje unificado de modelado*, conocido por sus siglas en inglés como UML (*Unified Modeling Language*)[20]. El UML es un lenguaje visual que permite construir y documentar, entre otras cosas, los elementos que integran un sistema de software orientado a objetos. Es interesante destacar que el UML es adoptado como un estándar desde 1997 por la OMG (*Object Management Group*) un consorcio internacional formado por más de 800 empresas, universidades e instituciones gubernamentales.

Mediante el UML es posible visualizar al sistema de software desde diferentes perspectivas. Los aspectos estáticos se capturan en diagramas de casos de uso, diagramas de clases y objetos, diagramas de componentes. Los aspectos dinámicos se capturan en diagramas de interacción, diagramas de estados y diagramas de actividades. Cada uno de estos diagramas son representaciones visuales de los componentes del sistema y sus interacciones mediante una notación simbólica estandarizada.

Entre otros aspectos importantes para llevar adelante el proceso de desarrollo debemos especificar algunos documentos como las especificaciones de diseño, manuales de procedimientos para: manejo de versiones, control de errores, especificación de la documentación, tanto técnica como para el usuario, estandarización de la codificación, etc.

4 METODOLOGIA DE CONTROL DE CALIDAD

El desarrollo de software es un proceso que consiste en la aplicación de métodos, procedimientos, instrucciones y normas para alcanzar un resultado. Si se desea obtener un resultados de Calidad, esto es, aplicaciones que posean un alto grado de reutilización, confiabilidad, corrección, precisión y robustez, como cualidades mínimas exigibles, es necesario introducir Calidad en todas las fases del proceso de desarrollo.

La serie de estándares ISO 9000 es un conjunto de documentos referidos a sistemas de calidad que pueden utilizarse para lograr el aseguramiento de la calidad del software. Estas normas proveen lineamientos para la selección y utilización de estándares internacionales sobre sistemas de calidad que pueden ser utilizados para administración de la calidad como ISO 9004 y para propósitos de aseguramiento de la calidad tales como ISO 9001, 9002 y 9003.

El estándar específico de esta serie referida a organizaciones de desarrollo de software, es la ISO 9001. Dicha norma se refiere a un modelo de aseguramiento de calidad en diseño, implantación y operación de un sistema de información. La norma ISO 9002 habla de sistemas de calidad en lo que hace a un modelo para aseguramiento de la calidad en la producción e instalación, mientras que la ISO 9003 es un modelo para el aseguramiento de la calidad en pruebas e inspecciones. La ISO 9004 describe un conjunto básico de elementos mediante los cuales los sistemas de administración de la calidad pueden ser desarrollados e implementados.

En particular, la ISO 9001 requiere que se establezca un sistema de calidad documentado, que incluya procedimientos e instrucciones. También requiere establecer procedimientos de control y verificación del diseño. Esto abarca el planeamiento de las actividades de diseño, la identificación de los recursos y los resultados, la verificación del diseño y el control de los cambios del diseño. Otro punto que se establece es la definición del planeamiento del proceso de producción, lo cual significa llevar adelante la producción bajo condiciones controladas, debidamente documentadas.

Dado que la ISO 9001 es una norma de aplicación amplia, se ha desarrollado un conjunto específico de directrices, agrupadas en la norma ISO 9000.3, para adaptar específicamente al estándar ISO 9001 al proceso de producción de software.

Un documento esencial a ser elaborado como parte de las herramientas necesarias para el desarrollo del framework es el Manual de calidad, donde se describe el formato y la estructura del sistema de calidad. Existe una correspondencia estricta entre la norma ISO 9001 y el Manual de Calidad. Entre los requisitos incluidos en el estándar ISO 9001 podemos citar: definición de responsabilidades, control del diseño, control de datos y documentos, trazabilidad de los elementos constituyentes del sistema, control de registros de calidad, etc.

Se puede mejorar la calidad del proceso de desarrollo utilizando el Modelo de Madurez de Capacidad o CMM[18] ("Capability Maturity Model"). Este modelo ha sido originalmente desarrollado por el Software Engineering Institute de la Carnegie Mellon University para ser utilizado en la evaluación de calidad de los proveedores de software. Se utiliza un cuestionario de evaluación que permite calificar a una empresa en una escala de cinco niveles: 1) nivel inicial, 2) nivel repetible, 3) nivel definido, 4) nivel administrado y 5) nivel

optimizado. Cada uno de estos niveles se define por el cumplimiento de determinadas prácticas en *áreas clave del proceso*.

Luego se adoptarán como documentos rectores para el control de la Calidad a la familia ISO 9000, tratando de cumplir las prácticas clave requeridas por los niveles más altos del modelo CMM.

5 CONCLUSIONES

Para muchos programadores, la distancia entre pensar una implementación y transformarla en código es casi cero. Lo piensas, lo codificas. De hecho, algunas cosas se modelan mejor directamente mediante código, ya que este permite hacerlo de manera clara y concisa. Tal es el caso de las expresiones y algoritmos. Sin embargo, esto plantea enormes problemas a la hora de reutilizar el código, tanto por la misma persona como por otros. Por ello, se ha tratado de presentar los aspectos básicos de una metodología de desarrollo de aplicaciones de simulación numérica que enfatiza la reusabilidad de la mayoría del código implementado cumpliendo requerimientos mínimos de calidad.

6 REFERENCIAS

[i] O.C. Zienkiewicz y R.L. Taylor, *The finite element method*, McGraw Hill, Vol. I., 1989, Vol. II, (1991).

[2] R.D. Cook, D.S. Malkus y M.E. Plesha, *Concepts and Applications of Finite Element Analysis*, Third Edition, Wiley, (1989).

[3] B.W.R. Forde, R.O. Foschi, S.F. Stiemer, "Object Oriented Finite Element Analysis", *Computers & Structures*, **34**, 355-374, (1990).

[4] G.L. Fenves, "Object-Oriented Programming for Engineering Software Development", Engineering with Computers, **6**, 1-15, (1990).

[5] J.S. Rodriguez Alves Filho, P. Devloo, "Object Oriented Programming in Scientific Computations: the Beginning of a New Era", *Engineering Computations*, **8**, 81-87, (1991).

[6] G.R. Miller, "An Object-Oriented Finite Approach to Structural Analysis and Design", *Computers & Structures*, **40**, 75-82, (1991).

[7] T. Zimmermann, Y. Dubois-Pèlerin, P. Bomme, "Object-Oriented Finite Element Programming: I. Governing Principles", *Computer Methods in Applied Mechanics and Engineering*, **98**, 291-303, (1992).

[8] Y. Dubois-Pèlerin, T. Zimmermann, P. Bomme, "Object-Oriented Finite Element Programming: II. A Prototype Program in Smalltalk", *Computer Methods in Applied Mechanics and Engineering*, **98**, 361-397, (1992).

[9] Y. Dubois-Pèlerin, T. Zimmermann, "Object-Oriented Finite Element Programming: III. An efficient implementation in C++", *Computer Methods in Applied Mechanics and Engineering*, **108**, 165-183, (1993).

[10] R.I. Mackie, "Object Oriented Programming of the Finite Element Method", *International Journal for Numerical Methods in Engineering*, **35**, 425-436, (1992).

[11] A. Cardona, I. Klapka, M. Gerardin, "Design of a New Finite Element Programming Environment", *Engineering Computations*, **11**, 365-381, (1994).
[12] R.E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations – Users' Guide 8.0*, SIAM Publications, (1998).
[13] W. Mitchell, *MGHAT netlib directory*. http://elib.zib.de/netlib/pdes/mgghat/, (2002).
[14] A.R. Bruaset y H.P. Langtangen, "Object-oriented design of preconditioned iterative methods in Diffpack", *ACM Transactions on Mathematical Software*, **23**(1), 55-80, (1997).
[15] R. Roitzsch, B. Erdmann y J. Lang, "An object-oriented finite element code: Design issues and application in hyperthermia treatment planning", in E. Arge, A.M. Bruaset y H.P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, 105-124, Birkauser Press, (1997).
[16] M.W. Beall, "An Object-Oriented Framework for the Reliable Automated Solution of Problems in Mathematical Physics", SCOREC report 1999/6, Rensselaer Polytechnic Institute, (1999).
[17] M. Jenner, *Software Quality Management and ISO 9000*, Wiley, (1995).
[18] M.C. Paulk, B. Curtis, M.B. Chrissis y C.V. Weber, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-93-TR-25, (1993).
[19] I. Jacobson, G. Booch y J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley Pub. Co., (1999).
[20] G. Booch, J. Rumbaugh y I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley Pub. Co., (1999).