UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

DIVISIÓN DE ESTUDIOS DE POSGRADO

MAESTRÍA EN ELECTRÓNICA Y COMPUTACIÓN

"DESARROLLO DE UNA METODOLOGÍA PARA EL DISEÑO DE SISTEMAS EMPOTRADOS BAJO EL PARADIGMA DE MEJORA DEL PROCESO SOFTWARE"

TESIS

Para obtener el grado de Maestro en Electrónica y Computación

PRESENTA
Ing. Andrea Ismeneé Herrera Huerta

DIRECTOR DE TESIS
Dr. Iván Antonio García Pacheco

Heroica Ciudad de Huajuapan de León, Oaxaca. Agosto 2011

2

EL CONTEXTO DE LOS SISTEMAS EMPOTRADOS Y LA CALIDAD DEL PRODUCTO

El desarrollo de dispositivos, equipos y sistemas electrónicos ha ido experimentando avances sucesivos desde que, hace décadas, surgieran los primeros componentes discretos. Sin embargo, las tareas que les son asignadas en la actualidad aumentan su complejidad, la cual sólo puede ser asumida por componentes altamente integrados cuya evolución más reciente es el SE [Salgado, 2008].

En este capítulo se abordarán los conceptos pertinentes a los SE, su historia, su clasificación y los retos actuales en cuanto a su desarrollo.

2.1 DEFINICIÓN FORMAL DE SISTEMA EMPOTRADO

Existen una gran variedad de definiciones para los SE, que son correctas pero subjetivas. A continuación se presentan algunas de estas:

1. Graaf, Lormans, y Toetenel definen a los SE como "*una mezcla de hardware y software que está dedicada a una aplicación especifica, y que forma parte de un sistema físico mayor unido por, al menos, una conexión lógica*" [Graaf, Lormans & Toetenel, 2002].
2. De acuerdo a Galeano [Galeano, 2009] un SE es "*un circuito electrónico computarizado que está diseñado para cumplir una labor especifica de un producto*". Así, la palabra empotrado ha reflejado el hecho de que estos sistemas se incorporan a un sistema de Ingeniería más general [Li & Yao, 2003], en el que se han realizado funciones de control, procesamiento y/o monitorización [Berger, 2002].

A partir de las definiciones anteriores y de la investigación realizada, se define a los SE de la siguiente manera:

3. Un SE es una combinación de software y hardware y algunas otras partes, mecánicas o de otro tipo, destinadas a desempeñar una función específica; cuyo objetivo es optimizar el producto final reduciendo su tamaño o costo, o mejorando características relacionadas con su funcionalidad (eficiencia, disponibilidad, etc.). Estos sistemas se incrustan en un producto más grande y normalmente no son visibles para el usuario. Además, el software empotrado es vital pues los convierte en sistemas de procesamiento informático.

Con base en estas definiciones se puede afirmar que el desarrollo de SE comprende tanto la parte hardware como software, e implica un diseño paralelo de hardware y software.

El software se vuelve un factor limitante (tanto en calidad como en tiempo de desarrollo y en costo), y el uso de técnicas rigurosas de desarrollo de software puede contribuir en gran medida a la calidad de los sistemas, así como al tiempo de comercialización de nuevos dispositivos o familias de dispositivos. Mientras que conocer el funcionamiento del hardware es trascendente y es uno de los requisitos fundamentales para realizar un buen diseño ya que, entre otras cosas, el hardware delimitará las capacidades del sistema que pueden mejorarse con el software.

Una decisión de hardware puede afectar al software y viceversa; por lo que es necesario considerar para su diseño y desarrollo dos elementos conceptuales: la arquitectura hardware (construida en base de un procesador y de dispositivos lógicos programables) y la arquitectura software (que involucra a sistemas operativos, lenguajes de programación, compiladores y herramientas de modelado) [Ball, 2002] (véase Figura 2.1).

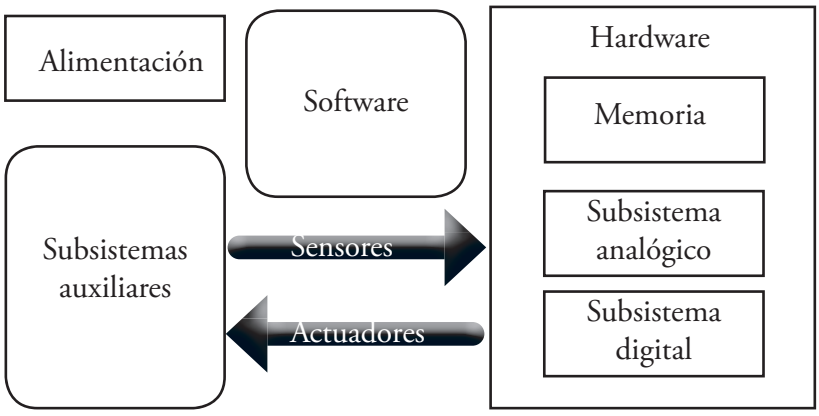

Figura 2.1. Arquitectura típica de un SE

El desarrollo de cada SE es muy específico en cuanto al producto y su aplicación. Sin embargo, el desarrollo de un SE básicamente incluye un ciclo de vida con los siguientes pasos:

- **Especificación del producto:** describe lo que será y lo que hará el producto final [Ball, 2002]; en esta fase se establecen los requisitos y en función a estos se eligen las herramientas de desarrollo hardware y software [Berger, 2002]. Para esta etapa del diseño existen diferentes herramientas para formalizar el SE de acuerdo a las funciones y restricciones que debe satisfacer.
- **División hardware/software:** identifica los subsistemas o módulos que serán implementados vía software o hardware [Berger, 2002]. En esta etapa se debe considerar que los requisitos de hardware son más rigurosos que los requisitos de software; la división depende en gran medida del procesador seleccionado.
- **Diseño de software:** la calidad del software debe ser inherente desde el principio e incorporada mediante el diseño [Calero, 2010]. Para el diseño del software empotrado (conocido también como firmware) existen diversas maneras de describirlo, esto depende de qué información ha de enviarse; algunos de los métodos más utilizados son los diagramas de flujo, diagramas de estado, pseudocódigo, etc. Así, para realizar un buen diseño se debe contar con los requisitos debidamente documentados y contemplar las características del producto.
- **Diseño de hardware:** una vez que el sistema está diseñado y los requisitos de hardware se han establecido, el siguiente paso es diseñar el hardware [Ball, 2002]. El objetivo es tener un diseño detallado del sistema a nivel hardware que cumpla con los requisitos del sistema. Por lo tanto en esta fase se realizan tareas específicas para el desarrollo del hardware (definición de la interfaz hardware, requisitos de tamaño, consumo, etc.).
- **Integración del sistema:** integra los componentes hardware con los componentes software. Una vez que se cuenta con un prototipo a nivel hardware y con el software empotrado compilado sin errores se puede iniciar con esta etapa. Es importante realizar pruebas de hardware y depurar el software simultáneamente usando las herramientas y métodos especiales para el manejo de la complejidad, por ejemplo: Matlab/Simulink, LabView y Proteus.
- **Pruebas del producto:** determinan si el sistema funciona correctamente; estas pruebas son más estrictas. Existen métodos de depuración para SE, sin embargo muchos de estos sistemas no pueden depurarse hasta que se encuentran operando.
- **Mantenimiento y actualizaciones:** la mayoría de los diseñadores de SE (alrededor del 60%) mantienen y mejoran los productos existentes, en lugar de diseñar nuevos productos. La mayoría de estos diseñadores no suelen ser miembros del equipo de diseño original, por lo que se debe confiar en su experiencia, conocimientos técnicos, la documentación existente y el producto; para entender el diseño original y mantenerlo y mejorarlo [Berger, 2002].

Regularmente, los SE son implementados en placas únicas o en un solo chip y deben de comprender tres aspectos: procesamiento, almacenamiento y comunicación [Vahid & Givaigis, 2002]. De acuerdo a [Noergaard, 2005], los sistemas de placas únicas son sistemas en los que se encuentran integrados los recursos de hardware en una tarjeta y son clasificados en cinco categorías de acuerdo a la funcionalidad del componente, estas son: Unidad Central de Procesamiento (CPU), memoria, dispositivos de entrada, dispositivos de salida, y las rutas de conexión (bus de datos). La Figura 2.1 muestra que los SE están conectados con el entorno físico a través de sensores, y su estructura

puede integrarse por FPGAs o partes dedicadas: dispositivos ASIC (Circuitos Integrados de Aplicación Específica), DSP (Procesador Digital de Señales), microcontroladores, etc. [Edwards *et al.*, 1997]. Dado que los SE interactúan continuamente con el medio ambiente que es de naturaleza analógica, normalmente se usan convertidores A/D y D/A, además de componentes electrónicos externos que realizan el resto de tareas necesarias. Los SE implementados en un solo chip cuentan con todos los recursos ya mencionados de forma interna y proporcionan un mayor rendimiento y menor consumo de energía; sin embargo su complejidad es mayor. Para el desarrollo de los SE, es necesario tener en cuenta todos los aspectos de hardware y de software; a continuación se analizarán los componentes más relevantes del hardware.

2.1.1. EL PROCESADOR

De acuerdo a [Marwede, 2006], en el 2006 se estimó que un 79% de todos los procesadores utilizados eran empleados en SE. El procesador (o CPU) es la unidad principal de los SE que ejecuta el software almacenado en la memoria y se encarga de realizar las operaciones como: cálculos matemáticos, ejecución de código para realizar una determinada tarea, además de gestionar el funcionamiento de los subsistemas. Los SE pueden emplear uno o varios procesadores digitales en formato microprocesador, microcontrolador, DSP, DSC, FPGA o usar tarjetas especiales para el desarrollo de SE dependiendo de la complejidad del sistema [Noergaard, 2005].

Inicialmente los SE eran construidos con procesadores de propósito general sin embargo estos procesadores complejos de diseño, incrementaban los costos del sistema debido a sus características y amplio espectro de funcionalidades. Los avances realizados en la tecnología de procesadores en los últimos años, permitieron diseñar e implementar los SE utilizando procesadores especiales en lugar de procesadores de propósito general. Estos procesadores son procesadores de propósito especial diseñados para una clase específica de aplicaciones [Li & Yao, 2003].

De acuerdo a la Figura 2.2, la selección del procesador a usar es una de las tareas principales para el desarrollo de SE, esta tarea se realiza por el staff de Ingeniera de hardware y/o de software considerando los siguientes puntos [Ball, 2002]:

- Número de pines de entrada y salida necesitados.
- Interfaces y memoria requerida.
- Número de interrupciones.
- Consideraciones de tiempo real.
- Entorno de desarrollo.
- Velocidad necesaria de procesamiento.
- Arquitectura de la memoria.
- Requisitos de alimentación.
- Disponibilidad, precio en el mercado y necesidades reales.
- Costos del ciclo de vida.

La oferta en el mercado de semiconductores, tanto de microprocesadores, microcontroladores, DSP y componentes similares, es elevada y se requiere de una cuidada fase de estudio inicial para seleccionar el más adecuado de acuerdo a la aplicación. En la Tabla 2.1 se presenta una lista de los procesadores más comunes para el desarrollo de SE y sus características.

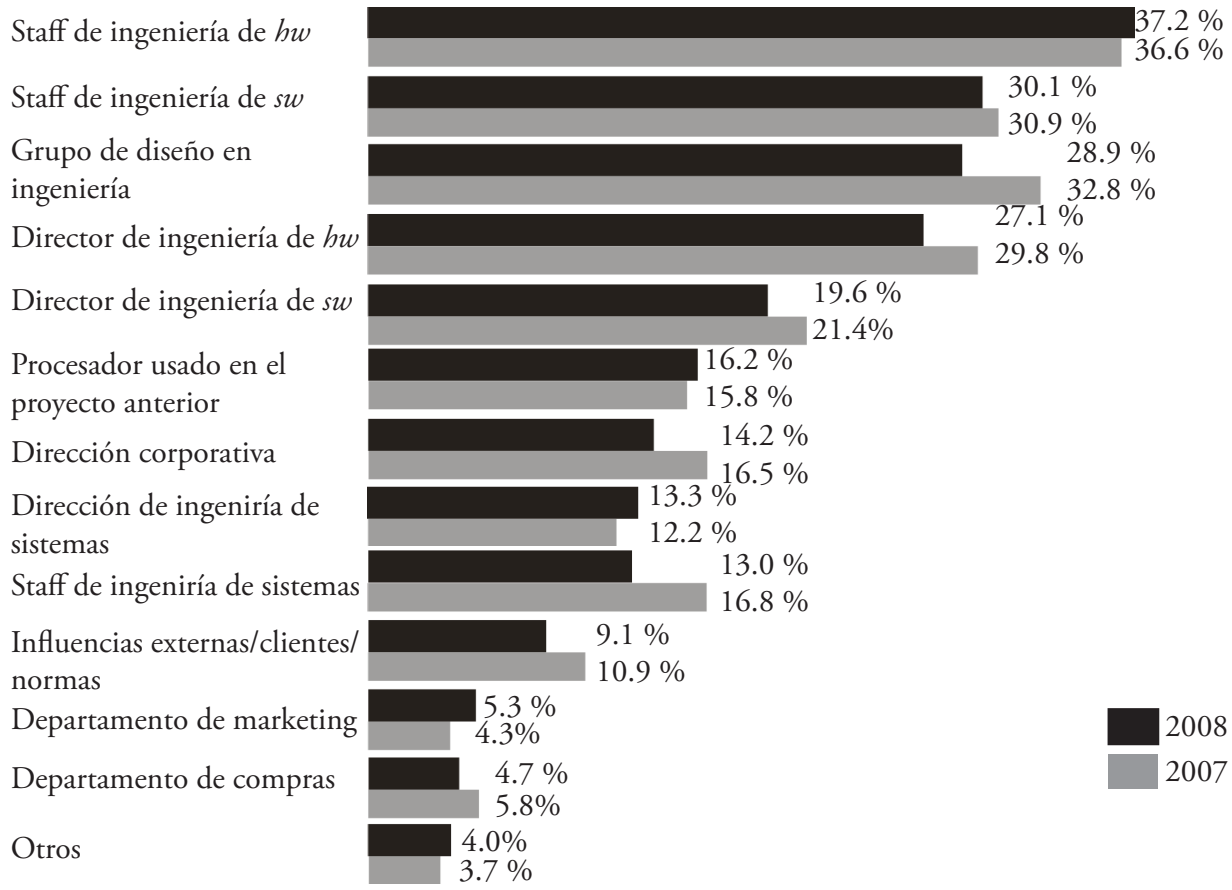

Figura 2.2. Influencias para seleccionar el procesador del SE a desarrollar [URL-2]

Tabla 2.1. Procesadores usados en el desarrollo de sistemas empotrados

Arquitectura	Descripción	Campo de aplicación
ARM	Familia de microprocesadores RISC, cuya principal ventaja es el consumo bajo de energía.	ARM11. Teléfonos celulares inteligentes/ PDA / equipamiento de redes. ARM Cortex-A. Telemedicina/ seguridad / vigilancia/ aviónica.
AMD	Familia de microprocesadores RISC, cuya principal ventaja son sus precios competentes y la tecnología usada.	AMD Phenom 64 Quad, AMD Athlon 64 y AMD Opteron
PowerPC	Desarrollada por IBM, Motorola y Apple. Son procesadores con arquitectura tipo RISC.	PowerPC E600. Redes/ telecomunicaciones.
AVR	Familia de microprocesadores RISC de ATmel.	Tiny AVR ATtiny11/ AVRATmega2560/ ATMEL AT90S1200/ ATmega2560

2.1.1.1. MICROPROCESADOR

Un microprocesador es un controlador complejo síncrono y programable en un solo circuito integrado (CI) que incluye los circuitos de las unidades de control y aritmética-lógica, en otras palabras la CPU, y un conjunto mínimo de memoria integrada y de componentes E/S (I/O) [Tocci & Widmer, 2010]. Para poder realizar su tarea y crear un sistema completo se necesitan otros chips adicionales, tales como memoria, circuitos de entrada salida E/S y reloj; y suelen ser de propósito general.

En 1971, Intel Corporation y el talento creativo de Hoff, Shima, Mazor y Faggin lanzaron el primer microprocesador: el 4004, de 4 bits [Brey, 2002]; lo que marcó una revolución en el campo del diseño de los controladores industriales y de los sistemas lógicos en general, teniendo impacto principalmente en sistemas complejos en lo referente a costo, flexibilidad y minimización de espacio físico ocupado. Desde entonces los fabricantes han realizado muchas mejoras, una de ellas es la expansión de la palabra del microprocesador de 4 a 8, 16, 32 bits (y en algunos casos hasta 64 bits).

Hasta hace unos años los principales fabricantes de microprocesadores eran AMD, Intel y Motorola [Kang, Kwon & Lee, 2005]. Existe una gran variedad de microprocesadores que varían en complejidad, velocidad, tamaño, y capacidad. Sin embargo, su arquitectura básica es la misma. Es común referirse a un microprocesador como la MPU (unidad del microprocesador).

Las unidades elementales de un microprocesador son: unidad de control, unidad aritmética-lógica (núcleo del microprocesador), registros internos, banderas y punteros, sistema de bus de datos, sistema de bus de direcciones y unidades funcionales. El funcionamiento básico de un microprocesador consiste en leer y ejecutar paso a paso todas y cada una de las órdenes (instrucciones) programadas por el diseñador del sistema. Éste se encarga del control y el procesamiento de datos en todo el ordenador. Para esta tarea es necesaria la ayuda de otros elementos capaces de realizar funciones específicas y así liberar de trabajo al microprocesador.

2.1.1.2. MICROCONTROLADORES

La mayoría de los SE han usado microcontroladores en lugar de microprocesadores [Berger, 2002]. El microcontrolador (MCU), es un circuito integrado programable que contiene todos los componentes de un procesador. En su arquitectura interna además de la CPU, están la memoria, los módulos de entrada salida y todos los recursos complementarios (buses, reloj, E/S, otros periféricos tales como conversores A/D, temporizadores, y demás) es decir, se trata de una computadora completa en un solo circuito integrado programable y se destina a gobernar una sola tarea con el programa que reside en su memoria. Sus líneas de E/S soportan la conexión de los sensores y actuadores del dispositivo a controlar. Por lo tanto, es un dispositivo programable capaz de ejecutar instrucciones almacenadas en su memoria de código.

Los microcontroladores están diseñados para proporcionar soluciones de bajo costo, por lo que su uso puede reducir drásticamente los costos de diseño y eliminar tareas redundantes de un proyecto [Craft, 2002]; por eso el tamaño de la CPU, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación.

Existen varias empresas que se dedican al desarrollo de MCU, algunas son: Microchip, Atmel, FreeScale (antes Motorola), Hitachi, Holtek, Intel, National Semiconductor, NEC, Parallax, Texas Instrument, Zilog, Silab, entre otras. La selección de un MCU depende de la tarea que se quiera realizar y por lo general éste está formado por siete componentes principales: CPU, ROM, RAM, entradas y salidas, temporizadores, buses e interrupciones [Craft, 2002].

2.1.2. MEMORIA

De acuerdo a [Ball, 2002] el determinar los requisitos de memoria también es una parte esencial para el diseño de los SE. En ésta se encuentra almacenado el software que el sistema puede ejecutar así como los datos. En estos sistemas puede existir una jerarquía y diferentes tipos de memoria (RAM o ROM) con diferentes tamaños; cualquier SE tendrá un poco de cada una. Si se requiere sólo una pequeña cantidad de memoria puede estar incluida en el mismo chip que el procesador. De lo contrario, uno o ambos tipos de memoria residirán en chips de memoria externa.

Los SE suelen almacenar todo su código objeto en la memoria ROM, mientras que, la memoria RAM es utilizada para almacenar los datos de entrada o de salida transitoria. La característica principal de la memoria de los SE es que debe tener un acceso de lectura y escritura lo más rápido posible para que el procesador no pierda tiempo en tareas que no son meramente de cálculo; además de considerar la densidad de la memoria. El diseñador de hardware por lo general debe hacer su mejor estimación por adelantado y estar preparado para aumentar o disminuir la cantidad real de memoria que el software utilizará [Barr, 1999]. La cantidad de memoria requerida también puede afectar a la selección del procesador.

2.1.3. ENTRADAS Y SALIDAS

Los SE siguen siendo una colección de piezas programables, y componentes estándar; que interactúan continuamente con un medio ambiente a través de sensores y actuadores [Balarin, *et al*, 1997]. Los sensores y actuadores de un SE son las entradas y salidas del sistema que permiten que el procesador se comunique con el mundo exterior para intercambiar datos. Los sensores proporcionan las entradas de datos hacia los procesadores.

Un sensor es un dispositivo eléctrico y/o mecánico que convierte magnitudes físicas en valores medibles de dicha magnitud. Los sensores van a aportar información tanto del entorno como del estado interno del componente que mide. En los SE suelen utilizarse diversos tipos de sensores como por ejemplo, sensores de luz, sensores de contacto, sensores de temperatura, etc. Las salidas se muestran normalmente como elementos encargados de llevar a cabo las acciones indicadas por el procesador y adoptan la forma de displays y dispositivos electromecánicos (que influyen directamente en el entorno a través de controladores de motores eléctricos, relés, conmutadores, motores eléctricos y otros equipos mecánicos) [Marwedel, 2006]. En esta etapa los SE hacen uso de convertidores digitales a analógicos para convertir la información digital (véase Figura 2.1).

El software determina las tareas que deben realizar los microprocesadores, microcontroladores, DSPs, PLCs, FPGAs u otros dispositivos programables utilizados para el control de equipos electrónicos, eléctricos, electromecánicos, y subsistemas.

El cambio que ha provocado el aumento del software empotrado en productos y servicios ha implicado un cambio riguroso en el desarrollo de los productos empotrados; pues el desarrollo de SE no solo es un desarrollo de hardware con un software añadido. El software empotrado tiene características que lo diferencian de las aplicaciones normales, por ejemplo: la interacción directa con los sistemas hardware, el funcionamiento sin errores y la capacidad de recuperarse por sí mismo en caso de que éstos ocurran y, regularmente, el funcionamiento con recursos limitados.

Un producto empotrado exitoso empieza con un diseño de sistemas flexible, el cual requiere de potentes herramientas de software y un entorno intuitivo de diseño. Es por esto que actualmente el software se ha convertido en el engranaje central de los SE, sin embargo, aunado a las ventajas que ofrece el software empotrado, este debe desarrollarse en el menor tiempo y con calidad para atender los retos de los SE. Así el software empotrado cuenta con diferentes retos entre los que se han encontrado: la creciente complejidad del software, la necesidad de satisfacer las exigencias de alto rendimiento de la convergencia tecnológica, entre otros [Graaf, Lormans & Toetenel, 2003].

El desarrollo de SE inicia con el desarrollo de hardware (que regularmente es necesario modificar hasta obtener el producto final) para después proceder con el desarrollo del software. Es importante considerar que las características que son construidas exclusivamente en hardware tienen costos demasiados altos, pero con el software esto puede cambiar. Las características adicionales que pueden ser implementadas utilizando software no provocan un aumento en los costos de producción comparado con la magnitud en que lo hace el hardware. Por otra parte, los problemas de hardware o defectos de hardware se pueden resolver utilizando las soluciones de software.

Así, las fases básicas para el desarrollo de software empotrado han sido las siguientes: diseño de la arquitectura software, definición de subsistemas software, diseño de bloques funcionales, e implementación de características y tareas [Kang, Kwon & Lee, 2005]. Sin embargo, debido a la importancia que ha adquirido la gestión de la calidad en los SE, el desarrollo de SE ha cambiado a un Desarrollo Guiado por Modelo (*Model-Driven Development,* MDD) y la tendencia de desarrollo considera las siguientes fases: requisitos, diseño funcional, arquitectura del sistema, arquitectura de software, diseño de software, implementación, pruebas de unidades especificas, integración del software y del sistema y pruebas, pruebas funcionales, y validación [Liggesmeyer & Trapp, 2009].

Un denominador común en todos los dominios de software empotrado es el uso de lenguajes de programación que permiten el acceso directo a las interfaces, la memoria, entre otras cosas. Más del 80% de todas las empresas están usando C (y C++), así como tambien, más del 40% están usando el ensamblador para las interfaces de nivel inferior. Mientras que Java se utiliza cada vez para el GUI y programación de aplicaciones.

Otros lenguajes utilizados para desarrollar SE son de alto nivel, por ejemplo: Microsoft .NET Micro Framework, Microsoft .NET Compact Framework o Java Platform Micro Edition, NesC. Actualmente se desarrollan sistemas operativos y herramientas de desarrollo para SE, como lo son: Microsoft (Windows CE, Windows XP Embedded), Wind River (VxWorks, Linux), Symbian (SymbianOS), Palm, Green Hills (INTEGRITY) [Kang, Kwon, & Lee, 2005], Intel® Embedded Design Center, e Intel C++ para Microsoft Embedded Visual C++, Android, Maemo, Network Embedded Systems C, Windows Mobile, eCos, LynxOS, QNX y Linux Embebido (véase Tabla 2.2). Cada proveedor sigue enfoques diferentes dependiendo de su experiencia y posicionamiento en el mercado.

Así, cada vez más la calidad del software está tomando mayor importancia en las organizaciones por su influencia en los costos finales y por ser un elemento diferenciador de la competencia frente a sus clientes.

Tabla 2.2. Sistemas Operativos usados para Sistemas Empotrados

Sistema	Descripción
Android	Android es un stack de software orientado a dispositivos móviles que incluye un sistema operativo, middleware y aplicaciones clave. Está basado en una versión modificada del núcleo Linux, por lo tanto Android es código libre.
ARM *Linux Mobile Platform*	ARM y otras empresas como Texas Instruments, Samsung, Marvell o Mozilla están creando una plataforma basada en Linux especialmente diseñada para dispositivos móviles.
Maemo	Maemo es una plataforma de desarrollo para dispositivos portátiles basado en Linux. Maemo cuenta con una interfaz de usuario optimizada para dispositivos móviles. La mayoría de los componentes Maemo son de código abierto, lo que ofrece a los usuarios y desarrolladores la libertad y flexibilidad para contribuir y modificar el desarrollo de la plataforma central.
OpenMoko	Proyecto para crear una plataforma para smartphones utilizando software libre. Utiliza el núcleo de Linux y está basado en el framework de OpenEmbedded.
QNX	QNX es un sistema operativo de tiempo real basado en Unix. QNX brinda una fuente fiable para aplicaciones en telecomunicaciones, sistemas electrónicos, sistemas automotores, instrumentación médica, control industrial, y más.
Symbian (SymbianOS)	Symbian es un sistema operativo desarrollado por Symbian Ltd. para adaptarse a los requerimientos de un teléfono móvil. Fue producto de la alianza de varias empresas de telefonía móvil (Nokia, Sony Ericsson, Siemens, Lenovo, Motorola, Panasonic, etc.).
Ubuntu Mobile and Embedded	Iniciativa de Ubuntu para crear una versión para MID (*Mobile Internet Devices*); con el objetivo de hacer portable Ubuntu en smartphones. Los programadores Ubuntu desarrollan una versión móvil de su sistema operativo en colaboración con Intel. Una de las ideas principales del proyecto es contar con un procesador de bajo consumo y una arquitectura chipset diseñada para permitir capacidades de Internet completas sobre dispositivos móviles.
Windows Embedded Devices (Windows CE, Windows XP Embedded)	Sistema operativo de Microsoft para dispositivos empotrados. Permite generar una instalación de Windows XP (o CE) adaptada al hardware especifico del dispositivo. Es común encontrarlo empotrado en Sistemas de Posicionamiento Global (GPS), cajeros automáticos, automóviles, dispositivos portátiles, servidores, y demás.

2.2 CLASIFICACIÓN Y CARACTERÍSTICAS DE LOS SISTEMAS EMPOTRADOS

De acuerdo a [Vahid & Givaigis, 2002] los SE pueden clasificarse en tres grupos dependiendo de su interacción con su entorno:

- **Sistemas reactivos:** aquellos sistemas que están en constante interacción con su entorno y se ejecutan a un ritmo determinado por éste. La velocidad de operación del sistema deberá ser la velocidad del entorno exterior.
- **Sistemas interactivos:** aquellos sistemas que siempre interactúan con el exterior de tal forma que la velocidad de operación del sistema deberá ser la velocidad del propio SE.
- **Sistemas transformables:** aquellos sistemas que no interactúan con el exterior, únicamente toman un bloque de datos de entrada y los transforman en un bloque de datos de salida.

Desarrollar un SE consiste en construir una implementación que satisfaga la necesidad deseada, que optimice métricas de diseño (energía, tamaño de código, eficiencia en el tiempo de ejecución, peso, costo) y que cumpla con ciertas características que los distinguen de otros sistemas de cómputo. A lo largo del tiempo, se ha exigido que el diseño de un SE cumpla con las siguientes características:

- **Concurrencia:** los componentes del sistema deben de funcionar de manera simultánea.
- **Fiabilidad y seguridad:** no debe presentar fallos y debe manejar los errores. El manejo de errores se puede hacer a través de hardware o de software [Vahid & Givaigis, 2002].
- **Capacidad de reacción y tiempo real:** ciertos componentes del SE deben reaccionar continuamente a los cambios en el entorno del sistema y deben computar ciertos resultados en tiempo real [Berger, 2002].
- **Eficientes y fáciles de mantener:** el SE debe permitir su modificación después de su lanzamiento inicial.
- **Interacción con dispositivos físicos:** los SE interaccionan con su entorno mediante diversos tipos de dispositivos físicos.
- **Robustez:** los SE deben operar bajo condiciones ambientales extremas [Berger, 2002].
- **Bajo consumo:** Debido al uso de lógica TTL muchos sistemas están alimentados con baterías o pilas.
- **Precio reducido:** un SE es más limitado en funcionalidad de hardware y/o software que una computadora personal lo que implica un bajo costo de fabricación [Berger, 2002].
- **Diseño especial:** los SE están dedicados a tareas específicas, por lo que este tipo de diseños combina hardware personalizado con software empotrado [Balarin, *et al*, 1997], lo que los hace únicos.
- **Flexibilidad:** los SE son especialmente sensibles a aspectos mercadotécnicos, y deben ser capaces de evolucionar con el mercado de una manera flexible y en un periodo determinado de tiempo. Sin embargo, a diferencia del software diseñado para computadoras de uso general, el software empotrado generalmente no se puede ejecutar en otros SE, sin modificaciones significativas [Barr, 1999].
- **Sistemas híbridos:** los SE son híbridos en el sentido de que incluyen partes analógicas y digitales.

2.3 EVOLUCIÓN Y VISIÓN GENERAL DE LA SITUACIÓN ACTUAL DE LOS SISTEMAS EMPOTRADOS

En la actualidad, los SE están presentes en el mundo y los hogares promedio tienen más de 50 SE [Srinivasan, Dobrin & Lundqvist, 2009]. Así, el mercado de los SE es uno de los que presentan mayor crecimiento a nivel mundial. Los SE contribuyen en gran medida a mejorar nuestra vida cotidiana y agregan valor a los productos que los contienen, convirtiéndose en una vía importante de innovación, por lo que se les ha identificado como un área de importancia capital en la mayor parte de los países desarrollados. De acuerdo a [Barr, 1999] parecía inevitable que el número de sistemas empotrados siguiera aumentando rápidamente. En el 2003, Graaf aseguraba que el mercado de los SE crecería exponencialmente en los siguientes 10 años [Graaf, Lormans & Toetenel, 2003]. En la Figura 2.3, se observa que la investigación, desarrollo y uso de los SE ha aumentado con el paso de los años; según las previsiones de ARTEMIS, habrá más de 16,000 millones de dispositivos empotrados en el 2010 y se superarán los 40,000 millones para 2020.

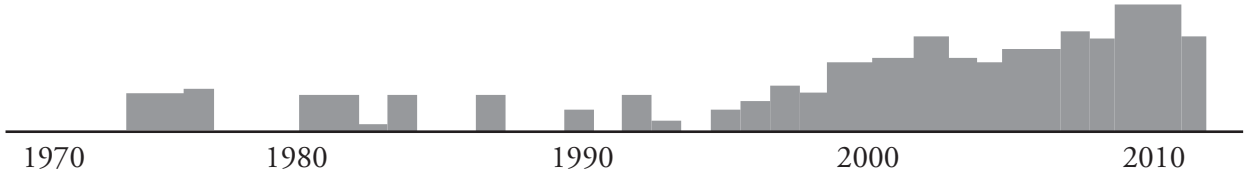

Figura 2.3 Historia de los SE [Gráfica generada con Google]

Los SE pueden ser de muy diferentes tipos y sus principales áreas de aplicación son: control industrial, vídeo e imagen, electrónica de consumo, sector aeroespacial, sector automotriz, equipos médicos y militares, computadoras y periféricos, comunicaciones de datos y telecomunicaciones (en la Figura 2.4 se presenta la distribución de los SE en la industria). Los SE se utilizan en aplicaciones críticas, sólo a modo de ejemplo, se mencionan algunas de estas:

- **Automotriz:** es posible encontrar SE como sistemas de control en automóviles. Los autos modernos sólo pueden venderse si contienen una cantidad significativa de electrónica: sistemas de control de las bolsas de aire, los sistemas de control del motor, los Sistemas de Frenado Antibloqueo (ABS), el aire acondicionado, Sistema de Posicionamiento Global (GPS), las características de seguridad, sistemas de encendido y control del motor.
- **Aplicaciones militares:** las compañías de SE actualmente deben concentrar sus esfuerzos en desarrollar mayor tecnología para el sector militar, en especial Vehículos Aéreos no-Tripulados (UAV), y todo lo relacionado con comunicación militar, además de otras aplicaciones tales como los DSP y de paquetes.
- **Sistemas médicos:** se encuentran SE en equipos de diálisis, equipos de monitorización fetal y todo tipo de equipos médicos y hospitalarios, incluso en equipos de fabricación de productos médicos y equipos de laboratorio.

Tal y como se mencionó, una de las características de los SE es que tienen que reaccionar en un tiempo adecuado y con el apropiado uso de recursos, por lo que estos sistemas también son omnipresentes en la telefonía móvil o la electrónica de consumo (áreas de gran impacto económico), por ejemplo:

- **Aparatos electrodomésticos:** televisores analógicos y digitales, DVD, VCR, asistentes de datos personales, aparatos de cocina (refrigeradores, tostadores, hornos de microondas), juguetes/juegos, teléfonos fijos, teléfonos celulares, cámaras, GPS.
- **Estaciones de red:** ruteadores, concentradores (HUB), gateways.
- **Control industrial:** robots y sistemas de control (manufactura).

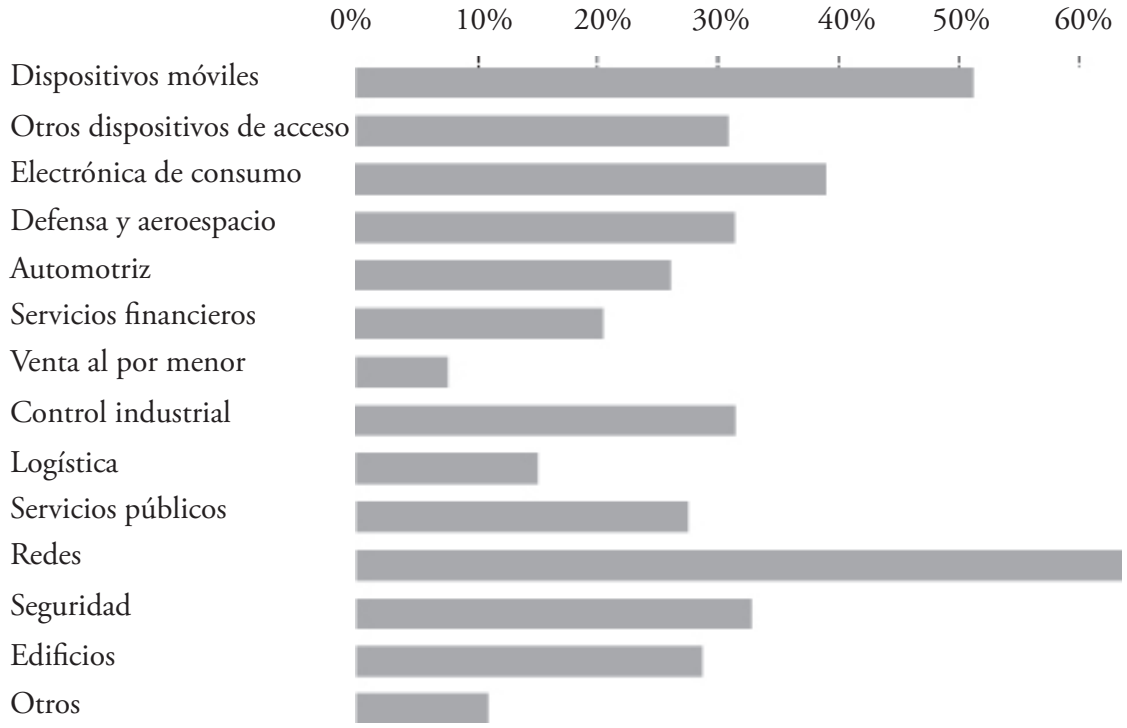

Figura 2.4. Distribución de los SE en la industria [URL-2]

A nivel de diseño de los SE, actualmente existen tres enfoques principales: codiseño hardware/software, plataforma de diseño, y diseño basado en componentes [Shaout, El-Mousa & Mattar, 2010].

- El **codiseño hardware/software** permite optimizar un diseño mediante la partición del sistema en componentes hardware y software (usando un enfoque top-down), considerando factores de calidad específicos. A diferencia del diseño tradicional, se parte de una especificación y se diseña de forma independiente a la arquitectura y posterior implementación del sistema. Sin embargo, surgen comúnmente limitaciones debido a indisponibilidad de herramientas adecuadas para el desarrollo de las distintas etapas del codiseño y a la factibilidad de realizar determinadas implementaciones por razones económicas. De acuerdo a [Srinivasan, Dobrin & Lundqvist, 2009] esta división de desarrollo en dos corrientes independientes, de software y de hardware, ocurre de forma paralela y el resultado de la integración puede convertirse en un punto crítico de fallo.
- ***Diseño basado en plataformas*** introducido desde hace varios años pretendiendo redefinir el futuro de los sistemas en chip (SoC); se ha convertido en un importante estilo de diseño. En general, una plataforma es una abstracción que oculta los detalles de la ejecución de varias posibles mejoras de nivel en capas subyacentes [Sangiovanni & Martin, 2001];

cada plataforma se trata de una colección de elementos. Su uso tiene ventajas y desventajas, pues, este enfoque limita opciones pero reutiliza diseños propiciando que el tiempo de comercialización sea rápido [URL-4]. La reutilización de diseños permite construir fácil y rápidamente SoC con características comunes.

- **Diseño basado en componentes**. El desarrollo de software basado en componentes permite reutilizar piezas de código pre-elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y un mayor retorno sobre la inversión. Los pasos básicos del diseño basado en componentes son: obtención de requisitos, partición en componentes, diseño interno, evolución del componente, e interconexión de componentes. Entre las ventajas que aporta el uso de este enfoque de diseño se encuentran: reutilización del software, simplificación de pruebas y del mantenimiento del sistema, mayor calidad, ciclos de desarrollo más cortos.

De acuerdo a [Noergaard, 2005], el diseño general de SE define siete fases de desarrollo: especificación del producto, división hardware y software, iteración e implementación, diseño detallado hardware y software, integración de componentes hardware y software, prueba y liberación del producto, mantenimiento y actualización. Esta división de desarrollo en dos corrientes independientes de software y de hardware ocurre de forma paralela, y el resultado de la integración puede convertirse en un punto crítico de fallo [Srinivasan, Dobrin, & and Lundqvist, 2009].

Para describir el ciclo de diseño de los SE se han usado modelos propios de la Ingeniería de Sistemas [Noergaard, 2005]. Estos modelos son:

- El modelo de gran explosión (*Bing-bang model*): en este modelo lo esencial es no planear o procesar los requisitos antes o durante el desarrollo del sistema.
- El modelo codificar-arreglar (*Code-and-fix model*): este modelo no define los procesos formales antes de empezar el desarrollo, pero si los requisitos.
- El modelo de cascada (*Waterfall model*): el modelo de cascada cuenta con procesos definidos para desarrollar el sistema; el resultado de un paso sirve como entrada para el siguiente.
- El modelo de desarrollo evolutivo: se entrelazan las actividades como la especificación, el desarrollo y la validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas y se refina basándose en las peticiones del cliente para producir un sistema que satisfaga los objetivos.
- El modelo de Ingeniera basada en Componentes: se basa en la existencia de un número significativo de componentes reutilizables. El proceso principal de desarrollo se centra en integrar los componentes existentes dentro del sistema más que en desarrollarlos desde cero.

El enfoque actual está introduciendo cada vez más a la Ingeniería de Software, ya que ésta proporciona técnicas y procedimientos que permiten ayudar a asegurar un software de calidad. Así, el desarrollo de SE de calidad bajo el enfoque de la Ingeniería de Software cubre los siguientes aspectos:

- Crecientes posibilidades del hardware.
- Creciente complejidad y tamaño en el software.
- Distribución y composición de equipos de trabajo.

Tal y como se mencionó en el Capítulo 1 de esta tesis, en el 2008 se realizó el estudio llamado "*Embedded Market Study*" que consistió en estudiar las principales áreas de aplicación de los SE; cuyas actividades se centran en la escritura de software, la depuración de software, la integración de hardware/software, elección o especificación de la arquitectura, diseño o el análisis del software, gestión de proyectos, y depuración de hardware.

En los resultados se observó que más de la mitad de los encuestados se encontraban trabajando en una actualización del producto, en lugar de trabajar en un proyecto nuevo. Más de la mitad de estas actualizaciones están aprovechando un microprocesador nuevo, por lo que requiere cambios de software. El tamaño del equipo de diseño promedio aumentó ligeramente, de 13.6 personas a 15.2 personas. Pero es interesante observar que el número de Ingenieros de Software en el equipo aumentó a casi dos, es decir, el número de Ingenieros de Hardware se mantuvo igual o se redujo ligeramente. En relación a lo anterior, la Figura 2.5 muestra el entorno de desarrollo que los desarrolladores de SE están usando. La Figura 2.6 muestra las principales herramientas hardware/software utilizadas para los SE. Es importante resaltar el lugar en el que se encuentran las herramientas de prueba de software, en la parte inferior de la lista. En relación a esto, los usuarios dicen que las pruebas son uno de los factores clave en el proceso de diseño y que toman a menudo la mayor parte del tiempo del mismo. Sin embargo en la lista aparecen en el último lugar, lo que deja como conclusión que los usuarios no están satisfechos con sus herramientas actuales de prueba de software. Es importante observar que las herramientas más usadas son el compilador/ ensamblador, el depurador y el osciloscopio. Por otra parte la necesidad de capturar las especificaciones en niveles altos de abstracción ha llevado al uso de herramientas de modelado como Matlab y Simulink[1] de MathWorks. Estas herramientas permiten a los diseñadores montar rápidamente algoritmos y simular su comportamiento.

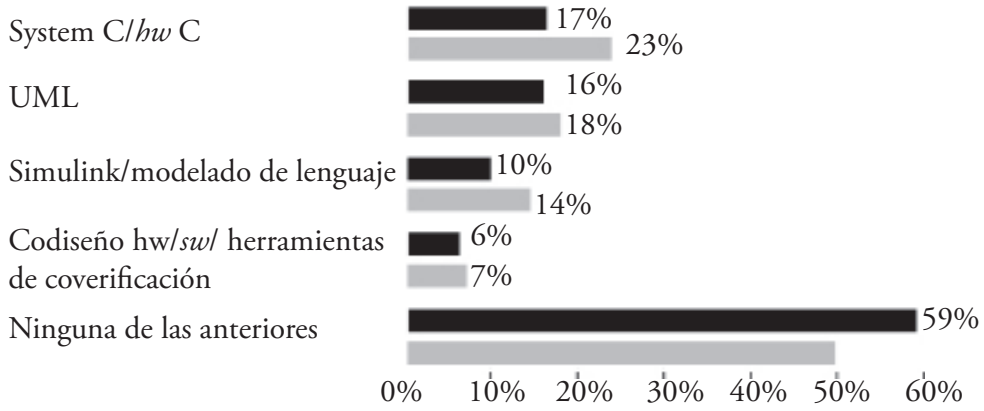

Figura 2.5. Entornos de desarrollo usados para SE [URL-2]

1 Simulink® es una plataforma para simulación y y diseño multidominio en base a modelos de sistemas-dinámicos y empotrados. Proporciona un entorno gráfico interactivo y un conjunto de librerías de bloques personalizables que permiten diseñar, simular, implementar y probar una gran variedad de sistemas con variación temporal, entre los que se incluyen sistemas de comunicaciones, control, procesado de señales, vídeo e imagen.

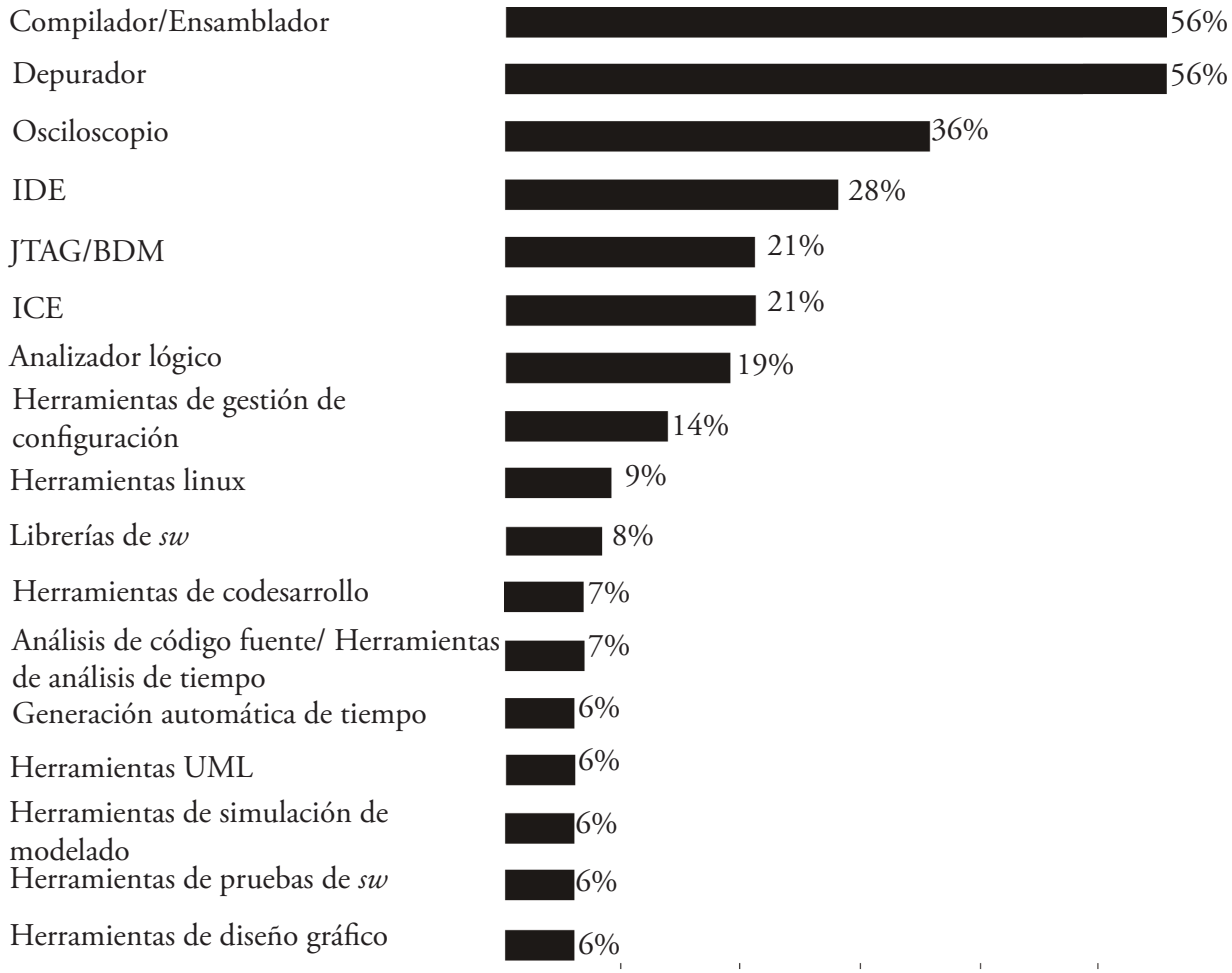

Figura 2.6. Herramientas hardware/software usadas para SE [URL-2]

2.4 RETOS EN EL DISEÑO DE SISTEMAS EMPOTRADOS

El campo de los SE es cada más difícil y las cuestiones de desarrollo de software empotrado están llamando la atención de un número creciente de investigadores, tanto en la industria como en la investigación. Hoy en día, se sigue pensando que los SE están diseñados con un enfoque *ad hoc* que está fuertemente basado en la experiencia anterior con productos similares y en el diseño de manuales, tal y como se determinó hace más de una década en [Balarin, *et al*, 1997].

Los problemas que surgen en el diseño e implementación de estos sistemas involucran prácticamente a todas las ramas de la ingeniería: Computación (tanto en software como en hardware), Comunicaciones, Control, Electrónica y Mecánica. Así, algunos de los principales retos de los SE son:

- **Estandarización:** Durante los últimos años se han utilizado diferentes tipos de metodologías (disponibles para el desarrollo de software que son adaptadas para SE), herramientas y lenguajes; debido a que existen diferentes tipos de enfoques (p.ej. herramientas para ingeniería de requisitos, herramientas de diseño, etc.) [Graaf, Lormans & Toetenel, 2002].

- **Tiempo menor de desarrollo:** antes los SE cumplían con grandes ciclos de vida; en losúltimos años estos sistemas han disminuido constantemente el tiempo destinado para su desarrollo. Las empresas que buscan ser líderes obteniendo una ventaja competitiva de estos productos, realizan una promoción de 6 a 9 meses del SE [Li & Yao, 2003]. Las empresas que no cumplen con estos tiempos para desarrollar o innovar su producto sufren pérdidas en la cantidad de ventas, lo que les ocasiona pérdidas muy importantes respecto a sus expectativas y en el avance tecnológico pues el poder de procesamiento de los microprocesadores sigue aumentando a un ritmo previsto por la Ley de Moore.
- **Bajo consumo de energía y alto rendimiento:** la componente software debe ser capaz de utilizar eficientemente las optimizaciones que la componente hardware pone a disposición de los diseñadores de SE para lograr disminuir el consumo de energía del sistema y aumentar su rendimiento.
- El desarrollo de software está cambiando de la programación manual al Desarrollo Guiado por Modelo.
- IBM *Software Group* afirma que el 80% de los costos de desarrollo se emplean en identificar y solucionar errores.

2.5 PROPUESTAS ACTUALES PARA EL DESARROLLO DE SISTEMAS EMPOTRADOS

En la actualidad se han desarrollado diferentes investigaciones en el dominio de los SE que pretenden gestionar adecuadamente un desarrollo de calidad buscando un equilibro entre los siguientes tres elementos: costo, calidad y tiempo (pues de estos elementos depende directamente el éxito de este tipo de sistemas, tal y como se ha manifestado a lo largo del documento). Los diversos marcos de desarrollo o Entornos de Desarrollo Integrados (*Integrated Development Environment*, IDEs) para SE tienen por objetivo establecer fases para guiar a los desarrolladores en el proceso de desarrollo de SE, así como dar respuesta a los cambios que surgen durante las diferentes fases. A continuación se analizarán las principales propuestas (y de actualidad) con el objetivo de identificar aspectos comunes y establecer así un proceso eficiente de diseño.

2.5.1. *EMBEDDED SOFTWARE COMPONENT MODEL (ESCM)*

De acuerdo a [Li *et al*., 2009], las tendencias del software empotrado representan una necesidad urgente para la cual es posible emplear técnicas avanzadas para su desarrollo. El desarrollo desde cero es una práctica común, sin embargo los beneficios potenciales del desarrollo basado en componentes; tales como la reducción del tiempo y costo de desarrollo, y el aumento en la calidad, así como la especialización de conocimientos, son muy atractivos en el ámbito de los SE. Pero a pesar de esto, el uso directo de las tecnologías de uso general basadas en componentes para el ámbito de los SE resulta complicado debido a varios retos ya mencionados.

En primer lugar, las aplicaciones empotradas se suelen utilizar en tiempo real y en entornos con alto grado de seguridad. Por lo tanto, las propiedades no funcionales (NFPs), como la fiabilidad en tiempo real y la memoria, tienen que ser modeladas explícitamente para permitir el razonamiento 'composicional' automatizado. Segundo, dado el tamaño pequeño y los requisitos para la movilidad, los SE a menudo suelen estar limitados en recursos.

2.5.1.1. OBJETIVO

Establecer un modelo formal para el desarrollo de software basado en componentes y orientado al campo de aplicación empotrado. El modelo ESCM (*Embedded Software Component Model*) consta de tres elementos esenciales: interfaz, componentes, y conector en combinación con contratos. ESCM se orienta a la especificación, la verificación y la composición de software empotrado y describe cómo especificar los componentes desde cuatro perspectivas: sintáctica, funcional, de calidad del servicio (QoS) y de sincronización. ESCM pone especial atención en los atributos no funcionales de los SE. Así, mediante un contrato llamado "contrato de NFPS" se especifican los atributos no funcionales de este tipo de sistemas.

2.5.1.2. ESTRUCTURA

La Figura 2.7 muestra que ESCM se basa en tres elementos: interfaz, componentes y conector. Cada elemento es especificado a través de uno o varios contratos e información adicional. El marco de ESCM especifica que el contrato de firma, el contrato de funcionalidad y el contrato de los NFPs están relacionados directamente con la interfaz. Mientras que el contrato de interacción es responsable de los componentes; y el contrato de adhesión (contrato "*glue*") es el responsable del conector.

El desarrollo basado en componentes tiene por objetivo desarrollar y mantener sistemas de software a través de componentes ya existentes. Es un sentido común que los componentes deben ser reutilizables y pueden interactuar entre sí en la arquitectura del sistema. A tal efecto, un componente de software empotrado es una unidad de ejecución reutilizable con interfaces especificadas contractualmente y comportamientos coherentes y dinámicos de especificación formal.

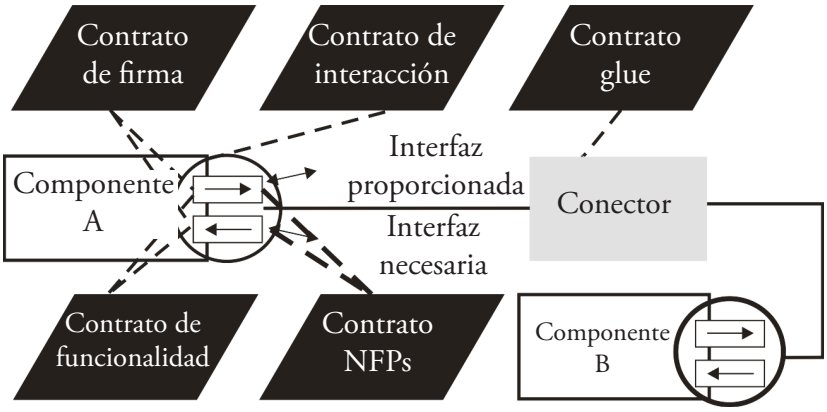

Figura 2.7. Marco de trabajo del modelo ESCM [Li *et al.*, 2009]

Por lo tanto, ESCM define a un componente mediante los siguientes elementos: un conjunto de interfaces (interfaces necesarias e interfaces requeridas) y la descripción del comportamiento dinámico de los componentes. Otro de los componentes del modelo es la interfaz que permite describir la interacción del componente con el resto del sistema de manera abstracta (usando al componente como una caja negra), y determina el comportamiento externo y las características de los componentes.

Así, el marco ESCM define un interfaz como una especificación de servicios expuestos a través de los componentes que interactúan y se puede utilizar en la construcción y mantenimiento de software empotrado. ESCM define una interfaz mediante los siguientes elementos:

- *Tipo*, que describe la interacción entre la interfaz y el entorno del componente,
- *Firma*, que es la especificación a nivel sintáctico que describe cómo usar la interfaz,
- *Funcionalidad*, que es una especificación de nivel semántico de la funcionalidad de la interfaz y presenta una descripción formal a través del contrato de funcionalidad, y
- *NFPs*, muchos modelos se enfocan sólo en los aspectos funcionales de un componente, sin embargo, para el software empotrado no es suficiente y mediante el contrato de NFPs se especifican las limitaciones no funcionales.
- El último elemento del modelo es el conector y este se encarga de proporcionar el «pegamento» para el diseño arquitectónico; es decir, es el elemento encargado de la comunicación y coordinación entre los componentes.

La Figura 2.8 muestra que ESCM distingue cinco tipos de contratos en cuatro niveles. Estos contratos se deben especificar estrictamente para la reutilización de componentes de forma segura. Las etapas para el desarrollo basado en componentes de ESCM son:

- **Especificaciones:** especificar el sistema y los componentes del mismo.
- **Validación:** obtener los componentes propios del repositorio de componentes que coinciden con los componentes especificados. Si no hay componentes existentes reutilizables, entonces se deben desarrollar nuevos componentes según las especificaciones.
- **Composición:** formar el sistema usando el conector y los componentes.
- **Predicción y análisis:** analizar las propiedades del sistema, especialmente los NFP.

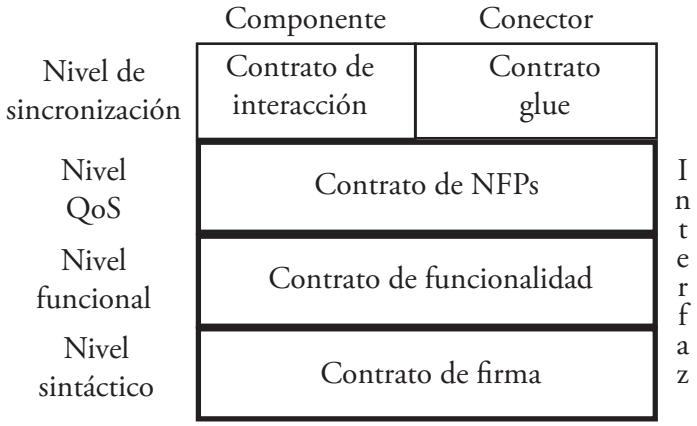

Figura 2.8. Contratos en ESCM [Li *et al.*, 2009]

2.5.2. *SAVE INTEGRATED DEVELOPMENT ENVIRONMENT (SAVE-IDE)*

Algunos dominios de los SE requieren tener una alta confianza en la calidad del desarrollo del producto.

Para ello, un deseo fundamental es tener la capacidad para hacer frente a requisitos tales como la fiabilidad (por ejemplo, disponibilidad y seguridad), la planificación (tales como la libertad y el tiempo de respuesta, tiempo de ejecución, el plazo de desarrollo), y la utilización de recursos (incluyendo la memoria, CPU, canales de mensajes, y/o el consumo de energía). De acuerdo a [Sentilles *et al.*, 2009], esto exige un fuerte énfasis en la posibilidad de analizar y automatizar el proceso de desarrollo para garantizar la necesaria calidad de los productos finales con respecto a tales requisitos. Al mismo tiempo, la creciente complejidad de los SE requiere métodos que incrementen el nivel de abstracción, mejoren la reutilización, y permitan la concurrencia en el proceso de desarrollo. Tal y como se dijo anteriormente, un enfoque para conseguirlo es la Ingeniería de Software basada en componentes. Sin embargo, la mayoría de las tecnologías basadas en componentes carecen de las herramientas de análisis formales necesarias para asegurar la confiabilidad. *Save Integrated Development Environment* (Save-IDE), reúne las herramientas y técnicas necesarias para el proceso de desarrollo de SE y los integra con el desarrollo basado en componentes. Esto incluye el desarrollo basado en un modelo de componentes, o SaveCMM, que está diseñado para permitir la eficiencia del diseño de los SE y el análisis temporal del modelo.

2.5.2.1. OBJETIVO

Save-IDE se ha diseñado como una plataforma con un conjunto ampliable de herramientas de apoyo integrado para lograr el enfoque *Save-Comp Component Technology* (SaveCCT). Así, Save-IDE aplica un nuevo enfoque que integra las siguientes actividades: diseño, análisis, transformación, verificación y síntesis.

2.5.2.2. ESTRUCTURA

Save-IDE se basa en el proceso de desarrollo SaveCCT el cual está diseñado como un enfoque top-down con énfasis en la reutilización de componentes. La Figura 2.9 muestra que el modelo comprende tres fases: diseño, análisis, y realización. Si los componentes coinciden con los requisitos que ya existen, se seleccionan y se adaptan a la actividad. De lo contrario, se desarrollan nuevos componentes. En consecuencia, los componentes son analizados y verificados individualmente en base a los requisitos. Después de haber reconstruido el sistema, partes del sistema o las composiciones del sistema deben ser analizadas o verificadas (lo que se denomina como verificación formal del sistema). El sistema y el diseño de componentes y el procedimiento de verificación se repiten hasta que los resultados son aceptables desde el punto de vista del análisis. Enseguida se continúa con la fase de realización, que consiste en la síntesis y la ejecución o simulación de las actividades.

El sistema se sintetiza de forma automática basándose en la entrada del diseño del sistema, en las implementaciones de los componentes y, en algoritmos estáticos para el uso de recursos y las limitaciones de tiempo. El proceso de desarrollo es semi-automático, con varias actividades automatizadas. Una actividad automatizada es la producción del esqueleto de los archivos de aplicación (archivos de cabecera C y sus correspondientes archivos) en base a la especificación del componente. Otra automatización reside en la generación de los de intercambios de archivos utilizados como medio de comunicación entre las herramientas. La tercera se produce durante la síntesis que incluye la transformación de los componentes en las unidades ejecutables en tiempo real, las tareas, la generación de código, la inclusión de un algoritmo de programación

particular, la compilación y la vinculación de todos los elementos en la imagen ejecutable. En la Figura 2.10 se muestra la estructura de Save-IDE, que está compuesta por el editor de la arquitectura donde el sistema y componentes del modelo pueden ser creados. Los componentes individuales pueden ser implementados desde archivos generados en plantillas de C a través de la herramienta del entorno C (CDT de Eclipse). El editor de arquitectura permite realizar la especificación de la interfaz funcional y asignar diferentes atributos a los componentes, tales como el tiempo de ejecución, o el modelo de comportamiento.

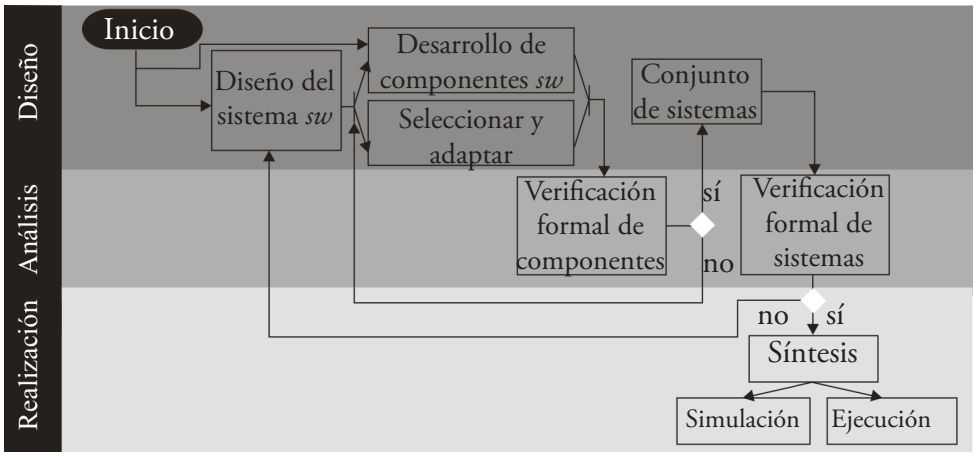

Figura 2.9. El proceso de desarrollo SaveCCT [Sentilles *et al.*, 2009]

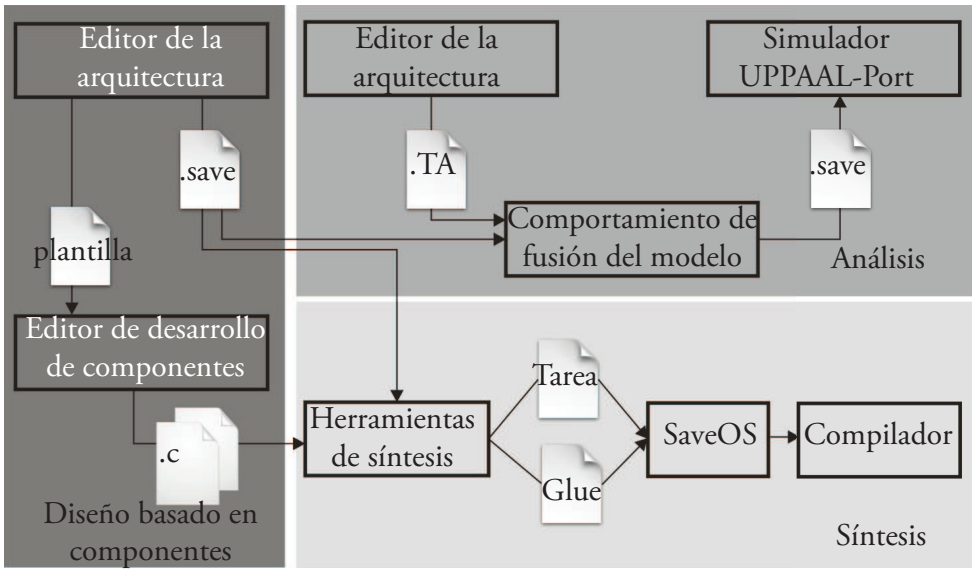

Figura 2.10. Vista general de la estructura de Save-IDE [Sentilles *et al.*, 2009]

2.5.3. *RESOURCE MODEL FOR EMBEDDED SYSTEMS (REMES)*

La importancia del conocimiento sobre los recursos en los SE está creciendo rápidamente pues la limitada disponibilidad de éstos representa la principal limitante de la introducción de nuevas características del producto y aplicaciones. Así, el análisis sistemático del consumo de recursos en un SE debe incluir los caminos de representación semántica de varios tipos de recursos, ya sean de tipo continuo (como la energía), o de naturaleza discreta (la memoria, puertos de I/O).

Una meta del análisis representativo es verificar la propiedad de viabilidad de los recursos. De acuerdo a [Seceleanu, Vulgarakis & Petterson, 2009], es posible afirmar que la composición de las necesidades de recursos para los componentes, se mantiene dentro de los disponibles proporcionados por la plataforma de ejecución, o que existe una ruta de ejecución que utiliza no más de los recursos disponibles para comportarse correctamente. En la práctica, a menudo puede ser necesario sustituir un componente por otro que tiene la misma funcionalidad. Es altamente deseable que el análisis del uso de los recursos aparezca en una fase al inicio del diseño. Esto permite la realización de un número potencialmente elevado de experimentos de diseño, sin aumentar los costos. Además, el análisis de los recursos puede orientar a los diseñadores en la toma de decisiones, como la selección de los componentes adecuados de un repositorio, o elegir entre varios modelos admisibles de diseño. REMES es un marco de modelado y análisis de aplicaciones que utiliza técnicas para realizar el análisis cuantitativo de la viabilidad y el análisis de recursos optimo-peor. REMES está diseñado específicamente para SE, pero generalmente es adecuado también para sistemas reactivos. El modelo da soporte a los recursos abstractos discretos y continuos; así, los recursos genéricos pueden ser modelados de este modo, incluyendo la memoria, puertos, energía, CPU, y buses.

2.5.3.1. OBJETIVO

El modelo REMES para SE introduce el modelado y análisis formales de los recursos empotrados tales como el almacenamiento, la energía, la comunicación y el cómputo. Este modelo es un lenguaje de comportamiento basado en una máquina de estados con soporte para el modelado jerárquico, anotaciones de recursos, tiempo continuo, y nociones de puntos explícitos de entrada y salida que resultan convenientes para el modelado de SE basado en componentes. El objetivo principal de REMES es reducir la brecha entre el modelado de arquitectura (por ejemplo, los Lenguajes para la Descripción de la Arquitectura (ADL)) y los modelos formales de análisis (por ejemplo, la programación de autómatas).

2.5.3.2. ESTRUCTURA

REMES está enfocado a describir el comportamiento racional de los recursos que interactúan entre los componentes empotrados. REMES depende en gran medida del lenguaje de modelado CHARON, utilizado para especificar la comunicación de los agentes en los SE. REMES agrega información sobre el consumo de los recursos y su tasa, así como otras construcciones que facilitan su aplicación al modelado funcional y extra-funcional de la conducta (en tiempo real) y los sistemas basados en componentes. La Figura 2.11 muestra que en REMES el comportamiento interno de un componente es descrito a través de un modo.

Se denomina 'modo atómico' si éste no contiene a su vez algún submodo, y 'compuesto' si contiene a su vez submodos. De manera similar que en CHARON, los datos son transferidos entre los modos a través de una interfaz de datos bien definida, es decir, variables de tipo global, mientras que el control (discreto) se realiza a través de una interfaz de control bien definida que consiste de puntos de entrada y salida.

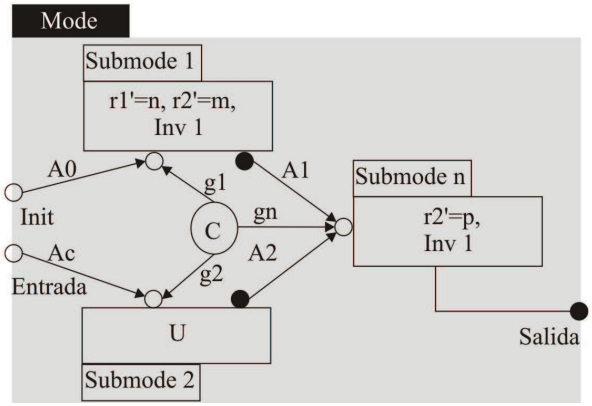

Figura 2.11. Modo compuesto de REMES [Seceleanu, Vulgarakis & Petterson, 2009]

2.5.4. *RAPID OBJECT-ORIENTED PROCESS FOR EMBEDDED SYSTEMS (ROPES)*

ROPES (*Rapid Object-Oriented Process for Embedded Systems*) se propone como un proceso completo para el desarrollo de SE. La metodología en que se enmarca ROPES, emplea directamente UML y añade un proceso de desarrollo que se define como iterativo (o en espiral) y que está basado en modelos [Powel, 2004]. ROPES está compuesto de diversas tendencias de la Ingeniería de Software, tales como el análisis de riesgos y la calidad del software.

2.5.4.1. OBJETIVO

ROPES está diseñado para emplearse en cualquier tipo de sistemas pero es especialmente útil para el desarrollo de SE pues hace énfasis en las características propias de las aplicaciones empotradas y de tiempo real. El objetivo de ROPES es producir aplicaciones software con un mínimo esfuerzo, sin fallos y con la máxima 'pronosticabilidad' en su desarrollo; por lo cual se centra en mejorar y ordenar el proceso haciendo una rápida realimentación, verificación temprana, reducción del riesgo, y mejora en la previsibilidad del proyecto en términos de esfuerzo y tiempo de programación.

2.5.4.2. ESTRUCTURA

El proceso se organiza en cuatro grandes fases: análisis, diseño, traducción y pruebas, siendo los artefactos resultantes de cada fase, modelos o diagramas UML que se refinan o corrigen en las fases siguientes. A continuación se resumen brevemente las cuatro fases de ROPES:

1. Análisis: esta fase cumple principalmente con dos propósitos: capturar y refinar los casos de uso, y asignar un conjunto de objetos que suministren el comportamiento del sistema. La fase se compone de:

- **Análisis de requisitos:** consiste en la especificación de caja negra de los requisitos funcionales y no funcionales del sistema.
- **Análisis del sistema:** determina la división de responsabilidades entre el hardware y el software, la arquitectura de alto nivel del sistema y los algoritmos de control necesarios.
- **Análisis de objetos:** se divide en dos sub-fases, en una se determinan los modelos estructurales de los objetos que han de componer el sistema y en la otra se modela su comportamiento mediante colaboraciones o diagramas de secuencia.

2. Diseño: optimiza al modelo de análisis, además de que se llevan a cabo los diseños de la arquitectura, de los mecanismos y del esquema detallado; el diseño define una única solución particular, que es óptima en algún sentido. La fase se compone de:

- **Diseño de arquitectura:** apunta a las decisiones estratégicas del diseño que afectan a la aplicación en su conjunto, tales como el modelo de despliegue, recursos de tiempo de ejecución y la concurrencia.
- **Diseño de mecanismos:** optimiza el comportamiento de las colaboraciones.
- **Diseño detallado:** optimiza el sistema de cara a su implementación.

3. Traducción: comprende la generación de código ejecutable a partir del diseño del sistema, implica tanto el código de la aplicación como el necesario para la verificación independiente de cada uno de sus objetos.

4. Pruebas: se trata de verificar la conformidad de la aplicación, ya sea para encontrar defectos o para observar un cierto nivel funcional. Incluye pruebas de integración y de validación.

Las cuatro fases antes descritas se encadenan de manera cíclica hasta lograr el nivel de aceptación y calidad deseadas. ROPES define los diagramas UML que deben desarrollarse en cada fase y que serán el resultado de esta, así como la información inicial de la siguiente. Así, de cada fase se obtiene como resultado artefactos (diagramas UML) que se refinan o corrigen en las siguientes. El proceso ROPES utiliza tres diferentes escalas de tiempo simultáneas [Powel, 1999]:

- El macrociclo (véase Figura 2.12) que hace referencia al desarrollo completo del proceso, desde que se inicia su especificación hasta que se entrega el producto. Dentro de él se desarrollan paralelamente las cuatro fases que estructuran a ROPES. Su objetivo es dirigir la evolución de los prototipos que se programan, de forma que los problemas sobre conceptos claves, requisitos, arquitectura y tecnología sean resueltos sucesivamente.

Figura 2.12. Iteración de cada microciclo en el proceso ROPES [Powel, 2004]

- El microciclo que tiene un ámbito más reducido de tiempo que el macrociclo y su objetivo es el desarrollo de un nuevo prototipo con una funcionalidad limitada que resuelva actividades consideradas de riesgo en el proceso de desarrollo.
- El nanociclo que tiene el ámbito más reducido y cuyo objetivo es modelar, ejecutar o establecer ideas que se necesitan. Su duración no debe exceder los dos días. La razón por la que se introduce esta fase es la conveniencia de probar continuamente y estar seguro de todas las ideas que se incorporan al proyecto.

Como herramienta de soporte para la elaboración y gestión de los artefactos, ROPES propone el uso de IBM® Rational ® Raphsody , en parte debido a que con esta herramienta se automatiza la generación del código.

2.5.5. *MODEL-DRIVEN DESIGN OF EMBEDDED SYSTEMS (ModES)*

Los enfoques basados en MDE[2] (*Model Driven Engineering*) se han propuesto como una metodología eficiente para el diseño de SE. Uno de los enfoques de la MDE es MDA (*Model Driven Architecture*), que es un marco propuesto por la OMG[3] para el desarrollo de software apoyado por modelos en diferentes niveles de abstracción. En un enfoque MDA, un sistema es modelado usando un Modelo Independiente de Plataforma (PIM), que se transforma en un Modelo Específico de la Plataforma (PSM), considerando un Modelo de Plataforma (PM). Los lenguajes utilizados para expresar estos modelos se definen por medio de meta-modelos que son capaces de definir la sintaxis abstracta y concreta, así como la semántica operacional de los lenguajes de modelado. MDE permite a los desarrolladores construir soluciones usando abstracciones, tales como las que proporcionan los lenguajes visuales adaptados al dominio de la solución.

En MDE, no se considera únicamente la generación de código como objetivo sino que además se incluyen operaciones tales como la transformación de artefactos software, su integración o composición, la interoperabilidad, la reutilización y la trazabilidad. El paradigma MDE tiene dos ejes principales: por un lado hace énfasis en la separación entre la especificación de la funcionalidad esencial del sistema, y por otro se enfoca en la implementación de dicha funcionalidad usando plataformas tecnológicas específicas. Existen muchos esfuerzos recientes de investigación sobre el diseño de SE basados en MDE. A través del enfoque MDE el diseñador puede especificar el sistema con un alto nivel de abstracción, buscando la mejor alternativa para desarrollarlo sin depender de costosos ciclos de evaluación de síntesis y de simulación, y generar el código fuente y scripts para automatizar y facilitar la transición de las fases de diseño inicial hasta la fase de ejecución.

2.5.5.1. OBJETIVO

ModES [Riccobene *et al.*, 2006]es una metodología y un conjunto de herramientas que aplican el enfoque MDE para el diseño de SE.

2 MDE es una metodología de desarrollo de software que se centra en la creación de modelos o abstracciones. Esta metodología tiene el propósito de aumentar la productividad mediante la maximización de la compatibilidad entre sistemas, simplificando el proceso de diseño, y promoviendo la comunicación entre los individuos y equipos que trabajan en el sistema.

3 Object Management Group o Grupo de Gestión de Objetos, es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA.

La metodología incluye una infraestructura de meta-modelos y herramientas para apoyar las tareas de diseño basadas en modelos tales como la exploración del espacio de diseño, la estimación, la generación de código, y la síntesis de hardware.

2.5.5.2. ESTRUCTURA

ModES se basa en un metamodelo que puede ser utilizado por varias tareas durante el diseño de SE. Desde la especificación hasta la generación y la síntesis del software/hardware, las herramientas de diseño pueden tener acceso a la infraestructura del metamodelo para generar los artefactos específicos del diseño, que servirán como insumos para las tareas del diseño. Los artefactos resultantes pueden ser almacenados en el repositorio de metadatos para continuar con el flujo de diseño. La Figura 2.13 muestra que ModES está compuesto por cuatro metamodelos internos cuyo objetivo es permitir su aplicación independiente y el modelado de la plataforma. La aplicación interna del metamodelo consiste en obtener un modelo uniforme con una sintaxis bien definida y una semántica operacional. Mientras que la plataforma interna del metamodelo corresponde a una taxonomía de los elementos de hardware y software con las propiedades de la caracterización. El mapeo del metamodelo describe las reglas utilizadas para transformar los casos de "aplicación interna del metamodelo" y la "plataforma interna del metamodelo" en una instancia de la aplicación del metamodelo y define cómo las funciones de aplicación se dividen entre los componentes arquitectónicos. El enfoque MDE se utiliza para generar las representaciones internas realizando transformaciones a base de modelos y apoyándose en QVT (*Query*, Vistas, Transformaciones). En la versión actual, se utilizan las características de *Eclipse Modeling Framework* (EMF) para implementar la infraestructura de metamodelado.

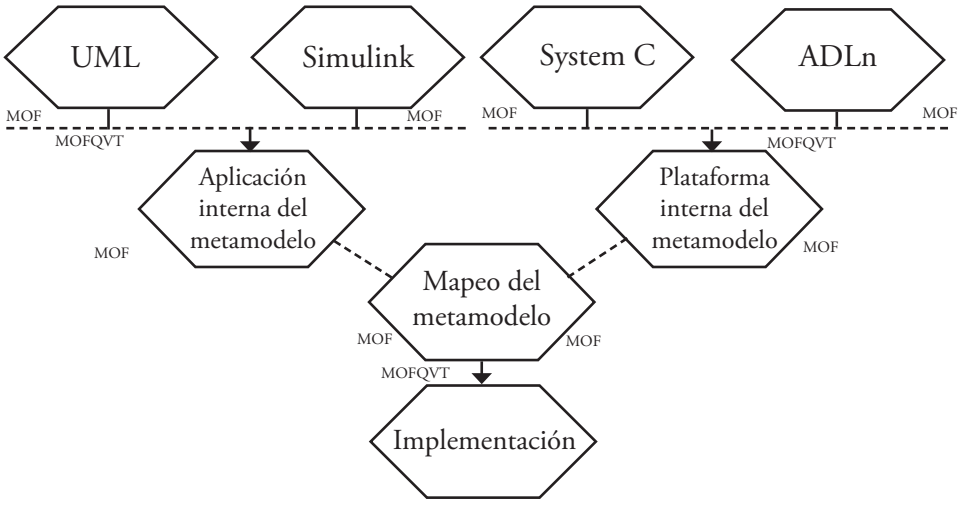

Figura 2.13. Estructura del meta-modelo [Riccobene *et al.*, 2006]

ModES utiliza un repositorio para almacenar los componentes hardware/software disponibles, la metainformación sobre ellos, así como los artefactos resultantes de aplicar el metamodelo. La metainformación puede corresponder a las propiedades físicas, tales como el rendimiento o el consumo de energía, las dependencias lógicas entre los componentes, o la información de versiones. Esta información será utilizada para apoyar las tareas de diseño, como la estimación, simulación y diseño.

2.5.6. *SIMPLIFIED PARALLEL PROCESSES (SPP)*

El desarrollo de SE, al igual que el desarrollo de software, presenta múltiples y variados conflictos, y muchas empresas ignoran una posible solución con la Mejora del Proceso Software (*Software Process Improvement*, SPI). Modelos como CMM/CMMI establecen las pautas para desarrollar software de calidad asumiendo que las empresas poseen recursos y tiempo para mejorar el proceso de software. Sin embargo puede necesitarse de una gran cantidad de recursos si cada área de proceso clave de CMM/CMMI ha sido ignorada. CMMI, describe en detalle la administración, mientras que los procesos de desarrollo tales como el desarrollo de los requisitos, el diseño, la codificación, las pruebas y el mantenimiento son sólo introducidos en un proceso llamado "Ingeniería de Producto" que sólo indica "qué" debe mejorarse sin indicar el "cómo" realizar las actividades de mejora de los procesos software, además de ser lineamientos demasiado generales como para ser adecuados para diferentes tipos de empresas. Es por esto que en el trabajo realizado por [Jun, Rui & Yi-min, 2007] se adapta y se presenta un método practico para orientar el desarrollo de SE bajo el enfoque de SPI, relacionado con el modelo CMMI.

2.5.6.1. OBJETIVO

El objetivo de *Simplified Parallel Process* (SPP) consiste en proporcionar una solución orientada a SPI y relacionada con el modelo CMMI en sus Niveles 2 y 3, y enfocada a empresas medianas. Esta investigación intenta proveer especificaciones de desarrollo técnico para los SE.

2.5.6.2. ESTRUCTURA

El modelo SPP consta de tres capas: el nivel superior consiste en los procesos para la gestión del proyecto, la capa intermedia está integrada por los procesos para el desarrollo del proyecto, y el nivel inferior sirve de apoyo a los procesos del proyecto. De acuerdo a los autores, SPP tiene las siguientes ventajas:

- La gestión de proyecto, el desarrollo y el apoyo son progresos dependientes y relacionados. Cada elemento en la organización sabe cuando hacer lo que debería hacer según la especificación.
- Se añade un inicio y terminación de la gestión de proyecto y los procesos de desarrollo son mejorados y detallados.
- SPP puede ser adaptada fácilmente a cualquier tipo de empresa.

SPP especifica los procesos para el desarrollo de SE y los incluye en forma de:

- Especificaciones técnicas de desarrollo para definir las etapas de los procesos, de las actividades y subactividades que conforman SPP.
- Directrices que indican cómo se debe realizar cada fase y las especificaciones pertinentes.
- Plantillas para ilustrar los formatos de los distintos documentos e indicar la forma de cómo realizarlas.

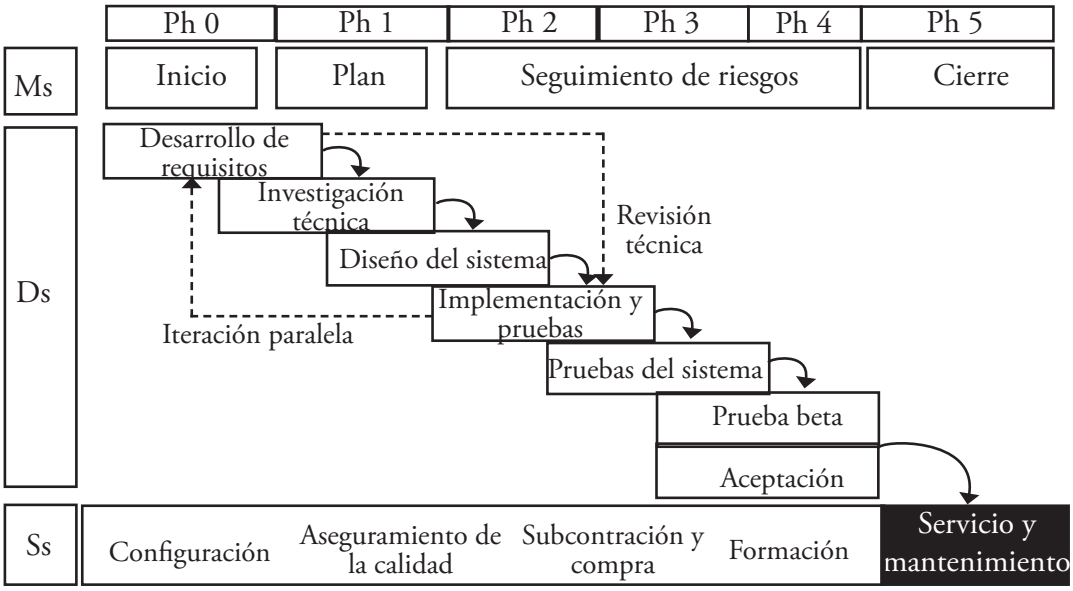

Figura 2.14. Modelo SPP [Jun, Rui & Yi-min, 2007]

La Figura 2.14 muestra la relación entre las áreas de procesos clave y el ciclo de vida del producto. Con el modelo SPP, el ciclo de vida del producto se divide en seis fases (véase Figura 2.14). Desde Ph0 a Ph5 hay 16 áreas de proceso clave, más de 40 procedimientos y 60 plantillas para documentar. SPP además aborda los procesos relacionados con el desarrollo de hardware.

2.5.7. *PRODUCT FOCUSED SOFTWARE PROCESS IMPROVEMENT*

Product Focused Software Process Improvement es el resultado de un trabajo de tesis doctoral de la *Eindhoven University of Technology, Netherlands.* La tesis se enfoca principalmente en la aplicación de SPI en el dominio del software empotrado. En el trabajo se presenta entre otras cosas un estudio sobre: los productos empotrados y su calidad, el proceso de software empotrado y la calidad de producto, y las fortalezas y debilidades de las metodologías de SPI para software empotrado. La investigación tuvo una duración de cuatro años, tiempo en el que se realizaron diferentes casos de estudio [Solingen, 2002].

2.5.7.1. OBJETIVO

El objetivo de la tesis doctoral es el desarrollo del modelo conceptual RMP centrado en la Mejora del Proceso Software para el desarrollo de productos empotrados. Así como también la descripción de un conjunto de directrices para el uso práctico de este modelo.

2.5.7.2. ESTRUCTURA

Product Focused Software Process Improvement se basa en un modelo conceptual llamado RPM. El modelo es el resultado de la combinación de tres áreas de trabajo:

- La *Ingeniería de Requisitos* es el proceso es el proceso en el que se describe lo que un producto debe hacer. El objetivo del proceso es darle a todas las partes una explicación escrita del problema. Por lo tanto la Ingeniería de Requisitos trata los principios, métodos, técnicas y herramientas que permiten obtener, documentar y mantener los requisitos. La importancia de la Ingeniería de Requisitos en *Product Focused Software Process Improvement* se debe al impacto que tienen los requisitos en la calidad del producto. La tesis hace uso de una definición práctica de Ingeniería de Requisitos que indica: "*es el proceso que recoger los deseos de todos los interesados en el producto y los transforma en una especificación de la calidad del producto, constante, inequívoca, y medible*". La Ingeniería de Requisitos proporciona una especificación de la calidad del producto y abarca los aspectos de: requisitos de calidad, percepción de la calidad que tienen los interesados o usuarios, y medición de la calidad. En la Figura 2.15 se presenta el proceso definido para Ingeniería de Requisitos.

- La *Ingeniería del Proceso* es la disciplina de ingeniería para diseñar, construir y adaptar los métodos, técnicas y herramientas para el desarrollo de un producto de software específico. Su importancia radica en el hecho de que el proceso de desarrollo más adecuado para un proyecto específico tiene que ser diseñado para su situación específica. Dentro de la tesis, Ingeniería del Proceso, "*es el diseño de un proceso medible para el desarrollo de un producto específico que obedece a la especificación de calidad de producto. Un proceso de desarrollo contiene un conjunto de acciones de proceso con efectos explícitos esperados sobre la calidad de producto*". En la Figura 2.16 se presenta el proceso definido para Ingeniería del Proceso.

- El *Programa de Medición* consiste en el diseño e implementación de un conjunto de procesos, productos y métricas de recursos, para alcanzar los objetivos predefinidos dentro de una organización. La importancia radica en el hecho de que las mediciones son objetivas y por lo tanto puede ser utilizado como un mecanismo de retroalimentación y evaluación. En *Product Focused Software Process Improvement* se define de marea práctica como "*el diseño e implementación de un conjunto de procesos, productos, recursos y métricas, para evaluar la calidad del producto y evaluar la relación producto-proceso*". En la Figura 2.17 se presenta el proceso definido para el Programa de Medición.

A partir de los casos de estudio realizados en [Solingen, 2002], se demostró que el modelo conceptual RPM es de gran beneficio para los proyectos de desarrollo industrial. El uso de este modelo en la práctica está totalmente integrado a los proyectos de desarrollo y a la mejora de los procesos, haciendo frente a los objetivos de calidad de los productos. El modelo se apoya de tres áreas de trabajo y contiene tres productos de trabajo que se relacionan directamente, tal y como se muestra en la Figura 2.18:

- **Especificación de la calidad del producto:** son los requisitos de calidad del producto debidamente documentados, especificados de un modo completo, constante, inequívoco, y medible.
- **Modelo de proceso de desarrollo:** es el modelo específico del proyecto que se integra por los pasos necesarios para realizar un producto empotrado.

- **Mediciones de productos y procesos:** son los datos y análisis realizado por el equipo de desarrollo en cuanto a la conformidad de las especificaciones de la calidad del producto, o en cuanto a la eficacia de acciones de proceso específicas.

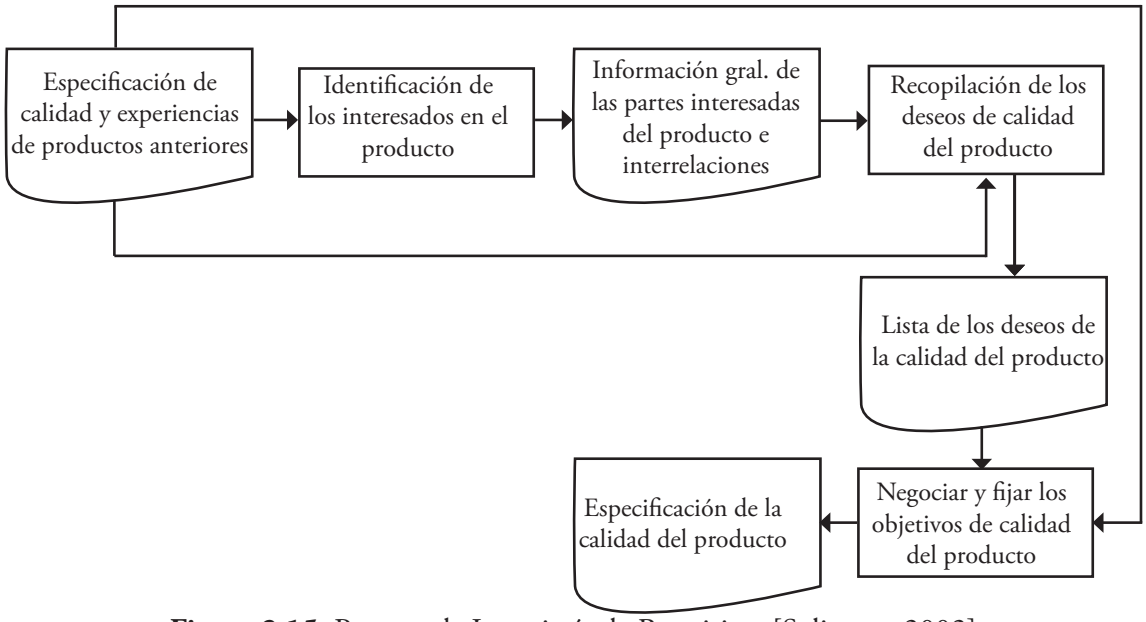

Figura 2.15. Proceso de Ingeniería de Requisitos [Solingen, 2002]

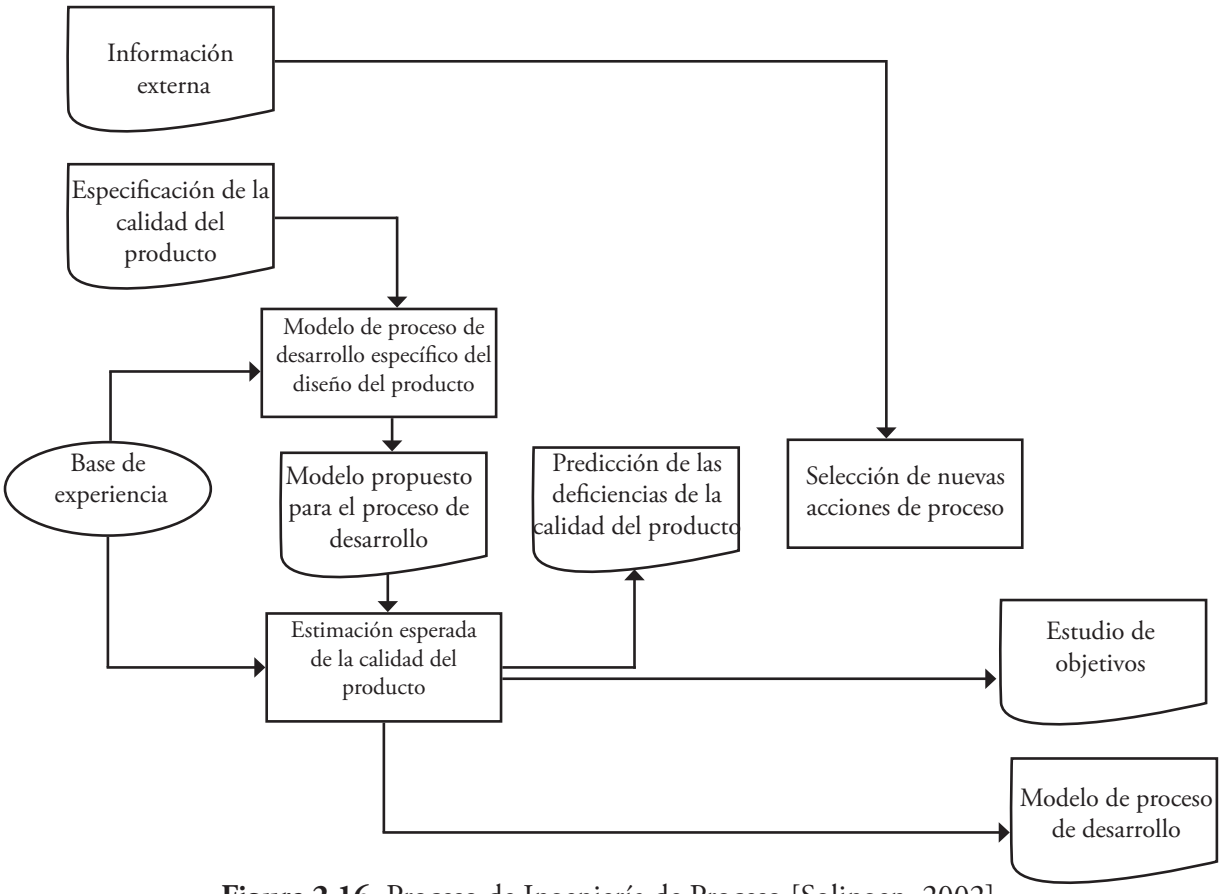

Figura 2.16. Proceso de Ingeniería de Proceso [Solingen, 2002]

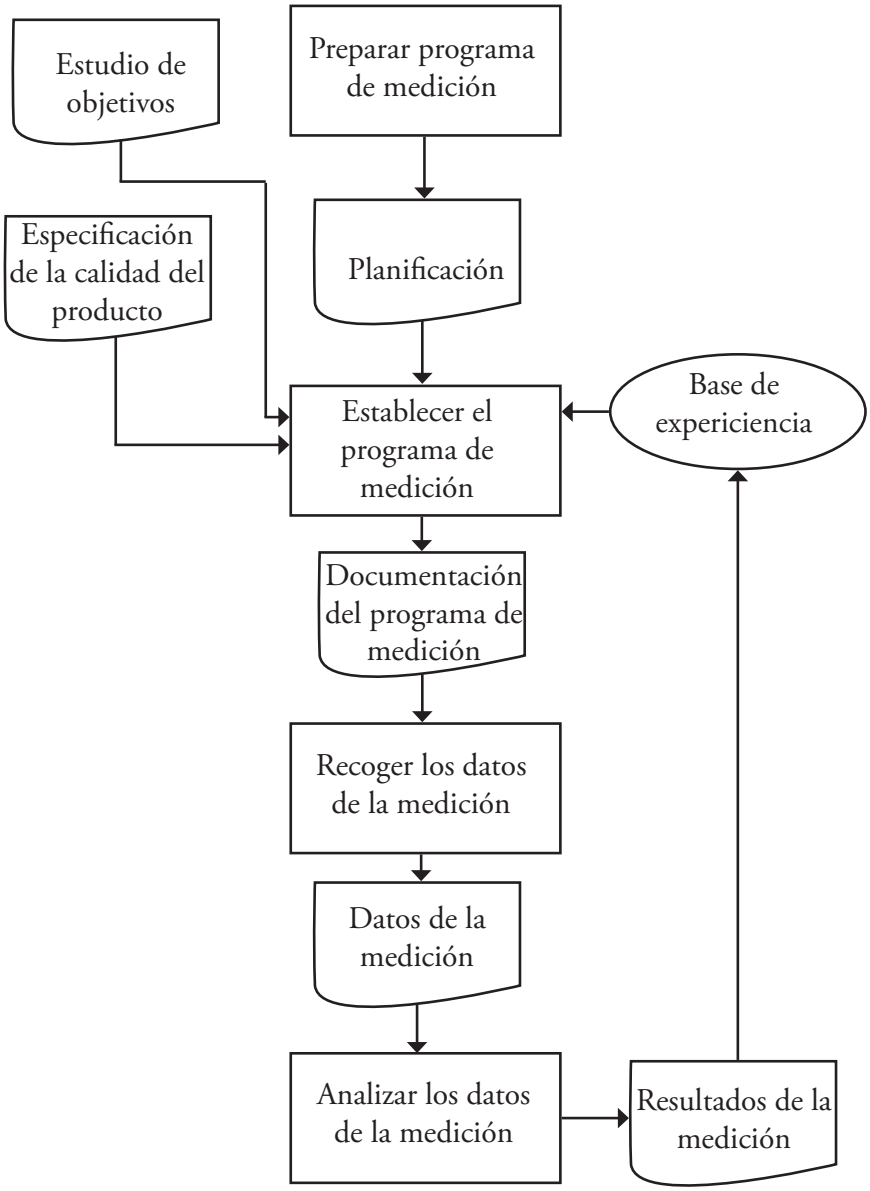

Figura 2.17. Proceso del Programa de Medición [Solingen, 2002]

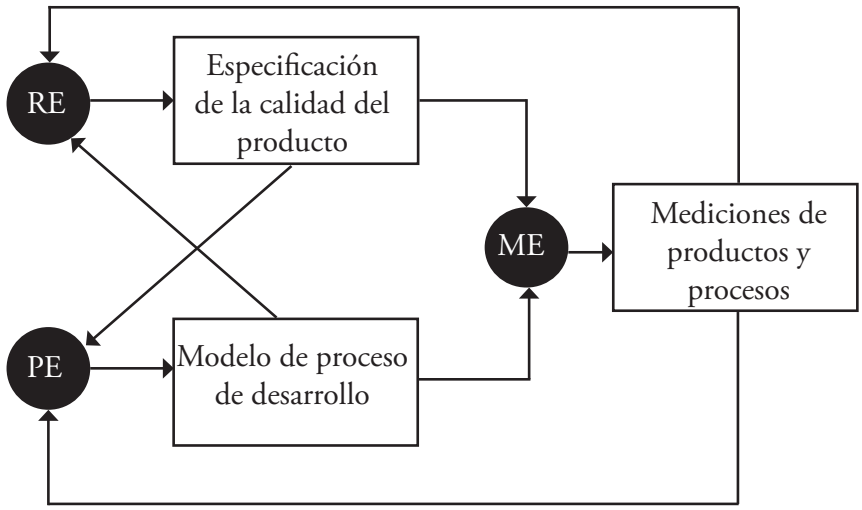

Figura 2.18. Modelo RPM [Solingen, 2002]

2.6 PRINCIPALES HALLAZGOS SOBRE LAS PROPUESTAS PARA EL DESARROLLO DE SISTEMAS EMPOTRADOS

Las metodologías analizadas son producto del esfuerzo de la industria e instituciones académicas, cuyo objetivo es incorporar formalidad al "proceso" de diseño de los SE.

Con el objetivo de identificar características comunes entre las metodologías actuales, se formuló un conjunto de criterios comparativos que a continuación son descritos de manera breve:

1. **Ámbito de aplicación:** este criterio se estableció para delimitar el uso de las metodologías, pues aunque se seleccionaron metodologías con características propias para SE, algunas de las metodologías son diseñadas y/o aplicadas a sistemas en general, sistemas empotrados, o software. Con este criterio se identificarán aquellas metodologías propias de los SE.

2. **Basado en:** de acuerdo al estudio exploratorio realizado, las metodologías actuales proporcionan soluciones orientadas principalmente al uso de Componentes (C), Herramientas (H) o a la Mejora del Proceso Software (SPI); todas con el fin de mejorar el proceso de desarrollo. Mediante este criterio se identifica la orientación hacia el objetivo de la metodología producto de esta tesis.

3. **Puntos a favor:** especifica la mejora real y/o beneficios que ofrece la metodología respecto al diseño de los SE.

4. **Puntos en contra:** especifica los inconvenientes de la metodología en el dominio de los SE.

La Tabla 2.3 resume los principales hallazgos sobre el análisis de cada herramienta.

Tabla 2.3. Tabla comparativa de las metodologías actuales para el desarrollo de SE

Metodología	Ámbito SG | SE | SW	Basado en C | H | SPI	Puntos a favor	Puntos en contra
ESCM	—	—	Reutilización, contratos a nivel interfaz, atención en los atributos no funcionales. Número de fases: 4.	Se enfoca únicamente en el desarrollo de software empotrado y no contempla el desarrollo de hardware.
SAVE-IDE	—	—	Reutilización, fases de desarrollo, herramientas asociadas. Número de fases: 4.	Falta de modelado del comportamiento de los componentes internos y externos, así como de los recursos del SE.
REMES	—	—	Modelado, gestión de recursos, lenguaje CHARON. Modelo formal.	Depende mucho de los recursos obtenidos para formular el diseño. No incorpora fases.
ROPES	—	—	Realimentación, verificación temprana, busca del mínimo esfuerzo para el desarrollo, reducción de fallos. Número de fases: 4.	No propone una estrategia que permita integrar la planificación en el proceso de desarrollo.
ModES	—	—	Abarca las áreas de especificación hasta la generación y la síntesis de software/hardware de SE.	Orientada únicamente al software empotrado y existe poca información. No incorpora fases.
SPP	—	—	Se integra por Especificaciones técnica, directrices y plantillas. Núm. de fases: 6.	Orientada únicamente al software empotrado.
Focused Software Process Improvement	—	—	Especificación de la calidad del producto. Establece directrices para las tres áreas de trabajo bajo las que está diseñada	No cubre todos los aspectos del desarrollo de un sistema empotrado. No incorpora fases.