ENTORNOS DE DESARROLLO UD-1

DESARROLLO DEL SOFTWARE (I)

ELEMENTOS DEL DESARROLLO DEL SOFTWARE

ÍNDICE

[CONCEPTOS BÁSICOS 3](#_Toc116113701)

[A. ¿QUÉ ES UN SISTEMA INFORMÁTICO? 3](#_Toc116113702)

[B. ¿QUÉ ES EL SOFTWARE? 3](#_Toc116113703)

[TIPOS DE SOFTWARE 3](#_Toc116113704)

[RELACIÓN HARDWARE-SOFTWARE 4](#_Toc116113705)

[CICLO DE VIDA DEL SOFTWARE 5](#_Toc116113706)

[A. ANÁLISIS 5](#_Toc116113707)

[B. DISEÑO 6](#_Toc116113708)

[C. CODIFICACIÓN 6](#_Toc116113709)

[D. PRUEBAS 6](#_Toc116113710)

[E. EXPLOTACIÓN 7](#_Toc116113711)

[F. MANTENIMIENTO 7](#_Toc116113712)

[G. DOCUMENTACIÓN 7](#_Toc116113713)

[MODELOS DE DESARROLLO DEL SOFTWARE 8](#_Toc116113714)

[A. MODELOS CLASICOS (PREDICTIVOS) 8](#_Toc116113715)

[B. MODELOS EVOLUTIVOS O INCREMENTALES 10](#_Toc116113716)

[C. MODELO DE CONSTRUCCIÓN DE PROTOTIPOS 12](#_Toc116113717)

[HERRAMIENTAS DE APOYO AL DESARROLLO DEL SOFTWARE 12](#_Toc116113718)

[FRAMEWORK 13](#_Toc116113719)

# CONCEPTOS BÁSICOS

## ¿QUÉ ES UN SISTEMA INFORMÁTICO?

Es el conjunto de recursos disponibles para la resolución de problemas mediante el uso de las ciencias de la computación. Incluye:

* Los equipos informáticos: ordenadores, periféricos.
* El software de dichos equipos.
* Los usuarios y administradores.
* Las relaciones entre todos estos elementos.

## ¿QUÉ ES EL SOFTWARE?

Es la suma total de los programas de ordenador, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo.

Un **programa** no es más que una serie de órdenes que se llevan a cabo secuencialmente, aplicadas sobre un conjunto de datos.

La tarea de un **programador** es escoger qué ordenes constituirán un programa de ordenador, el orden en el que deben funcionar y sobre qué datos hay que aplicarlas para llevar a cabo la tarea que se debe resolver.

Por tanto:

* **Software**: Conjunto de programas informáticos que actúan sobre el hardware para realizar una tarea específica.
* **Programa**: Conjunto de instrucciones escritas en un lenguaje de programación que indican a la máquina qué operaciones realizar sobre unos determinados datos.

# TIPOS DE SOFTWARE

Existen tres clasificaciones:

1. Según la **tarea** que realiza:

* **Software de sistema**. Permiten que el sistema pueda funcionar. Por ejemplo, el sistema operativo o los drivers.
* **Software de programación**. Permiten desarrollar programas informáticos. Son los entornos de desarrollo integrados (IDEs).
* **Software de aplicaciones**. Conjunto de programas que tienen una finalidad más o menos concreta.

1. Según el método de **distribución**:

* ***Shareware***. Software que se distribuye con limitaciones (periodo de prueba).
* ***Freeware***. Software que se distribuye de manera gratuita. No suelen permitir ver su código fuente ni modificarlo (si lo permitiera sería *Free Software*).
* ***Adware***. Software que se distribuye de manera gratuita pero que incluyen publicidad. Si se compra la licencia de programa, se elimina esa publicidad.
* **Software multimedia**. Software que permite realizar diferentes acciones.
* **Software de uso específico**. Software para realizar tareas concretas que necesite una organización y/o persona. Requiere de un experto en informática para su creación o adaptación.

1. Según **licencias**:

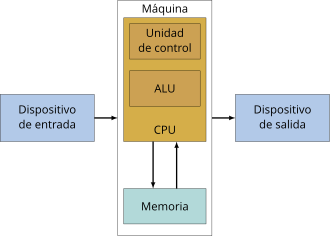
* **Software libre (*Free Software*)**. El autor cede al usuario la libertad de ver, usar y modificar el código. Se conoce como ***General Public License (GPL)***. Un ejemplo de esto sería el SO Ubuntu.
* **Código abierto (*Open Source*)**. El código puede distribuirse, copiarse y modificarse, pero en todo momento se necesita notificar al autor de estos cambios.
* **De dominio público**. Carecen de licencia y/o autor.
* **Software propietario**. Tienen licencia y/o autor. Es todo el software que no es libre.
* **Software comercial**. Aquel que tiene como fin hacer dinero (por ejemplo, vendiéndolo).

# RELACIÓN HARDWARE-SOFTWARE

El **hardware** lo forman los componentes físicos mientras que el **software** lo forma la parte lógica del ordenador.

La **relación** entre ellos es **inseparable**. El software se ejecuta sobre el hardware y el hardware necesita del software para cumplir sus funciones.

La primera arquitectura que se realizó fue la de **Von Neumann**:

* **Memoria principal (RAM)**. Se encarga de almacenar de forma temporal el código binario de los archivos ejecutables y los archivos de datos necesarios.
* **Procesador (CPU)**. Lee y ejecuta instrucciones almacenadas en la RAM, así como los datos necesarios.
* **Entrada/Salida (E/S)**. Recoge nuevos datos desde la entrada, se muestran los resultados, se leen/guardan a disco…

# CICLO DE VIDA DEL SOFTWARE

Es todo el proceso desde que se **concibe una idea** hasta que el **programa está funcionando** en los ordenadores de los usuarios.

Es una tarea bastante complicada, hay que pasar por una serie de fases:

Diagrama

Descripción generada automáticamente

La **documentación** debe estar **permanentemente actualizada** a lo largo de todo el proceso para facilitar el desarrollo.

## ANÁLISIS

Es la primera etapa del proyecto, la más complicada y la que depende de la capacidad del **analista**. Los **errores** en esta fase son los que tienen un **mayor impacto** en el proyecto.

Se hacen uso de diferentes herramientas como los **diagramas de flujo (DFD)**, diagramas de caso de uso, diagramas de transiciones…

Una vez finalizada la fase, tendremos dos documentos:

1. **Documento de especificación de requisitos software**. Determina qué requisitos debe tener el software. Dos tipos de requisitos:
   1. **Funcionales**. Describen qué funciones tendrá que realizar la aplicación. Por ejemplo, determinar que una aplicación bancaria haga transferencias.
   2. **No** **funcionales**. Describen las características del sistema. Por ejemplo, determinar el color de la aplicación.
2. **Documento de diseño de la arquitectura**. Determina las partes en que se va a estructurar y cómo se van a **relacionar entre ellas**.

## DISEÑO

Se va a diseñar cómo se va a hacer, qué partes va a tener, etc.

Se **descompone** y organiza el **sistema** en **componentes** o **bloques**:

* + Diseño de datos.
  + Selección del lenguaje de programación que se va a utilizar.
  + Selección del sistema gestor de base de datos.
  + Determinar los algoritmos necesarios.
  + Herramientas de desarrollo a utilizar (IDEs).
  + Diseño de la interfaz de usuario.

Una vez finalizada la fase obtendremos:

1. **Documento de diseño del software**. Contiene los aspectos anteriormente mencionados.
2. **Plan de pruebas**.

A veces el diseño de arquitectura es considerado como una primera fase del diseño.

## CODIFICACIÓN

Es básicamente escribir el **código fuente**. Se llama código fuente porque el ordenador no lo entiende, requiere de otros procesos para entenderlo.

Se pueden utilizar lenguajes de programación o de otro tipo.

El software pasa por un **entorno de desarrollo**.

## PRUEBAS

Hay que garantizar que el software hace lo que debe hacer y para ello hay que realizar una serie de pruebas que se llevan a cabo sobre un conjunto de datos de prueba (reciben el nombre de **datos límite**) a los que la aplicación es sometida.

Hay dos tipos de pruebas:

* + **Pruebas unitarias**. Se hacen pruebas a los módulos de forma unitaria. Como resultado se obtiene el **documento de procedimiento de pruebas.**

**Junit** es un entorno de pruebas unitarias para Java.

* + **Pruebas de integración**. Se realizan en todos los módulos y se comprueba que la relación entre ellos es correcta. Como resultado se genera el **documento de procedimiento de pruebas de integración.**

## EXPLOTACIÓN

Una vez generado el software, se instala en un entorno llamado **preproducción** (similar al de producción, pero sin entregar al usuario final), una vez probado ahí pasa al entorno de **producción** (o entorno de explotación).

El resultado de esta fase será el **reporte de errores**.

## MANTENIMIENTO

Se encarga de solventar los errores que se hayan encontrado y a implementar mejoras.

Existen diferentes tipos de mantenimiento:

* + **Correctivo**: Se corrigen errores.
  + **Perfectivo**: Se mejora la funcionalidad.
  + **Evolutivo**: Se añaden funcionalidades nuevas.
  + **Adaptativo**: Se adapta a nuevos entornos.

## DOCUMENTACIÓN

Hay que tener en cuenta que no es la última etapa, sino que es una etapa **constante** durante todo el proceso de vida del software, comienza en el análisis y termina cuando el software muere (cuando deja de recibir mantenimiento).

Existen tres tipos de documentación:

|  |  |  |  |
| --- | --- | --- | --- |
|  | GUÍA TÉCNICA | GUÍA DE USO | GUÍA DE INSTALACIÓN |
| Dirigido a | Personal técnico e informático (analistas y programadores) | Usuarios (clientes) | Personal responsable de instalación |
| Aspectos reflejados | * El diseño de la aplicación. * La codificación de los programas. * Las pruebas realizadas. | Descripción de la funcionalidad de la aplicación:   * Forma de comenzar a ejecutar la aplicación. * Ejemplos de uso del programa. * Requerimientos software de la aplicación. * Solución de los posibles problemas que se pueden presentar. | Puesta en marcha. |
| Objetivo | * Facilitar un correcto desarrollo. * Realizar correcciones en los programas. * Permitir un mantenimiento futuro. | Dar a los usuarios finales toda la información necesaria para la utilización de la aplicación. | Implantación de la aplicación de forma segura, confiable y precisa. |

# MODELOS DE DESARROLLO DEL SOFTWARE

## MODELOS CLASICOS (PREDICTIVOS)

Dentro de estos tenemos 3 modelos:

1. **Modelo en cascada**: consiste en seguir paso a paso las diferentes fases del ciclo de desarrollo.

Diagrama

Descripción generada automáticamente

Se caracteriza por:

* + - * Ser el más más clásico y el de mayor antigüedad.
      * Las fases han de realizarse en el orden indicado sin vuelta atrás.
      * Cualquier error en las fases iniciales ya no será subsanable (aunque se haya detectado más adelante) hasta la fase de mantenimiento.
      * El resultado de una fase es la entrada de la siguiente fase.
      * Es un modelo bastante rígido que se adapta mal al cambio continuo de especificaciones.
      * Este escaso margen de error lo hace **prácticamente imposible de utilizar**. Sólo es aplicable en pequeños desarrollos.

1. **Modelo en cascada con retroalimentación**:

Se caracteriza por:

* + - * Es uno de los más utilizados.
      * Permite **volver atrás** (a la fase anterior), es decir, se introduce la retroalimentación.
      * Si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.
      * Es el modelo perfecto si el proyecto es rígido y los requisitos están claros.

1. **Modelo en V**:

Imagen que contiene Diagrama

Descripción generada automáticamente

Se caracteriza por:

* + - * Modelo muy parecido al modelo en cascada.
      * Visión jerarquizada con distintos niveles.
      * El resultado de una fase es la entrada en la siguiente.
      * Permite que en cada etapa se puede **validar** que lo que se ha hecho, esté bien.
      * No es de los que más se usan.

## MODELOS EVOLUTIVOS O INCREMENTALES

Son modelos modernos y tienen en cuenta la naturaleza cambiante y evolutiva del software.

1. **Diagrama

   Descripción generada automáticamenteModelo en espiral (iterativos).** Simplifica las fases y las repite de forma cíclica.

Consiste en realizar ciclos del modelo en cascada, pero sin que las fases estén encorsetadas, lo que permite que haya muchos cambios en base a cambio de requisitos.

Corre el riesgo de alargarse en el tiempo y aumentar el coste del desarrollo debido a esos cambios.

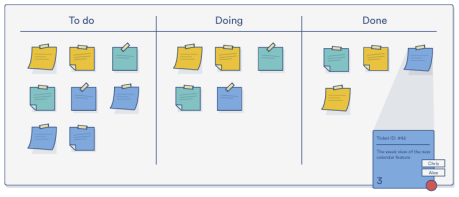
1. **Metodologías ágiles.** Aparecen a partir del año 2001 los principios en los que se basan estas metodologías. Es lo que se conoce como Manifiesto Ágil:
   * **Individuos e interacciones** sobre procesos y herramientas.
   * **Software funcionando** sobre documentación extensiva.
   * **Colaboración con el cliente** sobre negociación contractual.
   * **Respuesta ante el cambio** sobre seguir un plan.

Son métodos basados en el desarrollo iterativo e incremental en el que los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Además, el trabajo es realizada mediante la colación de equipos auto-organizados y multidisciplinares, inmersos en un proceso compartido de toma de decisiones a corto plazo.

Existen varios tipos:

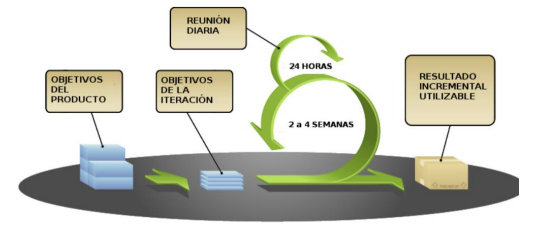
1. **Kanban**. También conocido como **“sistema de tarjetas”**. Fue desarrollado por Toyota y controla por demanda la fabricación de los productos necesarios en la cantidad y tiempo necesarios.

Es un método enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos.



1. **Scrum**. Es un tipo de desarrollo incremental basado en iteraciones (***sprints***) regulares cada 2-4 semanas.

Al inicio de cada sprint se establecen los objetivos priorizados (***sprint backlog***). Una vez que se termina la iteración se obtiene una entrega parcial utilizable para el cliente.

Durante los *sprints* hay reuniones diarias lo que aumenta la comunicación del equipo.

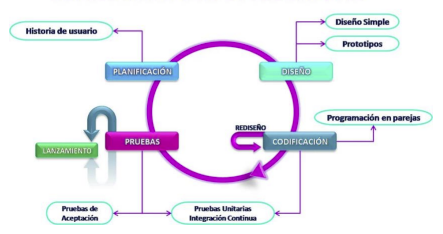
1. **XP (Programación Extrema)**. Se basa en potenciar las relaciones interpersonales del equipo de trabajo para facilitar el aprendizaje y el buen clima de trabajo.

Favorece la retroalimentación continua entre el cliente y el equipo de desarrollo.

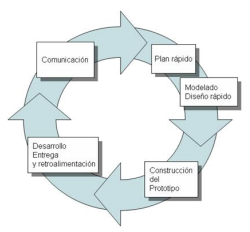
Es el mejor método para proyectos con requisitos imprecisos y cambiantes.

Tiene un diseño sencillo con pequeñas mejoras continuas. Hay una integración continua y siempre se programa por parejas.

Otra de las cosas positivas es que el cliente se integra dentro del equipo de desarrollo. En este método la propiedad del código es compartida entre cliente y el equipo desarrollador.



## MODELO DE CONSTRUCCIÓN DE PROTOTIPOS

Es adecuada cuando los requisitos no están especificados claramente, por ejemplo, porque no hay experiencia previa o porque el cliente no se acuerda de decir cosas obvias para él y el desarrollador se queda sin información valiosa que determina la dirección del desarrollo o porque simplemente cambie de idea a mitad de desarrollo.

El proceso de desarrollo es crear un prototipo durante la fase de análisis y se prueba con el cliente para probarlo e irlo cambiando. Se repite este proceso durante las veces que sean necesarias hasta que el software esté desarrollado.

# HERRAMIENTAS DE APOYO AL DESARROLLO DEL SOFTWARE

Son herramientas que permiten automatizar parte del trabajo. Esto permite que los desarrolladores puedan centrarse en los requerimientos del sistema.

Se denominan herramientas **CASE** **(*Computer Aided Software Engineering*)** y permiten:

* Mejorar la planificación del proyecto.
* Dar agilidad al proceso.
* Poder reutilizar partes del software en proyectos futuros.
* Hacer que las aplicaciones respondan a estándares.
* Mejorar la tarea del mantenimiento de los programas.
* Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

Pueden ayudar durante todas las fases del diseño:

* **U-CASE (*Upper-Case*).** En la planificación y análisis.
* **M-CASE (*Medium-Case*).** En el análisis y diseño.
* **L-CASE (*Lower-Case*).** En la programación, pruebas y documentación.

Ejemplos de herramientas son: Microsoft Project, Rational Rose, JDeveloper, etc

# FRAMEWORK

Son plataformas software que nos dan parte de los módulos software desarrollados y el programador puede partir de ellos para sus desarrollos sin necesidad de empezar de 0. Esto ayuda a agilizar los proyectos.

En estas plataformas software se definen programas soportes, bibliotecas, lenguajes interpretados, etc.

Ejemplos:

* **.NET.** Framework para desarrollar aplicaciones sobre Windows.
* **Spring de Java.** Es un conjunto de bibliotecas para el desarrollo y ejecución de aplicación Java.
* **Angular.** Framework de JavaScript para aplicaciones web.