Programas Informáticos en la Sociedad del Conocimiento:

- En la sociedad actual, el tratamiento de la información desempeña un papel crucial en ámbitos como la industria, la sociedad y el hogar. Los sistemas informáticos, encargados de este tratamiento, se componen de software. Abordaremos los conceptos de programa informático y programari, basándonos en estándares como IEEE, ISO e IEC.

Programa Informático

- Definición:
    - Conjunto de instrucciones y definiciones de datos que permite al hardware realizar funciones computacionales o de control.
    - Unidad sintáctica que sigue las reglas de un lenguaje de programación y está compuesta por declaraciones e instrucciones necesarias para una función específica.
- Analogía:
    - Se asemeja a una receta de cocina, donde los ingredientes (datos) y los pasos (instrucciones) son esenciales.

Programa (Software):

- Definición:
    - Incluye programas informáticos, procedimientos, documentación y datos relacionados con el funcionamiento de un sistema informático.
    - Comprende parte o la totalidad de los programas, procedimientos, reglas y documentación asociada a un sistema de procesamiento de la información.
- Complemento con maquina(Hardware):
    - Maquina se refiere a la parte física y tangible, mientras que el programa es la parte intangible. Ambos son complementarios, ya que el programa hace funcionar el maquina.
- Clasificación :
    - Programari de sistema: Facilita el funcionamiento y mantenimiento del sistema informático.
    - Programari d'aplicació: Ayuda al usuario a realizar tareas específicas.
    - Programari de suport: Dedicado al desarrollo y mantenimiento de otros programas.

Algoritmos y su Representación:

- Definición :
    - Conjunto finito de reglas definidas para resolver un problema en un número finito de pasos.
    - Secuencia de operaciones para realizar una tarea específica.
- Ejemplo:
    - Analogía con una receta de cocina.
    - Representación en lenguaje natural y posibilidad de expresarse en pseudocódigo o gráficamente (diagramas de flujo o actividades en UML).

Lenguajes de Programación:

- En el apartado anterior se habló de los algoritmos como secuencias de pasos para resolver problemas, expresados en lenguaje natural, pseudocódigo o diagramas. En este apartado, se abordan los lenguajes de programación, que permiten expresar algoritmos para que los comprenda un ordenador.
-

Definición de Lenguaje de Programación:

- Un lenguaje diseñado para especificar las reglas que un sistema informático debe seguir para resolver un problema.

Clasificación de los Lenguajes de Programación:

Nivel de Abstracción:

- Lenguajes de Bajo Nivel:
  - Cercanos al hardware, específicos para cada tipo de procesador (como el ensamblador).
- Lenguajes de Nivel Intermedio:
  - Con capacidades de alto nivel pero manteniendo algunas características de bajo nivel (como el lenguaje C).
- Lenguajes de Alto Nivel:
  - Más cercanos al lenguaje humano (como Java o Python).

Propósito:

- Lenguajes de Propósito General:
  - Diseñados para realizar diversas tareas (C, C++, Java, Python).
- Lenguajes de Propósito Específico:
  - Orientados a tareas específicas (como PHP para desarrollo web o R para análisis estadístico).

Evolución Histórica:

- Cinco generaciones, desde el código máquina hasta lenguajes más cercanos a la lógica humana.

Forma de Ejecución:

- Compilados:
  - Traducen el código completo a lenguaje de máquina (C, C++).
- Interpretados:
  - No requieren compilación, pero necesitan un intérprete (Python).

Estilo o Paradigma de Programación:

- Imperativa:
  - Estructurada (C) y orientada a objetos (C#, Java).
- Declarativa:
  - Funcional (Lisp/Haskell) y lógica (Prolog).

Lugar de Ejecución:

- Lenguajes de Servidor (Backend):
  - Utilizados en aplicaciones de servidor, como PHP.
- Lenguajes de Cliente (Frontend):
  - Ejecutados potencialmente en el cliente, como Javascript en navegadores web.

Ejecución de programas: del código fuente al ejecutable

- Los desarrolladores utilizan lenguajes de programación de alto nivel para expresar programas, los cuales deben ser traducidos a un lenguaje que el ordenador pueda entender y ejecutar, conocido como lenguaje máquina. En este apartado, se exploran las diferentes aproximaciones para generar programas ejecutables.

Traductores, compiladores e intérpretes

- Compiladores: Traducen programas de alto nivel a código máquina. Detectan errores antes de generar ejecutables. Ejemplos comunes son C o C++.
- Intérpretes: Traducen y ejecutan línea por línea, sin generar un ejecutable. Errores se detectan durante la ejecución. Ejemplos son Python.

Código fuente, código objeto y código ejecutable

- Código fuente: Instrucciones y definiciones expresadas en un lenguaje apto para traducción.
- Código objeto: Instrucciones y definiciones producidas por un compilador o ensamblador. Resultado de la traducción del código fuente.
- Código ejecutable: Se obtiene tras el enlazado de varios archivos de código objeto, combinando bibliotecas necesarias

Tecnologías de virtualización: Java y .NET

- Máquinas Virtuales: Permiten ejecutar múltiples máquinas virtuales en una máquina física.
- Java Virtual Machine (JVM): Compila a bytecode, interpretado por la JVM.
- Common Language Runtime (CLR): Compila a Common Intermediate Language (CIL), interpretado por el CLR. Ambos facilitan el desarrollo multiplataforma.

El concepto de multiplataforma

- Multiplataforma: Programas que pueden ejecutarse en diferentes plataformas, independientemente del hardware y sistema operativo.
- Ejemplos: LibreOffice, Mozilla Firefox, Gimp.

Sobre código fuente y software libre

- Programara Lliure: Basado en libertades como ejecución, adaptación, redistribución y mejora. Algunas requieren el código fuente.
- Software Propietario: Solo se distribuye el código ejecutable, no el fuente.

Desarrollo de Aplicaciones y Modelos de Desarrollo de Software

- El desarrollo de aplicaciones informáticas sigue un proceso denominado ciclo de vida del software, que consta de diversas fases y engloba a distintos actores. Hay varios modelos para abordar este ciclo, cada uno con su propia visión y enfoque. A grandes rasgos, las cinco fases del desarrollo de una aplicación son: análisis, diseño, codificación o implementación, implantación y mantenimiento.

Fases del Desarrollo de Aplicaciones:

1. Análisis:
    - Se establecen los requisitos de la aplicación mediante un análisis del problema.
    - Búsqueda de posibles soluciones en el mercado.
    - Especificación de modelos de datos y comportamiento utilizando herramientas gráficas.

2. Diseño:
    - A partir de los requisitos, se crea la solución considerando recursos del sistema.
    - Diseño de estructuras de datos y clases, dependiendo del paradigma utilizado.
    - Utilización de técnicas de diseño modular y procedimientos.

3. Codificación o Implementación:
    - Transformación de los algoritmos diseñados en programas en un lenguaje de programación.
    - Realización de pruebas para asegurar la calidad del software.

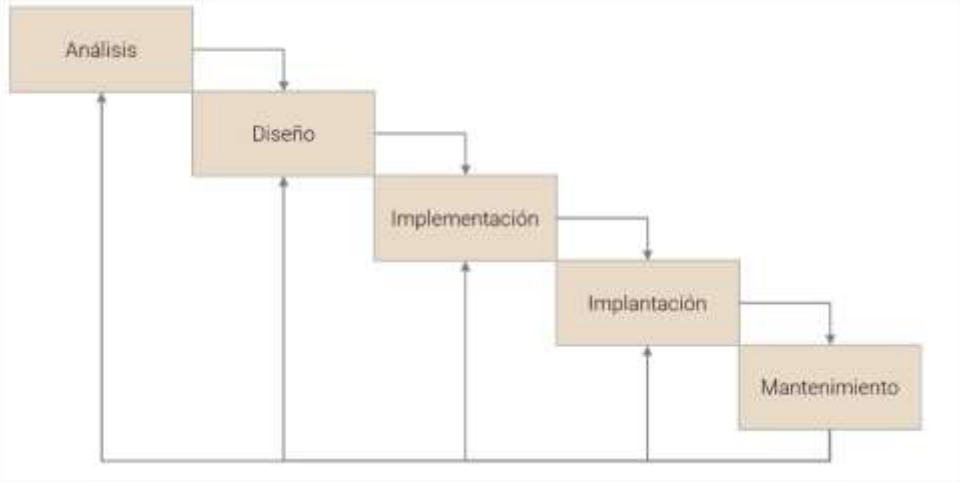
4. Implantación:
    - Puesta en producción del software, instalación en sistemas y realización de pruebas de aceptación.
    - Provisión de documentación para el usuario final.

5. Mantenimiento:
    - Corrección de errores, mejoras en la funcionalidad y adición de nuevas características.
    - Mantenimiento correctivo, adaptativo y perfectivo.

Modelos de Desarrollo de Software:

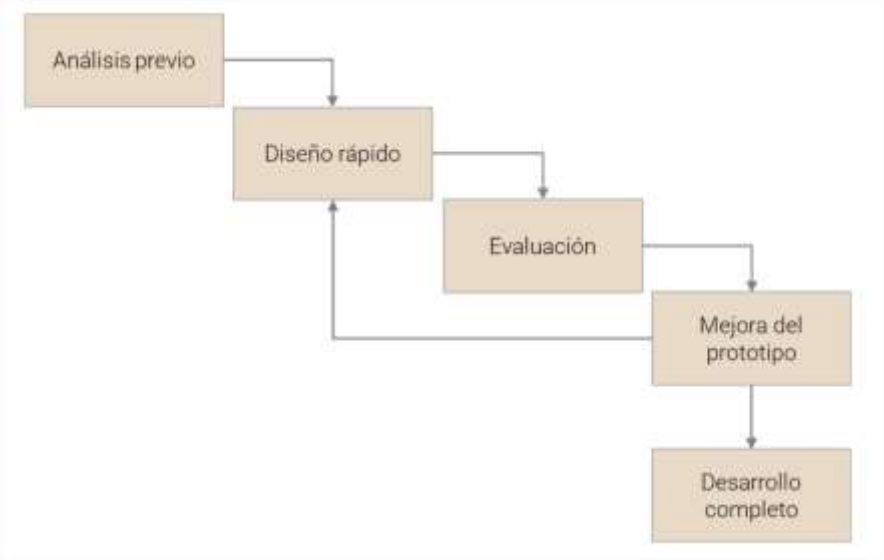
Ciclo de Vida en Cascada:

- Fases secuenciales con documentación de una fase como punto de partida para la siguiente.
- Metódico pero poco realista, ya que presupone claridad total de los requisitos desde el principio.



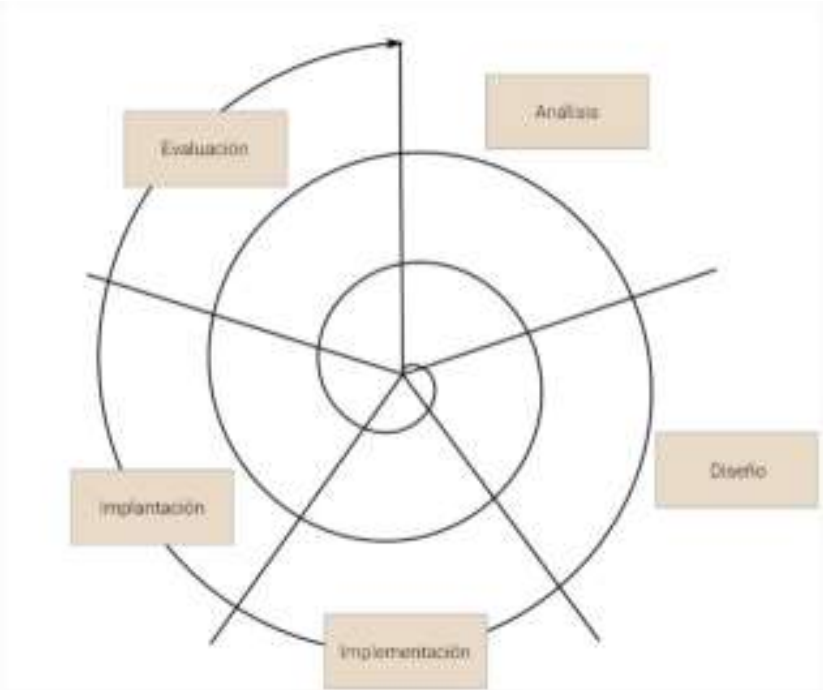
Modelos de Desarrollo Evolutivo:

- Construcción de Prototipos:
- Uso de prototipos para validar requisitos antes del desarrollo.
- El prototipo se rechaza después de la validación.



Modelo en Espiral:

- Similar al iterativo, pero continuo durante toda la vida útil del software.
- No hay un número preestablecido de iteraciones, permitiendo adaptarse a los cambios.



Metodologías Ágiles y Scrum

Introducción:

- La constante evolución de la sociedad demanda adaptación rápida por parte de las empresas y equipos de desarrollo. Las metodologías tradicionales, aunque sistemáticas, son demasiado rígidas, surgiendo así la necesidad de metodologías ágiles capaces de enfrentar los cambios continuos.

Desarrollo Ágil:

- La metodología ágil reconoce la posibilidad de evolución en los requisitos del producto y aborda esto mediante enfoques iterativos y evolutivos. Divide el proceso de desarrollo en bloques pequeños, centrados en proporcionar valor al producto final. La retroalimentación constante con el cliente y revisiones frecuentes son fundamentales.

Scrum:

- Scrum es una metodología ágil que maximiza la rapidez de los entregables, la capacidad de adaptación del equipo y la flexibilidad ante cambios en los requisitos. Divide el proyecto en sprints, periodos de dos semanas a un mes, con un enfoque en colaboración y comunicación.

Roles en Scrum:

- Scrum Master: Garantiza la correcta aplicación del framework, dirige reuniones y protege al equipo de influencias externas.
- Product Owner: Representante del cliente, define necesidades, prioriza y asegura resultados adecuados desde una perspectiva comercial.
- Equipo de Desarrollo: Responsable de entregar partes y el producto final en cada sprint. Equipos autogestionados y autosuficientes.

Reuniones en Scrum:

- Planificación del Sprint: Antes de iniciar el sprint, se definen tareas y objetivos a partir de las prioridades del Product Backlog.
- Scrum Diario: Reunión diaria para sincronizar tareas y discutir el progreso individual.
- Revisión del Sprint: Al finalizar el sprint, se revisa el proceso, se muestran resultados y se evalúan objetivos.
- Retrospectiva: Al concluir cada sprint, se analiza la forma de trabajo, los problemas surgidos y se buscan soluciones para aplicar en el siguiente.