**UNIDAD 1. INTRODUCCIÓN Y CONCEPTOS**

**Hardware - Software**

**Partes o elementos** de un dispositivo informático:

* **Hardware (HW):** Conjunto de componentes físicos o materiales que componen un dispositivo o PC.
* **Software (SW):** Conjunto de programas y aplicaciones que residen y se ejecutan sobre el HW.

Ambos elementos **se complementan** para que un dispositivo pueda funcionar.

**Tipos de Software**

**Según la función:**

* **Sistemas operativos (S.O.):** es el software base que ha de estar instalado y configurado en un ordenador para que las aplicaciones puedan ejecutarse y funcionar.
* **Software de programación:** herramientas para desarrollar otros programas informáticos. Ejemplo: **entornos de desarrollo**.
* **Aplicaciones:** son un conjunto de programas con finalidad más o menos concreta.

**Desarrollo de software**

Es el **proceso** desde que se detecta una necesidad o se concibe una idea, hasta que se dispone de una herramienta software en funcionamiento que la resuelve, o más bien, hasta que esta herramienta o solución deja de utilizarse:

1. **Inicio**: se detecta una necesidad o se concibe una idea.
2. **Objetivo**: desarrollar e implantar una **solución software**, que resuelva la necesidad y cumpla los acuerdos previstos (tiempo, forma y coste).
3. **Final**: la herramienta o solución deja de utilizarse y mantenerse.

**Fases clásicas de un proyecto de desarrollo de SW**

Análisis: se recopilan los requerimientos y se analizan en profundidad.

Diseño: se definen en detalle las funcionalidades necesarias para cubrir los requerimientos.

Desarrollo: se codifica la solución en base al diseño realizado.

Pruebas: se verifican y testean de los resultados del desarrollo de la solución.

Implantación: etapa de puesta en marcha de la solución. Instalaciones. Capacitación de usuarios.

Mantenimiento y evolución: esta etapa puede durar toda la vida útil de la solución.

* Análisis:

Es la primera fase de un proyecto. Las demás dependen de su precisión.

¿Qué hay que hacer en esta fase? Objetivos:

* Definir los requerimientos, funcionales o no funcionales.
* Definir una estrategia.
* Definir una planificación de tareas e hitos.

La comunicación e implicación de todo el personal implicado en el proyecto es un factor clave para definir ¿qué debe realizar la solución?

* Diseño:

¿Cómo se van a resolver los requerimientos? Dependerá de la naturaleza de los requerimientos, de la estrategia y de las circunstancias del proyecto.

¿Qué hay que hacer en esta fase?

* Análisis técnico de los requerimientos: mediante gráficos, diagramas, algoritmos, textos, etc.
* Decisiones para organizar y cuantificar recursos: personas, infraestructura, Entornos de desarrollo, lenguaje/s de programación, gestor de bases de datos.
* Desarrollo o codificación:

¿Qué hay que hacer en esta fase? Objetivos:

Codificar mediante uno o varios lenguajes de programación.

El código puede pasar por diferentes estados:

* Código fuente: lo generan los programadores. Habitualmente mediante el uso de lenguajes de programación de alto nivel.
* Código objeto: código binario resultado de compilar el código fuente. Código intermedio, no es inteligible por las personas, y todavía no es ejecutable por un sistema operativo.
* Código ejecutable o código máquina: código binario resultante de enlazar o linkar el código objeto con rutinas y bibliotecas utilizadas. El sistema operativo lo podrá cargar en memoria RAM y ejecutarlo directamente.
* Pruebas:

Conforme se va disponiendo de elementos desarrollados, es crucial realizar pruebas.

¿Qué hay que hacer en esta fase? Objetivos:

Se distinguen múltiples tipos de pruebas:

1. Unitarias: se testean pequeños elementos aislados.
2. Pruebas de integración: se testea que los elementos se integran correctamente.
3. Pruebas sistema: se testea el funcionamiento de un proceso completo.

El resultado de las pruebas puede provocar que algunos elementos retrocedan, incluso a la fase de análisis.

* Implantación:

Cuando una solución alcanza cierta fiabilidad, se prepara el escenario donde se usará.

¿Qué hay que hacer en esta fase?:

* Instalación de la solución en la infraestructura donde se ejecutará.
* Configuración de la solución.
* Capacitación de usuarios.
* Explotación/producción.
* Mantenimiento y evolución:

La naturaleza del software es cambiante, necesitará actualizarse, adaptarse y evolucionar.

¿Qué hay que hacer en esta fase?:

El mantenimiento de una solución software puede durar toda la vida del producto.

Existen varios tipos de mantenimiento:

* Correctivo: corrección de errores y fallos de funcionamiento.
* Adaptativo: modificaciones y actualizaciones de la funcionalidad existente.
* Evolutivo: nuevas funcionalidades, nuevas versiones.

**Modelos del ciclo de vida del software**

En función de la secuencia que se establezca entre las fases o etapas y de las reglas que establezcan entre ellas, podemos encontrar diferentes modelos:

En cascada:

* Es el modelo más clásico y menos flexible.
* Una variante muy interesante y flexible es el modelo en cascada con retroalimentación.

Modelos más actuales, que tienen en cuenta la naturaleza altamente cambiante del SW, son los modelos evolutivos:

En espiral:

* Es un modelo flexible.
* El software se va construyendo repetidamente en forma de versiones mejoradas.
* Se apoya en las fases clásicas.
* Puede ser complejo en su implementación.
* Permite reducir los riesgos en cada versión.

Existen más modelos (modelo en V, modelo por prototipos, etc.), y más que surgirán…

No existen modelos “buenos” y “malos”, dependerá de las circunstancias del proyecto.

Es habitual utilizar una combinación de modelos, en mayor o menor medida, a la hora de desarrollar un proyecto.

**Documentación**

La documentación es una labor muy importante.

Se puede considerar como una fase clásica más.

Desde la antigüedad, la documentación ha permitido universalizar los detalles de un proyecto.

**Mejora e integración continua**

Mejora continua:

Consiste en repetir este procedimiento de forma permanente:

* Analizar procesos y estudiarlos en detalle.
* Realizar adecuaciones para mejorarlos y minimizar los errores.

Integración continua:

* Consiste en integrar las mejoras de varios equipos o desarrolladores (contribuidores) en un único proyecto de software, de forma periódica.

**Gestión ágil de proyectos**

Consiste en realizar entregas de forma continua e iterativa.

Las personas y su interacción se posicionan como núcleo del proyecto. Por encima de las herramientas y los procedimientos.

La estrecha colaboración y comunicación entre los componentes del proyecto es una necesidad primordial para obtener éxito.

La unidad básica de ejecución es la iteración.

En cada iteración se debe marcar una serie de objetivos, de forma que cada iteración va aportando valor a la solución.

Existen definiciones específicas de pautas de trabajo basadas en la gestión ágil de proyectos, como, por ejemplo:

* Scrum.
* Kanban.

También existen herramientas orientadas a este tipo de gestión:

* Trello.
* Jira.

**Lenguajes de programación**

Son idiomas creados artificialmente para que una máquina los pueda ejecutar.

Formados por un conjunto de símbolos y unas normas aplicables.

Clasificación de los lenguajes:

Según la proximidad con el lenguaje humano:

* Alto nivel: próximos al lenguaje humano. Los usados habitualmente.
* Bajo nivel: más próximos al funcionamiento interno de una máquina:
* Ensamblador.
* Código máquina.

Según la técnica de programación utilizada:

Estructurados: conjunto ordenado de sentencias con inicio y final (Pascal, C, Fortran, Cobol, BASIC, etc.).

Modulares: permiten dividir un código en fragmentos o módulos reutilizables (Pascal, C, Cobol, RPG, etc.).

Orientados a objetos: los programas son un conjunto de objetos que interactúan entre sí. Es el paradigma más extendido (C++, C#, Delphi, Java, JavaScript, PHP, etc.).

Visuales: permiten generar programas mediante la manipulación de elementos gráficos (Scratch, Bolt, MakeCode, etc.).

**Entornos de desarrollo**

En inglés Integrated Development Environment (IDE).

Aplicación con un conjunto de herramientas para facilitar el desarrollo de software.

Características:

1. Diseñados para maximizar la productividad.
2. Ofrecen utilidades para la creación y depuración de software.
3. Facilitan la reutilización de componentes.
4. Ayudan a reducir la configuración.
5. Algunos soportan múltiples lenguajes de programación.
6. Permiten personalización mediante la activación o desactivación de herramientas y componentes.

Ventajas:

Facilita el aprendizaje.

* Dispone de mecanismos de ayuda avanzada, como auto completar.
* Proporciona mensajes de alerta o error durante todo el proceso.
* Incrementa la productividad del desarrollador mediante herramientas y plantillas.

Inconvenientes:

* Suelen consumir muchos recursos.
* Generan dependencia, cierta cautividad.

Ejemplos: en el mercado podemos encontrar múltiples opciones. Por ejemplo:

Visual Studio:

* Desarrollado por Microsoft y comercializado bajo licencia privativa.
* Es compatible con lenguajes como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP, etc.

Eclipse:

* Actualmente lo gestiona una fundación que fomenta el código abierto.
* Es multiplataforma (Windows, Linux y Mac).
* Es posible desarrollar usando Java, C, C++, JSP, perl, Python, Ruby, PHP, etc.

NetBeans:

* Desarrollado por Apache (Oracle), basado en código abierto.
* Es el “oficial” de Java, también podemos desarrollar en otros lenguajes como PHP, C, C++, etc.

**U2. UML COMPORTAMIENTO: CASOS DE USO UML INTRODUCCIÓN.**

**¿Qué es UML?**

Lenguaje de modelado unificado. Por sus siglas en inglés, Unified Modeling Language.

Según la RAE….

* Modelar: configurar o conformar algo.
* Unificar: hacer de muchas cosas, una o un todo.

Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un estándar desde el año 2004.  
Es un lenguaje gráfico utilizado para analizar, especificar, diseñar, visualizar, construir y documentar un sistema, servicio, aplicación, desarrollo o solución.

UML no es un lenguaje de programación.

Describe objetos, procesos, estados y relaciones en un sistema.

Su elaboración se realiza durante las fases de análisis y/o diseño y su uso se extiende durante el resto del proyecto, como casi toda la documentación importante.

Se basa en el uso de diagramas.

**¿Qué es un diagrama?**

Un diagrama es una representación de la realidad.  
 En nuestro caso, dicha realidad podría ser un proceso, un sistema o las relaciones existentes entre ciertos objetos de un sistema.

Se trata de una representación gráfica, habitualmente mediante dibujos geométricos.  
Es necesario conocer el significado de las formas y demás caracterización utilizada para poder elaborar o interpretar un diagrama.

**Tipos de diagramas UML**

En UML encontramos 2 tipologías principales de diagramas:

Diagramas de comportamiento: definen el comportamiento dinámico de un sistema. Tipos:

1. Diagramas de casos de uso.
2. Diagramas de actividades.
3. Diagramas de secuencias.
4. Diagramas de tiempos.

Diagramas estructurales: sirven para definir la estructura estática de un sistema (foto). Tipos:

1. Diagramas de clases.
2. Diagramas de componentes.

U3 UML COMPORTAMIENTO: CASOS DE USO

¿Qué es un diagrama de casos de uso?  
Es un tipo de diagrama de comportamiento, por lo tanto, define los aspectos dinámicos de un sistema.  
Dentro del ciclo de vida del SW, los diagramas de casos de uso formarían parte de la fase de análisis.  
Se recomienda que sean de los primeros diagramas a elaborar.  
Facilitan la especificación de requerimientos.

Representan un sistema desde los distintos puntos de vista de los distintos perfiles de usuario que interactuarán con él.  
Un diagrama de casos de uso describe un sistema, teniendo en cuenta sus distintas variedades de interactuación.

Cada una de estas variedades, equivaldría a un caso de uso.  
Los casos de uso son ideas sencillas y prácticas que no requieren habilidades ni conocimientos tecnológicos avanzados para ser analizadas.

Cada caso de uso se puede componer internamente de una secuencia de eventos y es iniciada por un tipo de usuario o agente, llamado actor.

Permiten modelar los requisitos funcionales y los perfiles de usuario involucrados en su utilización, para favorecer el análisis.

Elementos: Sistema

Es uno de los elementos más importantes en un diagrama de casos de uso.

Hace referencia a solución software integral que vamos a estudiar o analizar, para implementarla.

Se representa mediante un rectángulo.

Elementos: Actores

Cada uno de los agentes o tipos de usuario que interactúa con un sistema, se denomina actor.

Un actor representa un rol externo al sistema que interactúa con el propio sistema.  
No son únicamente humanos, un actor podría ser otro sistema, dispositivo o el tiempo.  
A modo de ejemplo, actores podrían ser:  
Un usuario que solicita la compra de un producto.  
Un dispositivo externo que mediante una aplicación intercambia datos con nuestro sistema.

Un temporizador que ejecuta un proceso cada 10 segundos.  
Un actor representa un perfil concreto de usuario, de modo que:  
Un usuario puede representar a varios actores.

Un actor puede ser representado por varios usuarios.

Elementos: Casos de uso

Los casos de uso representan un proceso o acción. Ejemplo: crear pedido, mover ficha, etc.  
Gráficamente, los casos de uso se representan en UML mediante una elipse.  
No podrán estar aislados en un sistema, siempre estarán enlazados con uno o varios actores y/o con uno o varios casos de uso.  
Para identificar los casos de uso existentes en un sistema, debemos averiguar qué funciones e interactuaciones realiza cada actor en el sistema.

Por ejemplo: generar pedido, solicitar listado de citas pendientes, actualizar catálogo de productos, mover un personaje en un laberinto, etc.  
Los casos de uso deben abstraer en gran medida lo fundamental o principal respecto a las interactuaciones de un sistema, por muy grande que éste sea.

En fases posteriores del análisis, se concretarán y detallarán las instrucciones que contiene cada caso de uso.

¿Cómo generar diagramas UML?

• Para poder elaborar un diagrama UML, como primer paso, debemos estudiar y analizar con el mayor detalle posible el escenario o supuesto sobre el que vamos a trabajar.

• La precisión en cada uno de los pasos favorece la evolución del proyecto.

• Si nos equivocamos u olvidamos algún aspecto importante al principio, nos costará un esfuerzo considerable durante las fases posteriores del proyecto.

Análisis y diseño de diagramas de casos de uso

• Identificación de elementos: uno de los resultados del análisis realizado, debe ser el inventario de elementos que formarán el diagrama. En nuestro caso, debemos identificar:

• Sistema

• Actores

• Casos de uso

• Relaciones

Análisis y diseño de diagramas de casos de uso

• Sistema: debemos tener claros los límites y objetivos de la solución software que vamos a analizar.

Análisis y diseño de diagramas de casos de uso

• Actores: los actores de un sistema se representarán como entidades externas al sistema, que interactúan con él.

Análisis y diseño de diagramas de casos de uso

• Casos de uso: son elementos internos del sistema y para identificarlos, habrá que estudiar las necesidades e interacciones que llevará a cabo cada uno de los actores. Podemos emplear preguntas del tipo:

• ¿Cuáles son las tareas principales de cada actor?

• ¿Qué procesos puede lanzar o ejecutar cada actor?

• ¿Tendrá un actor que consultar y/o modificar información del sistema?

• ¿Deberá el actor informar al sistema sobre cambios externos?

• …

Análisis y diseño de diagramas de casos de uso

• Ejemplo:

• Dado un sistema para la gestión de proyectos software.

• Habrá una persona que se encargará de planificar y asignar tareas en cada proyecto, ejerciendo las funciones de coordinación.

• También habrá un equipo de consultoría que analizará y diseñará los procedimientos del proyecto.

• Por último, un equipo de desarrollo codificará la funcionalidad del proyecto.

Análisis y diseño de diagramas de casos de uso

• Ejemplo:

• Pensemos ahora en un videojuego de ajedrez online.

• Los únicos que interactúan son l@s Jugadore/as.

• Las interacciones que pueden realizar son:

• Registrarse.

• Iniciar sesión.

• Configurar parámetros.

• Iniciar partida:

• Mover ficha.

• Salir.