2025

Entorno de Desarrollo

TEMA 1

OLIVER BITICA – 1°DAM

Contenido

Leyenda de Colores

- Amarillo: Para conceptos fundamentales, definiciones y puntos clave.
- Verde: Para ventajas, objetivos, propósitos y características positivas.
- Azul: Para tipos, clasificaciones, componentes, estructuras y ejemplos.
- Rojo/Salmón: Para problemas, inconvenientes, limitaciones o advertencias.
- Gris: Para tecnologías específicas, nombres propios, estándares o secciones explícitamente marcadas como “Contenido Prioritario”.

Fundamentos de la Computación: Un Viaje desde el Hardware hasta las Metodologías de Software

1. Introducción: Los Pilares del Mundo Digital

Para comprender verdaderamente el software que impulsa nuestra era digital, primero es necesario entender la máquina sobre la que se ejecuta y los procesos mediante los cuales se crea. La aplicación más sofisticada no es más que una serie de instrucciones que una pieza de hardware debe interpretar y ejecutar. Del mismo modo, el código más elegante es ineficaz si no se enmarca en un proceso de construcción coherente y planificado. Este documento propone un recorrido en tres etapas fundamentales para construir una visión integral de la computación. Comenzaremos explorando la arquitectura fundamental del computador, continuaremos con los lenguajes que actúan como puente entre el ingenio humano y la lógica de la máquina, y concluiremos con las metodologías que organizan la creación de software complejo y fiable.

Este viaje comienza en el nivel más elemental: el hardware. La base de casi todos los dispositivos informáticos modernos se asienta sobre un modelo conceptual definido hace décadas, la Arquitectura de Von Neumann, nuestro punto de partida.

2. La Máquina por Dentro: La Arquitectura de Von Neumann

Aunque fue definida en 1945, la Arquitectura de Von Neumann sigue siendo el modelo conceptual sobre el que se construyen la gran mayoría de los computadores actuales. Su diseño estratégico, que trata tanto las instrucciones como los datos como información almacenable en la misma memoria, revolucionó la computación y sentó las bases para la era del software. Comprender su estructura es, por tanto, esencial para entender cómo un conjunto de líneas de código se transforma en acciones concretas dentro de un dispositivo.

2.1. Las Unidades Funcionales Clave

La arquitectura define un sistema compuesto por cuatro unidades funcionales interconectadas de forma permanente, cada una con una misión específica:

- **Memoria**: Almacena tanto las instrucciones que componen un programa como los datos que este manipula. Está organizada en celdas, cada una con una dirección única.
- **CPU (Unidad Central de Proceso)**: Es el componente encargado de ejecutar los programas almacenados en la memoria principal, procesando las instrucciones de forma secuencial.
- **Sistema de E/S (Entrada/Salida)**: Gestiona la comunicación y conexión del sistema con los periféricos externos, como el teclado, el monitor o los dispositivos de almacenamiento.
- **Buses**: Actúan como las "autopistas" de la información, interconectando las distintas unidades funcionales para permitir la transmisión de datos, direcciones y señales de control.

2.2. El Cerebro de la Operación: La Unidad Central de Proceso (CPU)

La CPU es el auténtico cerebro del computador, responsable de ejecutar y controlar todas las operaciones del sistema. Sus funciones abarcan desde leer y escribir información en la memoria, decodificar instrucciones hasta realizar cálculos matemáticos y lógicos. Para cumplir su cometido, se compone de tres elementos internos fundamentales.

Registros Son unidades de memoria de muy alta velocidad y pequeño tamaño, integradas directamente en el procesador. Sirven para almacenar información de manera temporal durante la ejecución de un programa. Se dividen en dos categorías:

- **De uso general**: Almacenan datos o direcciones de memoria de forma temporal.
- **De uso específico**: Tienen un propósito predefinido, como el contador de programa (que apunta a la siguiente instrucción), el registro de instrucción (que almacena la instrucción actual) o el indicador de estado, que veremos en detalle en la ALU.

Unidad Aritmético-Lógica (ALU) Es la sección de la CPU encargada de realizar las operaciones matemáticas (suma, resta) y lógicas (AND, OR, NOT) que se le indican. Sus componentes principales son:

- **Registros de entrada (RE)**: Almacenan los operandos sobre los que se realizará la operación.
- **Circuito de operaciones (COP)**: Ejecuta la operación propiamente dicha.
- **Registro acumulador (RA)**: Guarda el resultado de la operación ejecutada.

- **Registro de estado (RS)**: Un excelente ejemplo de registro de uso específico. Almacena los "flags" o indicadores sobre el resultado de la última operación (si fue cero, negativo, si hubo acarreo, etc.).
- **Registros bandera**: Registros de 16, 32 o 64 bits (como EFLAGS o RFLAGS en la arquitectura x86) que contienen el conjunto de flags de estado del sistema.

Unidad de Control (UC) Actúa como la directora de orquesta del computador. Su función es leer las instrucciones de la memoria, interpretarlas y enviar las órdenes necesarias al resto de componentes (CPU, memoria, E/S) para que se ejecuten de forma sincronizada. Sus elementos clave incluyen:

- **Contador de programa (CP)**: Contiene la dirección de memoria de la próxima instrucción a ejecutar.
- **Registro de instrucción (RI)**: Almacena la instrucción que se está ejecutando en el momento.
- **Decodificador (D)**: Interpreta la instrucción y genera las señales de control para su ejecución.
- **Reloj (R)**: Proporciona una señal de pulsos eléctricos que sincroniza todas las operaciones del sistema.
- **Secuenciador (S)**: Genera las microórdenes detalladas para ejecutar cada instrucción paso a paso.

2.3. El Flujo de la Ejecución: El Ciclo de Instrucción

La CPU ejecuta un programa procesando sus instrucciones una por una, de forma secuencial. Este proceso fundamental se conoce como el ciclo de instrucción, y se divide en dos fases principales para cada instrucción del programa:

1. **Fase de Búsqueda (Captación)**: El objetivo de esta fase es traer la instrucción desde la memoria principal hasta la Unidad de Control para que pueda ser interpretada.
2. **Fase de Ejecución**: Una vez la instrucción ha sido decodificada, se llevan a cabo las acciones que esta especifica. El secuenciador envía las microórdenes a la ALU o a otras unidades, y el resultado se almacena donde corresponda.

El proceso detallado de la **Fase de Búsqueda** sigue los siguientes pasos:

1. Se lee la dirección de la instrucción desde el Contador de Programa (CP).
2. Esta dirección se transfiere al registro de dirección de memoria (MAR).
3. El decodificador de memoria activa la celda de memoria correspondiente y su contenido se transfiere al registro de datos (MBR).
4. La instrucción completa está ahora disponible en el MBR.
5. La instrucción viaja desde el MBR al Registro de Instrucción (RI) de la Unidad de Control.
6. El decodificador de la UC interpreta la instrucción e informa al secuenciador.
7. El Contador de Programa (CP) se incrementa para apuntar a la siguiente instrucción, preparándose para el próximo ciclo.

2.4. La Jerarquía de la Memoria

Un computador no utiliza un único tipo de memoria, sino un sistema organizado en niveles para equilibrar velocidad, capacidad y coste.

- **Memoria Principal**: Es donde se almacenan temporalmente los programas y datos en uso. Se divide en ROM (memoria de solo lectura, usada para el arranque o BIOS) y RAM (memoria de acceso aleatorio, volátil y de escritura-lectura).
- **Memoria Caché**: Una memoria más pequeña y rápida que la RAM, situada entre esta y la CPU. Almacena la información utilizada con más frecuencia para acelerar el acceso. Se organiza en niveles (L1, L2, L3), siendo L1 la más cercana a la CPU y, por tanto, la más rápida.
- **Memoria Virtual**: Una técnica que utiliza espacio en un dispositivo más lento (como el disco duro) para simular una mayor cantidad de memoria principal. Es útil cuando los programas requieren más RAM de la que está físicamente disponible.

Este sistema se estructura en una **Jerarquía de Memoria** para optimizar el rendimiento global:

- **Nivel 0: Registros** (la más rápida, de menor capacidad y mayor coste).
- **Nivel 1: Memoria caché**.
- **Nivel 2: Memoria primaria (RAM)**.
- **Nivel 3: Disco duro (almacenamiento masivo)**.
- **Nivel 4: Memorias extraíbles** (la más lenta, de mayor capacidad y menor coste).

Comprendida la arquitectura física que obedece ciegamente las órdenes, el desafío se traslada a cómo formular dichas órdenes de manera eficiente y comprensible para el ser humano. Este es el rol fundamental de los lenguajes de programación, el puente abstracto que construimos sobre el hardware.

3. El Puente entre Humano y Máquina: Los Lenguajes de Programación

Los lenguajes de programación son la herramienta esencial que nos permite instruir a la arquitectura de hardware. Proporcionan un conjunto de reglas sintácticas y semánticas que abstraen la complejidad del código máquina (secuencias de unos y ceros) y nos permiten expresar algoritmos y lógica de una manera más cercana al pensamiento humano. Existen múltiples formas de clasificar estos lenguajes, cada una ofreciendo una perspectiva diferente sobre su naturaleza y propósito.

3.1. Clasificación por Nivel de Abstracción

Este criterio mide la distancia entre el lenguaje y el hardware subyacente.

- **Bajo Nivel (Primera Generación)**: Es el código máquina, el único lenguaje que la CPU entiende de forma nativa. Consiste en secuencias de 1 y 0, y es impracticable para el desarrollo humano directo.
- **Medio Nivel (Segunda Generación)**: Corresponde al lenguaje ensamblador. Utiliza mnemónicos (palabras cortas) para representar instrucciones máquina básicas, ofreciendo una ligera abstracción pero manteniendo una alta dependencia del hardware específico.
- **Alto Nivel (3ª, 4ª y 5ª Generación)**: Son lenguajes mucho más cercanos al lenguaje humano, independientes de la máquina.
    - **Tercera Generación**: Incluye la mayoría de los lenguajes modernos como Java o Python, muchos de ellos orientados a objetos.
    - **Cuarta Generación**: Diseñados para un propósito específico, como la gestión de bases de datos (SQL) o el cálculo matemático (MatLab), reduciendo la cantidad de código necesario.
    - **Quinta Generación**: Orientados a la inteligencia artificial y la resolución de problemas mediante bases de conocimiento, como Prolog o Lisp.

3.2. Clasificación por Forma de Ejecución

Este criterio se enfoca en cómo el código fuente escrito por un programador se convierte en acciones ejecutables por la CPU.

- **Lenguajes Compilados**: Un programa llamado compilador traduce todo el código fuente a código objeto (una versión en bajo nivel) de una sola vez. Posteriormente, un enlazador (linker) une este código objeto con las librerías necesarias para generar un archivo ejecutable final. Ejemplos son C y C++.
- **Lenguajes Interpretados**: El código fuente no se traduce previamente. Un programa llamado intérprete lee y ejecuta el código línea por línea en tiempo real. Este enfoque tiene un claro compromiso: facilita un ciclo de desarrollo y depuración más rápido al no requerir un paso de compilación largo, pero a costa de un rendimiento en ejecución más lento en comparación con el código compilado.

- **Lenguajes Intermedios**: Combinan ambos enfoques. El código fuente se compila a un código intermedio llamado bytecode, que no es específico de ninguna máquina. Este bytecode es luego interpretado por una máquina virtual en el sistema de destino. Ofrecen la ventaja de ser multisistema ("escribe una vez, ejecuta en cualquier lugar"), aunque son algo más lentos que los compilados puros. Java es el ejemplo paradigmático. Python, aunque comúnmente citado como interpretado, en su implementación estándar (CPython) primero compila el código fuente a bytecode (archivos .pyc), que es luego interpretado por su máquina virtual.

3.3. Clasificación por Paradigma de Programación

Un paradigma es un estilo o una filosofía para estructurar el código y resolver problemas.

- **Imperativo**: El código describe una secuencia de pasos que modifican el estado del programa para alcanzar un resultado.
- **Declarativo**: Se enfoca en describir qué se quiere lograr, sin detallar el cómo. SQL es un ejemplo claro ("selecciona estos datos de esta tabla").
- **Procedimental**: Una evolución del imperativo, donde el programa se organiza en procedimientos o funciones reutilizables.
- **Orientado a Objetos**: El código se estructura en torno a "objetos" que encapsulan tanto datos (estado) como comportamiento (operaciones), modelando entidades del mundo real.
- **Funcional**: Trata la computación como la evaluación de funciones matemáticas, evitando los cambios de estado y los datos mutables.
- **Lógico**: Se basa en la lógica formal. El programador define un conjunto de reglas y hechos, y el sistema utiliza un motor de inferencia para deducir las respuestas a las consultas.

3.4. Tabla de Clasificación de Lenguajes Populares

La siguiente tabla resume la clasificación de algunos de los lenguajes de programación más utilizados según los criterios descritos:

Lenguaje	Nivel de Abstracción	Forma de Ejecución	Paradigma de Programación
C	Alto Nivel (3ª Gen)	Compilado	Imperativo, Procedimental
C++	Alto Nivel (3ª Gen)	Compilado	Orientado a Objetos, Procedimental
Java	Alto Nivel (3ª Gen)	Intermedio (Virtual)	Orientado a Objetos
Python	Alto Nivel (3ª Gen)	Interpretado	Orientado a Objetos, Imperativo, Funcional
Ruby	Alto Nivel (3ª Gen)	Interpretado	Orientado a Objetos, Funcional, Imperativo
Go	Alto Nivel (3ª Gen)	Compilado	Procedimental, Imperativo
Javascript	Alto Nivel (3ª Gen)	Interpretado	Orientado a Objetos, Funcional, Imperativo
SQL	Alto Nivel (4ª Gen)	Interpretado	Declarativo
PHP	Alto Nivel (3ª Gen)	Interpretado	Orientado a Objetos, Procedimental
Prolog	Alto Nivel (5ª Gen)	Interpretado	Lógico

Disponer de un lenguaje es fundamental, pero no suficiente. Para construir sistemas de software complejos, robustos y mantenibles, es imprescindible seguir un proceso estructurado que organice el esfuerzo colectivo. Esto nos lleva a la última etapa de nuestro viaje: las metodologías de desarrollo.

4. El Arte de Construir Software: Procesos y Metodologías de Desarrollo

La creación de software robusto y fiable va mucho más allá de la simple codificación. Requiere un enfoque estructurado que guíe a los equipos desde la concepción de una idea hasta la entrega y el mantenimiento del producto final. Este marco de trabajo

se conoce como el ciclo de vida del software, y las distintas formas de organizarlo dan lugar a diferentes metodologías de desarrollo, cada una con sus propias fortalezas y debilidades.

4.1. El Modelo Tradicional: Cascada

El Modelo en Cascada es un enfoque estrictamente secuencial. Cada fase del proyecto debe completarse en su totalidad antes de poder iniciar la siguiente, de forma similar a una cascada de agua que fluye en una única dirección. Es un modelo adecuado para proyectos con requisitos muy claros, estables y bien definidos desde el principio. Su principal desventaja es su rigidez: los cambios son difíciles y costosos de implementar, y los errores a menudo no se descubren hasta las etapas finales. Su filosofía se basa en la creación de una documentación formal y exhaustiva en cada etapa para garantizar la claridad y el acuerdo antes de avanzar.

Las fases del Modelo en Cascada son:

1. **Análisis**: Se definen los requisitos funcionales y no funcionales del software. El entregable clave de esta fase es la Especificación de Requisitos del Sistema (ERS), un documento formal que actúa como contrato entre el cliente y el equipo de desarrollo.
2. **Diseño**: Se planifica la arquitectura del sistema. El resultado se plasma en el cuaderno de carga, un documento técnico que detalla el funcionamiento general y los recursos necesarios, sirviendo de guía para los programadores.
3. **Codificación**: Los programadores traducen las especificaciones del diseño a un lenguaje de programación concreto.
4. **Pruebas**: Se verifica que el software funciona correctamente, está libre de errores y cumple con todos los requisitos especificados en la ERS.
5. **Documentación**: Se elaboran los manuales de usuario y la documentación técnica para futuros mantenimientos.
6. **Explotación**: El software se instala en el entorno del cliente y se pone en producción.
7. **Mantenimiento**: Se corrigen los errores (bugs) que surgen durante el uso y se implementan mejoras o nuevas funcionalidades.

4.2. La Evolución hacia la Flexibilidad: Modelos Iterativos

Como respuesta a la rigidez del modelo en cascada, surgieron los modelos iterativos y evolutivos. La idea central es desarrollar el software a través de ciclos repetidos (iteraciones), permitiendo la retroalimentación y el refinamiento continuo.

- **Modelo de Prototipado**: Se centra en construir una versión temprana y simplificada del sistema (un prototipo) para que el usuario pueda interactuar con ella y proporcionar feedback. Este proceso se repite, refinando el prototipo en cada iteración hasta que cumple las expectativas.
- **Modelo Incremental**: El software se construye y entrega en partes funcionales o "incrementos". Cada incremento añade nuevas capacidades al sistema, permitiendo que el cliente disponga de un producto funcional mucho antes de que el proyecto esté completamente terminado.
- **Modelo en Espiral**: Combina la naturaleza iterativa con el control sistemático del modelo en cascada, añadiendo un fuerte énfasis en el análisis y control de riesgos en cada ciclo de la espiral. Es ideal para proyectos grandes y complejos donde los riesgos son altos.

4.3. La Revolución Ágil: Adaptación y Colaboración

Los **Modelos Ágiles** representan un cambio de mentalidad. En lugar de seguir un plan rígido, priorizan la adaptabilidad, la colaboración con el cliente, la entrega continua de software funcional y la capacidad de responder rápidamente al cambio.

- **Scrum**: Es un marco de trabajo que organiza el desarrollo en ciclos cortos de duración fija llamados sprints (generalmente de 2 a 4 semanas). Al final de cada sprint, el equipo entrega un incremento de software potencialmente desplegable.
- **Kanban**: Se enfoca en la gestión visual del flujo de trabajo a través de un tablero. Su objetivo es optimizar el flujo, limitar el trabajo en curso (WIP) para evitar cuellos de botella y fomentar la mejora continua del proceso.
- **Programación Extrema (XP)**: Es una metodología ágil que se centra en un conjunto de prácticas de ingeniería de software para mejorar la calidad del código y la productividad del equipo, como el desarrollo guiado por pruebas (TDD), la programación en parejas y la integración continua.

4.4. Análisis Comparativo de Metodologías

La elección de una metodología depende en gran medida del tipo de proyecto, el equipo y la cultura de la organización. La siguiente tabla compara los modelos descritos:

Modelo	Ventajas	Inconvenientes
Cascada	- Sencillo y fácil de gestionar. - Fases y entregables bien definidos. - Ideal para proyectos con requisitos estables.	- Inflexible y resistente a cambios. - Los errores se detectan tarde y son costosos. - El software funcional no se ve hasta el final.
Prototipado	- Retroalimentación temprana del usuario. - Reduce el riesgo de no cumplir las expectativas. - Mejora la comprensión de los requisitos.	- Puede aumentar la complejidad. - El usuario puede confundir el prototipo con el producto final. - Puede consumir más tiempo y recursos.
Incremental	- Entrega temprana de software funcional. - Más flexible que el modelo en cascada. - Facilita la detección de errores.	- Requiere una buena planificación inicial. - El coste total puede ser superior al de cascada. - La integración de los incrementos puede ser compleja.
Espiral	- Fuerte enfoque en el análisis de riesgos. - Bueno para proyectos grandes y complejos. - Permite cambios y adaptaciones.	- Es un modelo complejo y costoso de gestionar. - El análisis de riesgos requiere personal experto. - No es adecuado para proyectos pequeños.
Scrum	- Alta adaptabilidad a los cambios. - Fomenta la colaboración y la comunicación. - Transparencia total sobre el progreso del proyecto.	- Requiere un equipo experimentado y autogestionado. - Riesgo de “scope creep” (aumento del alcance). - Puede ser difícil de escalar a grandes proyectos.
Kanban	- Flexible y enfocado en el flujo continuo. - Mejora la eficiencia y reduce el desperdicio. - Visualización clara del trabajo.	- Puede ser demasiado simple para proyectos complejos. - La falta de plazos fijos puede ser un problema. - No es un modelo de desarrollo en sí mismo.
P. Extrema (XP)	- Produce software de alta calidad. - Alta satisfacción del cliente. - Mejora la colaboración y productividad del equipo.	- Requiere una gran disciplina del equipo. - Puede ser difícil de implementar en grandes empresas. - Fuerte dependencia de la comunicación directa.

Hemos diseccionado la máquina, el lenguaje para comunicarnos con ella y los procesos para construir software de manera disciplinada. Es hora de ensamblar estas piezas para formar una visión unificada del campo.

5. Conclusión: De los Electrones a la Experiencia de Usuario

A lo largo de este recorrido, hemos viajado desde el nivel más fundamental del hardware hasta las estrategias más abstractas de gestión de proyectos. Hemos visto cómo la arquitectura física de Von Neumann establece las reglas del juego, definiendo cómo se procesan y almacenan las instrucciones. Hemos explorado cómo los lenguajes de programación, con sus diversos niveles de abstracción y paradigmas, nos proporcionan las herramientas para jugar ese juego de manera efectiva. Finalmente, hemos analizado cómo las metodologías de desarrollo, desde la rigidez secuencial de Cascada hasta la flexibilidad adaptativa de Agile, nos ofrecen las estrategias para ganar, entregando valor de manera consistente y organizada.

Es la comprensión de esta sinergia —entre la limitación física del hardware, la potencia abstracta del lenguaje y la disciplina estructural del proceso— lo que permite diseñar sistemas tecnológicos que no solo funcionan, sino que son robustos, mantenibles y verdaderamente alineados con su propósito. En última instancia, es el dominio de este camino, desde los electrones que fluyen en un circuito hasta la experiencia final del usuario, lo que define la esencia de la tecnología y el desarrollo de software.