# ENSEÑANZA DE UNA METODOLOGÍA PARA LA PROGRAMACIÓN DE MICROCONTROLADORES EN EL MARCO DE LA TITULACIÓN DE ELECTRÓNICA INDUSTRIAL

*Antonio Bono, Bonifacio Martín*

***Universidad de Zaragoza***

antoniob@unizar.es , nenet@unizar.es

## RESUMEN

Los microcontroladores son uno de los componentes cruciales que más están experimentando un gran aumento de potencia debido al imparable desarrollo tecnológico. La consiguiente tendencia es a aumentar la complejidad del software que llevan incorporado. Desde la docencia se debe apostar por adaptar la enseñanza de microcontroladores a estos cambios, aumentando la importancia de la fase de construcción del software y dotándola de una metodología eficaz. Se propone un método concreto, con fases de análisis, diseño, desarrollo, pruebas y documentación, que facilitará la resolución de problemas complejos. Los resultados serán una programación de calidad y que tendrá en cuenta otros aspectos como correcciones o modificaciones posteriores, trabajo en equipo y generación progresiva de la documentación.

## 1. INTRODUCCIÓN

El microcontrolador es un dispositivo que en un único circuito integra la unidad de procesamiento central (CPU), memoria (RAM, EPROM, EEPROM, etc.) y periféricos (UART, USB, I2C, conversores A/D, etc.). La labor principal del microcontrolador es la de control sobre el sistema al que está adscrito. La utilización de este dispositivo frente a la lógica convencional tiene varias ventajas claras, entre las que podemos destacar la reducción de tamaño y costes del sistema al tener integrados varios periféricos y, por otro lado, la programabilidad, que permite utilizar un mismo componente, el microcontrolador, para multitud de aplicaciones sin más que cambiar el programa que lleva grabado [Martín del Brío, 1999]. Prácticamente todo diseño o aparato electrónico contiene en la actualidad uno o más microcontroladores, aunque no seamos conscientes de ello (computación oculta).

El enorme avance de las tecnologías microelectrónicas de integración proporcionan cada vez más potentes microcontroladores industriales, a un precio muy reducido, lo que conlleva la simplificación del diseño del hardware que tradicionalmente llevaba a cabo un Ingeniero Técnico en Electrónica Industrial, aumentando sin embargo la complejidad del software que éste deberá programar [Martín del Brío, 1999]. Un claro ejemplo es el gran desarrollo que han experimentado los teléfonos móviles. Cada vez son más pequeños en tamaño, pero tienen un sistema de menús cada vez más completo que incluye funcionalidades adicionales, como juegos o acceso a internet, que nos puede dar una idea de la complejidad del software incorporado. Hoy en día el desarrollo del software es una parte crucial en el

diseño de sistemas que puede llegar a ocupar un 70% del tiempo total dedicado al proyecto.

## 2. PROBLEMA Y SOLUCIÓN

Nuestro contacto con la empresa nos permite apreciar las carencias formativas de los Ingenieros Técnicos, y una de las más importantes que observamos es no emplear un método para la programación de microcontroladores. Sin duda hay que rechazar la programación "artesanal" y la figura del programador "gurú". Desde la docencia universitaria debemos rechazar esta figura como alguien que hace lo que nadie más puede, sin un método. De esta manera evitaríamos proyectos sin documentación o cuya única explicación son los comentarios insertados en el programa (que si el tiempo apremiaba, ni estaban) o confusos y extensos diagramas de flujo de muchas páginas y que ha medida que iban creciendo en tamaño tendían a desaparecer o a estar incompletos (fig. 1).

Desde nuestra labor docente en la ingeniería debería inculcarse que la programación es otra actividad "industrial", que debe incluir un método, diagramas (planos), documentación, trabajo en equipo,... Pensamos que es necesario incluir esta metodología en la carrera para luego poder aplicarla en la industria. Ha pasado el tiempo en el que los programas eran muy sencillos y se podían implementar directamente, ahora hay que hacer un esfuerzo y enseñar a nuestros futuros ingenieros como desarrollar un software de calidad y que se puedan integrar en el mercado laboral con más garantías.

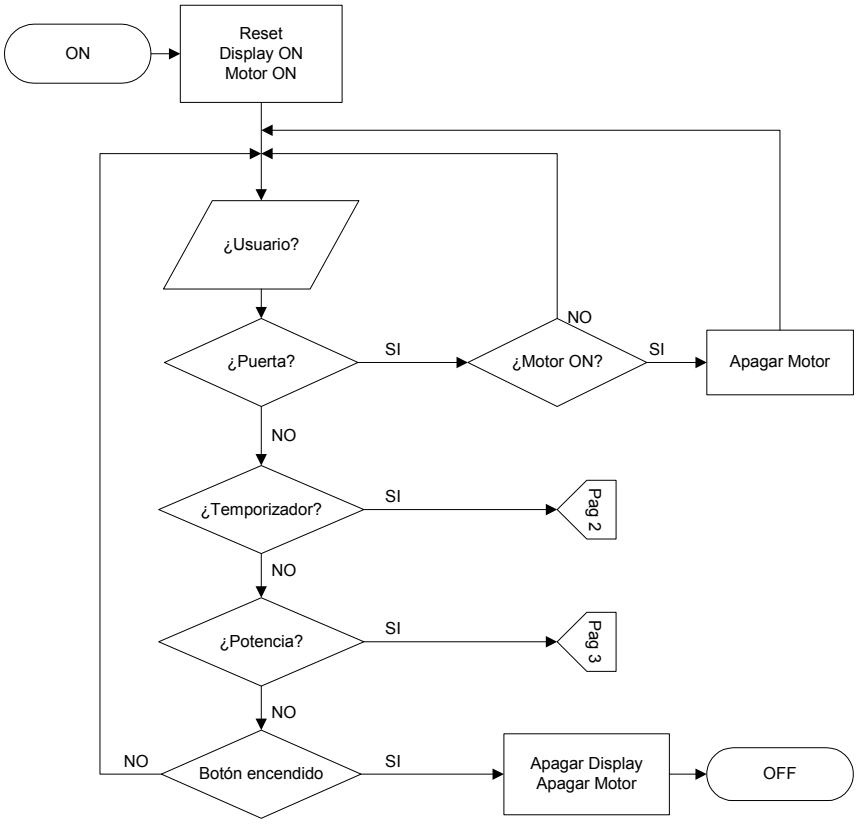


***Fig. 1.*** *Diagrama de flujo de un horno microondas*

Una vez vista la necesidad de enseñar un método la pregunta que surge es "¿qué método es el más idóneo?" Frente a un problema complejo la solución que se suele adoptar es dividirlo en problemas más pequeños y por ende, más sencillos de resolver. En el diseño hardware se divide el problema en grandes bloques y luego se trabaja por separado cada uno. Parece lógico hacer lo mismo con el diseño del software, pero, ¿cómo?

Antes de inventar nada podemos plantearnos el buscar algo similar que ya funcione, y una posibilidad es que la metodología a enseñar a nuestros futuros ingenieros técnicos esté basada en la ingeniería del software que emplean los profesionales de la informática. Sin embargo, la programación de computadores ha evolucionado y ahora se basa en el diseño orientado a objetos, lo cual hace que sus métodos no sean directamente aplicables al caso de los microcontroladores industriales.

Nosotros defendemos la idea de que hay que partir de las bases de las que nació dicha ingeniería del software, cuando lo que se utilizaba era programación estructurada y lenguajes como el ensamblador, C,... que es justo lo que se aplica ahora mayoritariamente con los microcontroladores. A lo largo del siguiente apartado estableceremos un método general, pero teniendo en cuenta que habrá que adaptarlo a cada proyecto concreto.

## 3. EL MÉTODO PROPUESTO

En primer lugar, es fundamental disponer de una serie de documentos de partida, cuya función será acotar la labor a realizar e instaurar el contexto donde va a incluirse el software. Dichos documentos serían los siguientes:

- Especificación de diseño funcional. Indica los objetivos a conseguir (a grandes rasgos). Por ejemplo*: Diseño de un horno microondas de aspecto y funcionalidad moderno, de especificaciones técnicas....*
- Especificación de diseño detallado. Indica cómo lograr los objetivos sin especificar construcción. Por ejemplo*: Diseño de un microondas con un LCD gráfico de puntos, teclado de membrana, sin mandos rotativos,....*
- Esquema básico de diseño del hardware y la mecánica del aparato o sistema. Nos proporciona el entorno de actuación del software.

El esquema propuesto tiene una serie de fases que, si bien al principio del proceso de aprendizaje sería lógico procesarlas secuencialmente para captar el sentido del método, en la práctica real habría que tratar de impulsarlas simultáneamente. Dichas fases serían las que a continuación se exponen.

### 3.1. Análisis

La fase de análisis consiste en realizar un grupo de diagramas y documentos que modelen o expliquen gráficamente las capacidades y funcionalidades del sistema [Yourdon 93]. Es importante que sean principalmente gráficos o diagramas porque ello facilita la abstracción del entorno y la separación en bloques funcionales.

Como veremos, un aspecto importante del modelado del sistema es subdividir el problema en partes más pequeñas, y cada una tendrá un diagrama explicativo. Es muy relevante, desde el punto de vista de su aplicación

práctica, el que los diagramas no sean muy extensos (la norma general es que no ocupen más de un A4). Si bien esta norma nos puede parecer muy arbitraria, la experiencia nos demuestra que es necesaria para no volver a caer en el problema de disponer de excesivas hojas con esquemas. Esto hace que a la hora de plantearse como implementar el diseño descendente haya que considerar las divisiones según funciones y según complejidad o extensión.

Esta es la fase en la que más habría que insistir desde la docencia, pues es la más ignorada y el esqueleto sobre el que crecerá el proyecto. Los distintos diagramas y documentación incluirían Diagramas de Flujo de Datos, Diagramas de Transición de Estados, Diccionarios de Datos y Miniespecificaciones.

### 3.1.1. Diagramas de Flujo de Datos (DFDs)

Se trata de diagramas que modelan funciones o procesos y sus relaciones mediante datos. Los procesos se dibujan con círculos y los datos con flechas que conectan y comunican los procesos. Aquí se hace especialmente patente la estructura de ir subdividiendo las funciones hasta llegar a procesos muy sencillos que se pueden implementar fácilmente.

Se parte desde un DFD de nivel 0, también llamado Diagrama de Contexto (fig. 2), que expresa la relación del sistema con el entorno. A continuación se toman los procesos de cierta complejidad y se van "explotando" en nuevos DFDs que expliquen su funcionalidad (fig. 3), y así sucesivamente hasta que los DFDs de más bajo nivel equivalen a funciones o módulos software sencillos. Para que el conjunto de diagramas sea consistente es importante que los diagramas de nivel inferior mantengan la misma conectividad (las flechas de entrada y salida al DFD) que el proceso del cual son "explosión".

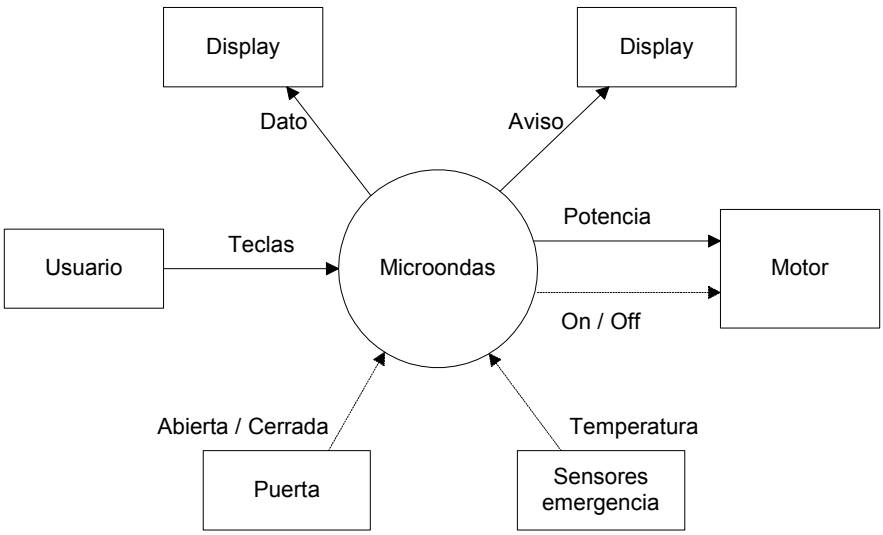


***Fig. 2.*** *Nivel 0: DFD Diagrama de contexto de un microondas*

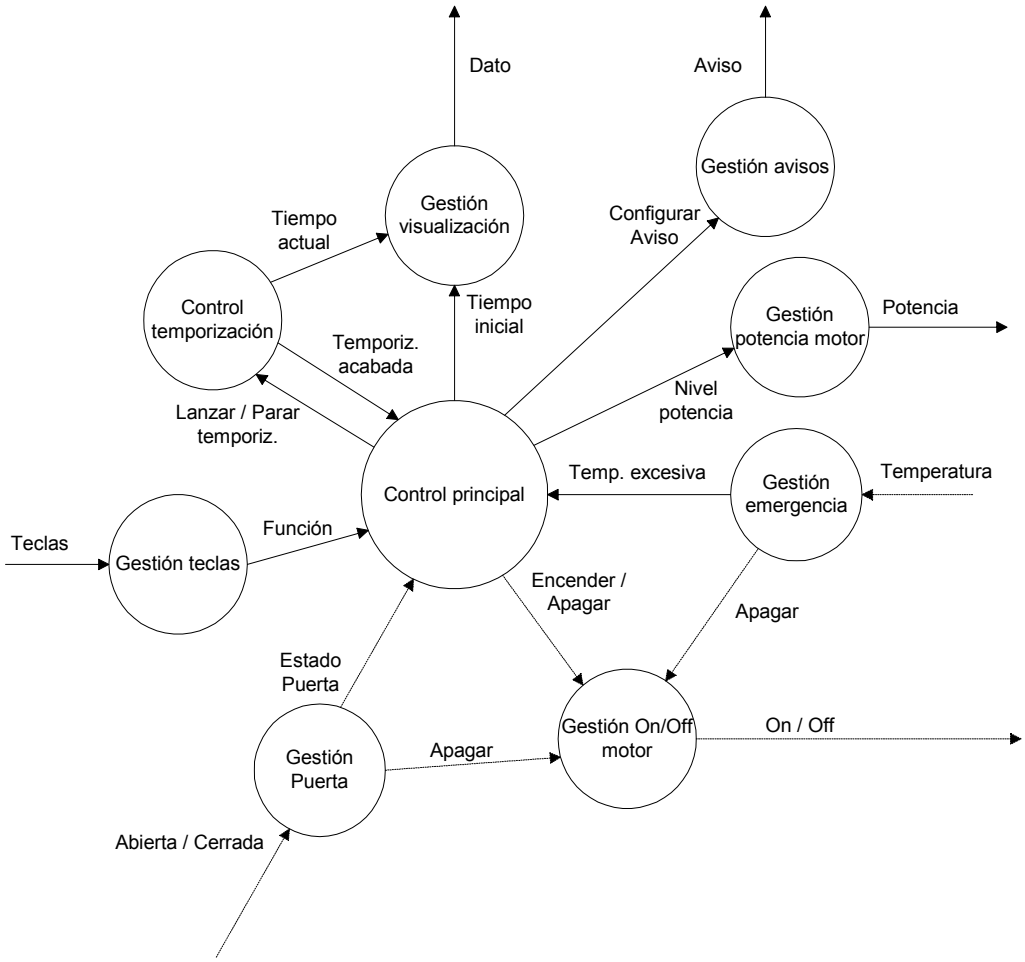


***Fig. 3.*** *Nivel 1: DFD del sistema Microondas*

### 3.1.2. Diagramas de Transición de Estados (STDs)

Modela el comportamiento dependiente del tiempo del sistema, así como eventos asíncronos (de los que desconocemos en qué momento van a aparecer y a los que hay que reaccionar inmediatamente).

El STD (fig. 4) como tal modela en base a estados temporales dibujados como rectángulos. De esta manera, en un momento dado, sólo se está en un estado y para pasar de uno a otro se tienen que cumplir ciertas condiciones y ejecutarse ciertas acciones. Estas transiciones se detallan en las flechas.

Estos diagramas son también susceptibles de subdivisión. Un estado puede desglosarse en otro STD de más bajo nivel, es decir, un estado puede contener internamente distintos subastados con sus condiciones y acciones.

### 3.1.3. Diccionarios de Datos (DDs)

Especifica en detalle los datos empleados en el sistema. Sólo en proyectos extensos y complejos consideramos que puede ser interesante desarrollar esta base de datos para completar todo el modelado. Desde un punto de vista práctico, antes de elaborar los DDs hay que considerar que van a suponer una gran cantidad de trabajo y la utilidad relativa que tienen.

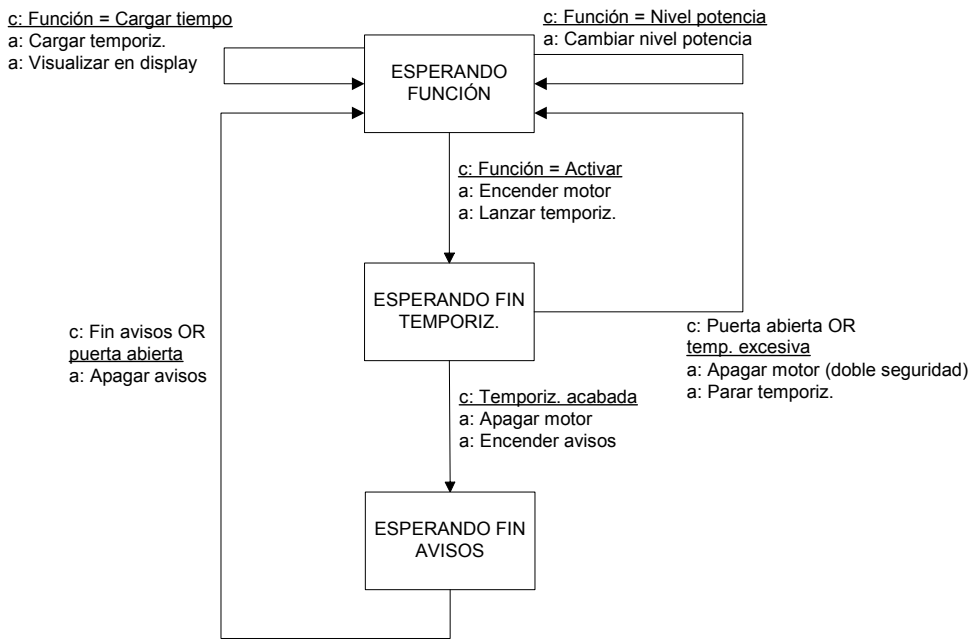


***Fig. 4.*** *Nivel 2: STD del proceso Control Principal*

### 3.1.4. Miniespecificaciones

Son las especificaciones de los procesos de más bajo nivel, de los que no se van a subdividir más. Define lo que hay que hacer exactamente para transformar las entradas en salidas mediante el uso de algún lenguaje genérico, como pseudocódigo o diagramas de bajo nivel (diagramas de flujo).

La utilidad de estas especificaciones es relativa, al ser muy cercanas a la codificación del software. En muchas ocasiones se podrán obviar y pasar directamente a la codificación, ya que hacerlas completas supone un gran trabajo. De todas maneras es muy interesante realizarlas cuando son relativas a partes del sistema complejas, cuando hay muchas condiciones distintas, cuando las alternativas son muy variadas, etc... En esos casos concretos puede ser conveniente realizar una miniespecificación que aclare el problema.

## 3.2. Diseño

En este punto se plantea la manera de implementar la información del análisis. Se concreta y añade información con el objetivo de implementar un software de calidad.

Esta fase la planteamos como una serie de “recetas” que hay que tener en cuenta a la hora de codificar el software. Hay que inculcar al alumno que tiene que pensar en no codificar de cualquier manera, sino con ciertos criterios. El primero de ellos es que debe plasmar el modelado gráfico desarrollado en la fase de análisis. Después debe hacerlo teniendo en cuenta los criterios clásicos de diseño del software [Monge 95 y Pressman 90], entre los que destacamos:

- Diseño descendente y refinamiento sucesivo: dividir un problema o proceso complejo en partes más pequeñas.

- Abstracción: consiste en realizar la codificación de un proceso buscando desligarse de la información de su entorno y de los niveles superiores e inferiores.
- Modularidad: es la división en bloques funcionales, en vez de realizar un software “monolítico”.
- Ocultación de información: se busca tratar los módulos como “cajas negras” de las que no necesitamos saber como están hechos, sólo como utilizarlos.
- Arquitectura del software: estructura general y relaciones de todos los módulos considerando el software como un todo.

### 3.3. Desarrollo

Consiste en la codificación del software propiamente dicha. Es de destacar que actualmente esta es la fase en la que más tiempo se consume. La experiencia en la empresa nos demuestra que el tiempo invertido en análisis y diseño es tiempo ganado aquí, ya que el software codificado sale con menos errores y es más “entendible”. El desarrollo incluye la codificación y las pruebas mínimas necesarias para tener la certeza de que el software responde a las especificaciones.

### 3.4. Pruebas

Consiste en la comprobación exhaustiva del software. Es una fase que se suele evitar y está muy desprestigiada porque supone el reconocer los propios errores. Hay que intentar evitar esta conducta y mentalizar al ingeniero de que una sesión de pruebas con éxito es precisamente la que encuentra errores.

Realizar estas pruebas conlleva generar un documento, protocolo de pruebas, donde se rellenan los resultados. Esto es vital para que sea algo exhaustivo y no solamente las pruebas que se hacen mientras se está codificando. Se pueden considerar los distintos tipos de pruebas [Pressman 90]:

- Pruebas de caja blanca: probar el funcionamiento interno de cada módulo en todas las posibilidades.
- Pruebas de caja negra: probar el funcionamiento externo. Es comprobar las salidas de los módulos frente a ciertas entradas.
- Pruebas de integración: juntar varios módulos y comprobar que responden adecuadamente
- Pruebas de validación: prueba de todo el sistema completo. Esta prueba es el mínimo indispensable para tener ciertas garantías.

La fase de pruebas es un apartado al que hay que dedicar mucho tiempo, pero son imprescindibles. Sin embargo habrá que valorar la complejidad del sistema para decidir qué tipo de pruebas son convenientes.

### 3.5. Documentación

Engloba textos descriptivos, comentarios en los programas, los diagramas ya desarrollados y protocolos de pruebas documentados [Pressman 90]. Como vemos es una fase que se ha ido desarrollando a medida que se hacían las otras con el consiguiente ahorro de tiempo y aumento de la eficiencia, además de quitar ese sentimiento de "tedio" que casi todo ingeniero tiene cuando debe realizar la documentación.

Ya que esta metodología no se está utilizando actualmente en la industria, pensamos que habría que considerar una implantación progresiva, evitando cambios bruscos. De todas maneras se ha de subrayar el hecho de que probablemente no habrá que utilizar el método en su totalidad, dependiendo de la aplicación y su dificultad. La decisión de hasta donde llegar es cuestión del ingeniero y, por tanto, en su formación se debe subrayar el carácter eminentemente práctico de la metodología. Para sistemas sencillos es mejor plantear simplemente algún diagrama tipo DFD de carácter general y obviar el resto. En el caso de sistemas grandes y complejos es donde el método demuestra sus puntos fuertes y donde su uso puede marcar una gran diferencia en tiempo y recursos. Este enfoque de implantación facilitará la aceptación por otras personas en el proyecto.

## 4. CONCLUSIONES Y TENDENCIAS

La adopción de la metodología del software propuesta permitirá abordar con garantías las tareas de desarrollo de programas complejos a los que se enfrenta el ingeniero electrónico actual especializado en el diseño con microcontroladores. Las ventajas de esta metodología son claras: desarrollo de un software de gran calidad (la más importante quizás), con pocos errores y fácil de corregir o modificar. Asimismo cabe destacar la facilidad para trabajo en grupo y confección de una documentación exhaustiva. Su principal inconveniente es que hay que realizar un esfuerzo importante para su implantación y que los principales beneficios se apreciarán a medio-largo plazo.

Trazando un paralelismo con la evolución de la ingeniería informática podemos intuir hacia donde irá este campo en el futuro: integración de métodos de diseño software, hardware y mecánicos; integración con gestión de proyectos, análisis de riesgos,..., así como nuevas metodologías, como programación orientada a objetos o programación extrema.

## REFERENCIAS

[Martín del Brío 99] B. Martín del Brío. *Sistemas Electrónicos Basados en Microprocesadores y Microcontroladores*. Prensas Universitarias de Zaragoza, 1999.

[Yourdon 93] E. Yourdon, *Análisis Estructurado Moderno*, Prentice Hall, 1993.

[Pressman 90] R. S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*. McGraw Hill, 1990.

[Monge 95] L. Monge, Apuntes del *Curso de Ingeniería del Software*, 1995.