Entornos de Desarrollo

1º DAM/DAW

Índice

Unidad Didáctica 1:

Sistemas informáticos

1.Sistema informático

Un ***sistema informático*** es cualquier dispositivo electrónico que permita la libre ejecución de una serie de programas informáticos, los cuales permitan realizar una serie de tareas, entre ellas, guardar y procesar información. Por ejemplo, un ordenador portátil es un sistema informático, porque permite ejecutar programas. Sin embargo, un reloj digital clásico no lo es porque las entradas que posee (botones), tan solo nos permiten gestionar una serie de comportamientos pre-establecidos en fábrica.

Algunos sistemas informáticos parecen no serlo porque el usuario final tan solo puede usar un conjunto limitado de opciones, mientras que un administrador sí que puede ejecutar todo tipo de programas. Por ejemplo, el expendedor de tiques de transporte público de la figura realmente es un equipo informático, algo que se puede adivinar por la pantalla (está ejecutando un programa con un interfaz de usuario), aunque el usuario final tan solo puede realizar ciertas funciones muy concretas.

Existen cierto hardware que realmente es un sistema informático, aunque más limitado, en el que un hardware ejecuta un programa escrito en memoria no volátil, a modo de firmware, como un router o una impresora. Este programa que ejecutan puede cambiarse actualizando dicho firmware. Estos sistemas informáticos se llaman ***empotrados***.

Ej. 1: ¿Cuáles de los siguientes son equipos informáticos, y cuáles son sistemas informáticos empotrados?:

- Un móvil.
- Una calculadora clásica.
- Un cajero de un banco.
- Un proyector.
- Una máquina expendedora de refrescos.
- Un smartwatch.
- Un frigorífico con conexión a Internet.
- Una PlayStation 5.
- Un iPod nano.

Solución: Un móvil, un smartwatch y una PlayStation son dispositivos que ejecutan aplicaciones, por lo que son, inequívocamente, sistemas informáticos.

Una calculadora clásica no es un sistema informático porque su función está implementada por hardware, y no puede ejecutar programas.

Un cajero de banco, actualmente, está implementado por un PC o similar, por lo que si es un sistema informático. Pudiera parecer lo contrario porque el usuario final solo tiene disponibles ciertas acciones, pero un administrador podrá configurar muchas más cosas. De hecho, el programa en sí que el usuario usa, es una aplicación.

Un proyector, una máquina expendedora de refrescos, un frigorífico con conexión a internet y un iPod nano son sistemas empotrados, porque ejecutan un programa específico en su firmware.

Ej. 2: De los siguientes ¿cuales son sistemas informáticos, y cuáles son empotrados?:

- Un portátil.
- Automóvil (los dispositivos internos, no Android Auto ni similar).
- Un reloj.
- Un termómetro digital.
- Sistemas de navegación GPS
- Un voltímetro.
- SmartTV

Capas de un sistema informático

El ***hardware*** es la parte "física" del sistema informático, es decir, todos los componentes electrónicos y el resto de materiales que pueden tocarse físicamente. El resto de elementos (aplicaciones y sistema operativo) son ***software***.

Aplicaciones	**Ej:** Writer, Firefox, Telegram, Steam, VSCode **Obj:** Tareas usuario: crear documento, jugar, etc.
Sistema operativo	**Ej:** Windows, Linux, MacOS, Android, CellOS. **Obj:** Organizar la ejecución de programas y Ocultar los detalles del hardware.
Hardware	**Ej:** PC, Arduino, Tablet, PS4. **Obj:** Proporcionar soporte para ejecución.

Para gestionar todos esos recursos hardware, está el ***sistema operativo***, que provee de una plataforma común a los programas y gestiona los recursos y también la ejecución de programa, para que éstos puedan ejecutarse, independientemente de los detalles hardware.

Las ***aplicaciones*** son los programas que se ejecutan para que el usuario pueda realizar la tareas que desee.

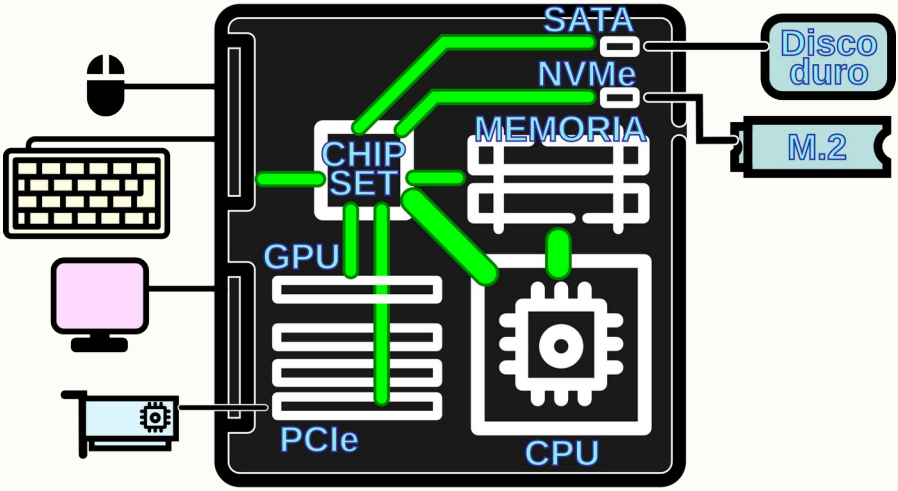
Ej. 3: Pon más ejemplos de aplicaciones. Esas aplicaciones, ¿necesitan a otros programas para ejecutarse (Google Maps en PC, etc.)?

Ej. 4: Pon un ejemplo de hardware que componga un sistema informático.

Ej. 5: Pon un ejemplo de sistema operativo. Dicho sistema operativo ¿está creado para un sistema hardware concreto, o para varios de ellos?

2.Hardware

La ***Unidad Central de Proceso (CPU)*** es la que ejecuta los programas, compuestos por muchas instrucciones. Para cada instrucción ejecutada, el ordenador envía las órdenes necesarias al resto de elementos.



La CPU tiene una conexión dedicada con la ***memoria***, donde se encuentran tanto los programas a ejecutar (juego, editor de texto, etc.) como los datos a utilizar. La CPU carga desde memoria esos programas y los ejecuta, leyendo y modificando los datos según éstos programas indiquen. Toda la información almacenada en la memoria se pierde cuando el computador se apaga.

Para comunicarse con el resto de elementos, la CPU tiene una conexión de alta velocidad con el ***chipset***, el cual se comunica con el resto de elementos para intercambiar datos entre los distintos elementos.

Conectados al chipset están los ***dispositivos de entrada y salida***, que intercambian, gestionan o almacenan información. Así, por ejemplo, puede haber un programa ejecutándose en la CPU que sea un editor de texto, que puede emitir la orden de mover ciertos datos desde el disco duro (correspondientes a un archivo de texto que se desea editar) hacia la

memoria. Un dispositivo es de entrada cuando el computador recibe datos (o le entran datos), y es de salida cuando el computador emite datos (saca datos). Muchos dispositivos son, principalemente, tato de entrada como salida, como por ejemplo una .tarjeta de red

El ***almacenamiento*** es un tipo de dispositivo de entrada/salida que almacena la información de forma permanente. Es más lenta pero de mayor tamaño que la memoria principal. Ejemplos son el disco duro y el almacenamiento interno de un smartphone. No hay que confundir este elemento con la memoria principal.

Otro tipo de dispositivo de dispositivo de entrada/salida son los ***periféricos***, que se encuentran en el exterior del sistema informático, conectados a éste a través de un cable, wifi, bluetooth o similar. Ejemplos de periféricos serían el ratón o el monitor.

Ej. 6: Un programa que se está ejecutando lee constantemente el teclado, en espera de que el usuario pulse una tecla, guardando la tecla pulsada en memoria. Al ser pulsada, esa tecla es mostrada en el monitor. Identifica los pasos generales de intercambio de datos entre los distintos elementos del computador.

Solución: Se realiza un envío de datos desde el teclado hasta la memoria. Eventualmente, la CPU lee esa memoria y envía la orden de escribir el carácter asociado a la tecla pulsada hacia el monitor.

Ej. 7: Otro programa que se está ejecutando trata de leer ciertas texturas guardadas en el disco duro y enviarlas a la GPU externa. Identifica los pasos generales de intercambio de datos entre los distintos elementos del computador.

Solución: la CPU inicia la lectura en disco, momento en el cual el chipset toma el control, moviendo los datos (las texturas en este caso), a la memoria de la GPU externa.

Ej. 8: Identifica si los siguientes dispositivos son dispositivos de entrada, de salida o ambos. Identifica también cuáles de ellos de tipo de almacenamiento, y cuáles son periféricos, o pueden serlo:

- Un teclado.
- Cascos de audio sin micro.
- Disco duro interno.
- Disco duro externo.
- Un ratón.
- Una pantalla.
- Una pantalla táctil.
- Altavoces externos.
- Cascos de Realidad Virtual.
- Un pendrive.
- Un mando de juego.
- Una tarjeta de red / wifi.
- Un "pinganillo" bluetooth para un móvil.

Solución: son de entrada: el teclado, el ratón y el mando de juego, puesto que, desde la perspectiva de la CPU, entran datos desde esos dispositivos. De forma inversa, son de salida los cascos, la pantalla, los altavoces y el pinganillo. El resto son de entrada y salida.
Periféricos son el teclado, los casos, el disco duro externo, los altavoces externos, los cascos de realidad virtual, el pendrive, el mando de juego y el pinganillo, pues son externos al PC (se conectan al PC por cable, bluetooth u otro).

Arquitectura

Existen diversas arquitecturas de computadores. Cada arquitectura consiste en definir una serie de características que un computador que sea acoja a dicha arquitectura debe implementar: el juego de instrucciones, microarquitectura, diseño lógico, etcétera. Ejemplos de arquitecturas son:

- ***x86***. Es la arquitectura de los antiguos PCs.
- ***x86_64*** (también llamado ***amd64***). Creada por AMD en 1999, es la arquitectura actual empleada en los PCs, y es retrocompatible con la arquitectura x86, es decir, puede ejecutar programas

creados con la arquitectura x86. Domina el mercado de escritorio y de consolas.

- ***ARM***. Es una arquitectura energéticamente eficiente, aunque no tan buena en computación intensiva (gaming, servidores, etc.). Domina el mercado de móviles y otros dispositivos portátiles, incluidos los computadores de Apple.
- ***RISC-V***. Libre y abierta, puede usarse sin pagar licencia. Está pensada para implementaciones sencillas, rápidas, pequeñas y de bajo consumo.

Los programas compilados para una arquitectura no son ejecutables en otra. Hay programas que son capaces de generar programas para varias plataformas.

Sistemas actuales

Actualmente, hay diversos mercados objetivo para los sistemas informáticos, algunos de los cuales son:

- ***PCs de escritorio y ordenadores portátiles***. Tienen arquitectura x86_64, y procesadores (CPUs) con 6 a 16 núcleos. Suelen ejecutar Windows o Linux.
- ***Portátiles Apple.*** Tienen una micorarquitectura propia dentro de la arquitectura ARM y ejecutan MacOs.
- ***Móviles.*** Tienen arquitectura ARM, con núcleos de rendimiento y núcleos de eficiencia energética. Ejecutan Android en móviles diseñados para este sistema, o iOs para iphones, aunque existen otros sistemas operativos como LineageOS.
- ***Servidores.*** Suelen ser computadores con arquitectura x86_64 con hardware de virtualización, gran cantidad de nucleos y, a menudo, varios procesadores.

- ***Routers.*** Aunque algunos aún usan la arquitectura MIPS (una arquitectura de bajo consumo y libre), la mayoría de ellos usan hoy día ARM. Los grandes routers de gran rendimiento, que pueden verse en data centers y organizaciones similares, suele utilizar x86_64. Suelen emplear sistemas propietarios o sistemas Linux adaptados.

Uno de los límites más importantes en los sistemas es su fuente de energía. Por ello, servidores y Pcs de escritorio pueden conseguir mucha más capacidad de cómputo que otros sistemas alimentados por batería.

Ej. 9: ¿Qué arquitecturas más comunes usan hoy día los portátiles? ¿Qué arquitectura usa una PS5?

Solución: los portátiles usan, hoy día, la arquitectura x86_64, ya que realmente son PCs, que también usan dicha arquitectura. La excepción a ello son los portátiles Apple, que usan una arquitectura ARM. Una PS5, si consultamos la página de la Wikipedia, vemos que es realmente un PC modificado, por lo que su arquitectura es también x86_64.

Ej. 10: ¿Podría el juego Cyberpunk 2077, que apenas va en una PS4, jugarse en un móvil?

Solución: Aunque hoy día los móviles son bastante potentes, tienen la limitación de la batería y un procesamiento 3D que no puede llegar al que puede mover un PC o un portátil con una tarjeta gráfica dedicada. Además, los desarrolladores de Cyberpunk 2077 se vieron forzados a introducir numerosos hacks para que el juego fuera ejecutable en PS4. Por tanto, hacer que se ejecute en un móvil sería inviable sin muchos sacrificios. Observa que Nintendo Switch es parecida en potencia a ciertos móviles, y es incapaz de ejecutar muchísimos juegos AAA.

3.Sistema operativo

El sistema operativo gestiona es una primera capa de software que gestiona el hardware subyacente, y permite ejecutar programas.

Sistema de arranque

Cuando un sistema informático se enciende, el hardware automáticamente ejecuta un pequeño programa que está escrito en la placa base, llamado BIOS, UEFI, Bootloader u otro según el hardware concreto. Este programa, que puede ser configurado, se encarga de cargar el núcleo del sistema operativo desde un dispositivo de almacenamiento, tras lo cual le otorga el control.

Tras ello, este núcleo de sistema operativo carga el resto de si mismo, y también ejecuta otros programas, como la interfaz de usuario que permite al usuario seleccionar y ejecutar programas.

Ej. 11: ¿Es posible que haya hardware cuyo sistema de arranque limite qué sistemas operativos pueden o no cargarse? ¿o bien realizar limitaciones de acceso a ciertos componentes hardware?

Solución: si, de hecho el sistema de arranque seguro de UEFI que hay en numerosos PCs y portátiles te impide instalar sistemas operativos distintos al Windows que traen por defecto o, como mínimo, te impide usar partes del hardware (puedes introducirte en la BIOS y desactivar el arranque seguro).

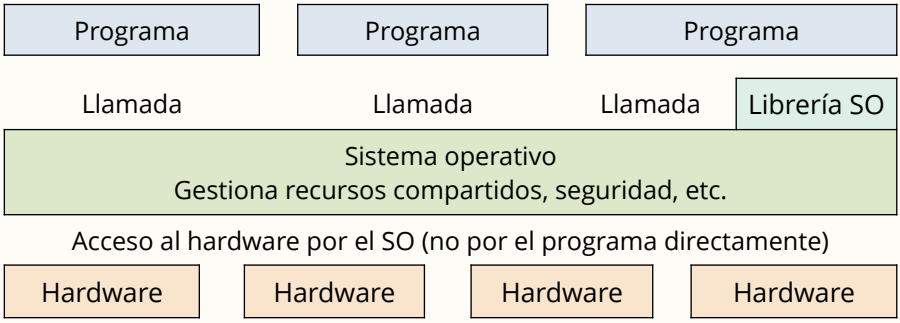
Los móviles y tablets son otro hardware cuyo sistema de arranque (llamado bootloader) que te impide cambiar el sistema operativo que tienen. Para hacerlo, debes hackear el bootloader para poder cambiar el sistema del móvil.

Funciones de un sistema operativo

Estando ya ejecutándose el sistema operativo, éste inicia la ejecución de ciertos módulos del sistema operativo como son la interfaz de usuario o programas en segundo plano que mantienen el correcto funcionamiento del sistema.

A la orden del usuario, o según esté configurado el sistema operativo, éste gestiona la ejecución de las diversas aplicaciones que permitirán al usuario realizar las tareas que desee.

El sistema operativo provee, además, de una plataforma base para que se ejecuten las aplicaciones. Proporciona a las aplicaciones una serie de herramientas, que las aplicaciones pueden usar para realizar una serie de funciones, como grabar un fichero a disco, mostrar un texto en pantalla, etcétera. De esta forma, las aplicaciones no tienen que preocuparse de cómo exactamente se guarda ese fichero, o cómo se muestra ese texto, sino que es el sistema operativo quien se encarga de ello. Cuando un programa hace uso de alguna de esta funciones se dice que realiza una ***llamada al sistema***.



Cuando se realiza una de éstas llamadas al sistema, el sistema operativo toma el control y realiza la función solicitada, si es posible. Esto permite al sistema operativo esconder los detalles del hardware, permitiendo a la aplicación funcionar de igual forma independientemente de que el hardware sea uno u otro.

Ej. 12: ¿Debería ser posible que dos programas tengan activas dos llamadas, las cuales hagan uso de un mismo disco duro?

Solución: si, dado que los ordenadores actuales ejecutan varios programas en paralelo, programas que puede perfectamente ser independientes, nada impide que, en un momento dado, varios quieran acceder al disco duro. Ahí, el sistema operativo será el responsable de ordenar esas peticiones de acceso, según las políticas establecidas.

Ej. 13: ¿Debería el sistema operativo resolver las llamadas que hagan uso de un hardware concreto de forma simultánea?

Solución: En algunos recursos (como el disco duro) el sistema operativo envía varias peticiones a la vez, que luego éste resuelve a su manera, pero en otros componentes hardware el sistema operativo decidirá cual enviar primero.

Existen alguna ***librerías del sistema*** que extienden la funcionalidad del sistema operativo, como por ejemplo las librerías DirectSound (que permite a las aplicaciones manejar la reproducción y captura de audio en aplicaciones y juegos en sistemas Windows) o Google Play Services (que permite que a las aplicaciones de Android acceder a funciones como la ubicación, autenticación, y servicios de Google). para dispositivos Android. Estas librerías proporcionan a las aplicaciones una nueva batería de llamadas al sistema que realizan funciones adicionales.

Ej. 14: Existe una librería de sistema de Windows necesaria para ejecutar juegos en dicho sistema ¿cuál sería?

Solución: DirectX es la librería de sistema típica en windows, aunque hoy día Vulkan también puede cumplir la misma función.

4.Aplicaciones

Las aplicaciones son el software que se ejecuta para que el usuario pueda realizar la tareas que desee.

Coste de desarrollo

Las aplicaciones cuestan mucho más tiempo y dinero del que suele pensarse: Desarrollar una aplicación profesional puede requerir años en la que trabajan desde decenas hasta miles de trabajadores. Por ejemplo, en 2013, la compañía Naughty Dog creó, tras 3 años de desarrollo, el juego Last of Us para el sistema operativo CellOS, que es el S.O. de la PlayStation 3, con la participación de más de mil personas y varias decenas de millones de dólares de presupuesto. Las aplicaciones se financian de distintas formas, como un precio de venta, publicidad, micropagos, venta de datos, etc.

Ej.15: Genshin Impact es un juego de móvil, en el que simplemente exploras un mundo luchando contra monstruos, looteando y realizando misiones ¿Qué coste ha tenido el juego? ¿Qué motivos puede haber para ese coste?

Solución: en la página «List of most expensive video games to develop» se dice que elcoste ha sido de más de 700 millones de dólares, aunque una parte importante de ese presupuesto se debe al márqueting.

Hay que observar que es un juego de mundo abierto, con mucho contenido en arte (personajes, escenarios, ciudades, monstruos, conjuros, etc.), diseño de niveles (quests, PNJs, tutoriales, escenario, etc.), reglas (conjuros, poderes, personajes, balance de juego), aparte del desarrollo general (3D, controles, prueba en distintas plataformas, etc.). Además, está el coste de servidores y personal de mantenimiento del juego, facturación, moderación, etcétera.

Plataformas de una aplicación

Cuando unos desarrolladores de software crean un programa, éste se realiza, generalmente, para un único sistema operativo. Por ejemplo, existen juegos que son exclusivos de XBOX (Halo). Sin embargo, a los desarrolladores puede interesarles sacar su aplicación para varios sistemas. Así, por ejemplo, la empresa Naughty Dog lanzó, en 2013, Last of US para CellOS, el sistema de Playstation 3. Un año después, lo lanzó para Orbis OS, el S.O. de la PlayStation 4.

Ciertas técnicas pueden ser de gran ayuda para los desarrolladores a la hora de sacar su aplicación para un segundo o sucesivos sistemas, pero siempre requiere de un esfuerzo adicional el soportar más sistemas, pues hay que testearlos y mantenerlos en varios sistemas. Otras veces, hay factores que impiden o hacen poco deseable el portar una aplicación a otras plataformas.

Ej. 16: El uego Among Us fue diseñado para ser jugado en local (son los jugadores en el mismo lugar y, más tarde, se añadió un modo online. Son capaces los desarrolladores de Among Us que, después de que su juego se hiciese viral, apenas consiguieron crear algún mapa nuevo y corregir errores, soportar plataformas adicionales?

Solución: los escasos desarrolladores quedaron desbordados por el número de jugadores que requería servidores y mantenimiento de esa estructura de juego online. Apenas pudieron hacer más puesto que el juego no estaba originalmente pensado para ser online. Esto les impidió crecer de forma adecuada e implementar mejoras de forma efectiva.

Ej. 17: ¿Le interesaría a Microsoft, propietaria del S.O. Windows, que el paquete Microsoft Office se ejecute en el sistema operativo rival Linux?

Solución: El deseo de Microsoft es seguir imponiendo su sistema operativo Windows a la comunidad de usuarios, para imponer su market o su publicidad dentro de Windows. Aunque Office es un programa que genera ventas, Microsoft a menudo proporciona descuentos (para estudiantes, ONGs, docencia, etc.), para que Windows

siga siendo el sistema dominante. Permitir que se ejecute en Linux facilitaría que algunos usuarios (aquellos cuya principal barrera para el cambio es tener que dejar de usar Office y poco más) pudieran abandonar Windows.

Ej. 18: ¿Interesaría a Discord, una empresa de audio y videollamadas gratuita, soportar nuevos sistemas? ¿En qué casos lo haría?

Solución: Discord es una aplicación que necesita que los usuarios a comunicar usen todos un sistema compatible con su aplicación, por lo que necesita extenderse lo más posible. Discord solo dejará fuera sistemas con pocos usuarios o que no tengan relevancia para su aplicación.

Ciertas aplicaciones especiales, llamadas ***emuladores***, que son capaces de ejecutar programas destinados a otros sistemas. Así, por ejemplo, Wine es capaz de ejecutar programas de Windows en entornos Linux, o el programa MAME puede ejecutar las ROMS de las antiguas máquinas recreativas de los 80 y 90.

Ej. 19: ¿Qué motivos tiene Valve, propietaria de la plataforma Steam, para crear y mantener la aplicación Proton (que es una extensión de wine), la cual permite jugar a juegos de Steam diseádos para Windows en entornos Linux? ¿Y si tenemos en cuenta que el 98% de usuarios de Steam son de Windows?

Solución: Linux no requiere pago para ser usado en la Steam Deck u otro sistema. Además, Valve consigue depender menos de Microsoft, que también tiene intereses en gaming (game pass y Xbox), que podría en un futuro decidir perjudicar a Steam, ya sea de forma legal o ilegal (como perjudicar su rendimiento como ya lo hace con Firefox y otras aplicaciones).

También existe la ***virtualización***, en la que el procesador posee na unidad en la que es capaz de acelerar las ejecuciones de programas escritos para otras arquitecturas, algo que se realiza con programas de virtualización como VirtualBox, Docker, Kubernetes y otros. Algunas virtualizaciones más concretas podemos encontrarlas en sistema conccretos, como por ejemplo PS3, que es capaz de ejecutar los juegos de

PS2. La virtualización nos ofrece una máquina o entorno estándar que podemos replicar en distintas plataformas, e incluso levantar ese mismo servicio múltiples veces sobre el sistema host, que puede ser muy útil en servidores compartidos, entornos empresariales y otros sistemas.

Ej. 20: Playstation 3, con una recepción inicial pobre, era compatible con PS2, pero PS4 no lo era con PS3 o anteriores? ¿Por qué esto pudo ser así?.

Solución: PS2 fue la consola más vendida de Sony, y dejar de ser compatible con ella hubiera sido un suicidio de marqueting, sin mencionar que muchos usuarios poseían infinidad de uegos en formato físico para la PS2. La PS tuvo también grandes jugos, pero la apuesta de Sony por la PS4 fue grande, en un momento de fuerza, por lo que Sony prefirió venderte el remake de diversos juegos para la PS4, que estaban en la PS4. Hoy día, esos juegos (PS1, PS2 y Ŝ·) se pueden jugar con la suscripción más cara de PS Plus.

Algunas aplicaciones se crean para ser usadas a través de una página ***web***. Salvo excepciones, estas aplicaciones requieren bastantes más recursos hardware y más intercambio de datos de red para funcionar, e incluso poseen ciertas limitaciones y necesitan un mayor cuidado. A cambio, tienen la ventaja que, una vez hechas, son accesibles por múltiples sistemas que soporten un navegador web, disponibles en la los sistemas móviles y de escritorio. Un ejemplo claro de aplicación web es el buscador de Google.

Ej. 21: ¿Por qué Google pagó a Apple 15M$ hace unos años para que su buscador siga siendo el buscador predeterminado en iphones?

Solución: Google ingresa gran cantidad de dinero a través de la publicidad en la web.

Ej. 22: ¿Qué tipo de aplicaciones son más adecuadas para que sus desarrolladores elijan el desarrollo web en vez de (o además de) desarrollar una aplicación para los distintos sistemas?

Solución: aplicaciones que deben soportar gran cantidad de sistemas, móviles y de escritorio, son adecuadas para el en desarrollo web, ya que los usuarios podrían no poder acceder a aplicaciones nativas.

También es una idea el desarrollar una aplicación web cuando el objetivo es facilitar el acceso a la aplicación, puesto que la instalación de aplicaciones suele ser una barrera de entrada para el usuario.

También se ha venido desarrollando el sistema de desarrollo ***multiplataforma***, en especial en ciertas áreas como es en los videojuegos. Hoy día, motores gráficos como Unreal Engine, Unity o Godot, permiten diseñar el juego en el entorno del PC, y exportarlo para multitud de sistemas objetivo, como Nintendo Switch, Playstation 4/5, Linux, MacOS, Windows, etcétera. Otras tecnologías, como Java o .NET, permiten realizar una aplicación y ejecutarla en diversas plataformas. Estos sistemas tienen que lidiar con la heterogeneidad de los múltiples sistemas que deben soportar.

Ej. 23: ¿Por qué Civilization V, que está disponible para Windows, Linux y MacOS, no está disponible en Xbox, PlayStation o Switch?

Solución: en este caso, el juego emplea teclado, un periférico muy poco común en Xbox, PlayStation o Switch, por lo que existirían pocos usuarios de estos sistemas que pudieran jugarlo.

Ej. 24: ¿Por qué Helltaker, un juego realizado por un artista para promocionarse, está solo disponible para Linux, MacOS y Linux (empleó un sistema de desarrollo que publicaba a la vez para esos 3 sistemas), pero no para consolas?

Solución: El autor eligió un sistema de desarrollo que publicaba en los citados sistemas, de forma que llegó de la forma más fácil al mayor número de usuarios. Su objetivo era promocionar su arte, y no le interesaba desarrollar y mantener para múltiples sistemas de consolas, cada uno con contratos de confidencialidad, mantenimiento, etcétera.

Ej. 25: ¿Y qué sucede con juegos como Age of Empires IV (publicado por Microsoft exclusivo para el Game pass de Microsoft) o The Legend of Zelda: Breath of the Wild (publicado por Nintendo y exclusivo para sus consolas)?

Solución: estos juegos son exclusivos para sus sistemas porque el objetivo de sus publicadores (que encargaron y pagaron el desarrollo)

era el de atraer usuarios a sus sistemas, no el llegar a la mayor cantidad de usuarios.

Unidad Didáctica 2:

Etapas del desarrollo

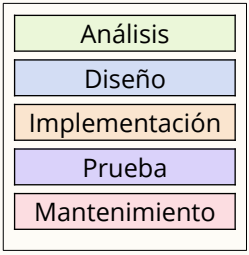
1.Desarrollo del software

Es desarrollo de software es un proceso que ocurre desde que se concibe una idea hasta que se construye una aplicación implementada en el sistema informático funcionando.

El proceso de desarrollo, que en un principio puede parecer una tarea simple, consta de una serie de pasos de obligado cumplimiento, pues sólo así podremos garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales (aquellos que van a utilizar el programa).

El modelo en cascada

El modelo en cascada es uno de los enfoques más antiguos y obsoletos, cuyas fases se ejecutan en secuencia (cada una debe completarse antes de pasar a la siguiente):



- Análisis: se recogen y estructuran los requisitos del software a desarrollar, se realiza una planificación y se realiza un modelo de alto nivel.
- Diseño: Realiza un diseño del nuevo sistema y se crea el plan de pruebas.
- Implementación. Se codifica según el diseño.
- Prueba. Se realizan las pruebas creadas en el diseño.
- Mantenimiento: se realiza la instalación del nuevo sistema, se arreglan los errores y, en su caso, se añaden nuevas funcionalidades.

El modelo en cascada, aún siendo un buen punto de partida pedagógico, es un modelo adecuado solo para proyectos con requisitos bien definidos y cambios mínimos, algo que no suele suceder, por lo que se usa muy poco.

2.Análisis

La importancia de esta etapa en el desarrollo de todo proyecto radica en que todo lo demás dependerá de lo bien detallada que esté esta etapa. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.

Planificación

Durante la planificación, se establece el ámbito del proyecto, definiendo el alcance de este, los actores implicados y se intenta estimar qué se necesitará para llevarlo a cabo:

- Se definen los ***objetivos y el alcance*** del proyecto, detallando las características generales que tendrá.
- Se identifican a las distintas ***partes*** que puedan estar implicadas por la realización del software, así como una primera concreción de cómo éstas serían afectadas. Esto puede incluir un estudio de mercado.
- Se identifican los ***riesgos*** del proyecto y, de ser posible, se trazan posibles alternativas en caso de que esos riesgos se conviertan en realidad.
- Se realiza una primera ***estimación*** de lo que se necesitará: personal, equipo, tiempo, costes, etcétera, lo que suele llevar a un plan de recursos y costes, a menudo usando un diagrama de Gantt.

Conforme se van definiendo los requisitos, y a lo largo de todo el desarrollo, estas estimaciones se van refinando.

Ej. 1: Un banco contacta con una empresa de desarrollo para que implemente un cajero automático ¿Cuáles serán las partes implicadas?

Solución: las partes implicadas serían: (1) el propio banco, cuyos intereses primarán sobre otros implicados, (2) los clientes del banco, que dispondrán del nuevo servicio (3) los trabajadores, que tendrán que saber operar con él y (4) los administradores e instaladores, que deberán instalar y mantener el sistema.

Ej. 2: Deseamos realizar un juego, financiándonos con la venta de éste. Comenta algunos de los posibles riesgos de este proyecto.

Solución: el juego podría no ser lo suficientemente atractivo para los usuarios, o puede que haya demasiada competencia de juegos parecidos. Es posible también que nuestro equipo no sea capaz de realizar el juego o que el márqueting sea inadecuado.

Ej. 3: Deseamos realizar una aplicación que consistirá en la creación y edición de notas, con un modelo de negocio basado en publicidad en la aplicación, con características novedosas para diferenciarnos de la competencia. Establece el alcance y objetivos; las partes interesada/afectadas, y los riesgos.

Solución: El objetivo del proyecto sería algo como «el realizar una aplicación de edición de notas que sea rentable a través de publicidad» En cuanto al alcance, deberemos decidir hasta donde deseamos llegar, en este caso (a grandes rasgos) se desea una complejidad suficientemente alta como para desmarcarse de la competencia, incluyendo, posiblemente, un márqueting de cierto nivel.

Ej. 4: Queremos desarrollar un juego para PlayStation. Somos un equipo de dos personas, un portátil y un 1 equipo de desarrollo y pruebas (Sony solo nos ha asignado un devkit). Establecemos las siguientes tareas:

- Concepto del juego: 2 sem. (requiere 1 portátil).
  (Necesario para Demo, programación, márqueting y arte).
  (Varios desarrolladores a la vez no requiere portátil extra).
- Pedir equipo de desarrollo: 1 sem. (requiere 1 portátil).
  (Necesaria para las tareas que necesiten equipo de desarrollo).
- Creación de demo. 2 sem. Requiere 1 equipo de desarrollo.
- Programación. 3 sem. (requiere 1 equipo de desarrollo).
- Arte. 2 sem. (requiere 1 portátil).

- Márqueting. 1 sem. (requiere 1 portátil).
  (Requiere Arte)
- Pruebas. 1 sem. (requiere 1 equipo de desarrollo).
  (Requiere de todas las demás tareas, excepto márqueting).

Realiza un diagrama que minimice el tiempo a emplear.

Solución: En primer lugar, puesto que tenemos que realizar el concepto, que es llave para todo lo demás, empleamos todos los recursos disponible a tal tarea. Tras ello, debemos pedir el equipo de desarrollo para poder acceder a la programación, la creación de la demo y las pruebas, pero el pedir los equipos de desarrollo requiere de un portátil. Una vez conseguido el entorno de desarrollo, las tareas se dividen según el recruso necesario.

1(ED)	Concepto		Programación		Demo	Pruebas
2(prt)	Concepto	Pedir ED	Arte	Márquet.		

Obtención de los requisitos

Aquí, se investigan y definen los requisitos del sistema y del software, comprendiendo las necesidades de los usuarios y las limitaciones técnicas. Se documentan los objetivos y las restricciones clave que guiarán el desarrollo.

A la hora de recoger los requisitos, deben tenerse en cuenta los distintos actores que interactúan con el sistema, no solo en usuario final. Con ello en cuenta, se realizan dos recogidas de requisitos, en el siguiente orden:

- ***Requisitos del sistema:*** se definen los requisitos generales del sistema al completo, tanto aspectos de negocio, de software o de hardware. Incluye tanto requisitos internos al equipo informático que ejecutará el software como a toda la estructura (redes, dispositivos externos, interfaces, etc.).

- ***Requisitos de software:*** se definen lo que debe hacer el software y cómo debe hacerlo: temas como el rendimiento, la escalabilidad, la seguridad o la mantenibilidad, pero siempre desde el punto de vista del software.

Los requisitos obtenidos en las dos fases anteriores pueden ser de dos tipos:

- ***Requisitos funcionales:*** Qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
- ***Requisitos no funcionales:*** Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

Una vez tenemos los requisitos, debemos filtrarlos (resolviendo contradicciones, eliminando duplicidades, etc.) y priorizados.

Tenemos un crear un software cuya idea inicial es que permita a los usuarios crear, gestionar y marcar como completadas sus tareas diarias. Como modelo de negocio, planeamos que se integre un sistema de anuncios.

Requisitos del Sistema:

- Generación de ingresos con anuncios (No Funcional).
- Almacenamiento en la nube (Funcional).

Requisitos del Software:

- Creación, edición y eliminación de tareas (Funcional).
- Marcado de tareas como completadas (Funcional).
- Integración con Sistema de anuncios (Funcional).

- Tiempo de respuesta de la carga de tareas (No Funcional).
- Seguridad en la transferencia y almacenamiento de datos (No Funcional).

Ej. 1: En el ejemplo anterior, se tienen en cuenta el gestionar las tareas en distintos dispositivos. Añade lo necesario para que las tareas sean gestionadas conjuntamente por un grupo de usuarios. Considera la inclusión de un administrador de grupo.

Solución: Como hay un concepto nuevo, el de grupos, habría que añadir funcionalidades para crear y finalizar grupos, así como añadir o eliminar usuarios del grupo. También sería conveniente incorporar el concepto de administrador, quizás por defecto el creador del grupo, que pueda realizar estas gestiones de usuarios, así como el eliminar el grupo.

Ej. 2: Crea una lista de requisitos de un software para dispositivo móvil que se encargará de activar o desactivar un sistema de alarma.

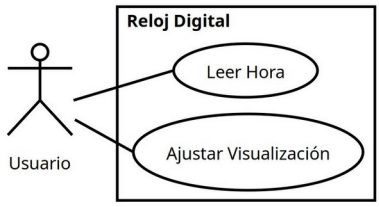
Solución: Los requisitos del software: Activar alarma (funcional), Desactivar alarma(funcional), Ver estado de alarma (funcional), intuituva (no funcional), seguridad (no funcional).

Respecto a los requisitos del sistema, serían la conexión con la alarma en cuestión (funcional).

Ej. 3: Crea una lista de requisitos sobre un sistema de gestión de biblioteca escolar.

Casos de uso

Una vez que tenemos los requisitos, es hora de darles una estructura. Se define una estructura de alto nivel del software a realizar empleando los diagramas de casos de uso.



En estos casos de uso se detallan todas las funcionalidades desde el punto de vista de los usuarios que emplearán el sistema. Cada una de estas funcionalidades tendrá descritos los distintos pasos para que estas tareas realizadas por el usuario sean realizadas.

Otras tareas

Durante el análisis, puede ser necesario realizar otras tareas, según la naturaleza de cada proyecto. Entre otros, pueden ser:

- Prototipado: a veces es difícil obtener ciertos requisitos o la forma en la que han de ser realizadas ciertas tareas. Para ello puede ser útil realizar un prototipo del software a realizar, que luego es descartado.
- Estudio de impacto: cuando se tiene que decidir si el reemplazar un sistema será beneficioso o no, en qué puntos habrá que realizar adaptaciones y cuán complicadas pueden ser éstas.
- Estudio de viabilidad, que puede contener aspectos técnicos o mercantiles que permitan decidir si el proyecto es posible con el equipo y recursos de los que se dispone.

3.Diseño

Durante el diseño, se planifica la estructura y la arquitectura del software. Esto implica determinar cómo se organizarán los componentes del sistema para lograr un funcionamiento eficiente. Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es definir cómo hacerlo.

Diseño arquitectónico

Consiste en dividir el sistema en partes y establecer las relaciones entre esta partes, especificando qué hace cada parte. El resultado será un modelo funcional-estructural de los requerimientos del sistema global, dividido en partes.

Este diseño se detalla, parte a parte, resultando en un diseño mucho más completo que también establece las relaciones entre las partes.

Selección de tecnologías

Las decisiones aquí pueden ser tomadas a lo largo de la la fase de diseño, e incluyen el seleccionar:

- Los lenguajes de programación a usar.
- Frameworks y librerías de mayor relevancia.
- Las arquitecturas a soportar
- Los sistemas de gestión de bases de datos.
- Tecnologías a emplear.

Modelado de datos

Se defines las entidades, atributos, atributos clave y relaciones de los datos, así como el proceso para acceder, insertar, modificar o borrar esos datos, junto con las políticas correspondientes. Se busca una estructura eficiente y coherente que facilite el acceso, la manipulación y el almacenamiento de la información de acuerdo con los requisitos del sistema.

Interfaz de usuario

Es el diseño de la interfaz de usuario, donde se diseñan las pantallas con las que el usuario usará con la aplicación. Este diseño buscará una interactividad intuitiva, de forma que el usuario encuentre fácilmente las funcionalidades que desea emplear en cada momento.

Dentro del diseño de la interfaz estaría la definición las tareas de usuario . Éstas describen las tareas que un usuario final podrá realizar con nuestro futuro programa. Por ejemplo, el usuario de un programa de edición fotográfica podría tener la tarea de:

- (1) Cargar imagen
- (2) Seleccionar filtro de imagen
- (3) Modificar valores de filtro
- (4) aplicar filtro.

Diseño del plan de pruebas

Se diseñan las pruebas que serán realizada en la fase de pruebas.

4.Codificación

En esta fase, los programadores escriben el código fuente del software basándose en los diseños previamente establecidos. Se siguen estándares de codificación para asegurar la coherencia y la calidad del código. Las características deseables de todo código son:

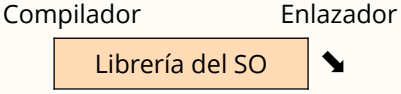
- Modularidad: que esté dividido en trozos más pequeños.
- Corrección: que haga lo que se le pide realmente.
- Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
- Eficiencia: que haga un buen uso de los recursos.
- Portabilidad: que se pueda implementar en cualquier equipo.

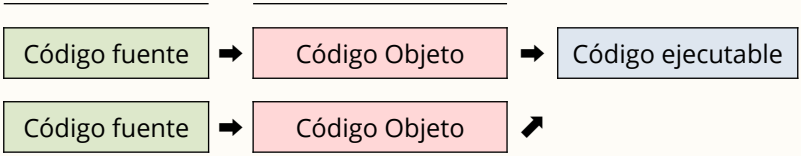
Durante esta fase, el código creado por los desarrolladores se llama código fuente, que es un texto legible por un humano, pero no es ejecutable por el sistema informático.

Compiladores e intérpretes

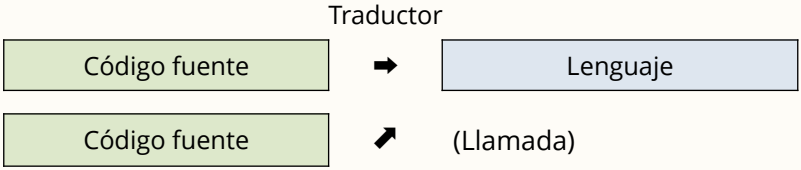
Una vez tenemos un código fuente, tendremos que realizar un proceso para que el código fuente sea ejecutado. Existen varias formas:

- ***Lenguajes compilados.*** Un programa llamado compilador convierte cada uno de los ficheros del código fuente en código objeto. Luego, otro programa llamado enlazador los une, empleando librerías del sistema operativo si es necesario, para generar un programa ejecutable. A partir de ahora, para ejecutar el programa, ya no serán necesarios ni el compilador ni el enlazador, solo el programa ejecutable resultante.

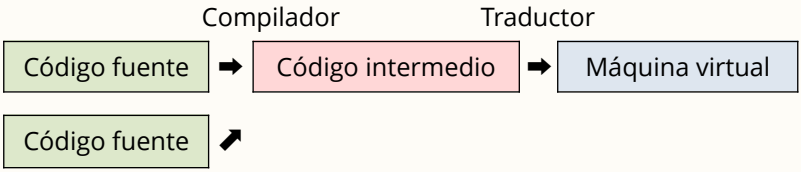




- ***Lenguajes interpretados.*** Un programa llamado intérprete, lee las instrucciones del código fuente y las va ejecutando. Cada vez que queramos ejecutar el programa deberemos usar el intérprete. Este método es más lento que el anterior, porque se deben ir traduciendo las instrucciones antes de poder ejecutarlas.



- ***Máquina virtual.*** Este método, usado por lenguajes como Java o C#, es un caso especial del anterior. El código se compila generando un código intermedio (también llamado bytecode), que luego es interpretado por el intérprete. Aunque este método no es tan eficiente como el primero (la compilación pura), pero si que es más eficiente que el anterior, pues el código intermedio es más competo y está optimizado para ser interpretado de la forma más rápida posible.



Frameworks y librerías

Las librerías son trozos de código fuente, bien encapsulados, que realizan una función concreta, que serán llamados por nuestro código para realizar esas funciones. Por ejemplo, una librería puede encargase de encriptar un fichero, de forma que le pasamos un fichero y una clave, y nos devuelve el fichero encriptado. Es muy común usar librerías realizados por terceros, de forma que nosotros nos ahorramos de desarrollar todo lo que la librería hace.

Los lenguajes de programación proporcionan una serie de librerías del lenguaje, que están disponibles en el lenguaje. Por ejemplo, en java tenemos la librería java.sql, que contiene las diversas clases e interfaces para comunicarse con bases de datos relacionales.

Los frameworks, tales como Angular, Spring o ASP.NET son estructuras más completas que proporcionan todo un conjunto de funcionalidades ya listas para usar en nuestros programas. En un framework, el flujo de trabajo lo dicta él, de forma que el código del desarrollador debe adaptarse a él.

Un problema de los frameworks suele ser la dependencia de estas funcionalidades y que es posible que su uso haga que nuestro programa final necesite más recursos. Algunos frameworks poseen una interfaz propia que facilita aún más el desarrollo, tales como Unity, y Unreal Engine.

Tipos de lenguajes

En esta fase también se deciden los lenguajes a usar. Se pueden realizar muchas clasificaciones de los lenguajes de programación. Algunas de ellas pueden ser:

- Alto nivel, medio nivel o bajo nivel. Los de alto nivel (Java, C#, etc.) son mas potentes y requieren menos esfuerzo de desarrollo, pero son menos eficientes. Los de bajo nivel (ensamblador) son muy cercanos a la máquina. El nivel intermedio (C), consiguen un compromiso entre ambos.

- Estructurados y orientados a objeto. Los orientados a objetos (Java) permiten encapsular código en objetos y ayudan a que el código sea modular. Los estructurados solo puede llegar a implementar funciones para ello.
- Visuales vs textuales. Los textuales (Java, C, C++, C#, etc.) requiren que se escriba texto para realizar el código. Los visuales (Microbit, Scratch) disponen de una interfaz en la que se van juntando bloques para realizar el código.
- Lenguajes de marcas (XML, HTML, JSON). Definen un programa o un contenido con texto que posee marcas para estructurar o identificar los elementos.
- Lenguajes declarativos (SQL). No se detallan los pasos a seguir, sino el resultado deseado.

5. Prueba

La fase de prueba es esencial para garantizar que el software funcione correctamente. Se diseñan casos de prueba para verificar que el software cumple con los requisitos y se identifican y corrigen los errores antes de la entrega al cliente.

Normalmente, éstas se realizan sobre un conjunto de datos de prueba, que consisten en un conjunto seleccionado y predefinido de datos límite a los que la aplicación es sometida.

La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido. Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

- ***Pruebas unitarias.*** Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). JUnit es el entorno de pruebas para Java.
- ***Pruebas de Integración.*** Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.

La prueba final se denomina comúnmente Beta Test, ésta se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa).

6.Mantenimiento

Explotación

La explotación es la momento en que los usuarios finales conocen la aplicación y comienzan a utilizarla. La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.

En el proceso de instalación, los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados. Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación. En este momento, se suelen llevan a cabo las Beta Test, que son las últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo.

Una vez instalada, pasamos a la fase de configuración. En ella, asignamos los parámetros de funcionamiento normal de la empresa y probamos que la aplicación es operativa.

También puede ocurrir que la configuración la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación. Y también, si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla. (Si el software es "a medida", lo más aconsejable es que la hagan aquellos que la han fabricado).

Una vez se ha configurado, el siguiente y último paso es la fase de producción normal. La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.

Mantenimiento

Después de la implementación, el mantenimiento implica realizar actualizaciones, correcciones de errores y mejoras en el software para mantenerlo actualizado y funcionando de manera óptima a lo largo del tiempo.

En cualquier otro sector laboral esto es así, pero el caso de la construcción de software es muy diferente. La etapa de mantenimiento es la más larga de todo el ciclo de vida del software. Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo.

Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó. Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores.

Por todo ello, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

El mantenimiento se define como el proceso de control, mejora y optimización del software. Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo. Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

- Perfectivos: Para mejorar la funcionalidad del software.
- Evolutivos: El cliente tendrá en el futuro nuevas necesidades. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- Adaptativos: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- Correctivos: La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).

7.Documentación

Se crea documentación que describe el funcionamiento y el uso del software. Esto incluye manuales de usuario, descripciones técnicas y registros de cambios para futuras referencias. Es necesaria para poder dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

La documentación no es una fase en sí, sino que se desarrolla durante el resto de fases. Algunos resultados de la documentación son:

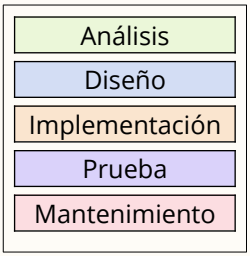
- Guía Técnica: Aspectos técnicos de las etapas anteriores (análisis, diseño, implementación, prueba y mantenimiento), para facilitar el mantenimiento a los propios desarrolladores..
- Guía de uso: Guía para el usuario final de cómo utilizar la aplicación.
- Guía de instalación: Los requisitos de la aplicación y cómo instalarla y ponerla a funcionar, para el administrador y/o el usuario final.

8.Ciclos de vida software

Los ciclos de vida del software son modelos o enfoques que describen cómo se desarrolla y se mantiene el software a lo largo de su ciclo de vida. Cada ciclo de vida tiene sus propias fases y actividades específicas.

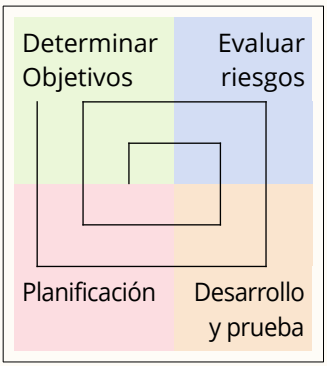
Modelo en Cascada

El modelo en cascada es uno de los ciclos de vida más antiguos y lineales. Las fases se ejecutan en secuencia, y cada una debe completarse antes de pasar a la siguiente. Tan solo es adecuado para proyectos con requisitos muy bien definidos, y que se prevean los mínimos cambios, algo que no suele suceder, salvo en los proyectos más pequeños, por lo que se usa muy poco.



Modelo en Espiral

El modelo en espiral incorpora iteraciones y se enfoca en la gestión de riesgos (el equipo debe ser tener la experiencia suficiente para evaluarlos). Es un ciclo de vida flexible y adaptativo, para grandes sistemas complejos que presentan diversos riesgos, como el software de control de un avión comercial. También es usado en proyectos experimentales y de investigación.



El desarrollo progresa a través de ciclos repetidos, cada uno de los cuales permite la incorporación de mejoras y cambios.

Desarrollo en Espiral Incremental: Es similar al modelo en espiral, pero se enfoca en entregar incrementos funcionales del software en cada iteración. Cada iteración agrega nuevas capacidades al sistema, lo que permite una entrega temprana de funcionalidad.

Desarrollo Ágil

Los métodos ágiles, como Scrum y Kanban, se centran en la flexibilidad y la colaboración continua. Los proyectos se dividen en iteraciones cortas llamadas "sprints" y se adaptan a medida que se avanza. La priorización de las características y la retroalimentación constante son fundamentales.

En el desarrollo ágil, el proceso se organiza en ciclos cortos llamados sprints (de 1 a 4 semanas) y suele seguir estos pasos clave:

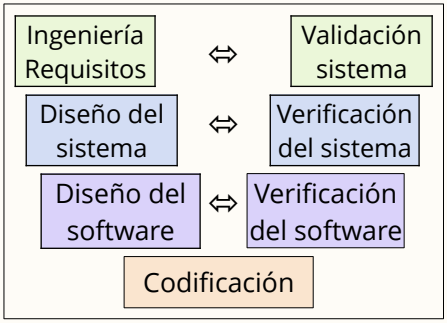
- Planificación: Se identifican los requisitos y se priorizan en una lista de trabajo (backlog). Se selecciona una o varias características, que son grupos de requisitos.
- División en tareas: Las características se dividen en tareas pequeñas para completar en el sprint.
- Desarrollo: El equipo trabaja en las tareas, revisando el progreso en reuniones diarias.
- Pruebas y validación: Se realizan pruebas continuas para asegurar la calidad.
- Revisión del sprint: Se presenta el trabajo completado y se obtiene retroalimentación.
- Retrospectiva: El equipo evalúa el proceso y mejora para el siguiente sprint.
- Entrega continua: Se liberan versiones funcionales del software de manera frecuente.

Este ciclo se repite, permitiendo adaptarse rápidamente a cambios y mejorar el producto de forma continua.

Timeboxing: En metodologías ágiles es posbile usar el la técnica de Desarrollo en Espacios de Tiempo, o Timeboxing, que se basa en la asignación de un tiempo fijo para cada fase del proyecto. El desarrollo se adapta a ese marco de tiempo y los recursos disponibles. Se enfoca en la entrega de incrementos de funcionalidad de manera regular.

Modelo en V

El modelo V ((Validación y Verificación)) refleja la relación entre las fases de desarrollo y prueba. Cada fase de desarrollo tiene una fase de prueba correspondiente. Las pruebas se realizan para verificar que cada etapa cumple con los requisitos y para validar que el producto final satisface las necesidades del cliente.



Modelo RAD

El Desarrollo Rápido de Aplicaciones se enfoca en la rápida creación de prototipos y la iteración continua. Se utiliza cuando los requisitos no están bien definidos y se busca una entrega rápida. Se construyen prototipos para refinar los requisitos antes de la implementación final.

9.Herramientas

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo. Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis del mismo, que son las causas principales de los fallos del software.

Las herramientas CASE son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.

El desarrollo rápido de aplicaciones o RAD es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE.

Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario o entornos de desarrollo integrado completos. La tecnología CASE trata de automatizar las fases del desarrollo de software para que mejore la calidad del proceso y del resultado final. En concreto, estas herramientas permiten:

- Mejorar la planificación del proyecto.
- Darle 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.

Clasificación

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- U-CASE: ofrece ayuda en las fases de planificación y análisis de requisitos.
- M-CASE: ofrece ayuda en análisis y diseño.
- L-CASE: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Ejemplos de herramientas CASE libres son ArgoUML, Use Case Maker y ObjectBuilder.

Ej. 4: ¿Cuáles de estas son herramientas del desarrollo software?:

Compilador, editor de texto, entorno de pruebas, Programa de diagramas UML.

Solución: todas ellas son herramientas del desarrollo, ya que todas son empleadas en éste.

Unidad Didáctica 3:

Casos de uso

1.Casos de uso

Los casos de uso especifican cada uno de los comportamientos de nuestro sistema: abarcan, por tanto los requisitos funcionales del sistema. Cada caso de uso es, fundamentalmente, una lista de pasos que detallan el comportamiento de una funcionalidad del sistema.

Los casos de uso (y los diagramas de casos de uso que veremos más adelante) se crean en la primera fase del desarrollo del software, la fase de análisis. Su principal función es dirigir el proceso de creación del software, definiendo qué se espera de dicho software. Los diagramas de casos de uso se emplearán luego para crear, durante la fase de diseño, los diagramas de clases.

La ventaja principal de los casos de uso es su facilidad para ser interpretados, lo que hace que sean especialmente útiles en la comunicación entre desarrolladores entre si y entre éstos y el cliente. Han de ser definidos, eso si, con cuidado y precisión, de forma detallada, pues cualquier característica del software que no se consiga tener en cuenta en los casos de uso ocasionará inconsistencias más adelante.

Actores

Es cualquier entidad externa que interactúa con la aplicación. Los ***actores*** son las entidades que desencadenan los casos de uso y participan en las interacciones con el sistema para lograr ciertos objetivos.

Los actores pueden ser personas, otros sistemas, dispositivos o incluso otros programas. Cada actor tiene un papel específico y puede desempeñar un papel activo al iniciar o participar en un caso de uso, o un papel más pasivo al recibir los resultados de las interacciones.

Por ejemplo, en un sistema de gestión de biblioteca, los actores podrían ser "bibliotecario", "estudiante" y "sistema de inventario". Cada uno de estos

actores desempeña un papel diferente y participa en casos de uso específicos, como "prestar libro", "devolver libro" o "actualizar inventario". La identificación clara de los actores y sus roles es fundamental para comprender y modelar los requisitos del sistema en el análisis de casos de uso.

Flujo principal, descripción y notas

En el flujo principal, se detallan los pasos habituales que tanto los actores como el sistema realizan para que se lleve a cabo la funcionalidad que se está describiendo.

Visualizar fecha
Actores: Usuario
Descripción: El usuario visualiza la fecha del sistema.
Flujo Principal: 1. El usuario selecciona la opción de visualizar fecha. 2. El sistema muestra la fecha del sistema.
Notas: El sistema trabajará con la hora local.

El caso de uso anterior, es posible que el diseño final de la aplicación no tenga una “opción de mostrar lista de chats activos”, sino que se muestre la lista de chats directamente al iniciar la aplicación: es algo que aún no sabemos como será, pero en el caso de uso debemos especificar que seleccionamos esa opción.

Por lo general, en especial en estos primeros ejemplos, los pasos del flujo principal alternarán entre una acción del usuario y una del sistema. Solo cuando un paso sea muy complejo, se dividirá en varios paso

Se suele incluirse una ***descripción*** del caso de uso, que resumirá lo que realiza dicho caso de uso.

Cuando sea necesario, habrán unas ***notas*** , que podrán ser sobre el caso de uso en general, sobre alguno de los puntos del flujo principal o de los alternativos, etcétera. En muchas ocasiones, estas notas adicionales surgen de los requisitos no funcionales.

Flujos alternativos

Los flujos alternativos describen las excepciones al comportamiento del flujo principal, de forma que reemplazan al paso con el que comparten número en caso de que ocurra algún suceso: en el caso de uso siguiente, si no hubieran chats activos, se mostraría el paso 2a, en vez de el 2.

Hay que destacar que, en el caso de uso no se incluyen los errores, tales como que el usuario haya introducido mal los datos que se pidan, que no haya conexión a Internet, etcétera.

Visualizar Chat
Actores: Usuario
Descripción: Interacción entre el usuario y el sistema para visualizar un chat activo.
Flujo Principal: 1. El usuario selecciona la opción de mostrar lista de chats activos. 2. El sistema muestra la lista de chats activos 3. El usuario selecciona un chat 4. El sistema muestra el chat seleccionado.
Flujo secundario: 2a. Si no existen chats activos, el sistema muestra un mensaje indicando que no hay chats para mostrar.

Ej. 1: En un sistema que es un portal de noticias, realiza un caso de uso, llamado `Leer Noticia`, en la que un usuario selecciona y accede a una noticia para leerla.

Ej. 2: Realiza un caso de uso, llamado `Buscar Producto`, en la que un cliente busca un producto que se corresponda a un criterio que dicho cliente desee.

Precondiciones y postcondiciones

Las ***precondiciones*** son condiciones que deben cumplrse antes de que un caso de uso o una función comiencen a ejecutarse. En un caso de uso, las precondiciones pueden incluir cosas como la autenticación del usuario, el estado del sistema, o cualquier otro requisito necesario para que el caso de uso se desarrolle adecuadamente.

Las ***postcondiciones*** son condiciones que deben ser verdaderas después de que un caso de uso o una función se ha ejecutado correctamente. Son los resultados esperados o los cambios de estado que se supone que ocurren como consecuencia de la ejecución de la acción principal.

Comprar Moneda Virtual
Actores: Usuario, Banco
Descripción: Proceso de un cliente al realizar un pago con tarjeta a través de un sistema en línea, donde el banco gestiona la transacción.
Precondiciones: El usuario debe haberse identificado en el sistema.
Flujo Principal: 1. El usuario selecciona la cantidad de monedas a comprar. 2. El sistema solicita la información de la tarjeta de crédito. 3. El usuario proporciona los detalles de la tarjeta de crédito. 4. El sistema envía la información de la transacción al banco para su procesamiento y muestra un mensaje con el resultado de la operación.
Flujo secundario: 4a. Si el banco rechaza la transacción, el sistema muestra un mensaje

de error al cliente y vuelve al punto 2.
Postcondiciones: La cantidad de moneda virtual seleccionada será añadida a la cuenta del usuario.
Notas adicionales: • El usuario proporciona los detalles de la tarjeta de crédito. • El sistema garantiza la seguridad de la información de la tarjeta mediante la implementación de estándares de seguridad. • En caso de rechazo del pago, se proporcionará información clara sobre el motivo y las acciones a seguir.

Ej. 3: Crea el caso de uso `Crear tarea`, en la que se creará y registrará una nueva tarea en el sistema. Un usuario deberá estar identificado en el sistema para crear la tarea.

Ej. 4: Respecto a una tienda virtual, crea el caso de uso `Comprar productos`, en la que el usuario seleccionará uno o varios productos y luego procederá a pagarlos, especificando la dirección de envío.

Desglosar un paso

Es posible que un flujo principal, o en el alternativo, una tarea sea algo compleja o que conlleve un número de pasos, para ello, se emplea una numeración que empieza por el paso a desglosar seguido de un punto:

Cambiar foto de perfil
Actores: Usuario
Descripción: El usuario cambiará su foto de perfil con una nueva captura fotográfica.
Flujo Principal: 1. El usuario selecciona "Cambiar foto de perfil". 2. El sistema muestra una vista de la cámara frontal. 3. El usuario realiza una fotografía. 3.1. El sistema le indica al usuario que debe encuadrar su cara.

3.1. El usuario encuadra su cara en la vista de la pantalla. 3.1. Cuando es sistema detecte un encuadro correcto, lo indicará al usuario. 3.3. El usuario podrá seguir moviendo el encuadre 3.4. Estando el encuadre como el válido, el usuario pulsará “tomar foto”. 4. El sistema tomará la foto y la establecerá como foto de perfil.
Postcondiciones: La foto de perfil habrá cambiado.

Ej. 5: Realiza un caso de uso `Silenciar notificaciones`. El paso en que el usuario selecciona la opción de “escoger opción” es un poco compleja, pues tendrá que elegir primero entre varias alternativas (que podría ser algo como "1 hora", "8 horas", o "Permanente"). Tras ello, el usuario deberá confirmar su elección (si respondiese que no, se cancela la tarea, por lo que no es necesario indicar ese hecho de seleccionar que no en el caso de uso).

Repeticiones y cambios de paso

También es posible que se produzca saltos entre los pasos, de forma que se indicará el paso al que se debe de saltar:

Leer noticias del día
Actores: Usuario
Descripción: El usuario leerá las noticias del día que desee.
Flujo Principal: 1. El usuario selecciona la opción de “leer noticias del día”. 2. El sistema muestra las noticias del día. 3. El usuario lee una noticia. 3.1. El usuario selecciona una noticia que desee, o bien elige “terminar”, en cuyo caso pasa al punto 4. 3.2. El sistema muestra la noticia seleccionada.

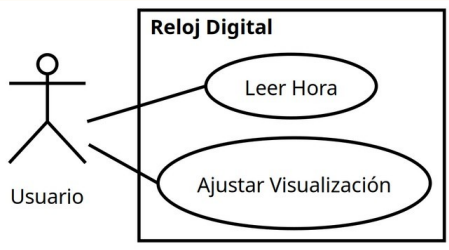
3.3. El usuario lee la noticia. 3.4. El usuario selecciona “volver”, y se pasa al punto 2. 4. El sistema cierra la lista de noticias.
Flujo secundario: 3.3a. El usuario selecciona “ver más tarde”.
Postcondiciones: Las noticias seleccionadas como “ver más tarde” se añadirán a la lista de “noticias guardadas”.

Ej. 6: Cambia el caso de uso de `Silenciar notificaciones`, Realizado anteriormente, de forma que, esta vez, cuando el usuario responda que no a la confirmación, se vuelva a la selección de alternativas.

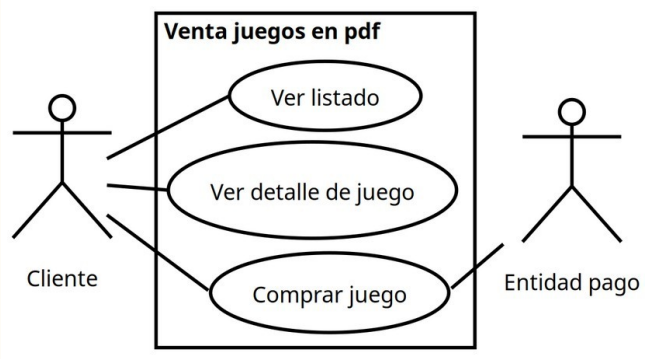
Ej. 7: Crea el caso de uso `Verificar email`. El usuario introducirá un email y el sistema enviará un correo de verificación con un código. El usuario deberá introducir el código o bien solicitar que se reenvíe un código nuevo. Observa que el flujo principal trata de verificar el email, de forma que el no conseguirlo se considera una cancelación de la tarea debido a un error. En notas, se establecerá que el uso repetido de esta función estará limitado, por temas de seguridad.

2.Diagrama frontera

El diagrama frontera incluye todos los casos de uso genéricos del sistema, enmarcados con un recuadro, dejando a los actores fuera.

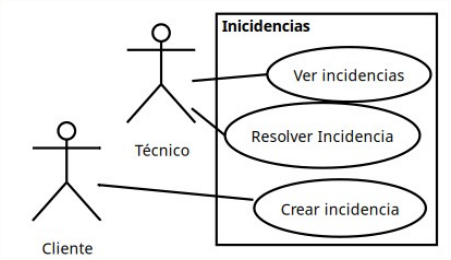


Hay que recordar que existen actores primarios y secundarios, ambos deben de representarse en el diagrama frontera (a menudo los actores primarios se dejan a la izquierda y los secundarios a la derecha). En el diagrama siguiente, se modela un portal de ventas de juegos de mesa en formato pdf muy simple, que no requiere registro ni identificación:



Ej.8: Crea el diagrama frontera de un sistema de notas (textos/notas de recordatorio). Un usuario podrá ver la lista de notas ya creadas, crear nuevas notas, editarlas y borrarlas.

Obviamente, el sistema podrá tener varios actores principales. El siguiente es un sistema en el que el cliente podrá crear una incidencia, mientras que un técnico podrá ver las incidencias, y también podrá resolver una incidencia.



Ej. 9: Crea un diagrama frontera de una aplicación en la que hay dos tipos de usuarios, el usuario normal y el usuario de pago. El usuario podrá acceder a "generar créditos con publicidad", y el usuario de pago podrá "comprar moneda virtual". Ambos usuarios podrán realizar la tarea de "ver película".

Ej. 10: Crea un diagrama frontera de un teléfono, donde un usuario podrá realizar llamadas (con el número de teléfono: el sistema no posee agenda) y podrá también recibir llamadas.

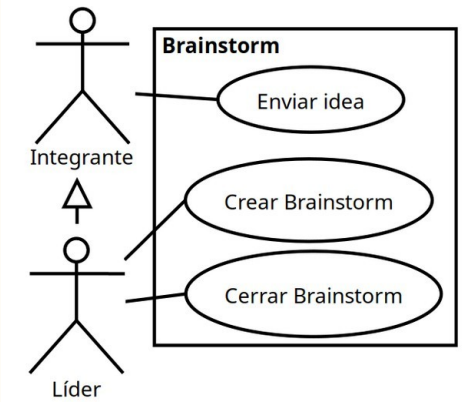
3.Relaciones

Interacción o asociación.

La asociación es el enlace entre actor y los casos de uso. La asociación se representa con un linea continua entre el actor y el caso de uso. Por ejemplo, en el diagrama frontera de Venta de juego en pdf, descrito en el apartado anterior, existen cuatro asociaciones, una entre `Comprar juego` y `Entidad de pago`, y otras tres entre `Cliente` y cada uno de los tres casos de uso.

Generalización o especialización

La generalización/especialización ocurre entre actores, de forma que uno o varios actores pueden interactuar con los casos de uso de la clase superior y, además, tienen otras interacciones.



En el caso frontera anterior tenemos un integrante de un sistema de Brainstorming, que puede enviar una idea al grupo de Brainstorm. El líder, además de ser un participante, pudiendo también enviar ideas, es capaz de realizar otras acciones. Todo lider es un participante, pero no todos los participantes son líderes.

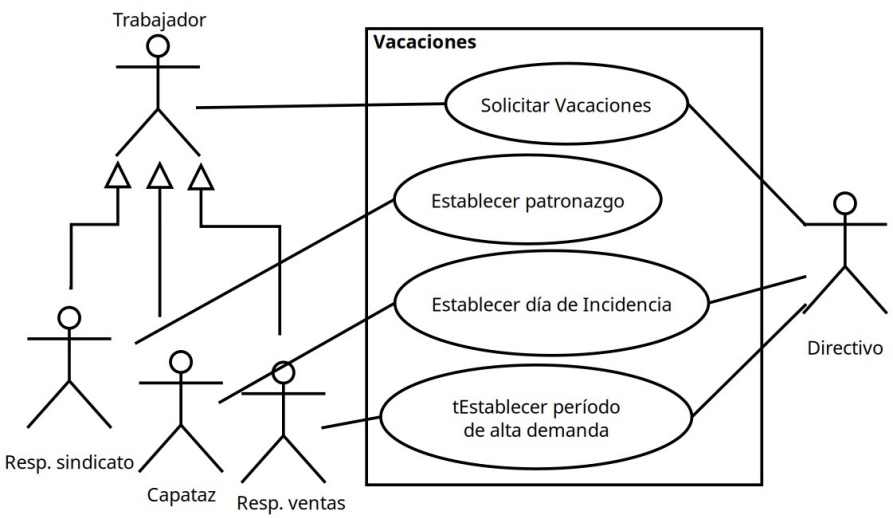
Ej. 11: Crea un diagrama frontera de un tablón de anuncios interno. En el tablón de anuncios, los usuarios pueden enviar mensajes, mientras que el administrador también puede borrarlos. El sistema identifica al administrador usando servicios en red, por lo que ni usuarios ni administrador tienen que identificarse o registrarse.

Ej. 12: Crea un diagrama frontera de una web de registro. Los poseedores de una propiedad (piso, casa, chalet, etc.) podrán inscribir su propiedad. Los que sean propietarios de una vivienda que no sea unifamiliar (piso y similares), también podrá especificar división (escalera, número, puerta, etc.).

Por otra parte, los inquilinos podrán también registrar la propiedad en la que vivan. Los inquilinos de una de una vivienda que no sea unifamiliar (piso y similares), también podrá especificar división (escalera, número, puerta, etc.).

Es posible que varios actores hereden de un mismo actor, pero no se permite la multiherencia: en ese caso se crea un actor nuevo con las correspondientes asociaciones que sean necesarias.

Lo siguiente es un sistema interno (sin identificación ni registro) en el que todos los trabajadores pueden entrar y solicitar el período de vacaciones. Los responsables de ventas pueden también registrar períodos de alta demanda, los capataces pueden registrar días de incidencia, y los responsables del sindicato pueden registrar días de patronazgo. Finalmente, los directivos pueden hacer todo lo anterior, salvo los días de patronazgo.



Observa que una persona física podría ser, a la vez, un capataz y sindicalista, o capataz y responsable de ventas, o capataz sindicalista, pero los perfiles son distintos.

Ej. 13: Una aplicación en la entrada de un evento te pide que escanees tu ticket para dejarte pasar. Los usuarios que tengan una suscripción podrán usar ticket o bien su tarjeta de socio y podrán, además, elegir un asiento en el auditorio general si así lo desean. Los usuarios miembros VIP podrán reservar un asiento en tribuna.

También hay artistas que tienen acceso únicamente a una segunda puerta, hacia los camerinos, y tendrán que registrar sus datos para poder entrar (no usan ticket), tras lo cual el sistema les informará del número de camerino. Por último los artistas invitados (artistas famosos) pueden entrar a todos lados.

La compra y generación de entradas y la gestión de clientes es realizada por una empresa de "ticketing" externa. Nuestro sistema, cada vez que escanea un ticket, consulta a una web que la empresa externa tiene a nuestra disposición.

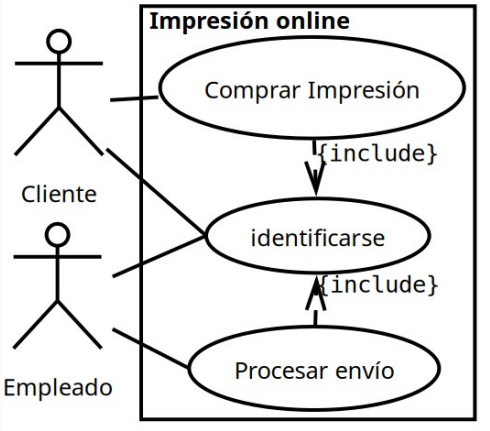
4.Inclusión

La inclusión se produce cuando un caso de uso principal depende de otro para completar su funcionalidad o cumplir con sus precondiciones.

La inclusión se representa con una flecha discontinua con la palabra include , que empieza en el caso de uso que incluye y se dirige hacia el caso de uso incluido. Existen tres casos en los que se emplea la inclusión.

Precondición

El primero es que el caso de uso incluido tenga una ***precondición*** que se realiza a través de otro caso de uso. En el ejemplo siguiente, tanto cliente como empleado deben estar identificados en el sistema para poder realizar los casos de uso de Comprar Impresión y Procesar Envío respectivamente. Observa que el tanto el caso de uso incluyente como el caso de uso incluido tienen sus debidas asociaciones a sus respectivos actores:



Como decíamos, estas inclusiones se reflejarán en los casos de uso en las precondiciones de `Comprar Impresión` y de `Procesar Envío`. Por ejemplo:

Comprar impresión
Actores: Cliente
Descripción: El usuario realiza la compra de una impresión
Precondiciones: El cliente debe estar identificado.

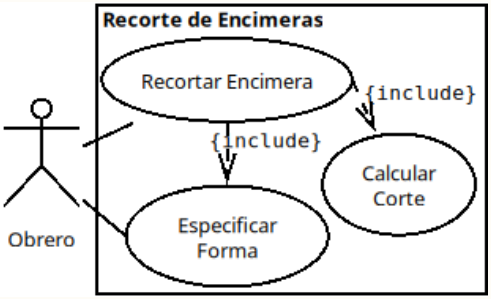
Ej. 14: Un sistema de identificación, el cual se halla en un sistema informático empotrado (un USB seguro), permite la `identificación por biometría` y la `identificación por datos personales`. En ambos casos, sin embargo, es necesario que el usuario haya `desbloqueado el dispositivo`. También es necesario que el usuario haya `registrado el dispositivo`.

Realiza el diagrama frontera y cada uno de los 4 casos de uso. Para el flujo principal de todos ellos, asume un flujo muy sencillo del estilo:

1. El usuario selecciona la opción de identificarse por datos personales
2. El sistema pide los datos personales.
3. El usuario introduce los datos personales.

Dividir un proceso complejos

El segundo tipo de inclusión es cuando un caso de uso tiene cierto paso o conjunto de pasos no triviales, que constituyan una funcionalidad aparte suficientemente diferenciada. Por ejemplo:

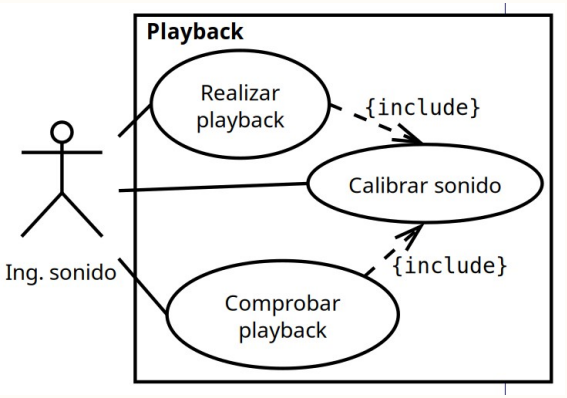


Recortar encimera
Actores: Obrero
Descripción: El obrero procede a definir las formas necesarias que desea se recorten en la tabla para encimera, y el sistema calcula la mejor forma de hacerlo.
Flujo Principal: 1. El obrero selecciona la opción de Recortar encimera. 2. El obrero añade una o varias formas: 2.1. El obrero selecciona, tantas veces como desee "Añadir una nueva forma", y el sistema ***iniciará el caso de uso Especificar Forma***, añadiendo la nueva forma a la lista de formas. 2.2. El obrero puede seleccionar "Borrar forma", y el sistema borrará la forma seleccionada. 2.3. Si no hay ninguna forma en la lista, *ve al punto 2.1* 2.3. El obrero podrá *volver al punto 2.1 o al punto 2.2* 4. El obrero selecciona "Calcular corte". 5. El sistema ***inicia el caso de uso Calcular Corte***. 6. El sistema guarda y muestra al usuario el diseño de corte calculado.
Flujo Secundario: 6a. Si las formas no caben en la tabla base, notifica al usuario y *vuelve al punto 2*.
Postcondiciones: El sistema ha guardado el diseño de corte.

Ej. 15: Un sistema de gestión financiera permite realizar diversas operaciones al cliente. Una de ellas es `realizar un análisis financiero`. Este caso de uso necesita hacer un `recopilado de datos`, un `análisis de mercado` y, finalmente, una `creación de diagramas financieros`. El análisis de mercado requiere, a su vez, un `recopilado de oportunidades financieras`.

Realizar una tarea común

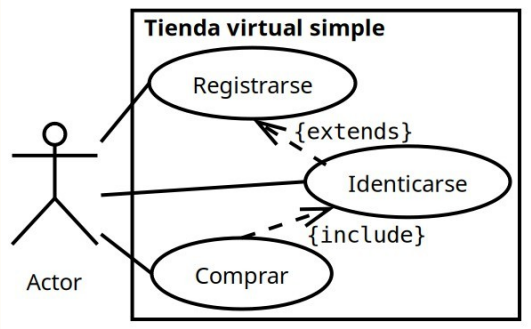
Por último, cuando un conjunto de pasos no triviales se realiza en distintos casos de uso, se aíslan dichos pasos en un caso de uso aparte.



Ej. 16: Un sistema de scripting ejecuta pequeños programas (también llamados scripts). El sistema permite `ejecutar scripts`, `guardar scripts` y `modificar scripts`. En todos ellos se realiza un `verificar script`.

5.Extensión

La extensión es similar a la inclusión, a excepción de que el caso de uso que extiende no siempre empleará al caso de uso extendido. El caso más típico es el del registro, donde al identificarse, es posible que el usuario realice el registro, o bien puede que se identifique directamente:



La invocación de un caso de uso extendido dentro de otro, se realiza de forma similar a la inclusión.

Ej. 17: Crea un diagrama frontera de una aplicación que es un lector de noticias. Un usuario podrá `Leer Noticias` estando o no identificado (con `Identificarse`), pero solo podrá `Añadir a favoritos` si está identificado.

Ej. 18: Crea una diagrama frontera de una mini-tienda virtual, donde el cliente podrá `Buscar producto`. Si el cliente está identificado (con `Identificarse` y, si fuera necesario, `Registrarse`) podrá, cuando haya encontrado el producto, `Añadir Producto a la Cesta`.

Además, el usuario identificado podrá seleccionar `Realizar Compra`, la cual podrá o no ocasionar el caso de uso `Usar Cupón`.

El realizar la compra Invocará siempre, eso si, a `Realizar Pago`, empleando un actor externo de `Sistema de Pago`, y también usará siempre `Introducir datos de envío`.

Finalmente, el usuario podrá `Ver perfil` e `Introducir Dirección de Envío`.

6.Resumen y ejercicios

Resumen general de los casos de uso

- Los actores principales son los que realizan las tareas (realizan los casos de uso), y los secundarios son los que son llamados por el sistema. Los actores están comunicados con los casos de uso a través de relaciones de asociación (líneas y no flechas).
- Es posible que un actor sea primario en un caso de uso y secundario en otro. La comunicación entre los actores fuera del sistema (fuera de la aplicación a desarrollar) no se refleja en los diagramas.
- Si de un caso de uso pasamos a otro directamente (ejemplo, si un usuario va a consultar su perfil pero necesitar estar identificado, y ello se hace de forma inmediata para luego volver), entonces se pondrá el {include} o el {extends} que corresponda entre ambos casos de uso. En caso contrario, si el programa no “pasa el control” directamente al otro caso de uso, entonces no se pone {include} ni {extends}, aunque haya prerequisito que los relacione.
- El {include} tiene la flecha desde el caso de uso “llamante” hacia el caso de uso “llamado”, pues lo incluye. El {extend} lo tiene al revés porque el “llamado” es extendido.
- La herencia solo es posible entre actores, y solo si el actor hijo puede hacer todos los casos de uso del padre y puede que alguno más. Aún así, si no hay relación entre ambos actores (ej: un cliente y un encargado, ambos de ellos pueden realizar un “cambio de contraseña” y luego el cliente puede “consultar perfil”), podría no haber herencia.

Ej. 19: [1] Una clínica veterinaria almacena datos de contacto de todos sus clientes, que son introducidos y gestionados por los auxiliares, que ejercen las funciones administrativas.

[2] También es posible dar de alta a un nuevo animal. Un animal solo puede pertenecer a un único cliente, pero es posible que éste cambie de dueño. Para animales con la obligación de estar identificados, al darlo de alta en el sistema, se comprobará el registro REIAC (Red Española de Identificación de Animales de Compañía) para saber si el animal está correctamente dado de alta.

[3] Cada vez que un veterinario examina un animal, dicha consulta queda almacenada, con todo los siguiente: (1) los síntomas del animal, (2) las recetas que se le prescriban,, (3) los tratamientos realizados, (4) los tratamientos y cuidados prescritos. Si fuera necesario, el veterinario podrá ordenar el veterinario el ingreso en clínica, cuya orden quedará también registrada.

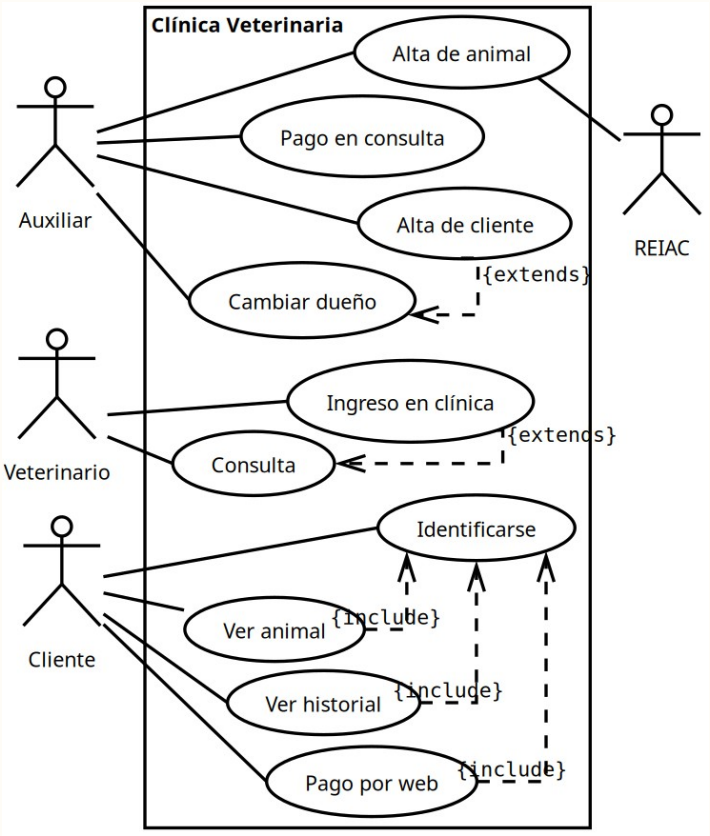
[4] Si un animal queda ingresado en la clínica, el cliente debe ser capaz de ver, en tiempo real (a través de una cámara fija), al animal ingresado, pudiendo ver también su historial, recetas prescritas y cuidados/comentarios indicados por el veterinario en la consulta o después.

[5] Mientras se está tratando al animal ingresado, todas las anteriores son añadidas al historial.

[6] La consulta realizada a sus animales, podrán ser consultados por el cliente desde la web.

[7] El cliente podrá realizar el pago justo tras realizar la consulta, con la ayuda de un auxiliar. También puede identificarse en la web para hacerlo en la siguiente semana.

[8] La identificación de los clientes en la web se realiza con un usuario y clave, pero el registro se hace a través de los auxiliares, en la clínica. Los auxiliares y veterinarios no han de identificarse (ésto se realiza de forma automática, pues cada uno usa su propio sistema informático).



[1] Este párrafo nos indica un caso de uso llamado `Alta de cliente`, que es realizado por un actor llamado `Auxiliar`. Se considera que es el auxiliar el que realiza el proceso (habla con el cliente, recibe sus datos por teléfono o por email, etc.), por lo que, en dicho caso, el cliente no forma parte del caso uso.

En caso de que el cliente usara la aplicación a desarrollar (se le enviara un enlace que tuviera que rellenar, usara una aplicación nuestra en el móvil, etc.), entonces el cliente formaría parte del caso de uso, y habría que asociarlo con una línea, NO con una flecha) al caso de uso.

[2] Nos indica el caso de uso `Alta de Animal`, con las mismas consideraciones respecto al cliente que en [1]. En todo caso, podría ser que

tengamos que consultar el sistema externo llamado `REIAC`, por lo que enlazamos a ese actor, que sería actor secundario) con el caso de uso.

Si se considerase un caso de uso para hacer la conexión con REIAC (por ejemplo, `Consulta con REIAC`), éste caso de uso estaría conectado con `Alta de animal` con un {extends} desde `Consulta con REIAC` el hacia `Alta de animal`. Sin embargo, la consulta parece ser más bien solo un paso en el flujo secundario de `Alta de Animal`, por lo que evitamos esta opción.

En este párrafo también se indica que es posible un cambio de dueño. Será un caso de uso realizado por el Auxiliar. En este caso, si el nuevo dueño no está en el sistema, tendremos que hacer un `Alta de cliente`, algo que se realizará al momento, por lo que supone un {extends} con dicho caso de uso. Observa que, en caso de que el animal no estuviera dado de alta, entonces no se podría hacer un cambio de dueño, sino simplemente sería un alta de animal.

[3] Nos indica que el veterinario realiza el caso de uso `Consulta` y, el de `Ingreso en clínica`. Éste último siempre se realizará a partir del primero, pero no siempre. Por tanto hay una relación de {extends}. Ambas tareas son iniciadas y realizadas por el Veterinario, por lo que deben estar enlazadas a él con una asociación (una línea).

Observa que la dirección de la flecha de {extends} es al contrario que la de {include}. En este caso, `Consulta` llama a `Ingreso en clínica` (aunque no sea siempre), por lo que la flecha va desde el primero al segundo. Si la llamada fuera siempre (si siempre que se hace una consulta se ingresara en clínica, entonces sería un include con la dirección de la flecha al revés.

El sistema no nos indica aquí que un animal deba estar registrado para ser consultado pero, en vista de lo dicho en [2], es posible que esto sea necesario. En ese caso, nos planteamos si hay que realizar un {include} o un {extends} hacia `Alta de animal`. En principio, son procesos distintos, puesto que la tarea de `Alta de animal` puede haberse realizado mucho antes, por lo que no habría {include} ni {extends}. Existe una dependencia NO funcional que se materializará en un prerequisito, pero no habrá {include} ni {extends}. Si, por

el contrario, se decidiera que la consulta conlleva a un `Alta de animal` en ese mismo momento, entonces habría un {extends} desde `Alta de Animal` a `Consulta`.

[4] Se crean los casos de uso `Ver animal` y `Ver historial` que son asociados al actor principal `Cliente`. Todos los objetos de recetas, tratamientos y demás son objetos, pero no son casos de uso ni actores. Podrán aparecer en los flujos principales y secundarios de los casos de uso, pero no en el diagrama frontera.

[5] Este párrafo indica que, mientras un animal está ingresado, se realiza una consulta, aunque no se aclara cuándo sucede esto. No se añadiría nada al diagrama, pues el caso de uso `Consulta` ya está.[6] Vuelve a incidir en el caso de `Ver historial`, que ya está hecho. Nos queda claro, si no lo estaba ya, que `Ver historial` y `Ver animal` son independientes.

[7] El pago puede realizarse a través de la web directamente por el `Cliente`, por lo que creamos un caso de uso llamada `Pago por web`, y lo asociamos con el actor en cuestión. El pago por consulta será realizado por el auxiliar, y tiene las mismas consideraciones que `Alta de cliente` o `Alta de animal` respecto a enlazar a Cliente o no.

[8] Finalmente, el `Identificarse`, por parte del cliente, será un caso de uso que será incluido por los 3 casos de uso relativos a la web. En cuanto al registro, éste debe ser realizado por el auxiliar. Consideramos que es parte de `Alta de Cliente`, Si tuviera entidad para ser un caso de uso aparte, habría una flecha de {include} desde `Alta de animal` hacia `Registro`.

`Ej. 20:` Realiza el caso de uso `Consulta` del ejercicio anterior.

En este caso de uso tan solo se trata de que el sistema pida información de las consulta, y el veterinario introduzca los datos de la consulta. Respecto a

estos datos, el enunciado deja claro que hay 4 apartados de información (Síntomas", "Recetas" "Tratamientos realizados", "Tratamientos y cuidados prescritos"). Aunque podríamos haber agrupado todas estas opciones en algo como pedir o insertar datos de consulta, ya que parece que el procedimiento de la consulta es algo claro, se han puesto ya esos apartados. Si esto no fuera así, más tarde tendríamos que arreglarlo, lo que sería un coste en nuestro desarrollo.

Aparte de los 4 apartados citados, existe un quinto que es el de "ingreso en clínica". Este apartado se resuelve llamando al caso de uso "Ingreso en clínica", que será el encargado de guardar los datos relativos a ese ingreso. Observa que, tras la llamada, se sigue el flujo, que en este caso será volver al punto 2.1. También sería correcto modelar lo de "ingreso a clínica" usando un flujo alternativo.

Consulta
Actores: Veterinario
Descripción: El veterinario realizará una consulta a un animal y registrará los tratamientos y recetas
Flujo Principal: 1. El veterinario selecciona "Consulta". 2. Se introducen cada uno de los datos de la consulta. 2.1. El sistema pide que se seleccione entre "Síntomas", "Recetas" "Tratamientos realizados", "Tratamientos y cuidados prescritos", "ingreso en clínica" o "Terminar". 2.2. El veterinario selecciona una de las opciones. 2.3. En caso de que se haya seleccionado "Terminar", se termina. 2.3. En caso de que se haya seleccionado "Ingreso en clínica, se llamará al caso de uso **"Ingreso en clínica"**, y se vuelve a 2.1 2.4. El sistema le pide al veterinario que ingrese la información relativa a la opción seleccionada 2.5. El veterinario introduce dicha información. 2.6. El sistema guarda la información y vuelve al punto 2.1
Postcondiciones: La consulta habrá quedado grabada en el sistema

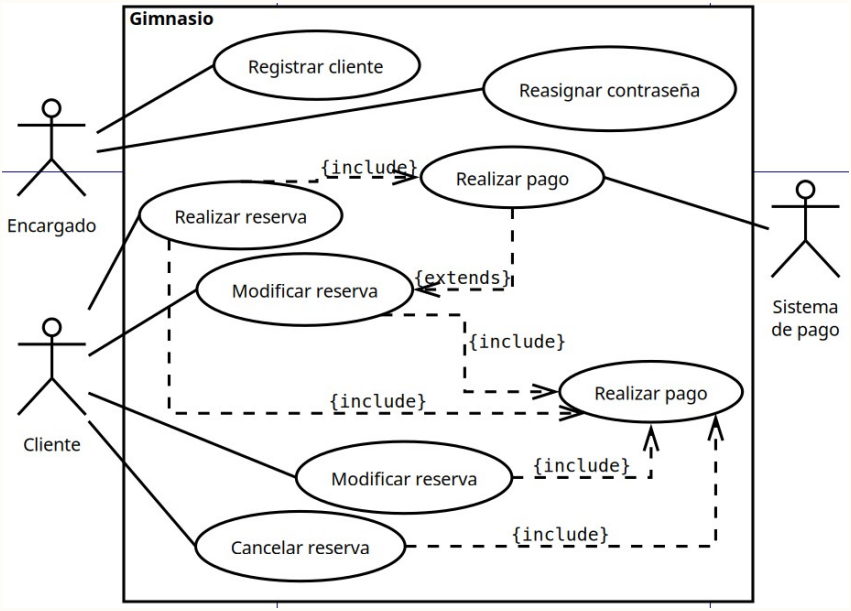
Ej. 21: Un gimnasio quiere implementar un sistema para gestionar reservas de sus clases.

[1] El registro de clientes es realizado por el encargado del gimnasio en el propio gimnasio, proporcionando al cliente los datos de acceso a la web (login y contraseña). El encargado no ha de identificarse ni registrarse en el sistema, pero el cliente si tiene que hacerlo para todo lo siguiente.

[2] Los clientes pueden reservar clases pero, para completar la reserva, deben realizar un pago (un proceso complejo de por sí) el cual debe ser validado por un proveedor externo de pagos.

[3] Los clientes pueden cancelar sus reservas y modificarlas. Puesto que las clases tienen distintos precios, el modificar una reserva puede conllevar un pago, al igual que en la reserva. Cancelar una reserva, o modificar a una reserva hacia una de menor valor supone la pérdida del dinero correspondiente.

[4] El cambio de contraseña puede ser realizado por el encargado del gimnasio, o bien en la web, por el cliente.



[1] El registro de clientes se realiza por parte del Encargado. Como el recabado de datos lo hace el encargado (por lo general, hablando directamente con el cliente cuando acude al mostrador del gimnasio), este proceso se hace fuera del sistema y no hay que reflejarlo. Por tanto, creamos el caso de uso de `Registro` asociado únicamente con el Encargado.

Dejamos pendiente la identificación del cliente para modelarla para más tarde.

[2] Los clientes pueden `Realizar reserva` pero, una parte integral e indispensable es `Realizar pago` que, al ser un proceso complejo, posee su propio caso de uso. Al ser un paso obligado, los conectamos con un {include}. Como este caso de uso necesita al actor secundario de la plataforma de pago, lo conectamos con ella: este actor NO estará conectado al caso de uso de `Realizar reserva`.

[3] Añadimos los casos de uso `Modificar reserva` y `Cancelar reserva`, realizados por el cliente. En caso del primero, es posible recaer en un pago adicional, dado que la nueva reserva sea más cara, por lo que conectamos con `Realizar pago` con un extends (si la nueva reserva vale igual o menos, no se necesitaría ese pago adicional).

[4] Creamos dos casos de usos distintos relativos a cambiar contraseña, ya que estimamos que el proceso sería distinto (en uno el cliente debe estar identificado, etc.). En caso de que fuera el mismo proceso, entonces sería un mismo caso de uso con ambos conectados a él.

Finalmente, conectamos todas la operaciones del cliente con el caso de uso `Identificarse`, que habíamos dejado pendiente en [1].

`Ej. 22:` Realiza el caso de uso Realizar reserva.

En este caso de uso, realizamos una llamada a otro caso de uso, el de `Realizar Pago`. Todo lo relacionado con éste (el pedir datos, la

comunicación con el actor secundario, etcétera, quedan en su ámbito, y no son incluidos aquí: observa que el único actor de este caso de uso es el cliente, y la grabación del pago no se realiza aquí.

Realizar reserva
Actores: Cliente
Descripción: El cliente selecciona y realiza la reserva de una clase.
Precondiciones: el Cliente debe estar identificado.
Flujo Principal: 1. El veterinario selecciona "Realizar reserva". 2. El cliente pide al usuario que seleccione una clase de entre las que haya con plazas libres. 3. El usuario selecciona una clase. 4. El sistema llama al caso de uso **"Realizar Pago"**. 5. El sistema guarda la reserva.
Flujo secundario: 5a. Si el pago no se realiza con éxito, vuelve al punto 2.
Postcondiciones: La reserva queda guardada.

En cuanto al flujo secundario, aunque trata sobre un error, al producirse ese error no se cancela la tarea ni se re-intenta en el mismo paso (en vez de ello, se redirige al punto 2), por lo que se incluye como un flujo secundario en vez de obviarse.

Ej. 23: Tenemos una aplicación web para recomendar libros. El administrador de la web puede subir los datos de un libro (título, autor, etc.), junto con una valoración opcional.

Por su parte, los usuarios pueden realizar comentarios del libro, así como compartir la página web del libro, valoración y comentarios por red social, algo que precisa de una entidad externa llamada "Red Social".

Todos los usuarios deben identificarse y, en su caso, registrarse en la web. El administrador también es capaz de comentar y compartir.

Unidad Didáctica 4:

1. Diagramas de Estado

1.Elementos

Los diagramas de estado muestra los distintos estados que un sistema adopta, y ante qué eventos puede dicho sistema pasar de un estado a otro. Los diagramas de estado no tienen en cuenta las actividades realizadas por el sistema, solo ciertos eventos ocurridos que hagan cambiar su estado.

Inicio
Representa el inicio de del diagrama de estados. Tan solo puede existir uno de estos símbolos, y éste apuntará siempre a un único estado, que será el estado inicial.

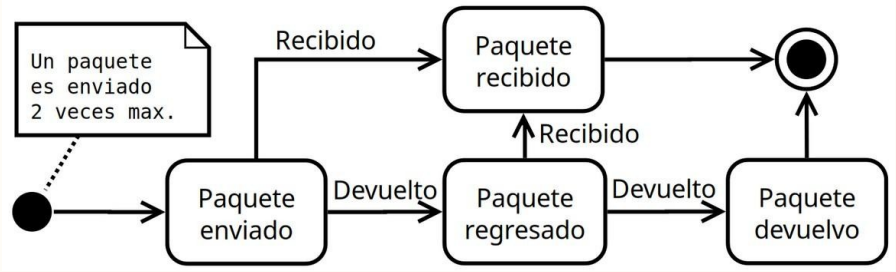
Estado
Indica un estado posible del sistema.

Evento
Un evento es una ocurrencia que puede causar la transición del sistema de un estado a otro.

Nota
Permite añadir notas a los distintos elementos del diagrama de estados.

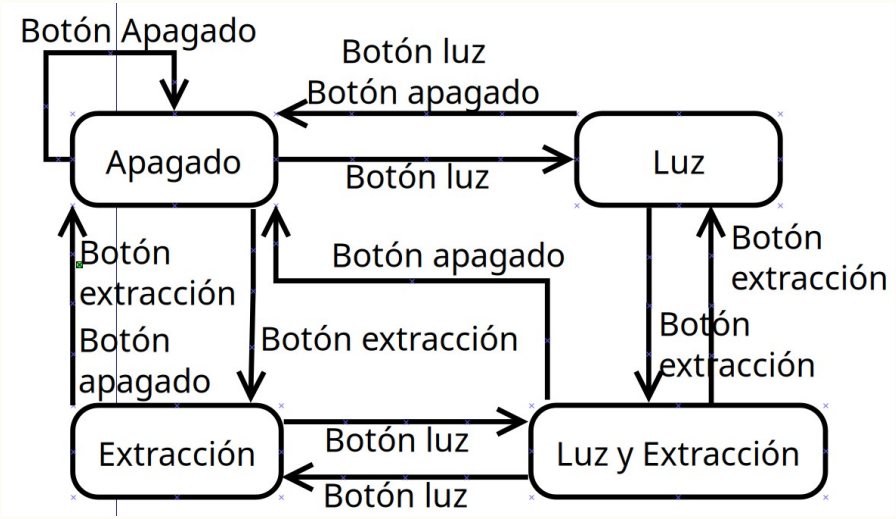
Finalización
Establece el estado final del diagrama de estados, y representa la conclusión de todos los eventos.

El siguiente diagrama de estados refleja un diagrama de los estados de un sistema que realiza envíos de paquetes, y éstos pueden ser devueltos o no. Se envía, como máximo, dos veces un paquete: si es devuelto dos veces, se queda como devuelto, en caso contrario quedará como recibido.

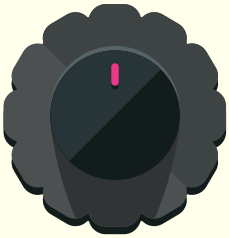


Observa que iniciamos desde un estado de `Paquete enviado`, puesto que se considera que siempre se envía, sin excepción. En este caso, poner un estado previo del estilo `Paquete a enviar` no tiene sentido, a menos que quisiéramos resaltar el evento de envío, o bien si el paquete pudiera no ser enviado en ciertas circunstancias (dirección desconocida, paquete defectuoso, etc.).

Otro ejemplo. Tenemos una campana extractora de cocina. Tiene un botón de apagado (si es pulsado, se apaga todo), un botón para encender o apagar la luz y otro para encender o apagar la función extractora:



Ej. 1: Realiza el diagrama de un dial, que inicialmente estará apuntando al `0`. El dial puede ser manipulado para que pase al `1`, al `2`, al `3` y, finalmente, al `4`, y también se podrá volver a posiciones inferiores. En principio, no podría pasar directamente del `4` y el `0` ni viceversa. De cara a los elementos de inicio o fin, el objetivo es establecer un valor y, posteriormente, volver al `0`.

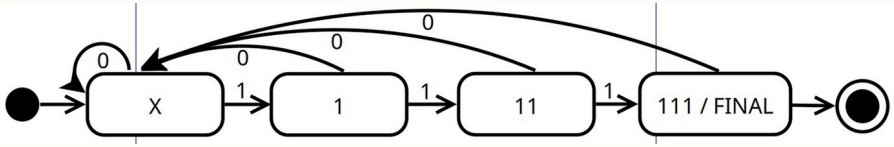


¿Qué cambiaría en el diagrama de estados para que si se pueda cambiar entre el `0` y el `4`?

Ej. 2: Realiza un ejercicio que modele una máquina expendedora. La máquina puede recibir solamente monedas de 0,5€, 1€ y 2€. El único producto que vende es de 2€. La máquina saca el producto en cuanto tiene dinero suficiente, expulsando el resto de monedas que excedan 2€ ¿Es posible que la máquina no pueda devolver el cambio completo?

2.Eventos de salida

Los diagramas de estados son buenos indicadores de cuando un sistema debe producir un evento de salida. El siguiente diagrama corresponde a un sistema que va recibiendo unos y ceros y detecta cuando la ha llegado la marca de fin, compuesta por tres unos seguidos. El estado llamado `111` posee una salida, `FINAL`:



Observa, además, que en este diagrama hay un evento que, de ocurrir, lleva al mismo estado.

Ej. 3: Realiza el diagrama de estados de un calentador con termostato. El calentador comprueba continuamente la temperatura y, si ésta es mayor que la temperatura establecida, se apaga, en caso contrario, se enciende.

Añade luego los eventos en los que se cambia la temperatura establecida.

Ej. 4: Realiza el diagrama de estados de una máquina expendedora. En dicha máquina, es posible introducir monedas, tras lo cual puede seleccionarse uno de los productos, y entonces la máquina sacará el producto y devolverá la cantidad restante.

Sin embargo, es posible que la cantidad de monedas no sea suficiente para el producto, o que el producto esté agotado, en cuyo caso la máquina volverá a pedir una selección. También es posible que el exista un error, o bien que el usuario pulse el botón de cancelar, en cuyo caso se devuelve el dinero.

Ej. 5: En D&D, cuando tu personaje pierde todos sus puntos de vida, cae moribundo. Cada turno, lanzas un dado de veinte y, según la siguiente tabla, sufre los siguientes efectos:

1: dos puntos 💀, 2-10: un punto 💀, 11-19: un punto ❤️, 20: dos puntos ❤️.

Si el personaje acumula tres o más puntos de 💀 (independientemente de los puntos ❤️ que tenga), muere. Si acumula tres o más puntos de ❤️ (independientemente de los puntos 💀 que tenga), se estabiliza. Haz un diagrama que modele este sistema.

Ej. 6: En 1989, la editorial R. Talsorian Games publicó el juego, creado por Mike Pondsmith, "Cyberpunk", junto a una segunda edición "Cyberpunk 2020" en 1990, que fue traducida el español y, varios años después, en 2005, una tercera versión "Cyberpunk V3.0" que tuvo mucha peor acogida. La segunda versión de este juego de rol fue considerado, en 1996, uno de los 10 juegos más populares de todos los tiempos. El juego está ambientando en el año 2013 (2020 en la segunda edición, 2030 en la tercera edición).

Mucho tiempo después, en 2020, el estudio CD Projekt RED creó el videojuego "Cyberpunk 2077", ambientado en el mismo año, sacando también el juego de rol asociado Cyberpunk RED", basado en los juegos de rol, aunque se establece la tercera edición como una versión alternativa no contemplada para esta continuación. Este videojuego dio origen la serie animada "Cyberpunk: Edgerunners", ambientada en el mismo año y creada por Studio Trigger.

Todo este universo está originado por las novelas de los '80, especialmente "El neuromante" de William Gibson", que originó una de las películas más míticas del género, "Blade Runner" (publicada en 1982) ambientada en 2019, que tuvo un remake llamado Blade Runner 2049 (publicada en 2017), ambientada en dicho año.

Crea un diagrama de estados por los que ha pasado el universo de Cyberpunk. Como estados, emplea los nombres de juegos o series, tales como "Cyberpunk 2077, ambientado en 2077". Como transiciones, emplea el nombre de la editorial o estudio que hizo la nueva creación y el año, por ejemplo "R. Talsorian Games, 1990".

Ej. 7: Realiza el diagrama de estados de una llamada telefónica. Cosas como "pulsar un número" NO serán un evento, sino que lo serán cosas como "introducir un número telefónico".

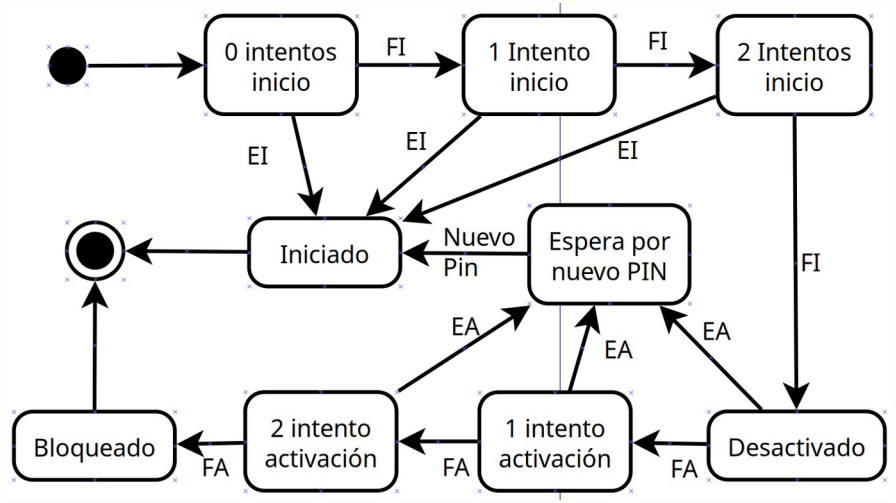
Ej. 8: Realiza el diagrama de estados de un mensaje de correo electrónico.

Considera que todos los correos son internos a un mismo servidor de correo.

Ej. 9: Realiza un sistema de autentificación de PIN telefónico, donde el usuario debe introducir su PIN para iniciar el móvil. Si la clave es introducida mal 3 veces, el sistema se desactiva.

Para activarlo, puede introducirse un segundo código (que viene con la tarjeta telefónica) que, si es introducido correctamente, el sistema esperará para poder introducir un nuevo pin. Tras introducir el nuevo pin, el teléfono se inicia (no hay que introducir nuevamente el pin para iniciarlo).

Sin embargo, si se introduce mal el código de activación 3 veces, la tarjeta se bloquea definitivamente.



En este diagrama de estados, el estado inicial será uno en el que todavía no hayamos realizado ningún intento, puesto que desde el símbolo de inicio (el punto de arriba a la izquierda) hasta el estado inicial no puede existir ninguna transición (solo una flecha). Este estados se ha llamado “o intentos inicio” (podríamos haberlo llamado “inicio” o “reposo”).

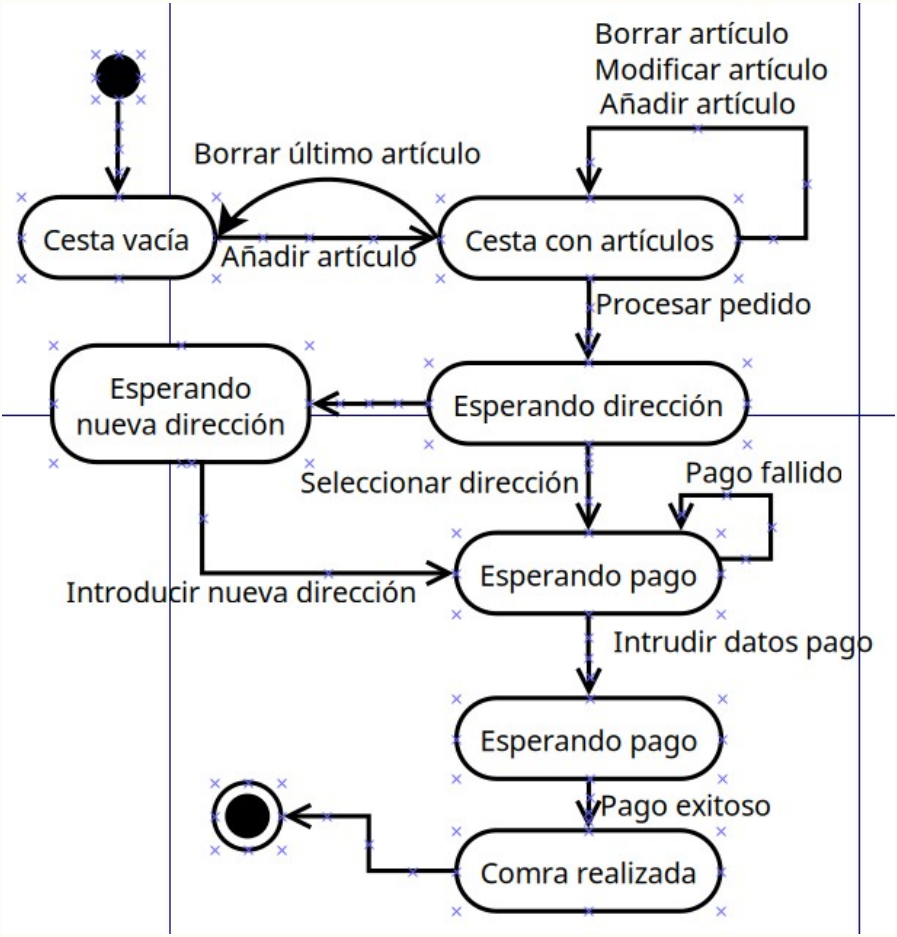
A partir de ahí, están los 3 estados de 0, 1 y 2 intentos fallidos. Si, estando en cualquiera de ellos, realizamos un éxito de inicio (EI), el teléfono se iniciará y terminamos. Si, por el contrario, realizamos un fallo de inicio (FI), nos iremos a donde haya más intentos de inicio fallidos (de “0 intentos de inicio a 1, etcétera). Sin embargo, no hay un estado de ”3 intentos de inicio”, puesto que si se han realizado 3 intentos, el teléfono está ya desactivado.

En cuanto a los intentos de activación, sería similar, con los “Fallos de activación” (FA) haciendo progresar de estado. Tampoco existe un “3 intentos de activación”, porque en ese punto el teléfono está ya bloqueado y ahí terminaríamos.

En caso de estar el teléfono desactivado, si conseguimos un “éxito de activación” (EI), entonces se nos pedirá un nuevo pin y, tras ponerlo el usuario, nos vamos a un teléfono activado directamente. Observa que, desde el estado “espera nuevo pin” solo hay una transición posible, hacia “iniciado”.

Ej. 10: Realiza el diagrama de estados de un proceso de compra en un ecommerce. Los estados incluyen: "Cesta vacía", "Cesta con productos", "Proceso de pago", "Pago exitoso", "Pago fallido", "Compra confirmada" y otros. Considera transiciones basadas en las acciones del usuario y de la plataforma de pago, y unos estados basados en el estado de la cesta/compra.

Aquí, no es posible, ni interesante, controlar el número exacto de artículos que tiene la cesta, tan solo es necesario controlar si está vacía o no, puesto que de estarlo, no cebe ser posible el procesar pedido.



Ej. 11: Realiza el diagrama de una máquina electrónica de café. La máquina estará, inicialmente, apagada. El usuario deberá pulsar un botón para encenderla, momento en el que la cafetera encenderá una luz y esperará que el usuario introduzca café y elija un programa. La cafetera empezará cuando ambas cosas se realicen, puesto que tiene un detector de peso para saber si se ha introducido café o no. Nota que es posible cambiar el programa mientras no se haya iniciado.

Cuando el programa está en funcionamiento es posible que ocurran varias cosas: (1) el usuario cancele el programa, (2) se produce un error

(3) se completa el programa seleccionado o (4) el usuario apaga la cafetera. La máquina entonces volverá a solicitar un programa.

Considera que el proceso finaliza cuando se vuelva a apagar la cafetera.

Ej. 12: Realiza el diagrama de un conjunto de semáforos que controla una intersección en forma de T, donde hay una calle principal (los lados izquierdo y derecho) y un acceso secundario (la parte hacia abajo), donde el tráfico es mucho menor.

Por defecto, los semáforos permiten el tráfico en la cale principal y bloquean el tráfico desde la calle secundaria. Cuando un coche es detectado en la calle secundaria, pueden suceder dos cosas: (1) si no hay tráfico en la calle principal, se permite el paso a la calle secundaria por 20 segundos y luego se da preferencia de nuevo a la calle principal. (2) Si hay tráfico en la calle principal, esperamos 30 segundos antes de dar paso al acceso secundario durante 20 segundos y también volvemos luego a dar preferencia a la calle principal.

También es posible que suceda otro evento, que es que sean las 11:00pm, en el que el semáforo se establece con luz amarilla intermitente hasta las 7:00am, donde se vuelve al estado normal, empezando con la preferencia a la calle principal.

Ej. 13: Un PNJ de un juego tiene está, por defecto, quieto. Si se pulsa sobre él, éste recitará una misión (pulsar de nuevo mientras recita la misión no tendrá efecto. Una vez complete de recitar su misión, podrán pasar 3 cosas.

(1) El usuario podrá declinar la misión propuesta, en cuyo caso el PNJ dirá alguna frase y luego volverá al estado inicial. La misión seguirá disponible si se pulsa de nuevo sobre el PNJ.

(2) El usuario podrá aceptar la propuesta, en cuyo caso el PNJ dirá alguna frase más y entonces el usuario podrá ir a obtener todas las condiciones de la misión. Considera que cada una de estas condiciones es independiente, pero pueden ser varias. Cuando todas están satisfechas, el usuario puede volver a hacer click en el PNJ, que dirá algo más y la misión ya no volverá a estar disponible.

Ej. 14: Haz el diagrama de estados de una aplicación de cámara. La cámara podrá establecer parámetros de zoom, flash y filtro, aunque tiene unos parámetros por defecto. En todo caso, el botón para realizar la fotografía solo estará activo cuando haya iluminación suficiente, a menos que el uso del flash esté activado. Cuando se pulse el botón para hacer la fotografía, ésta se guardará en la galería, y se dará opción a compartirla (una vez se selecciona compartir, los pasos siguientes pertenecen al sistema, no a nosotros). En ambos casos, se llega de nuevo al estado inicial (se considera final ya en este punto).

Ej. 15: [ej. examen] Realiza el diagrama de estados de un sistema de autentificación de una puerta con apertura automática a través de un pin numérico.

La puerta, inicialmente cerrada, esperará a que sea insertado un PIN y, si éste es correcto, la puerta se abrirá. Tras pasar 30 seg, la puerta se cierra (se considera ésto como la finalización del diagrama de estados).

Si el PIN es introducido incorrectamente, no sucederá nada pero, si la el pin es introducido mal 2 veces, la puerta se bloquea, no admitiendo pin ninguno. Pasados 10 minutos, la puerta vuelve al estado de reposo (estableciéndose el número de intentos fallidos a 0).

Desde el interior, si la puerta está bloqueada, es posible accionar un pulsador y la puerta se abrirá durante 1h, tras lo cual se cerrará (estableciéndose el número de intentos fallidos a 0).

Ej. 16: [ej. examen] Crea el diagrama de estados de un programa de facturas. El usuario podrá añadir o eliminar productos a una lista de artículos (considera que el evento de "añadir artículo" incluye también especificar artículo). Una vez seleccionados los artículos deseados, el usuario seleccionará "continuar", en ese caso, el programa pedirá los datos del destinatario. Si el destinatario es una empresa, aplicará un porcentaje de descuento preestablecido, en caso contrario, no se aplicará descuento. Finalmente, el sistema generará la factura.

Especifica, de alguna forma en lenguaje UML, las situaciones en las que una factura es, finalmente, creada.