405ssedm

Software para Sistemas Empotrados y Dispositivos Móviles(嵌入式和移动设备软件)

Actividad#1

Tema 1: "Introducción a los Sistemas Empotrados y Dispositivos Móviles"

Conferencia 1: "Introducción a los Software para Sistemas Empotrados. Conceptos básicos"

Introducción

> Los sistemas embebidos (o empotrados) son sistemas computacionales diseñados para realizar una o varias funciones específicas dentro de un sistema más grande, sin ser percibidos como un ordenador convencional. A diferencia de los sistemas de propósito general, combinan hardware y software de forma que el algoritmo que define su comportamiento suele ser fijo y no modificable por el usuario. Su arquitectura típica es heterogénea, integrando componentes como microcontroladores, procesadores de señal digital (DSP) y circuitos integrados de aplicación específica (ASIC), lo que implica una decisión crítica de diseño sobre qué funcionalidades se implementarán por hardware y cuáles por software.
>
> Estos sistemas se caracterizan por su concurrencia, fiabilidad, interacción con dispositivos físicos a través de interfaces no convencionales, robustez para operar en entornos hostiles, bajo consumo y reducido coste y tamaño. Su aplicación es omnipresente en la vida cotidiana, abarcando desde electrodomésticos y sistemas de entretenimiento hasta dispositivos de comunicación, automoción y control industrial. Esto los sitúa como la forma de computación más numerosa y extendida, a menudo operando autonómicamente y sin intervención humana directa.

Objetivo

> Caracterizar los principales elementos, funciones y aplicaciones de los sistemas empotrados.

Parte 1: Sistemas empotrados o embebidos

¿Qué son los sistemas empotrados o embebidos (SE)?

> Un sistema empotrado es **un sistema que usa un computador para realizar una función específica**, pero ni es usado ni es percibido como un computador. Los distinguen las siguientes características:
>
> - Tienen una funcionalidad de hardware y/o software más limitada que una computadora personal (PC). En términos de limitaciones de hardware, esto puede significar limitaciones en el rendimiento del procesamiento, el consumo de energía, la memoria, la funcionalidad del hardware, etc. En el caso del software, esto generalmente significa limitaciones tales como: menos aplicaciones,

aplicaciones de menor escala, ausencia de sistema operativo (SO) o un SO limitado, o código con menor nivel de abstracción.
- Está diseñado para realizar una función dedicada. La mayoría de los dispositivos embebidos están diseñados principalmente para una función específica. Sin embargo, ahora vemos dispositivos como los híbridos de asistentes personales de datos (PDA) y teléfonos celulares, que son sistemas embebidos diseñados para realizar diversas funciones principales.
- Es un sistema de cómputo con mayores requisitos de calidad y fiabilidad que otros tipos de sistemas informáticos. Algunas familias de dispositivos integrados tienen un umbral muy alto de requisitos de calidad y fiabilidad.

Componentes de los sistemas embebidos

La CPU (Unidad Central de Procesamiento) se encuentra en el centro de estos sistemas, actuando como el cerebro que coordina todas las actividades. La CPU se comunica con varios componentes esenciales:

- **Sensores**: Recogen datos del entorno externo y los envían a la CPU a través de una conversión A/D (analógica a digital).
- **Actuadores**: Ejecutan acciones basadas en las instrucciones de la CPU, recibiendo señales a través de una conversión D/A (digital a analógica).
- **Memoria**: Almacena tanto el software como los datos necesarios para el funcionamiento del sistema.
- **Software**: Proporciona las instrucciones necesarias para que la CPU realice sus tareas.
- **Sistema Auxiliar**: Incluye componentes como la potencia de refrigeración que aseguran el funcionamiento adecuado del sistema.
- **Interfaz de Usuario**: Permite la interacción entre el usuario y el sistema embebido.

Arquitectura común de un SE

Por lo general los sistemas embebidos tienen la estrutura que se muestra en la siguiente figura:

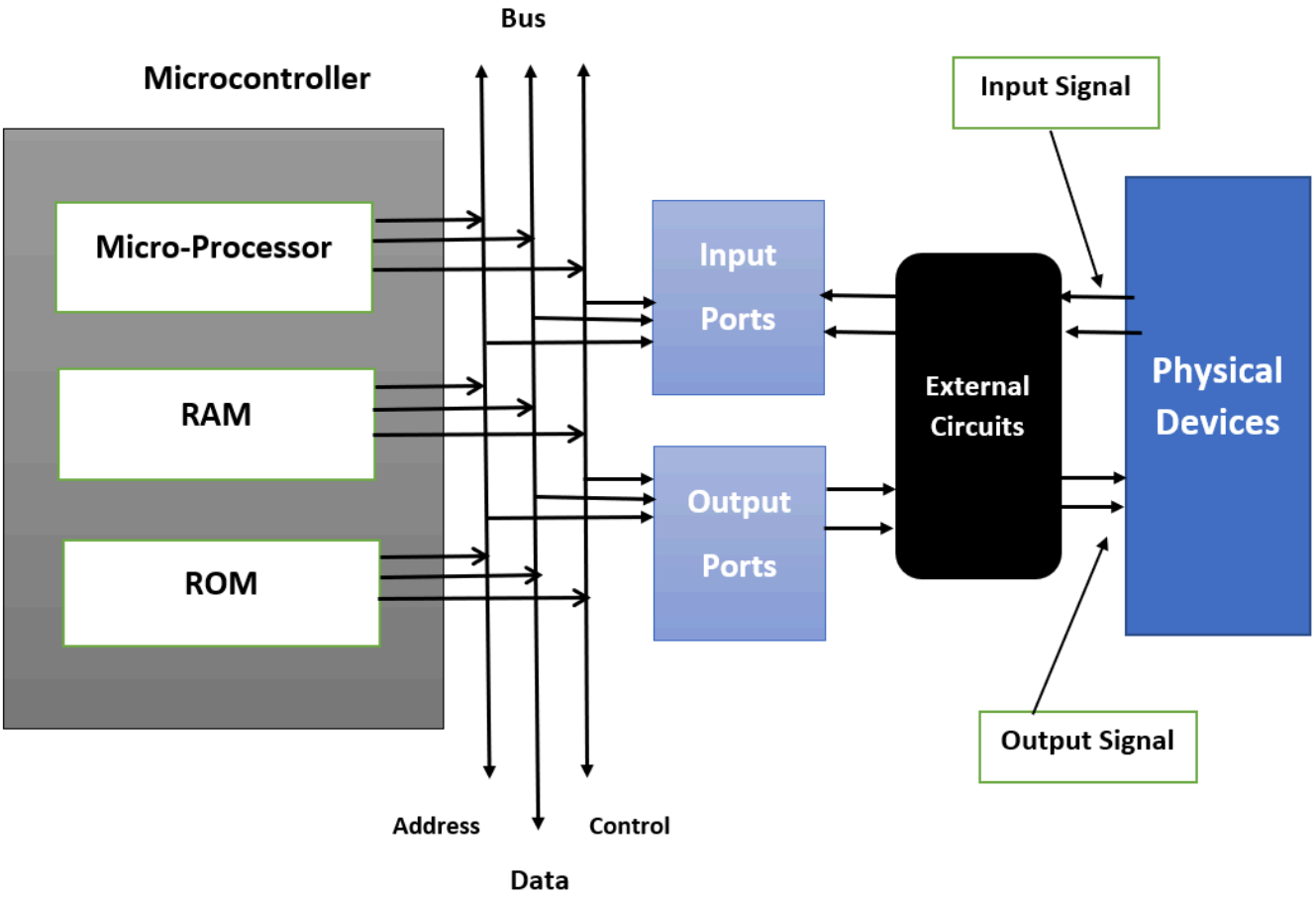

Figura 1. Arquitectura común de un SE en cuanto a componentes

Donde:

1. Se centra en el **microcontrolador** (Microcontroller), que generalmente incluye:
    - **Micro-processor**: Realiza las operaciones de cálculo y dirige el funcionamiento de los demás componentes.
    - **RAM**: Almacena datos temporales.
    - **ROM**: Contiene el firmware necesario para el funcionamiento del sistema.
2. Los **puertos de entrada y salida** (Input Ports y Output Ports) permiten la comunicación del microcontrolador con dispositivos externos a través de circuitos externos (External Circuits). Estos puertos reciben señales de entrada de dispositivos físicos, las procesan y generan señales de salida para controlar dichos dispositivos.

Por qué los sistemas empotrados son diferentes

- Los sistemas embebidos se dedican a tareas específicas, mientras que las PC son plataformas informáticas genéricas.
- Los sistemas embebidos son compatibles con una amplia gama de procesadores y arquitecturas de procesador.
- Los sistemas embebidos suelen ser sensibles a los costes.
- Los sistemas embebidos tienen limitaciones de tiempo real.

- Si un sistema embebido utiliza un sistema operativo, lo más probable es que utilice un sistema operativo en tiempo real (RTOS), en lugar de Windows 9X, Windows NT, Windows 2000, Unix, Solaris o HP-UX.
- Las consecuencias de un fallo de software son mucho más graves en los sistemas embebidos que en los sistemas de escritorio.
- Los sistemas embebidos suelen tener limitaciones de energía.
- Los sistemas embebidos a menudo deben operar en condiciones ambientales extremas.
- Los sistemas embebidos tienen muchos menos recursos del sistema que los sistemas de escritorio.
- Los sistemas embebidos suelen almacenar todo su código objeto en ROM.
- Los sistemas embebidos requieren herramientas y métodos especializados para un diseño eficiente.
- Los microprocesadores integrados a menudo tienen circuitos de depuración dedicados.

En la trabla 1 se realiza una comparación resumida entre los sistemas embebidos y las PC (Personal Computer).

Tabla 1 Distinción entre diseño tipo PC y diseño de sistemas embebidos. Tomado de [1]

	Embebido	**Tipo PC/Servidor**
Arquitecturas	Frecuentemente heterogéneas, muy compactas	Mayormente homogéneas, no compactas (x86, etc.)
Compatibilidad ×86	Menos relevante	Muy relevante
¿Arquitectura fija?	A veces no	Sí
Modelo de computación (MoCs)	C + múltiples modelos (flujo de datos, eventos discretos, ...)	Mayormente von Neumann (C, C++, Java)
Objetivos de optimización	Múltiples (energía, tamaño, ...)	Predomina el rendimiento promedio
¿La seguridad es crítica?	Posiblemente	Usualmente no
Relevancia en tiempo real	Frecuente	Casi nunca
Aplicaciones	Se requieren garantías para varias aplicaciones concurrentes	Enfoques de mejor esfuerzo para ejecutar aplicaciones
Aplicaciones conocidas en el diseño	Sí, para sistemas en tiempo real	Solo algunas (por ejemplo, WORD)

Aspectos clave en la gestión de recursos en Sistemas Embebidos

La gestión de recursos es un aspecto crítico en el diseño de sistemas embebidos debido a las limitaciones inherentes de estos sistemas y a los múltiples objetivos de diseño que deben equilibrarse. A continuación, se resumen los principales aspectos a considerar:

1. Energía

> Los sistemas embebidos convierten energía eléctrica en otras formas (generalmente térmica), por lo que la eficiencia energética es crucial.
> Razones para ser conscientes del consumo de energía:
>
> - Calentamiento global: Relevante para sistemas conectados a la red.
> - Costo de la energía: Importante cuando el suministro es costoso.
> - Rendimiento: Mayor rendimiento generalmente implica mayor consumo.
> - Refrigeración y fiabilidad: Temperaturas altas reducen la vida útil y confiabilidad.
> - Escasez de energía: Crítico en sistemas autónomos o con batería limitada.

2. Tiempo de Ejecución (Run-time)

> Se debe optimizar el uso del hardware para evitar ciclos de procesador desperdiciados.
> La eficiencia en el tiempo de ejecución debe considerarse en todos los niveles: algoritmos, software y hardware.

3. Tamaño del Código

> Muchos sistemas embebidos tienen memoria limitada (ej.: sistemas implantables o SoCs).
> El código debe ser compacto, aunque esto puede cambiar con nuevas tecnologías de memoria o carga dinámica.

4. Peso

> Los sistemas portátiles deben ser livianos para ser prácticos y competitivos.

5. Costo

> En mercados masivos (como electrónica de consumo), el costo es un factor decisivo.
> Se debe minimizar el uso de recursos hardware y software.
> Estrategias:
> -Reducir frecuencia de reloj y voltaje.
> -Evitar hardware innecesario (ej.: cachés o unidades de gestión de memoria que no mejoren el peor caso de tiempo de ejecución).

6. Enfoque Integral Hardware-Software

> No es posible diseñar software sin considerar el hardware subyacente.
> Se requiere cooperación entre ingeniería eléctrica y ciencias de la computación.
> Es necesario encontrar un equilibrio entre eficiencia (hardware personalizado) y flexibilidad (software).

7. Otros Desafíos Relacionados

> - Big Data: Sistemas CPS/IoT generan grandes volúmenes de datos que deben almacenarse y analizarse bajo restricciones de recursos.
> - Aspectos no técnicos: Legales, económicos, sociales, humanos y ambientales (ej.: responsabilidad en coches autónomos, interfaces usuario-máquina, impacto ambiental).
> - Concurrencia: Los sistemas embebidos son concurrentes por naturaleza; gestionar múltiples tareas es un desafío.
> - Heterogeneidad: Integración de componentes hardware y software de diversos proveedores en entornos cambiantes.

> - Diseño Composicional: Necesidad de estudiar el impacto de combinar componentes (ej.: añadir un GPS sin saturar el bus de comunicación).

Claro. Aquí tienes un resumen de los flujos de desarrollo para sistemas embebidos, sintetizado a partir de fuentes confiables (como el libro de texto *Embedded System Design* de Peter Marwedel, estándares de la industria y mejores prácticas).

Flujos de desarrollo de Sistemas Embebidos.

El desarrollo de sistemas embebidos es inherentemente complejo debido a la **co-diseño hardware/software** y a las estrictas **restricciones no funcionales** (tiempo real, potencia, energía, costo, fiabilidad). Por ello, se utilizan flujos de desarrollo estructurados y iterativos.

1. Características comunes de un flujo de diseño embebido

Todos los flujos comparten una serie de etapas y conceptos comunes:

- **Especificación:** Captura formal y semi-formal de los requisitos funcionales y no funcionales del sistema. Es la fase más crítica.
- **Arquitectura del Sistema (Exploración):** Decisión sobre los componentes hardware (procesadores, ASICs, sensores, actuadores) y software (SO, middlewares) que compondrán el sistema.
- **Partición Hardware/Software (Hardware/Software Partitioning):** Decisión crucial sobre qué partes de la funcionalidad se implementarán en hardware (para eficiencia y rendimiento) y cuáles en software (para flexibilidad).
- **Diseño Co-Curricular:** Desarrollo paralelo e integrado de ambos dominios:
  - **Diseño Hardware:** Diseño de PCBs, sistemas en chip (SoC), FPGA, etc.
  - **Diseño Software:** Programación de bajo nivel (drivers), sistemas en tiempo real (RTOS), lógica de aplicación.
- **Integración:** Unión de los componentes hardware y software en una plataforma física. Suele ser la fase más problemática.
- **Verificación y Validación (V&V):** Proceso continuo para asegurar que el diseño cumple con la especificación ("¿se construyó el sistema correctamente?") y que el sistema cumple con las necesidades del usuario ("¿se construyó el sistema correcto?"). Incluye:
  - **Simulación:** Modelado del comportamiento del sistema antes de tener el hardware físico.
  - **Emulación:** Uso de FPGAs o hardware especial para prototipar el sistema.
  - **Pruebas (Testing):** Pruebas unitarias, de integración, de sistema y de regresión.
- **Síntesis y Optimización:** Procesos automáticos o semi-automáticos para generar implementaciones eficientes a partir de descripciones de alto nivel (ej.: síntesis de alto nivel - HLS para hardware, optimizaciones de compilador para software).

2. Modelos de proceso de desarrollo comunes

No existe un único flujo universal, sino varios modelos que se adaptan a la naturaleza del proyecto.

1. **Modelo en "V" (V-Model):**
   - **Fuente:** Ampliamente adoptado en industria automotriz, aeroespacial y por agencias gubernamentales (como el estándario alemán).

- **Descripción:** Es un modelo secuencial que enfatiza la **verificación y validación** en cada etapa. La forma de "V" representa cómo cada fase de diseño (brazo izquierdo) tiene una fase de prueba correspondiente (brazo derecho).
- **Fortalezas:** Muy estructurado, documentado y con una clara trazabilidad entre requisitos y pruebas. Ideal para sistemas de alta integridad y seguridad.
- **Debilidades:** Poco flexible a cambios una vez avanzado el proyecto.

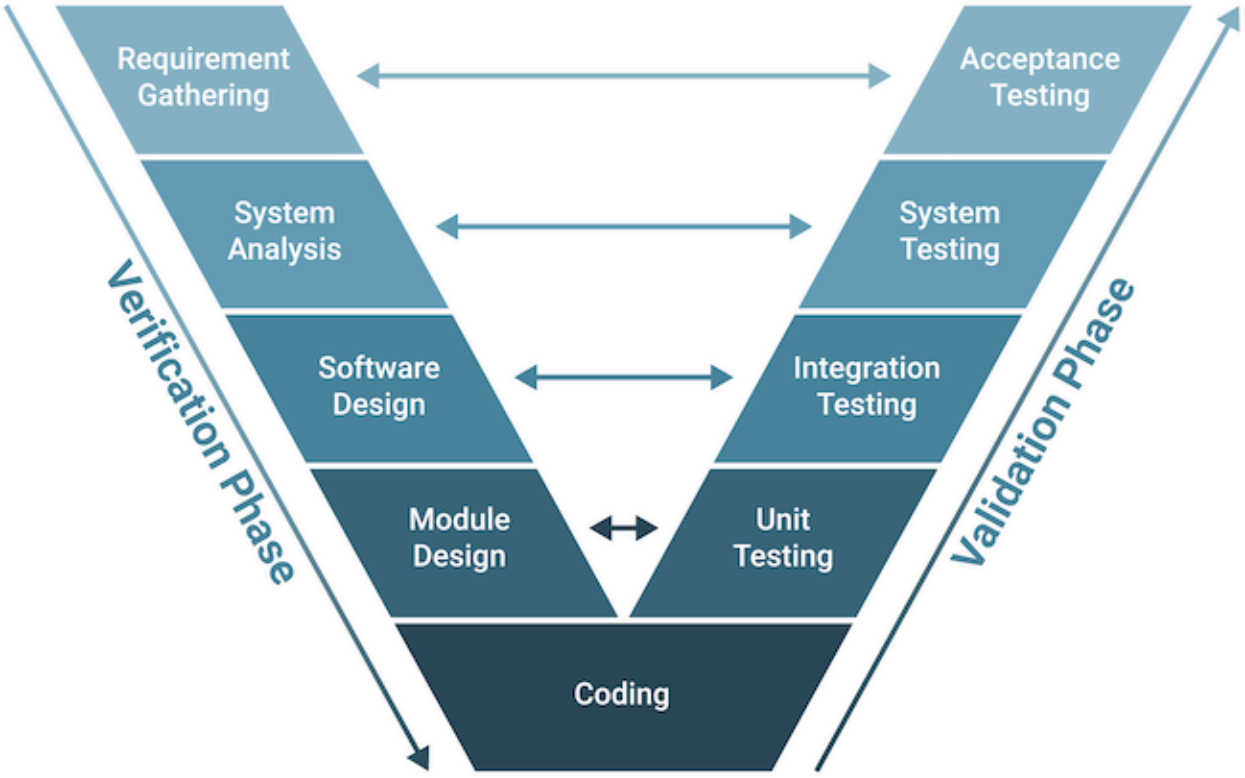

Figura 2 Flujo de trabajo V para el desarrollo de sistemas empotrados

2. **Diseño Basado en Plataformas:**
   - **Fuente:** Práctica industrial moderna para reducir tiempo y costo.
   - **Descripción:** Se basa en reutilizar una plataforma hardware/software preexistente y bien probada (ej.: una familia de microcontroladores, un RTOS específico, librerías). El diseño se centra en personalizar y agregar valor a esta plataforma base.
   - **Fortalezas:** Acelera enormemente el time-to-market, reduce riesgos y costos.
   - **Debilidades:** Puede limitar la optimización extrema al estar restringido a las capacidades de la plataforma.
3. **Desarrollo Ágil para Sistemas Embebidos (Agile/Scrum):**
   - **Fuente:** Adaptación de las metodologías ágiles de software al contexto embebido.
   - **Descripción:** Utiliza iteraciones cortas (sprints) para desarrollar incrementalmente el sistema. Se enfoca en la colaboración y la respuesta al cambio.
   - **Fortalezas:** Muy adaptable a cambios en los requisitos. Mejora la comunicación.

- **Debilidades:** La integración continua con hardware físico es un desafío. Requiere técnicas como "hardware-in-the-loop" (HIL) para simular el hardware en sprints tempranos.

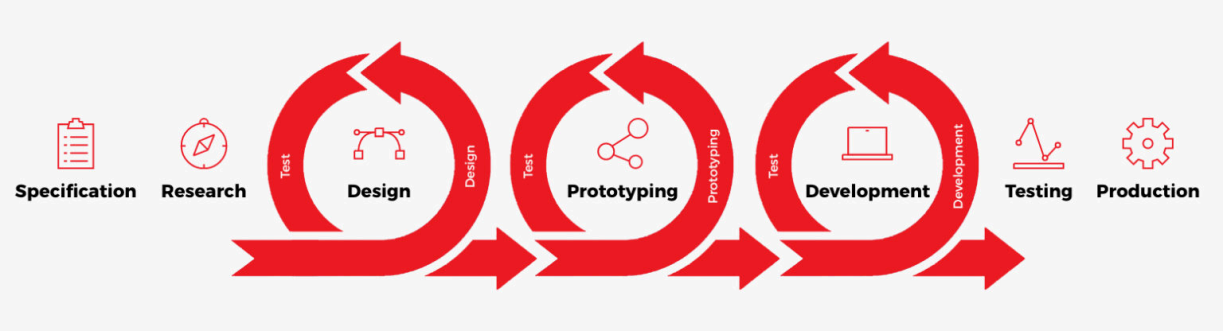

Figura 3 Flujo de trabajo Spring para el desarrollo de sistemas empotrados

4. **Carta Y de Gajski:**

- **Fuente:** Modelo académico e industrial para el diseño de sistemas electrónicos.
- **Descripción:** Visualiza el diseño en tres dominios que se refinan de manera concurrente:
  - **Comportamental:** *¿Qué* hace el sistema? (Algoritmos).
  - **Estructural:** *¿Con qué* se hace? (Componentes e interconexiones).
  - **Físico:** *¿Dónde* se ubica? (Layout, geometría).
- **Fortalezas:** Enfatiza la naturaleza multidimensional del diseño (HW/SW/físico).

Figura 4 Flujo de trabajo Y de Gajski para el desarrollo de sistemas empotrados

5. Herramientas Clave en el Flujo (Toolchain)

Un flujo de desarrollo moderno depende de un ecosistema de herramientas:

- **Entornos de Desarrollo Integrado (IDE):** Keil µVision, IAR Embedded Workbench, STM32CubeIDE, MPLAB X.
- **Herramientas de Modelado y Análisis:** MATLAB/Simulink (para diseño basado en modelo - MBD), Stateflow.
- **Compiladores Cruzados (Cross-Compilers):** GCC para ARM, LLVM, compiladores propietarios.
- **Simuladores y Emuladores:** QEMU, modelos de ISS (Instruction Set Simulator).
- **Sistemas de Control de Versiones:** Git, SVN.
- **Herramientas de Integración Continua (CI):** Jenkins, GitLab CI (para automatizar builds y pruebas en software, incluso con emulación).

Aplicaciones de los sistemas embebidos

Los sistemas embebidos están presentes en prácticamente todos los ámbitos de la vida moderna, facilitando funciones críticas y mejorando la eficiencia en diversos sectores. En el transporte, permiten el funcionamiento de sistemas de seguridad como airbags, frenos ABS y control de estabilidad en vehículos, así como la navegación GPS y la asistencia al conductor. En el ámbito industrial, son fundamentales para la automatización de procesos, el control de maquinaria y la logística, integrando tecnologías como RFID para la gestión inteligente de inventarios y cadenas de suministro. Además, en el sector salud, posibilitan el desarrollo de

dispositivos médicos avanzados, como marcapasos, ventiladores y sistemas de monitorización remota de pacientes, mejorando el diagnóstico y el tratamiento.

En la vida cotidiana, estos sistemas optimizan el consumo energético y la seguridad en hogares y edificios inteligentes, regulando la climatización, la iluminación y el acceso. También son clave en infraestructuras críticas, como redes eléctricas inteligentes (smart grids), donde equilibran la demanda y generación de energía, y en sistemas de comunicación, incluyendo **teléfonos móviles** y equipos de telecomunicaciones. Otras aplicaciones destacadas incluyen la robótica, la agricultura de precisión, la gestión de desastres naturales y la seguridad pública, mediante tecnologías de identificación y monitoreo. Su versatilidad y capacidad de integración los convierten en componentes indispensables para el funcionamiento de la sociedad contemporánea.

Parte 2: Introducción al desarrollo de aplicaciones para teléfonos móviles

El desarrollo de aplicaciones móviles se ha convertido en uno de los campos más dinámicos y relevantes dentro de la industria tecnológica contemporánea. La expansión de los dispositivos inteligentes, la conectividad permanente y el acceso inmediato a la información han transformado no solo la manera en que los usuarios interactúan con la tecnología, sino también la forma en que las organizaciones diseñan sus servicios y productos. Las aplicaciones móviles no se limitan a un sector específico; abarcan desde el ámbito educativo hasta el comercial, pasando por el entretenimiento, la comunicación y la gestión empresarial. En este contexto, el documento analizado presenta un recorrido por los principales elementos del desarrollo móvil, abordando desde los retos técnicos hasta las estrategias metodológicas que lo sustentan.

Panorama global: uso de dispositivos móviles y sistemas operativos

- En 2025, Android posee aproximadamente un **73.9 %** del mercado mundial de sistemas operativos móviles, seguido por iOS con cerca del **25.7 %**.
- En los dispositivos móviles (smartphones), estas dos plataformas dominan con más del 99 % de la cuota combinada, relegando a otros sistemas operativos como KaiOS, sistemas propios de marcas, u otros, a porcentajes muy pequeños.
- En cuanto al mercado de teléfonos inteligentes, los envíos globales se mantienen elevados (cientos de millones de unidades al trimestre). Por ejemplo, en el segundo trimestre de 2025 se reportaron **295.2 millones de unidades** entregadas mundialmente.

Enfoques de desarrollo de aplicaciones móviles

1. Aplicaciones Nativas

- **Desarrolladas** específicamente para una plataforma (iOS o Android).
- **Ventajas**: Alto rendimiento, acceso completo al hardware del dispositivo y experiencia de usuario fluida.
- **Desventajas**: Costos elevados y necesidad de desarrollar múltiples versiones para diferentes plataformas.

2. Aplicaciones Híbridas

- **Combinan** elementos de aplicaciones nativas y web.
- **Ventajas**: Single codebase para múltiples plataformas, menor costo y tiempo de desarrollo.
- **Desventajas**: Rendimiento inferior comparado con las nativas y acceso limitado a funcionalidades del dispositivo.

3. Aplicaciones Web Progresivas (PWA)

- **Accesibles** mediante navegadores web pero ofrecen experiencia similar a una aplicación nativa.
- **Ventajas**: No requieren instalación, actualizaciones automáticas y menor uso de almacenamiento.
- **Desventajas**: Funcionalidad limitada sin conexión a internet y acceso restringido a hardware del dispositivo.

4. Cross-Platform Applications

- **Compiladas** en código nativo pero desarrolladas con frameworks como React Native o Flutter.
- **Ventajas**: Balance entre rendimiento y coste de desarrollo.
- **Desventajas**: Pueden requerir bridges para acceso a APIs nativas.

A copntinuación, en la tabla 2, se resumen algunos elementos de los enfoques, con ventajas y desventajas distintas:

Tabla 2 Resumen de características de las aplicaciones móviles según el enfoque.

Tipo de App	Rendimiento	Costo	Acceso a Hardware	Mantenimiento	Ventajas	Desventajas	Caso de Uso Ideal
Nativa	Alto	Alto	Completo	Complejo	Alto rendimiento, acceso total a hardware	Desarrollo costoso y no reusable entre plataformas	Juegos complejos, aplicaciones empresariales críticas
Web Móviles	Bajo-Medio	Bajo	Limitado	Sencillo	Acceso inmediato sin instalación, bajo costo	Limitada integración con dispositivo, dependencia de conexión	Sitios informativos, portales de noticias
Híbrida	Medio	Medio	Parcial	Moderado	Balance entre costo y performance	Mayores desafíos de depuración	Apps corporativas, catálogos de productos
PWA	Bajo-Medio	Bajo	Limitado	Sencillo	Instalables, funcionamiento offline	Limitaciones en acceso a hardware avanzado	Plataformas de educación, herramientas de productividad
Cross-Platform	Alto-Medio	Medio	Parcial a Completo	Moderado	Código reusable (70-90%), rápido desarrollo, amplio alcance de mercado	Rendimiento inferior en apps complejas, dependencia de frameworks de terceros	Startups, MVP, aplicaciones empresariales, apps de contenido

Fuente: Adaptado de [2] [3] [4] [5] [6]

Factores que impulsan el desarrollo de aplicaciones móviles

Algunos de los principales impulsos para que el desarrollo móvil sea hoy una disciplina clave:

1. **Penetración de teléfonos inteligentes y conectividad**
   Millones de personas en el mundo usan smartphones; la conectividad móvil se ha expandido notablemente, lo que hace que muchas aplicaciones se diseñen para uso continuo, online/offline, sincronización, etc.
2. **Demanda de experiencia "always-on" e integración de hardware**
   Usuarios esperan que las apps aprovechen GPS, cámara, sensores como giroscopio/acelerómetro, biometría, etc. Cuanto más nativa sea la app, mejor puede integrar estas funciones con alto rendimiento.
3. **Ecosistemas de distribución establecidos**
   Las tiendas de apps (Google Play, Apple App Store, etc.) ofrecen infraestructuras robustas de distribución, monetización, seguridad y revisión. Esto influye en decisiones sobre tipo de app, modelo de negocio y alcance geográfico.
4. **Diversidad de dispositivos y plataformas**
   Desde teléfonos de gama alta hasta dispositivos económicos, tablets, relojes inteligentes, etc. Esto implica que hay que pensar en rendimiento, compatibilidad, tamaño de pantalla, modos de interactuar, disponibilidad de hardware, versiones antiguas del sistema operativo, etc.

Estrategias de desarrollo: Enfoques Ágiles y Centrados en el Usuario

El desarrollo móvil requiere **métodos ágiles y iterativos** debido a:

- Ciclos de desarrollo cortos
- Alta volatilidad del mercado
- Necesidad de prototipos rápidos

Algunos métodos comunes incluyen:

1. Desarrollo Ágil:

- Iteraciones cortas (sprints) para responder rápidamente a cambios.
- Priorización de funcionalidades basada en valor de usuario.
- Ideal para entornos volátiles con alta incertidumbre.

2. Mobile-D:

- Ciclos de desarrollo ultra-rápidos (menos de 10 semanas).
- Combinación de XP, Crystal y RUP para equilibrar velocidad y calidad.
- Enfocado en equipos pequeños y co-locados.

3. Pruebas Continuas:

- Unitarias, de integración y de usabilidad en dispositivos reales.
- Uso de herramientas cloud-based para testear en múltiples dispositivos en paralelo.
- Criticalidad de pruebas de estrés y rendimiento en condiciones de red variables.

Fases de un proyecto móvil: Consideraciones clave

A nivel global, todo proyecto de desarrollo móvil suele seguir estas fases:

1. **Planificación y levantamiento de requisitos**:
    - Análisis de mercado y usuarios objetivo.
    - Evaluación de viabilidad técnica y económica.
    - Consideración de **brechas digitales** y contexto socioeconómico del público meta .
2. **Diseño (UI/UX)**:
    - Prototipado iterativo con feedback de usuarios.
    - Patrones de diseño como **MVC** para separar lógica y presentación.
    - **Threading** para operaciones no bloqueantes y **Delegation** para manejo de eventos.
3. **Implementación**:
    - Elección de stack tecnológico (nativo, híbrido, cross-platform).
    - Desarrollo guiado por pruebas (TDD) para mayor robustez.
    - Integración de APIs y servicios cloud.
4. **Pruebas**:
    - Enfocadas en usabilidad, rendimiento y seguridad.
    - Es crucial probar en **condiciones de red reales** y con los dispositivos más comunes en el mercado objetivo .
5. **Distribución y Mantenimiento**:
    - Publicación en stores (App Store, Google Play) o distribución empresarial.
    - Monitoreo continuo de crash reports y métricas de uso.
    - Actualizaciones periódicas para adaptarse a nuevos OS y dispositivos.

Herramientas y Lenguajes de Programación

Para Android

- **Lenguajes**: Java, Kotlin.
- **Entorno de Desarrollo**: Android Studio.
- **Frameworks**: React Native, Flutter.

Para iOS

- **Lenguajes**: Swift, Objective-C.
- **Entorno de Desarrollo**: Xcode.
- **Frameworks**: React Native, Flutter.

Frameworks Multiplataforma

- **React Native**: Utiliza JavaScript y React.
- **Flutter**: Utiliza Dart y ofrece alto rendimiento.
- **Xamarin**: Utiliza C# y .NET.

Tendencias actuales en desarrollo móvil

1. **Inteligencia Artificial (IA) y Machine Learning**: Integración de chatbots y recomendaciones personalizadas.
2. **Realidad Aumentada (AR) y Virtual (VR)**: Experiencias inmersivas en aplicaciones de retail y educación.
3. **Internet de las Cosas (IoT)**: Conexión con dispositivos inteligentes para automatización del hogar.
4. **Seguridad y Privacidad**: Enfoque en protección de datos personales y cumplimiento de regulaciones como GDPR.
5. **5G**: Mayor velocidad de conexión y capacidades para aplicaciones en tiempo real.

Escenarios recomendados: cuándo usar cada tipo de aplicación

Aquí algunas recomendaciones prácticas para elegir el tipo de aplicación según el contexto:

Escenario	Tipo recomendado
Aplicaciones con alta demanda de rendimiento, gráficos pesados, interacción compleja (por ejemplo videojuegos, edición de multimedia, realidad aumentada)	**Nativas**
Presencia en múltiples plataformas con presupuesto limitado, funcionalidades moderadas, interfaz sencilla, tiempo de desarrollo corto	**Híbridas** o frameworks cross-platform nativo (“write once, run everywhere”) como React Native, Flutter
Prototipos, aplicaciones informativas, contenido ligero, sitios web que desean estar disponibles como apps, sin necesidad de distribución en tiendas	**Web Apps / PWAs**

Fuentes:

---

1. Embedded System Design. Peter Marwedel, 2018. Springer ↩

2. IBM. "What Is Mobile Application Development?" (2023) ↩

3. AWS. "Mobile Application Development: Native vs. Hybrid vs. Cross-Platform" (2023) ↩

4. GitHub. "State of the Octoverse: Cross-Platform Development Trends" (2023) ↩

5. StatCounter. "Mobile Operating System Market Share Worldwide" (2024) ↩

6. Gartner. "Market Guide for Multiplatform Application Development" (2023) ↩