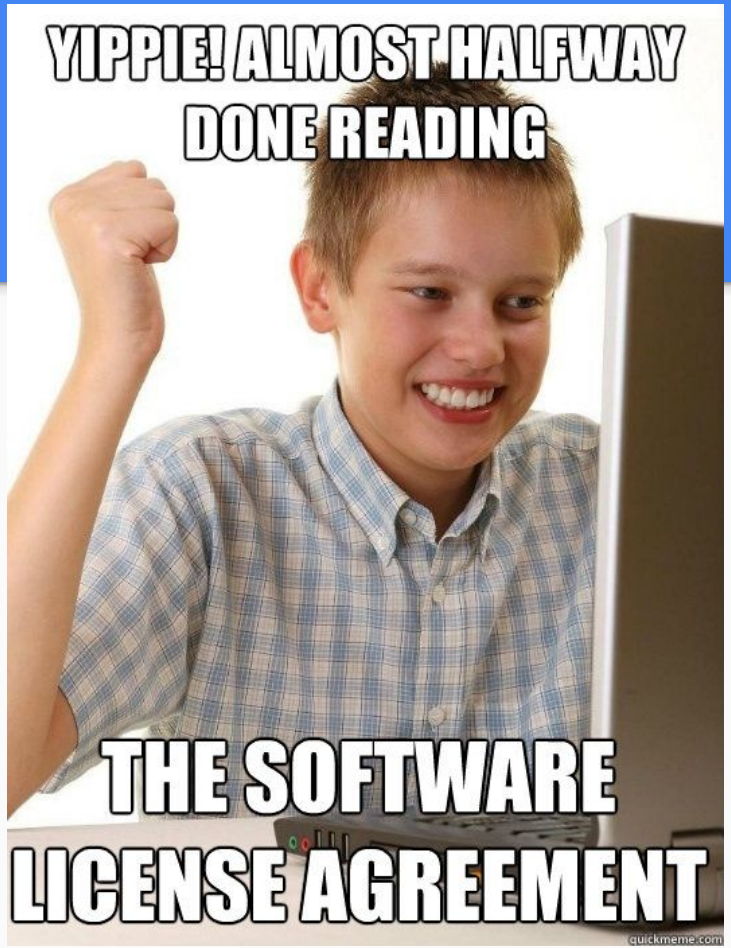
Ciclo de Vida del Software

Licencias de Software

Una **licencia** de software es un contrato que se establece entre el desarrollador de un software, sometido a propiedad intelectual y a derechos de autor, y el usuario, en el cual se definen con precisión los derechos y deberes de ambas partes.



Licencias de Software: Software Libre

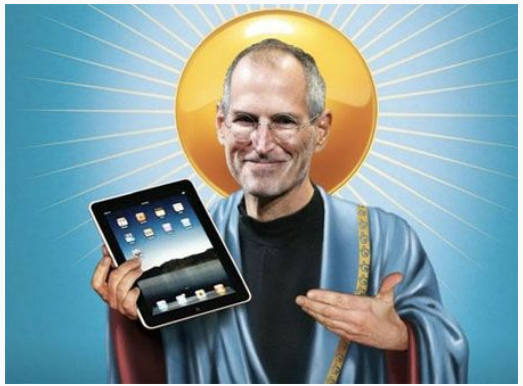
El **software libre** es aquel en el cual el autor cede una serie de libertades básicas al usuario:

- Libertad de utilizar el programa con cualquier fin en cuantos ordenadores se desee.
- Libertad de estudiar cómo funciona el programa y de adaptar su código a necesidades específicas.
- Libertad de distribuir copias a otros usuarios (con o sin modificaciones).
- Libertad de mejorar el programa

Licencias de Software: Software Propietario

El **software propietario** es aquel que, habitualmente, se distribuye en formato binario, sin posibilidad de acceso al código fuente según una licencia en la cual el propietario.

Decompiladores



Ejercicio: ¿Todo es ‘free’? - Libre vs Gratis

¿Qué significa ‘free software’?

¿Quién es Richard Stallman?

¿Richard Stallman sabe cantar? ¿canta bien?

Ciclo de Vida del Software

¿Ciclo de vida del software? ¿Eso quiere decir que el software está vivo?

¿Alguna vez habéis visto a un arquitecto hacer un edificio sin planos?

Construir un edificio vs Crear un programa

¿Por qué crees que se debe planificar el software?

La importancia de planificar el desarrollo de software

El proceso de desarrollo del software implica un conjunto de actividades que se tienen que planificar y gestionar de tal manera que aseguren un producto final que dé solución a las necesidades de todas aquellas personas que lo van a utilizar.

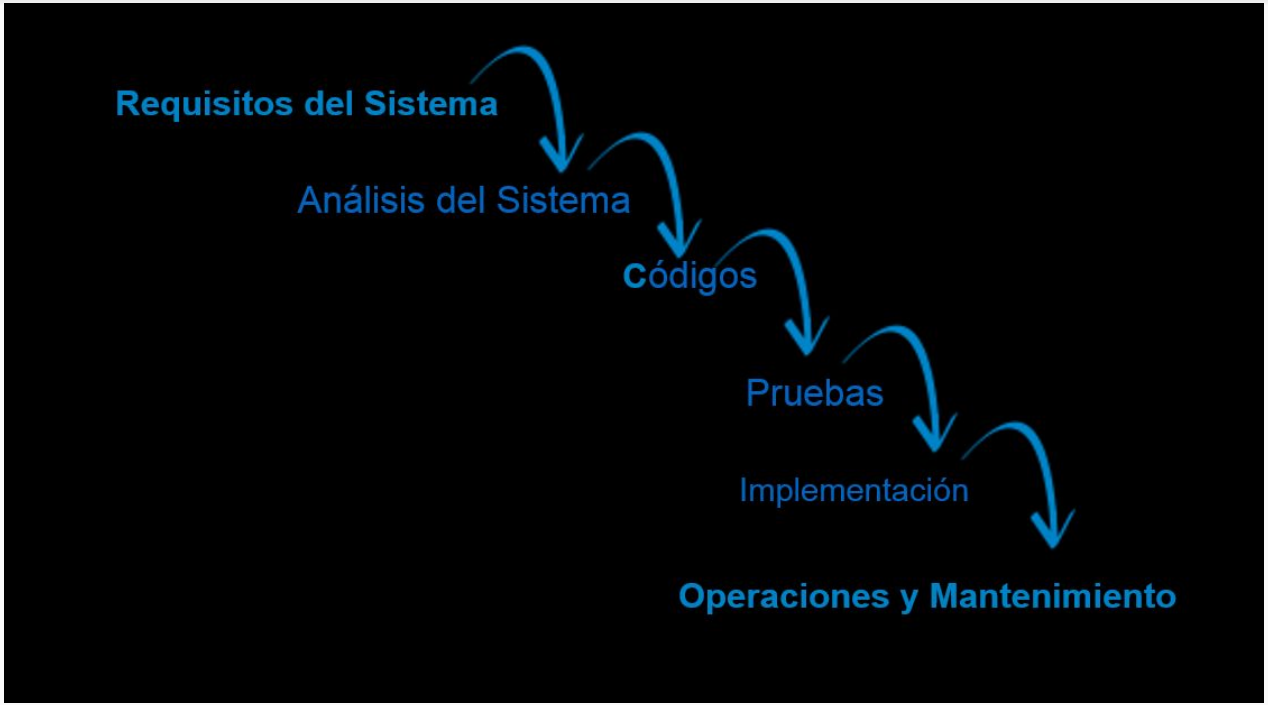
¿Qué es el ciclo de vida?

Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.

Definición de verdad

Esquema/Documento/Servilleta de papel de un bar donde se recogen todos los pasos que hay que dar desde que alguien tiene una idea para desarrollar una web/app/programa hasta que se deja de usar.

Etapas del Ciclo de Vida



Etapas del Ciclo de Vida Episodio I: **Requisitos del Sistema**

Requisitos: necesidades y requerimientos que tiene un sistema. Se documentan de forma ordenada para después poder analizarlos.

- Es decir, **qué** debe hacer el sistema (pero no cómo).
- Se suelen obtener a base de reuniones con los clientes.

Etapas del Ciclo de Vida Episodio II: **Análisis**

- **Análisis**. Construye un modelo de los requisitos. En esta etapa se debe entender y comprender de forma detallada el problema que se va a resolver. Es muy importante producir en esta etapa una documentación entendible, completa y fácil de verificar y modificar.

Etapas del Ciclo de Vida Episodio III: **Diseño**

Diseño: En esta etapa ya sabemos qué es lo que hay que hacer, ahora hay que definir cómo se va a resolver el problema. Se deducen las estructuras de datos, la arquitectura de software, la interfaz de usuario y los procedimientos.
Por ejemplo, en esta etapa hay que seleccionar el lenguaje de programación, el Sistema Gestor de Bases de Datos, etc.

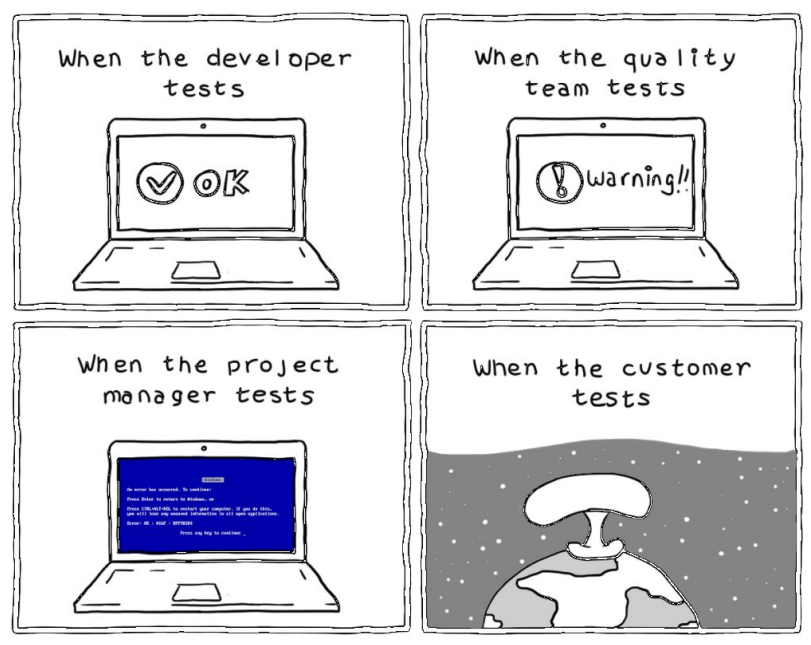
Etapas del Ciclo de Vida Episodio IV: **Implementación**

En esta etapa se traduce lo descrito en el diseño a una forma legible por la máquina. La salida de esta fase es código ejecutable.

- Tomar café
- Escribir código

Etapas del Ciclo de Vida Episodio V: **Pruebas**

Se comprueba que se cumplen criterios de corrección y calidad. Las pruebas deben garantizar el correcto funcionamiento del sistema

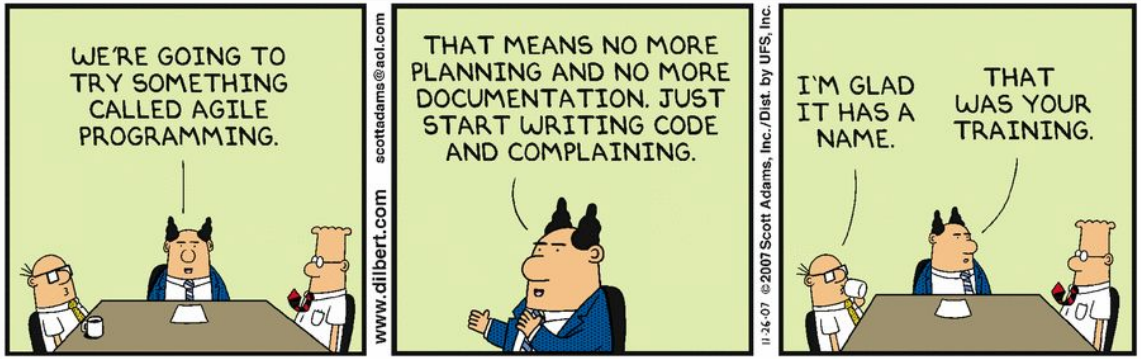


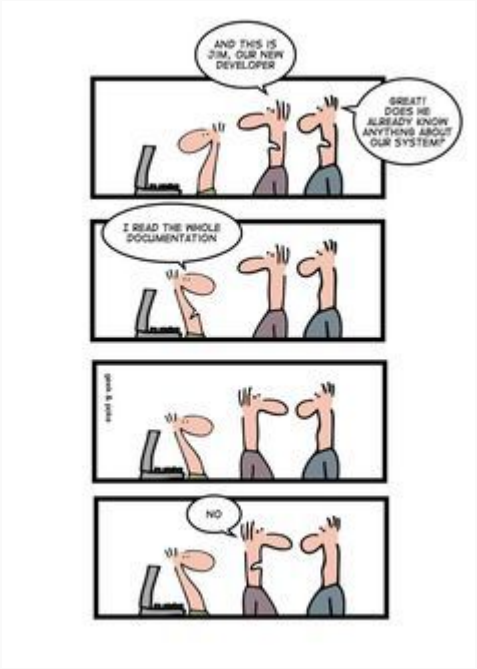
Etapas del Ciclo de Vida Episodio VI: **Mantenimiento**

Esta fase tiene lugar después de la entrega del software al cliente. En ella hay que asegurar que el sistema pueda adaptarse a los cambios. Se producen cambios porque se han encontrado errores, es necesario adaptarse al entorno (por ejemplo se ha cambiado de sistema operativo) o porque el cliente requiera mejoras funcionales.

Etapas del Ciclo de Vida Rogue One: **Documentación**

Es necesario documentar en todas las etapas.



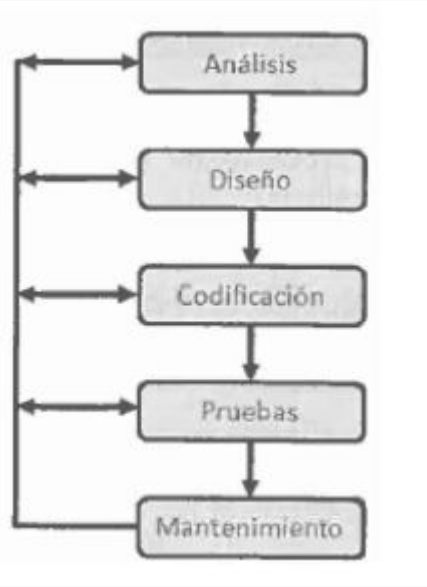


Modelos del Ciclo de Vida

Modelo en Cascada

En este modelo las etapas para el desarrollo del software tienen un orden, de tal forma que para empezar una etapa es necesario finalizar la etapa anterior.

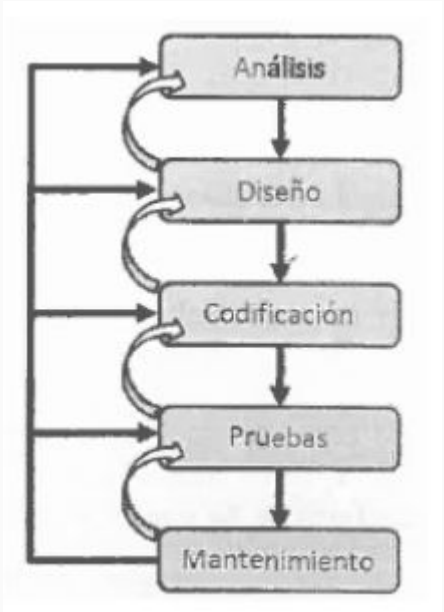
- Iterativo



Modelo en Cascada

Tiene varias variantes, una de la más utilizada es la que produce una realimentación entre etapas, se la conoce como *Modelo en Cascada con Realimentación.*

Por ejemplo, supongamos que la etapa de *Análisis* ha finalizado y se puede pasar a la de *Diseño.* Durante el desarrollo de esta etapa se detectan fallos, entonces será necesario retornar a la etapa anterior, realizar los ajustes pertinentes y continuar de nuevo con el *Diseño.* A esto se le conoce como realimentación



Modelo en Cascada

Ventajas:

- Fácil de comprender, planificar y seguir.
- La calidad del producto resultante es alta.
- Permite trabajar con personal poco cualificado.

Inconvenientes:

- La necesidad de tener todos los requisitos definidos desde el principio (algo que no siempre ocurre ya que pueden surgir necesidades imprevistas).
- Es difícil volver atrás si se cometen errores en una etapa.
- El producto no está disponible para su uso hasta que no está completamente terminado.

Modelo en Cascada

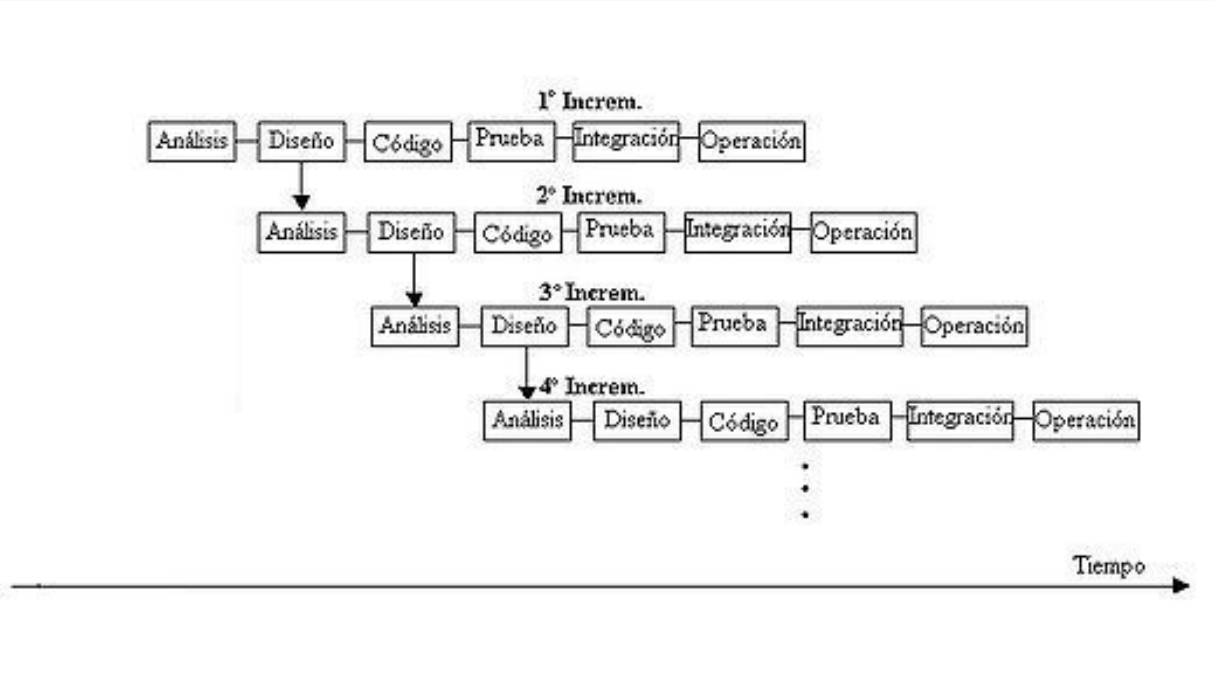
Se recomienda cuando:

- El proyecto es similar a alguno que ya se haya realizado con éxito anteriormente.
- Los requisitos son estables y están bien comprendidos.
- Los clientes no necesitan versiones intermedias.

Modelo Iterativo Incremental

- Basado en varios ciclos cascada realimentados aplicados repetidamente. El modelo incremental entrega el software en partes pequeñas, pero utilizables, llamadas «incrementos» (prototipos).
- Ejemplo: un procesador de textos, en el primer incremento se desarrollan funciones básicas de gestión de archivos y de producción de documentos; en el segundo incremento se desarrollan funciones gramaticales y de corrección ortográfica, en el tercer incremento se desarrollan funciones avanzadas de paginación

Modelo Iterativo Incremental



Modelo Iterativo Incremental

Ventajas:

- No se necesitan conocer todos los requisitos al comienzo.
- Permite la entrega temprana al cliente de partes operativas del software.
- Las entregas facilitan la realimentación de los próximos entregables.

Inconvenientes:

- Es difícil estimar el esfuerzo y el coste final necesario.
- Se tiene el riesgo de no acabar nunca.
- No recomendable para desarrollo de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos.

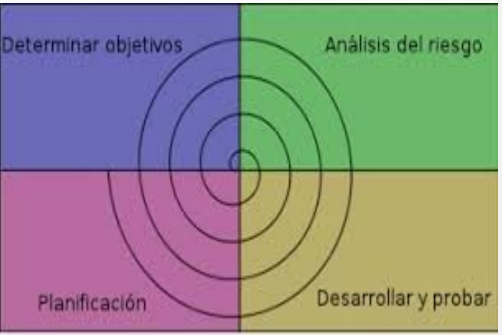
Se recomienda cuando:

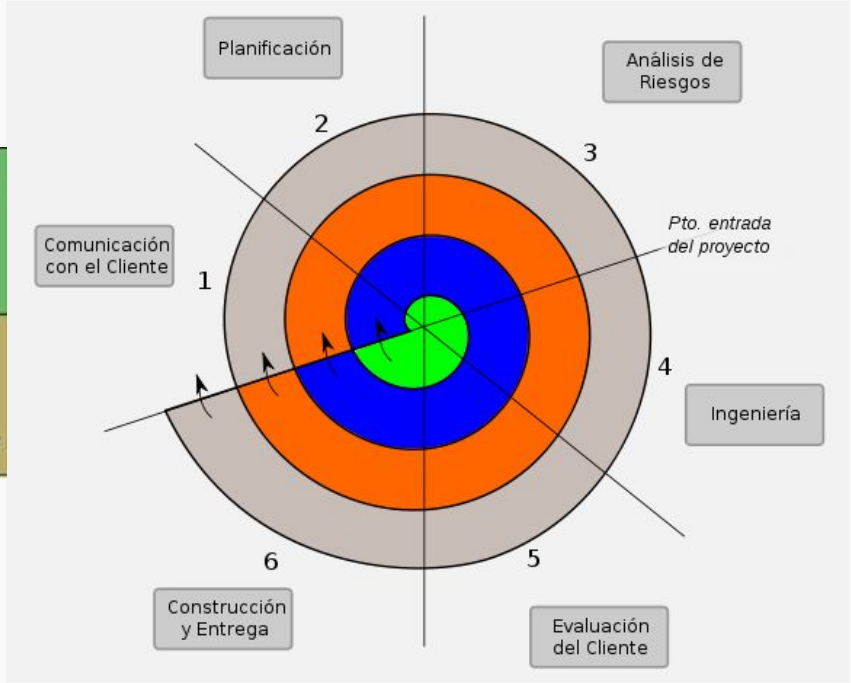
- Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.
- Se están probando o introduciendo nuevas tecnologías.

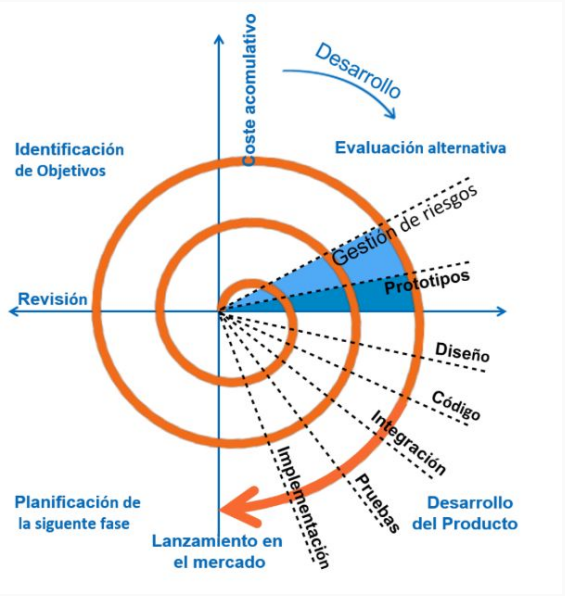
Modelo en Espiral

- Combina el *Modelo en Cascada* con el modelo iterativo de construcción de prototipos
- Se representa como una espiral, donde en cada ciclo se desarrolla una parte del mismo.
  Cada ciclo está formado por cuatro fases, y cuando termina produce una versión incremental del software con respecto al ciclo anterior
- Se parece al *Modelo Iterativo Incremental* con la diferencia de que en cada ciclo se tiene en cuenta el análisis de riesgos.
- Durante los primeros ciclos la versión incremental podría ser maquetas en papel o modelos de pantallas
- En el último ciclo se tendría un prototipo operacional que implementa algunas funciones del sistema.

Modelo en Espiral







Modelo en Espiral

Determinar objetivos: identificación de los objetivos, las alternativas para alcanzar los objetivos y las restricciones impuestas a la aplicación de las alternativas

Análisis del riesgo:

- evaluar las alternativas en relación con los objetivos y limitaciones
- se identifican los riesgos involucrados y (si es posible) la manera de resolverlos. Utiliza la construcción de prototipos

Desarrollar y probar: Desarrollar la solución al problema en este ciclo, y verificar que es aceptable.

Planificación: Revisar y evaluar todo lo que se ha hecho, y con ello decidir si se continúa, entonces hay que planificar las fases del ciclo siguiente.

Modelo en Espiral

Ventajas:

- No requiere una definición completa de los requisitos para empezar a funcionar.
- Análisis del riesgo en todas las etapas.Reduce riesgos del proyecto
- Incorpora objetivos de calidad

Inconvenientes:

- Es difícil evaluar los riesgos.
- El costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
- El éxito del proyecto depende en gran medida de la fase de análisis de riesgos.

Se recomienda para:

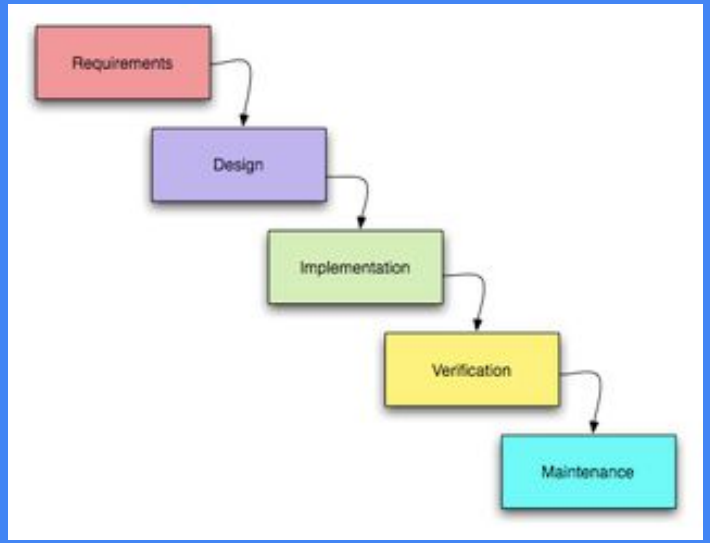
- Proyectos de gran tamaño y que necesitan constantes cambios.
- Proyectos donde sea importante el factor riesgo.
- Este sistema es muy utilizado para el desarrollo de sistemas orientados a objetos.

Ejercicios

En grupo comentad qué modelo crees que se adapta mejor a...

- la app Whatsapp
- a la web ForoCoches
- la web de Carrefour
- a Fortnite
- a la app Linterna
- al S.O. android
- a Microsoft Word
- a http://www.milliondollarhomepage.com/

Cada fase del Ciclo de Vida en Profundidad



¿De qué depende cada fase?

- Hay que elegir un modelo de ciclo de vida.

- Será necesario examinar las características del proyecto para elegir un modelo u otro.

- Hay una serie de etapas mínimas que se deben seguir para construir un proyecto de calidad

Fase 0: Requisitos

Fase 0: Requisitos

- Objetivo: Determinar todas las necesidades o condiciones para crear o modificar un programa.
- Es decir **qué** debe hacer el sistema (el **cómo** lo debe hacer se decide después)
- Los requisitos se obtienen a través de reuniones con los clientes
- Unos requisitos obtenidos correctamente llevarán a un buen análisis, diseño…
- Si los requisitos son inconsistentes, resultará en problemas en las demás fases
- Se podrán realizar reuniones en diferentes iteraciones para ir actualizando los requisitos a medida que se va desarrollando el software.
- Cuando el proyecto está avanzado, las reuniones no deben ser necesarias
- Todos los requisitos deben quedar reflejados en un documento para poder justificar la planificación de las siguientes fases, y defender el proyecto.

Fase 0: Requisitos

Pueden surgir problemas:

- Al captar requisitos: El cliente puede no tenerlos claros, pueden surgir nuevos requisitos, puede cambiar lo especificado, pueden existir malos entendidos por falta de conocimiento del equipo de desarrollo sobre el problema a resolver, el cliente puede no expresarse de forma clara debido a la falta de conocimientos informáticos, etc.
- Después: El cliente puede quejarse de un aspecto del sistema. La documentación de requisitos debe aclarar si el cliente ha solicitado o no un aspecto del sistema.

Fase 0: Requisitos

Los requisitos se dividen en:

- Funcionales: descripción de lo que un sistema debe hacer. Este tipo de requisito especifica algo que el sistema entregado debe ser capaz de realizar.
- No funcionales: rendimiento, de calidad, etc; especifica algo sobre el propio sistema, y cómo debe realizar sus funciones. Algunos ejemplos de aspectos solicitables son la disponibilidad, el testeo, el mantenimiento, la facilidad de uso,fiabilidad, etc.

Fase 0: Requisitos

Preguntas:

- ¿Cuál es el proceso básico de la empresa?
- ¿Qué datos utiliza o produce este proceso?
- ¿Cuáles son los límites impuestos por el tiempo y la carga de trabajo?
- ¿Qué controles de desempeño utiliza?
- ¿De qué otros sistemas depende?
- ¿En qué sistema operativo / navegador web se va a ejecutar el sistema?
- ¿En qué equipos se va a instalar el sistema?
- ¿Cuál es la finalidad de la actividad dentro de la empresa?
- ¿Quiénes van a ser los usuarios del sistema? ¿Qué nivel de conocimiento tienen?
- ¿Cuál es la finalidad de la actividad dentro de la empresa?
- ¿Qué pasos se siguen para realizarla?
- ¿Dónde se realizan estos pasos?
- ¿Cuánto tiempo tardan en efectuarlos?
- ¿Con cuánta frecuencia lo hacen?
- ¿Quiénes emplean la información resultante?
- ¿Cuántos empleados van a trabajar en el sistema a la vez?
- ¿Cómo debe estar presentada la información resultante?
- ¿Cuáles son los requerimientos de seguridad?
- ¿Existe alguna base de datos ya creada con la información?
- ¿Cómo se realiza hasta ahora este proceso en la empresa?

Fase 0: Requisitos

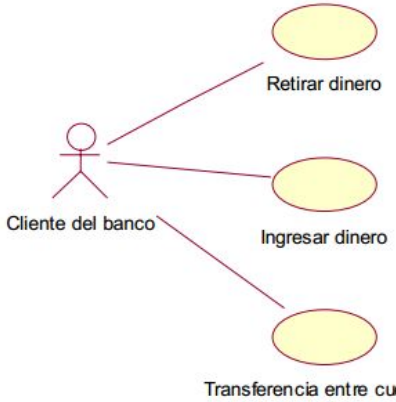
Requisitos funcionales	Requisitos no funcionales
El usuario puede agregar un nuevo contacto	La aplicación debe funcionar en sistemas operativos Linux y Windows
El usuario puede ver una lista con todos los contactos	El tiempo de respuesta a consultas, altas, bajas y modificaciones ha de ser inferior a 5 segundos
A partir de la lista de contactos el usuario puede acceder a un contacto	Utilizar un sistema gestor de base de datos para para almacenar los datos
El usuario puede eliminar un contacto o varios de la lista	Utilizar un lenguaje multiplataforma para el desarrollo de la aplicación
El usuario puede modificar los datos de un contacto seleccionado de la lista	La interfaz de usuario es a través de ventanas, debe ser intuitiva y fácil de manejar
El usuario puede seleccionar determinados contactos	El manejo de la aplicación se realizará con el teclado y el ratón
El usuario puede imprimir la lista de contactos	Espacio libre en disco, mínimo: 1GB. Mínima cantidad de memoria 2GB

Fase 0: Casos de Uso

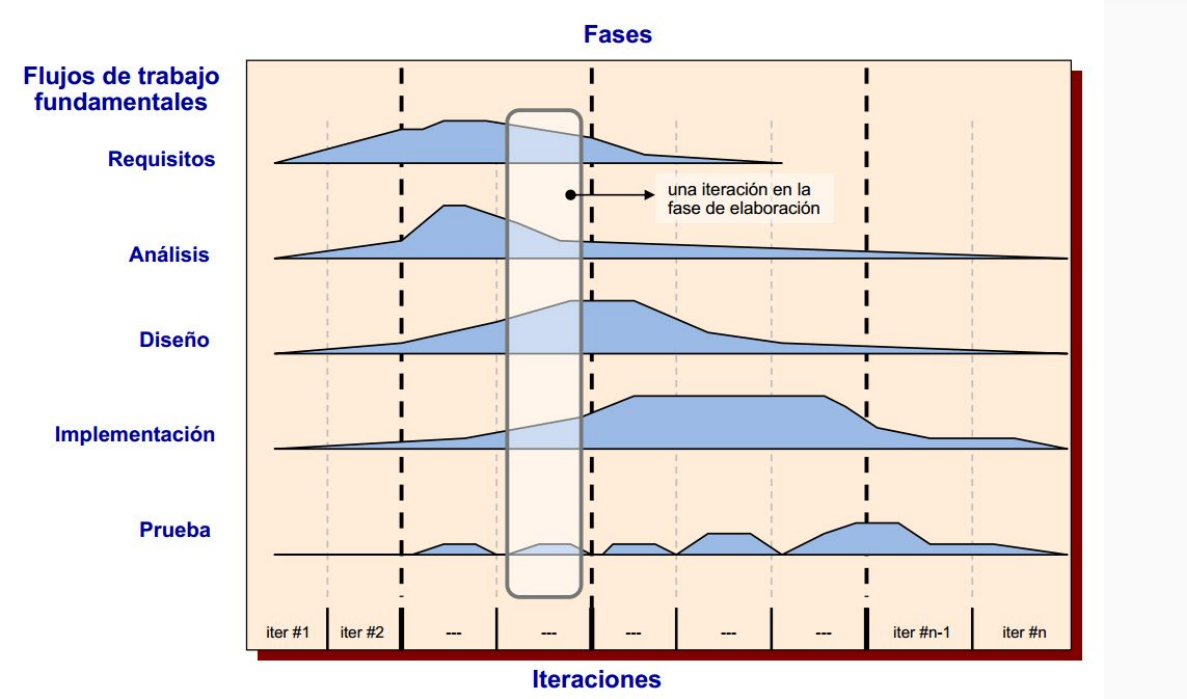
- para construir un sistema con éxito hay que conocer las necesidades y deseos de los futuros usuarios
  - usuario
    - personas que trabajan y necesitan el sistema
    - otros sistemas que interactúan con el que estamos desarrollando
  - interacción:
    - usuario inserta tarjeta en cajero automático
    - usuario responde sobre la pantalla a las preguntas que realiza el cajero
    - el usuario recibe una cantidad de dinero y su tarjeta
  - una interacción así es un ***caso de uso***
    - fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante

Fase 0: Casos de Uso

- utilidad casos de uso
  - herramienta para especificar los requisitos de un sistema: representan los requisitos funcionales y juntos constituyen el ***modelo de casos de uso***, que describe la funcionalidad total del sistema
  - guían el proceso de desarrollo (diseño, implementación y prueba)
    - basándose en el modelo de casos de uso, se crean modelos de diseño e implementación
    - se revisa cada modelo para que sean conformes al modelo de casos de uso
    - se prueba la implementación para garantizar que los componentes del modelo de implementación implementan correctamente los casos de uso
  - no sólo inician el proceso de desarrollo sino que éste sigue un hilo de trabajo que parte de los casos de uso



Requisitos en las fases siguientes



Fase 1: Análisis

Fase 1: Análisis

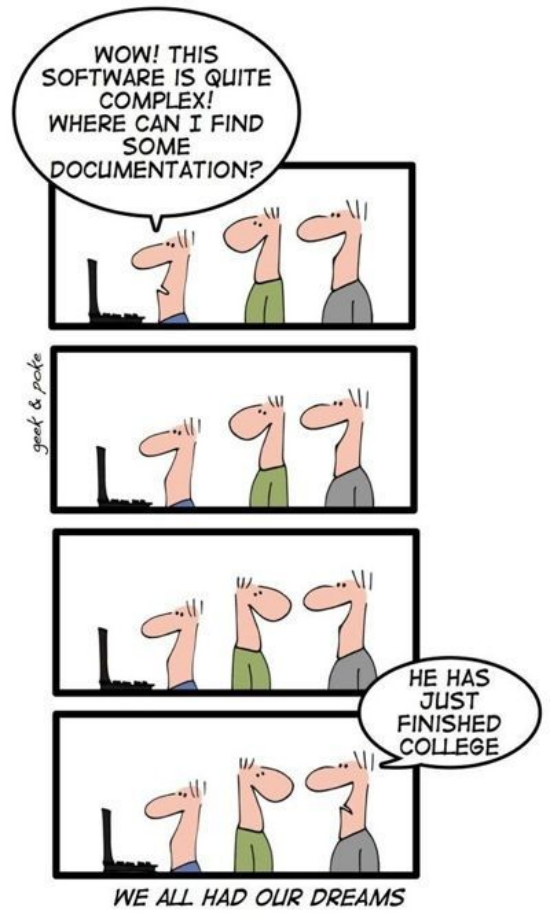
- Muy importante entender y comprender el problema que se necesita resolver, y una vez comprendido darle solución.
- En esta fase se analizan ~~y especifican~~ **los requisitos** o capacidades que el sistema debe tener porque el cliente así lo ha pedido.
- De los requisitos captados en la fase anterior se realiza un orden identificando las partes del sistema consideradas fundamentales y las partes accesorias.
- **Brainstorming:** los miembros del proyecto aportan ideas desde diferentes puntos de vista para la resolución de un problema, es decir para definir cómo desarrollar cada apartado.
- **Prototipos:** Se planea una versión inicial del sistema**.** Después se descarta, se modifica o se usa como base para añadir más cosas.
- **Diagramas de Casos de Uso:** Es la técnica definida en UML, se basa en la representación de escenarios que describen el comportamiento deseado del sistema, es decir lo que queremos que haga el sistema. Representan **requisitos funcionales** del mismo.

Fase 1: Análisis: Documentación

Todo lo realizado en esta fase debe quedar reflejado en el documento de **Especificación de Requisitos del Software** (ERS),

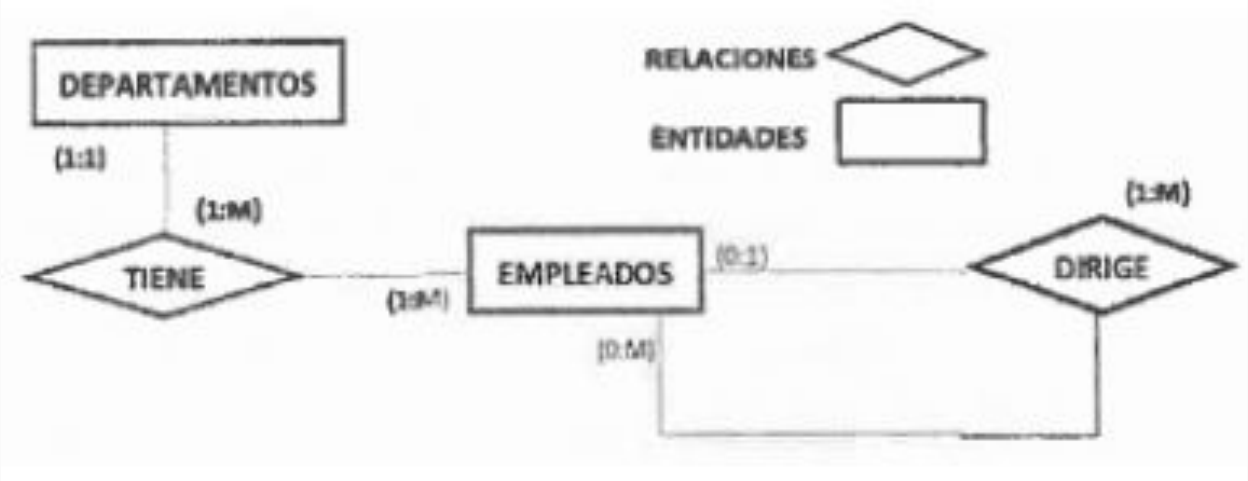
Este documento no debe tener ambigüedades, debe ser completo, consistente, fácil de verificar y modificar, fácil de utilizar en la fase de explotación y mantenimiento, y fácil de identificar el origen y las consecuencias de los requisitos.

Sirve como entrada para la siguiente fase en el desarrollo de la aplicación


WOW! THIS SOFTWARE IS QUITE COMPLEX! WHERE CAN I FIND SOME DOCUMENTATION?
geek & poke
HE HAS JUST FINISHED COLLEGE
WE ALL HAD OUR DREAMS

Fase 1: Análisis: Diagrama Entidad / Relación

Usado para representar los datos y la forma en la que se relacionan entre ellos



Fase 2: Diseño

Fase 2: Diseño

Una vez identificados los requisitos es necesario componer la forma en que se solucionará el problema.

En esta etapa se traducen los **requisitos funcionales y no funcionales** en una representación de software.

Fase 2: Diseño

El diseño estructurado (diseño clásico) produce un **modelo de diseño con 4 elementos**:

Fase 2: Diseño: Fases

Fase 1: Diseño de datos. Se encarga de transformar el modelo de dominio de la información creado durante el análisis, en las estructuras de datos que se utilizarán para implementar el software. El diseño de datos está basado en los datos y las relaciones definidos en el diagrama entidad relación y en la descripción detallada de los datos

Fase 2: Diseño arquitectónico. Se centra en la representación de la estructura de los componentes del software, sus propiedades e interacciones. Partiendo de los DFD se establece la estructura modular del software que se desarrolla.

Fase 2: Diseño: Fases

Fase 3: Diseño de la interfaz. Describe cómo se comunica el software consigo mismo, con los sistemas que operan con él, y con las personas que lo utilizan. El resultado de esta tarea es la creación de formatos de pantalla.

Fase 4: Diseño a nivel de componentes (diseño procedimental). Transforma los elementos estructurales de la arquitectura del software en una descripción procedimental de los componentes del software. El resultado de esta tarea es el diseño de cada componente software con el suficiente nivel de detalle para que pueda servir de guía en la generación de código fuente en un lenguaje de programación.
Diagramas de flujo, diagramas de cajas, tablas de decisión, *pseudocódigo*, etc.

Fase 2: Diseño: UML

El **lenguaje unificado de modelado** (**UML**, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el *Object Management Group* (OMG).

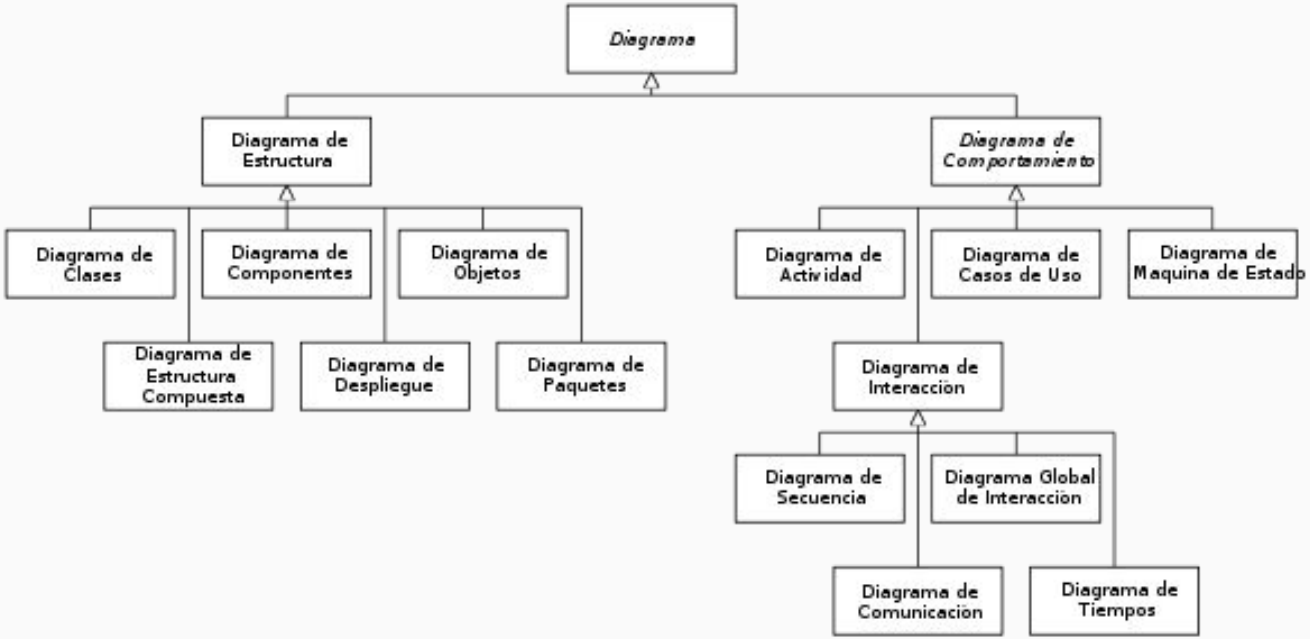
Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Fase 2: Diseño: UML

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional, *Rational Unified Process* o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

Fase 2: Diseño: UML



Fase 3: Codificación / Implementación / Picar Código

Fase 3: Codificación

Una vez realizado el diseño se realiza el proceso de codificación.

- En esta etapa, el programador recibe las **especificaciones del diseño** y las transforma en un conjunto de instrucciones escritas en un lenguaje de programación, almacenadas dentro de un programa.
- A este conjunto de instrucciones se le llama **código fuente**.
- El programador debe conocer la sintaxis del lenguaje de programación utilizado.
- En cualquier proyecto en el que trabaja un grupo de personas debe haber unas normas de codificación y estilo, claras y homogéneas

Fase 4: Pruebas

Fase 4: Pruebas

- Ya se dispone del software y se trata de encontrar errores
- Durante la prueba del software se realizarán tareas de verificación y validación del software
- Si se está construyendo el producto correctamente
- Si el software construido se ajusta a los requisitos del cliente
- El objetivo en esta etapa es planificar y diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y de esfuerzo

Fase 4: Pruebas: Recomendaciones

- Cada prueba debe definir los resultados de salida esperados.
- Un programador o una organización debe probar sus propios programas.
- Es necesario revisar los resultados de cada prueba en profundidad.
- Las pruebas deben incluir datos de entrada válidos y esperados, así como no válidos e inesperados.

Fase 4: Pruebas: Recomendaciones

- Centrar las pruebas en dos objetivos:
    1) comprobar si el software no hace lo que debe hacer
    2) comprobar si el software hace lo que no debe hacer.
- Evitar hacer pruebas que no estén documentadas ni diseñadas con cuidado.
- No planear pruebas asumiendo que no se encontrarán errores.
- La probabilidad de encontrar errores en una parte del software es proporcional al número de errores ya encontrados.
- Las pruebas son una tarea creativa.

Fase 4: Pruebas: El flujo de proceso para probar el software

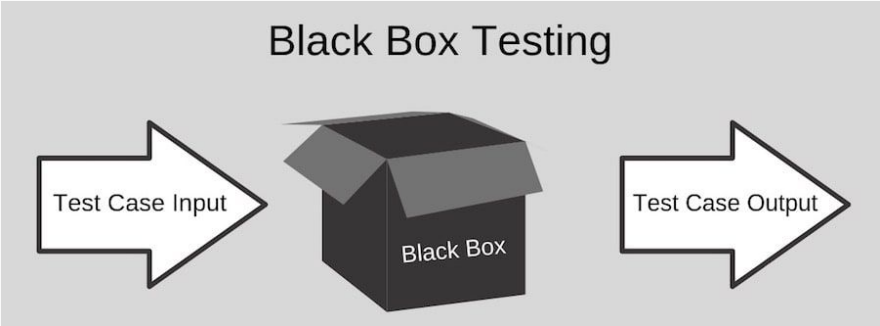
- **Generar un plan de pruebas**
- **Se diseñan las pruebas**
- **Generación de los casos de prueba**
- Especificar cómo se va a llevar a cabo el proceso, quién lo va a realizar, cuándo, etc.
- **Ejecución de las pruebas**
- **Evaluación.**
- **Depuración**
- **Análisis de errores**

Fase 4: Pruebas: Tipos de Pruebas

- **Pruebas de Caja Blanca:** se centran en validar la estructura interna del programa (necesitan ver el código)

Fase 4: Pruebas: Tipos de Pruebas

- **Prueba de Caja Negra**: Se centran en validar los requisitos funcionales sin fijarse en el funcionamiento interno del programa.
- No ven el código.
- Hacen como si fuesen un usuario



Fase X: Documentación

Fase X: Documentación

- Todas las etapas del desarrollo deben quedar perfectamente documentadas
- Los documentos relacionados con un proyecto de software:
  - Deben actuar como un medio de comunicación entre los miembros del equipo de desarrollo.
  - Deben ser un repositorio de información del sistema para ser utilizado por el personal de mantenimiento.
  - Deben proporcionar información para ayudar a planificar la gestión del presupuesto y programar el proceso del desarrollo del software.
- Algunos de los documentos deben indicar a los usuarios cómo utilizar y administrar el sistema.

Fase X: Documentación

Se puede decir que la documentación presentada se divide en dos clases:

- **La documentación del proceso.** Son documentos en los que se indican planes, estimaciones y horarios que se utilizan para predecir y controlar el proceso de software, que informan sobre cómo usar los recursos durante el proceso de desarrollo, sobre normas de cómo se ha de implementar el proceso.
- **La documentación del producto.** Define dos tipos de documentación:
  - la documentación del sistema que describe el producto desde un punto de vista técnico, orientado al desarrollo y mantenimiento del mismo;
  - la documentación del usuario que ofrece una descripción del producto orientada a los usuarios que utilizarán el sistema.

Fase X: Documentación del Sistema

Incluye todos los documentos que describen el sistema, desde la especificación de requisitos hasta las pruebas de aceptación.

La documentación debe incluir:

- **Fundamentos del sistema**
- **El análisis y especificación de requisitos**
- **Diseño**
- **Implementación**
- **Plan de pruebas del sistema**
- **Plan de pruebas de aceptación**
- **Los diccionarios de datos**

Fase 5: Despliegue (Explotación)

Fase 5: Despliegue

Esta etapa se lleva a cabo la instalación y puesta en marcha del producto software en el entorno de trabajo del cliente

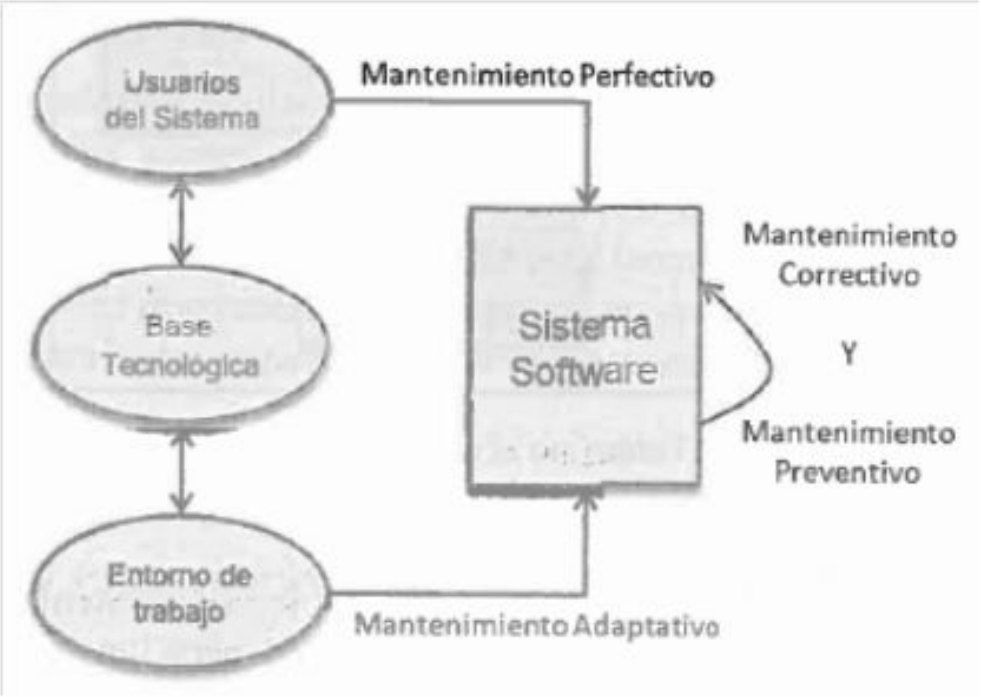
- **Se define la estrategia para la implementación del proceso**
- **Pruebas de operación**
- **Uso operacional del sistema**
- **Soporte al usuario**

Fase 6: Mantenimiento

Fase 6: Mantenimiento

La modificación de un producto de software después de la entrega para corregir los fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno modificado

Fase 6: Mantenimiento: Tipos



Lenguajes de Programación

Partes de un Lenguaje de Programación

- **Un alfabeto o vocabulario (léxico):** formado por el conjunto de símbolos permitidos.
- **Una sintaxis:** son las reglas que indican cómo realizar las construcciones con los símbolos del lenguaje.
- **Una semántica:** son las reglas que determinan el significado de cualquier construcción del lenguaje.

Clasificación

Según su nivel de abstracción	Lenguajes de bajo nivel Lenguajes de nivel medio Lenguajes de alto nivel
Según la forma de ejecución	Lenguajes compilados Lenguajes interpretados
Según el paradigma de programación	Lenguajes imperativos Lenguajes funcionales Lenguajes lógicos Lenguajes estructurados Lenguajes orientados a objetos

Según su Abstracción

Lenguajes de bajo nivel

- Se acercan al funcionamiento de un ordenador
- Las instrucciones están formadas por cadenas de ceros y unos
- A este le sigue el lenguaje ensamblador. Este lenguaje es difícil de aprender y es específico para cada procesador

Lenguajes de nivel medio

- tienen ciertas características que los acercan a los lenguajes de bajo nivel, pero a la vez también tienen características de los lenguajes de alto nivel.
- Un lenguaje de programación de este tipo es el lenguaje C. Se suelen utilizar para aplicaciones como la creación de sistemas operativos.

Lenguajes de alto nivel

- más fáciles de aprender
- Para poder ejecutarlos en el ordenador se necesita un programa intérprete o compilador
- son independientes de la máquina

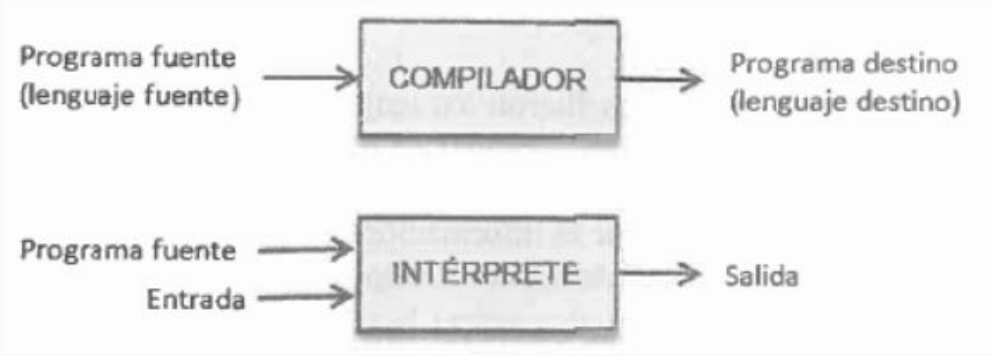
Según su ejecución

Lenguajes compilados

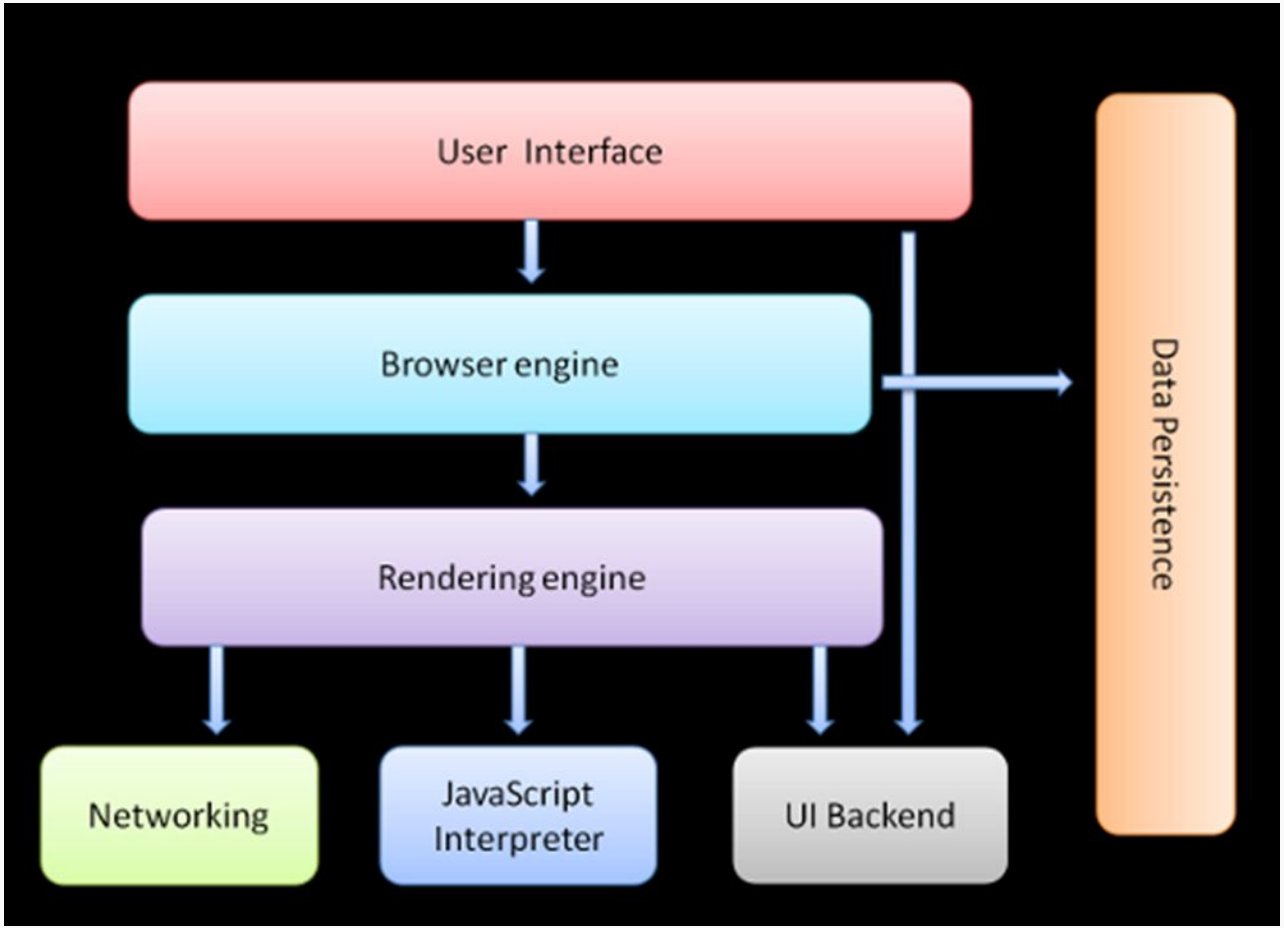
- se escribe en un lenguaje de alto nivel tiene que traducirse a un código que pueda utilizar la máquina
- Los programas traductores que pueden realizar esta operación se llaman compiladores o intérpretes

Lenguajes interpretados

- en vez de producir un programa destino como resultado del proceso de traducción, el intérprete nos da la apariencia de ejecutar directamente las operaciones especificadas en el programa fuente con las entradas proporcionadas por el usuario
- Algunos ejemplos de lenguajes interpretados son: PHP, JavaScript, Python, Perl, Logo, Ruby, ASP, Basic, etc
- Los procesadores del lenguaje Java combinan la compilación y la interpretación


Programa fuente
(lenguaje fuente)
COMPILADOR
Programa destino
(lenguaje destino)
Programa fuente
Entrada
INTÉRPRETE
Salida


Programa fuente
TRADUCTOR
Programa Intermedio
Entrada
MÁQUINA VIRTUAL
Salida


User Interface
Browser engine
Rendering engine
Networking
JavaScript Interpreter
UI Backend
Data Persistence