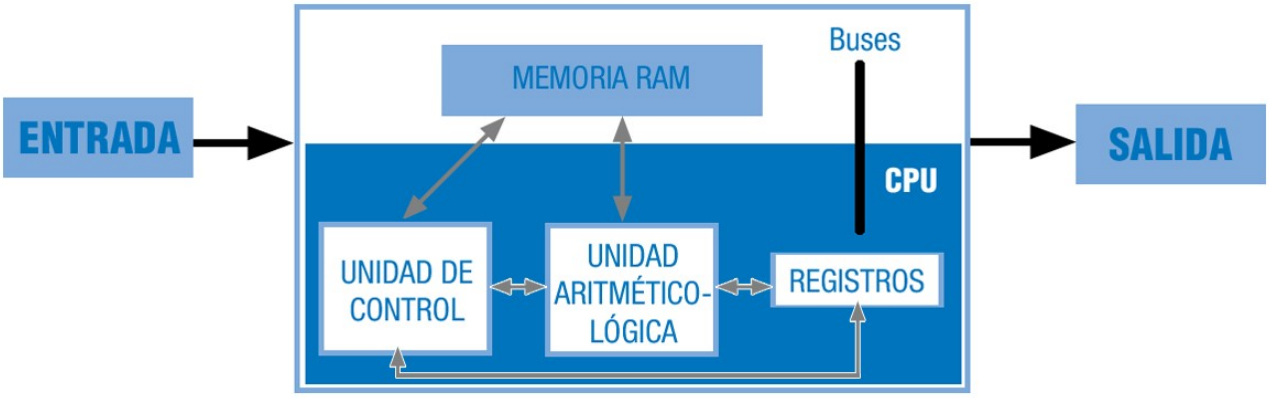
RESUMEN U1: Desarrollo del Software

TIPOS DE SOFTWARE

Para saber más
Tipos de software

https://es.slideshare.net/susahhreyyhha/tipos-de-software-15979630

El software, sea del tipo que sea, se ejecuta sobre los dispositivos físicos del ordenador. ¿Qué relación hay entre ellos?



Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:

- Desde el punto de vista del sistema operativo

El sistema operativo es el encargado de **coordinar** al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.

Todas **las aplicaciones necesitan recursos hardware** durante su ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de Entrada/Salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).

- Desde el punto de vista de las aplicaciones

Ya hemos dicho que una aplicación no es otra cosa que un conjunto de programas, y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Hay multitud de lenguajes de programación diferentes (como ya veremos en su momento). Sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, **el hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión)** que, en informática, se traducen en secuencias de 0 y 1 (código binario).

Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?.

AUTOEVALUACIÓN
Para fabricar un programa informático que se ejecuta en una computadora:

a) Hay que escribir las instrucciones en código binario para que las entienda el hardware.
b) Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
c) Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
d) Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

Reflexiona

Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito. La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir. ¿Por qué el porcentaje de fracaso es tan grande? ¿Por qué piensas que estas causas son tan determinantes?

CICLOS DE VIDA DEL SOFTWARE

1. Modelo en Cascada

Es el modelo de vida clásico del software.

Es prácticamente imposible que se pueda utilizar, ya que requiere conocer de antemano todos los requisitos del sistema. Sólo es aplicable a **pequeños desarrollos**, ya que las etapas pasan de una a otra sin retorno posible. (se presupone que no habrá errores ni variaciones del software).

2. Modelo en Cascada con Realimentación

Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, **si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo**.

Es el modelo perfecto **si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros**.

3. Modelos Evolutivos

Son más modernos que los anteriores. Tienen en cuenta la **naturaleza cambiante y evolutiva del software**.

Distinguimos dos variantes:

1. Modelo Iterativo Incremental

Está basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.

2. Modelo en Espiral

Es una combinación del modelo anterior con el modelo en cascada. En él, el software se va construyendo repetidamente **en forma de versiones** que son cada vez mejores, debido a que incrementan la funcionalidad en cada versión. Es un modelo bastante complejo.

AUTOEVALUACIÓN
Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?

Herramientas de apoyo al desarrollo - CASE

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.

¿En qué fases del proceso nos pueden ayudar?

En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores…

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:

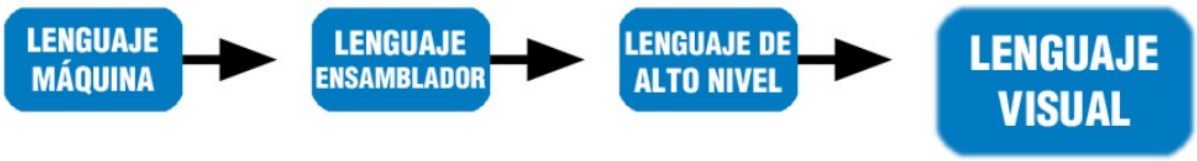
- U-CASE: ofrece ayuda en las fases de **planificación y análisis** de requisitos.
- M-CASE: ofrece ayuda en análisis y diseño.
- L-CASE: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder…

Para saber más

https://oposicionestic.blogspot.com/2016/10/herramientas-case-temario-de-las.html

Lenguajes de Programación



Características de los Lenguajes de Programación

- **Lenguaje máquina**:
  - Sus instrucciones son combinaciones de **unos y ceros**.
  - Es el único lenguaje que entiende directamente el ordenador. (No necesita traducción).
  - Fue el primer lenguaje utilizado.
  - Es **único para cada procesador** (no es portable de un equipo a otro).
  - Hoy día nadie programa en este lenguaje.
- **Lenguaje ensamblador**:
  - Sustituyó al lenguaje máquina para facilitar la labor de programación.
  - En lugar de unos y ceros se programa usando **mnemotécnicos** (instrucciones complejas).
  - Necesita t**raducción al lenguaje máquina para poder ejecutarse**.
  - Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo.
  - Es difícil de utilizar.
- **Lenguaje de alto nivel basados en código:**
  - Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación.

•En lugar de mnemotécnicos, se utilizan **sentencias y órdenes derivadas del idioma inglés**. (Necesita traducción al lenguaje máquina).
•Son más cercanos al razonamiento **humano**.
•Son utilizados hoy día, aunque la tendencia es que cada vez menos.

•**Lenguajes visuales:**

•Están sustituyendo a los lenguajes de alto nivel basados en código.
•En lugar de sentencias escritas, se programa **gráficamente usando el ratón** y diseñando directamente la apariencia del software.
•Su correspondiente **código se genera** automáticamente.
•Necesitan traducción al lenguaje máquina.
•Son **completamente portables** de un equipo a otro.

Para saber más

Evolución de los lenguajes de programación

https://www.monografias.com/trabajos38/tipos-lenguajes-programacion/tipos-lenguajes-programacion.shtml

Listado con los lenguajes de programación y características

http://www.larevistainformatica.com/LENGUAJES-DE-PROGRAMACION-listado.html

Programación Esctructurada

Sentencias de control: http://programacioncitlajessiyaz.blogspot.com/p/unidad-3-estructuras-de-control.html

La programación estructurada fue de gran éxito por su sencillez a la hora de construir y leer programas.
Fue sustituida por la programación modular, que permitía dividir los programas grandes en trozos más pequeños (siguiendo la conocida técnica "divide y vencerás").
A su vez, luego triunfaron los lenguajes orientados a objetos y de ahí a la programación visual (siempre es más sencillo programar gráficamente que en código, ¿no crees?).

VENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA

•Los programas son fáciles de leer, sencillos y rápidos.
•El mantenimiento de los programas es sencillo.
•La estructura del programa es sencilla y clara.

INCONVENIENTES

•Todo el programa se concentra en un único bloque (si se hace demasiado grande es difícil manejarlo).
•No permite reutilización eficaz de código, ya que todo va "en uno". Es por esto que **a la programación estructurada le sustituyó la programación modular**, donde los programas se codifican por módulos y bloques, permitiendo mayor funcionalidad.

La Programación estructurada evolucionó hacia la Programación modular, que divide el programa en trozos de código llamados módulos con una funcionalidad concreta, que podrán ser reutilizables.

Ejemplos de lenguajes estructurados: Pascal, C, Fortran.

AUTOEVALUACIÓN
¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?

ANÁLISIS

Se especifican y analizan los **requisitos funcionales y no funcionales** del sistema.

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.

- No funcionales: Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

Lo fundamental es la **buena comunicación entre el analista y el cliente** para que la aplicación que se va a desarrollar cumpla con sus expectativas.

La culminación de esta fase es el **documento ERS (Especificación de Requisitos Software)**.

En este documento quedan especificados:

- La planificación de las **reuniones** que van a tener lugar.
- Relación de los **objetivos del usuario cliente y del sistema**.
- Relación de los requisitos funcionales y no funcionales del sistema.
- Relación de **objetivos prioritarios y temporización**.
- Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.

Como **ejemplo** de requisitos funcionales, en la aplicación para nuestros clientes de las tiendas de cosmética, habría que considerar:

•Si desean que la lectura de los productos se realice mediante códigos de barras.
•Si van a detallar las facturas de compra y de qué manera la desean.
•Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.
•Si van a operar con tarjetas de crédito.
•Si desean un control del stock en almacén.
•Etc.

DISEÑO

Se debe **dividir el sistema en partes** y establecer qué relaciones habrá entre ellas.

Decidir q**ué hará exactamente cada parte**.

En este punto, se deben tomar decisiones importantes, tales como:

•Entidades y relaciones de las bases de datos.
•Selección del lenguaje de programación que se va a utilizar.
•Selección del Sistema Gestor de Base de Datos.
•Etc.

CITA
Design is not just what it looks like and feels like. Design is how it works. *Steve Jobs ("El diseño no es sólo lo que parece y cómo parece. Diseño es cómo se trabaja"*

CODIFICACIÓN

Durante la fase de codificación se realiza el proceso de programación.

Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

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.

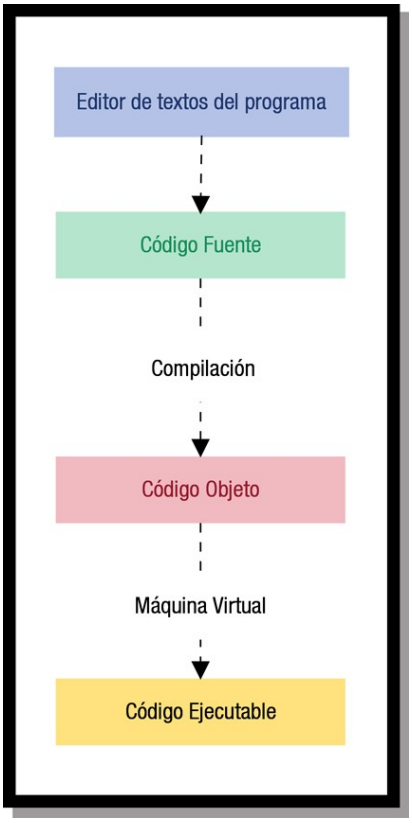
AUTOEVALUACIÓN

¿Crees que debemos esperar Para obtener código fuente a partir de toda la información necesaria del problema:

a) Se elige el Lenguaje de Programación más adecuado y se codifica directamente.
b) Se codifica y después se elige el Lenguaje de Programación más adecuado.
c) Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.

Generación código objeto

https://www.monografias.com/trabajos11/compil/compil2.shtml#co



Tipo de código.	Relación.	Características.
Código Ejecutable		1. Escrito en Lenguaje Máquina pero no ejecutable.
Código Objeto		2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable.
Código Fuente		3. Escrito en Lenguaje Máquina y directamente ejecutable.

Cómo instalar JVM

https://www.java.com/es/download/help/download_options.xml

FRAMEWORKS

Tutorial java Spring v4

https://www.uv.es/grimo/teaching/SpringMVCv4PasoAPaso/index.html

PHP

http://www.maestrosdelweb.com/los-frameworks-de-php-agilizan-tu-trabajo/

AUTOEVALUACIÓN
Señala la afirmación falsa respecto de los entornos de ejecución:

a) Su principal utilidad es la de permitir el desarrollo rápido de aplicaciones.
b) Actúa como mediador entre el sistema operativo y el código fuente.
c) Es el conjunto de la máquina virtual y bibliotecas necesarias para la ejecución.

Instalar JRE

https://www.java.com/es/download/help/linux_install.xml
https://www.java.com/es/download/help/windows_manual_download.xml

PRUEBAS

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).

El período de prueba será normalmente el pactado con el cliente.

AUTOEVALUACIÓN
Si las pruebas unitarias se realizan con éxito, ¿es obligatorio realizar las de integración?

a) Sí, si la aplicación está formada por más de cinco módulos diferentes.
b) Sí, en cualquier caso

Para saber más

Pruebas software

http://isg2.pbworks.com/w/page/7624280/Pruebas%20del%20Software

	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	El diseño de la aplicación. La codificación de los programas. Las pruebas realizadas.	Descripción de la funcionalidad de la aplicación. Forma de comenzar a ejecutar la aplicación. Ejemplos de uso del programa. Requerimientos software de la aplicación. Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: Puesta en marcha. Explotación. Seguridad del sistema.
¿A quién va dirigido?	Al personal técnico en informática (analistas y programadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su objetivo?	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

EXPLOTACIÓN

Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.

La explotación es la fase 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**.

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.
Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente: será el momento crítico del proyecto.

MANTENIMIENTO

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.

Los tipos de mantenimiento del software:

- 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).

AUTOEVALUACIÓN
¿Cuál es, en tu opinión, la etapa más importante del desarrollo de software?

a) El análisis de requisitos.
b) La codificación.
c) Las pruebas y documentación.
d) La explotación y el mantenimiento.

Para saber más

- Definiciones de INGENIERÍA DEL SOFTWARE, CICLO DE VIDA DEL SOFTWARE y ETAPAS DEL CICLO DE VIDA DEL SOFTWARE: http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060024/Lecciones/Capitulo%20I/problemas.htm
- MODELOS DE CICLO DE VIDA: https://www.incibe.es/file/N85W1ZWFHifRgUc_oY8_Xg
- COMO FUNCIONA EL LENGUAJE JAVA: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html
- Conceptos de los Lenguajes POO: https://docs.oracle.com/javase/tutorial/java/concepts/index.html

Resumen de respuesta correcta:

Análisis -> Diagramas de Control de Flujo DCF y Diccionario de Datos DD.

Diagramas de casos de uso CU -> Análisis;
Diagramas de flujo de datos DFD -> Análisis;
Diagramas de Entidad-Relación DER -> Análisis;

Diagramas de transición de estados -> Ambos

Diagramas de clases -> Diseño;
Diagramas de secuencias -> Diseño;