TIPOS DE SOFTWARE

De sistema → Interactúa directamente con el hardware (de bajo nivel, muy cerca del hardware)
De aplicación → programas que están más lejos que el microprocesador y se valen del SO
De desarrollo → programas para hacer programas (Visual Studio Code…)

Firmware → es el software de sistema más cercano al hardware (como por ejemplo cualquier tipo de código de programa que se ejecute en un microcontrolador) (Ratones teclados…)

Drivers → programa entre el SO y el firmware del hardware, mandas instrucciones básicas al hardware del firmware

RELACIÓN HARDWARE SOFTWARE

Disco duro: Almacena de forma permanente los archivos ejecutables y los de datos
RAM: Almacena de forma temporal el código binario de ejecutables y archivos de datos.
CPU: Lee y ejecuta instrucciones almacenadas en RAM
E/S: Recoge nuevos datos desde la entrada.

Códigos fuente, objeto y ejecutable

Código fuente: Archivo de texto legible escrito en lenguaje de programación
Código objeto: Archivo binario no ejecutable
Código ejecutable: Archivo binario ejecutable

Sólo válido para lenguajes compilados: C, C + +, Java.

En lenguajes interpretados no existe código objeto, ni binario, solo código fuente.

PHP, Javascript…

El código objeto en Java se denomina ByteCode.

CICLO DE VIDA DE SOFTWARE

INGENIERÍA DE SOFTWARE

Disciplina que estudia los principios y metodologías para el desarrollo y mantenimiento de sistemas softwares

DESARROLLO DE SOFTWARE

- ANÁLISIS
- DISEÑO
- CODIFICACIÓN
- PRUEBAS
- MANTENIMIENTO

análisis: analizar y definir requisitos
diseño: ver ideas y cómo implementarlas
codificación: programar el código a realizar
pruebas: realizar test para verificar que todo funciona correctamente
documentación/mantenimiento: actualizaciones parches solución de los fallos que surjan

ANÁLISIS

Se determina y define claramente las necesidades del cliente y se especifica los requisitos que debe cumplir el software a desarrollar

Características a cumplir: (para que el cliente lo entienda)

- Se completa y sin omisiones
- Ser concisa y sin trivialidades, información realmente necesaria.
- Evitar ambigüedades y utilizar un lenguaje formal
- Evitar detalles de diseño o implementación, solo requisitos que tiene que tener
- Ser entendible por el cliente
- Separar requisitos funcionales y no funcionales (por ejemplo que sea bonito para la vista,no funcional)
- Dividir y jerarquizar el modelo (hacer divisiones o categorías distintas partes que tiene la aplicación)
- Fijar criterios de validación, es decir una vez tener requisitos tenemos que dar los de validación que cosas de consideran válidas y que cosas no validas

DISEÑO

Gente que imagina la solución, se le presenta al cliente varias soluciones. Hay que descomponer el sistema en elementos que puedan ser desarrollados por separado.

Una vez realizadas las partes hay que detallar muy bien la relación que hay entre ellas.

Las actividades habituales son las siguientes:

- Diseño arquitectónico, definir como voy a solucionar el problema a nivel bloques
- Diseño detallado, definir que va a hacer cada bloque
- Diseño de datos, definir que tipo de datos usamos y cómo lo repartimos
- Diseño de interfaz de usuario, es decir cómo interactúa el usuario con nuestro programa, hay algunos que no tienen interfaz de usuario

MOCK UP → diseño digital de una web y / o aplicación, representación del prototipo del proyecto que se va a realizar

CODIFICACIÓN

Escribir el código fuente del programa, se pueden usar distintos lenguajes de programación.
Lenguajes de programación: C,C + +,JAVA, Javascript…
Lenguajes de otro tipo: HTML, XML, JSON…

PRUEBAS

Objetivo, es conseguir que el programa funcione incorrectamente para detectar los fallos del programa, hay que someter al programa al máximo número de situaciones diferentes.

MANTENIMIENTO

Durante la explotación del sistema software es necesario realizar cambios ocasionales
Correctivo: corregir los defectos una vez a ocurrido el fallo
Perfectivo: funciona correctamente pero hace que funcione mejor
Evolutivo:donde se añaden funcionalidades nuevas
Adaptativo: se adapta a nuevos entornos (por ejemplo un juego de android pasarlo también a IOS)

RESULTADO TRAS CADA FASE

***Ingeniería de sistemas**: Especificación del sistema*

***ANÁLISIS**: Especificación de requisitos del software*

***DISEÑO** arquitectónico: Documento de arquitectura del software*

***DISEÑO detallado**: Especificación de módulos y funciones*

***CODIFICACIÓN**: Código fuente*

***PRUEBAS de unidades**: Módulos utilizables*

***PRUEBAS de integración**: Sistema utilizable*

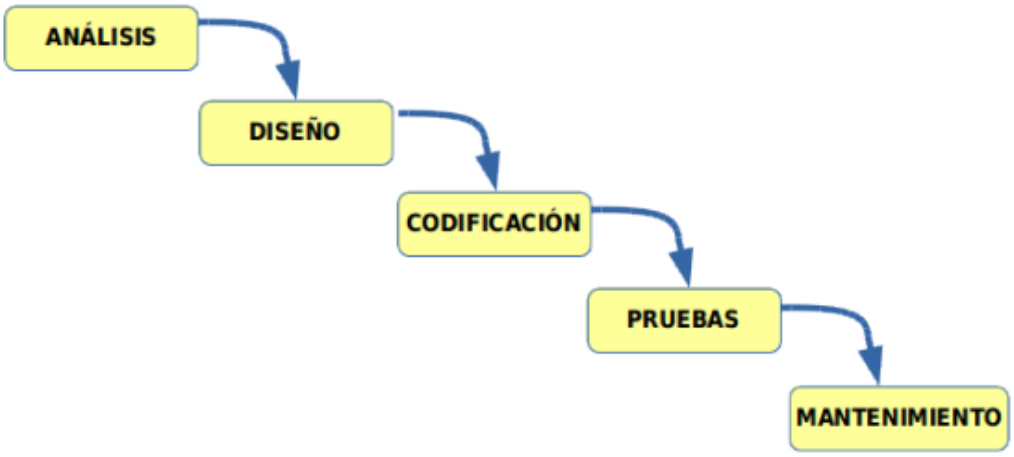
***PRUEBAS del sistema**: Sistema aceptado*

***Documentación**: Documentación técnica y de usuario*

***MANTENIMIENTO**: Informes de errores y control de cambios*

MODELOS DE DESARROLLO DEL SOFTWARE

MODELO EN CASCADA.



Modelo de mayor antigüedad.

• Identifica las fases principales del desarrollo software

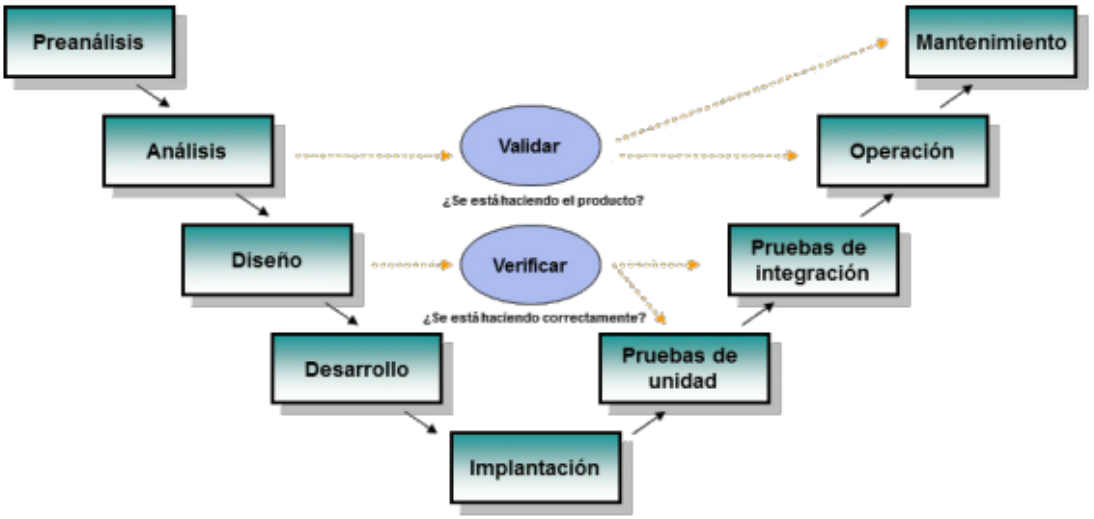
• Las fases han de realizarse en el orden que vemos

• El resultado de una fase es la entrada de la siguiente fase

• Es un modelo bastante rígido que se adapta mal al cambio continuo de especificaciones

• Existen diferentes variantes con mayor o menor cantidad de actividades

MODELO EN V



Modelo muy parecido al modelo en cascada.

• Visión jerarquizada con distintos niveles

• Los niveles superiores indican mayor abstracción

• Los niveles inferiores indican mayor nivel de detalle

• El resultado de una fase es la entrada de la siguiente fase

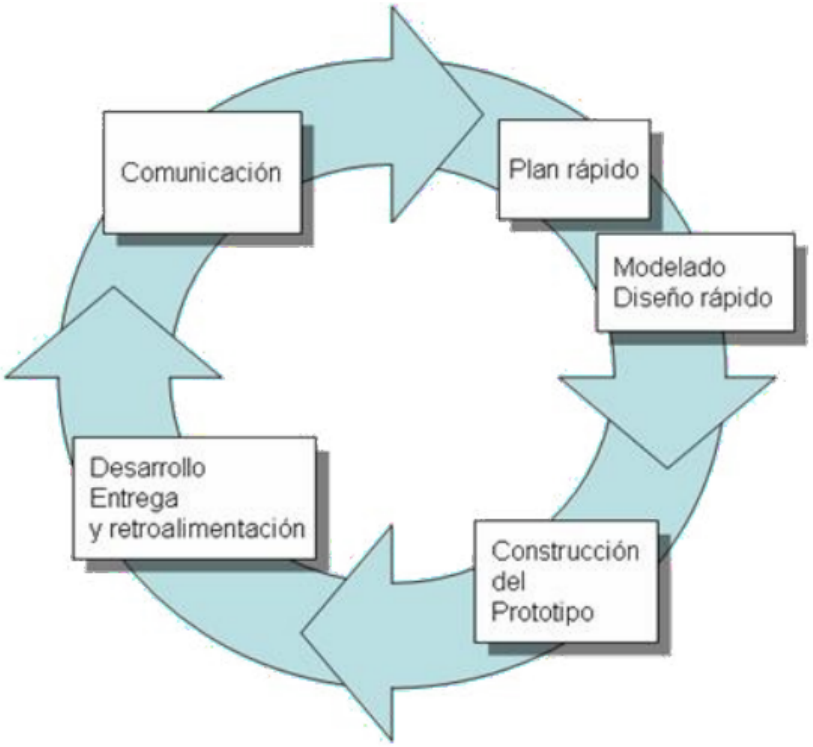
• Existen diferentes variantes con mayor o menor cantidad de actividades

PROTOTIPOS

A menudo los requisitos no están especificados claramente:

Por no existir experiencia previa

Por falta de concreción del usuario/cliente



Modelo de construcción de prototipos

- Se crea un prototipo durante la fase de análisis y es probado por el usuario/cliente para refinar los requisitos del software a desarrollar
- Se repite el paso anterior las veces necesarias

Tipos de prototipos:

Prototipos rápidos

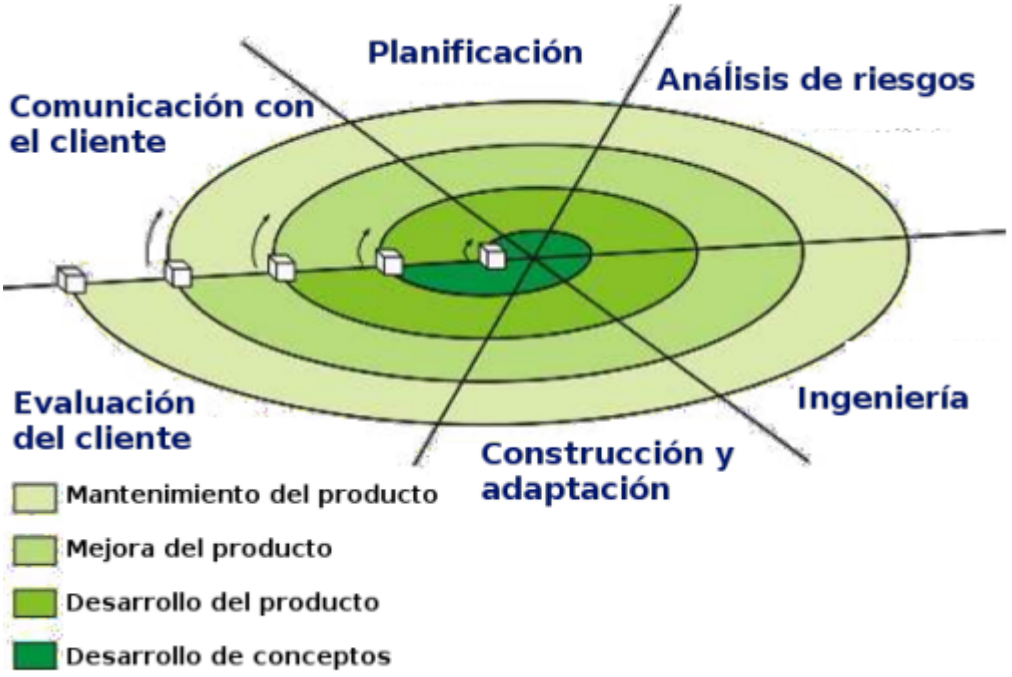
- El prototipo puede estar desarrollado usando otro lenguaje y/o herramientas
- Finalmente el prototipo se desecha

Prototipos evolutivos

- El prototipo está diseñado en el mismo lenguaje y herramientas del proyecto
- El prototipo se usa como base para desarrollar el proyecto

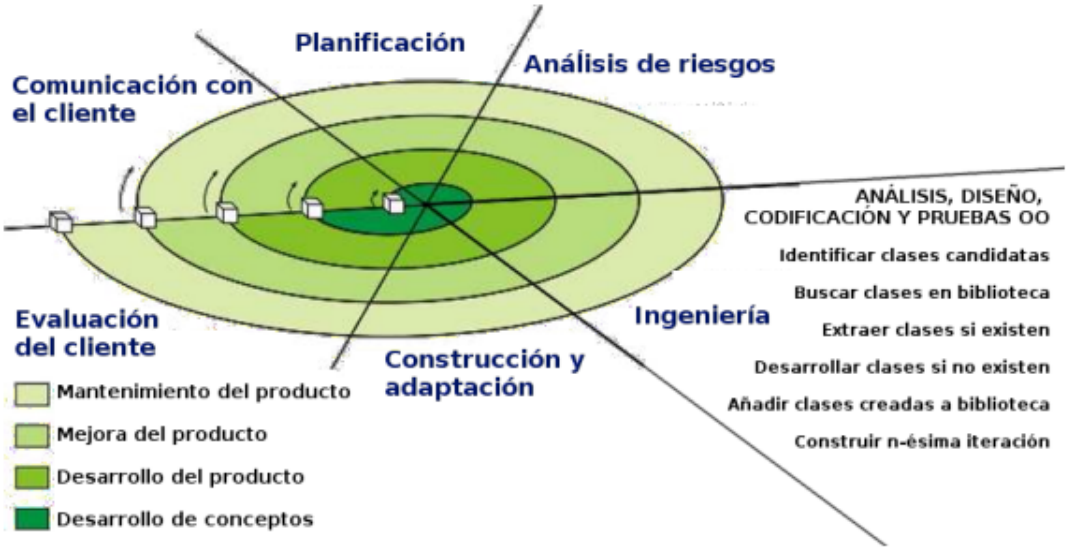
MODELO EN ESPIRAL

- Desarrollado por Boehm en 1988.
- En la actividad de ingeniería corresponde a las fases de los modelos clásicos: análisis, diseño, codificación, ..



MODELO EN ESPIRAL (II)

- Aplicado a la programación orientada a objetos
- En la actividad de ingeniería se da gran importancia a la realización de código



Metodologías ágiles

- Son métodos de ingeniería del software basados en el desarrollo iterativo e incremental
- Los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto
- El trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo

Las metodologías más conocidas son:

- Kanban
- Scrum
- XP (eXtreme Programming)
-

MANIFIESTO POR EL DESARROLLO ÁGIL

Individuos e interacciones sobre procesos y herramientas

- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

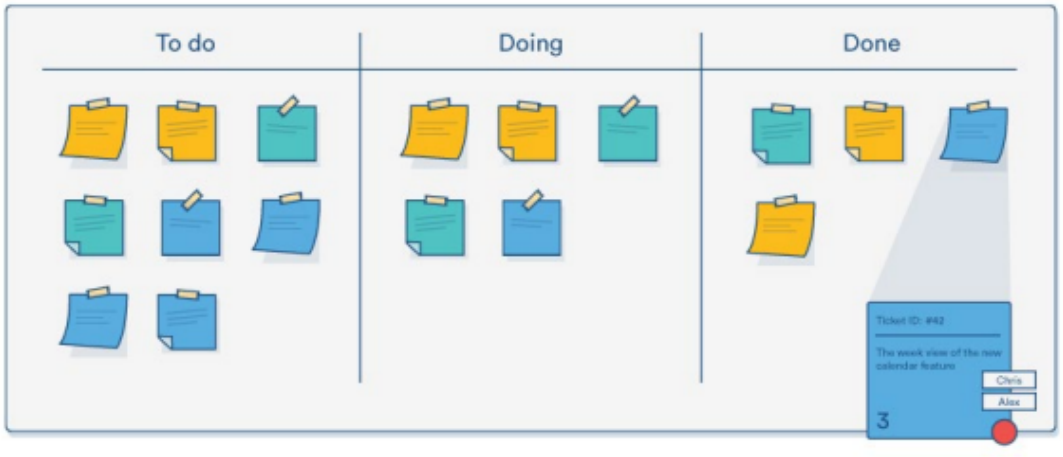
KANBAN

También se denomina "sistema de tarjetas".

- Controla por demanda la fabricación de los productos necesarios en la cantidad y tiempo necesarios
- Enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos

LEAN MANUFACTURING →

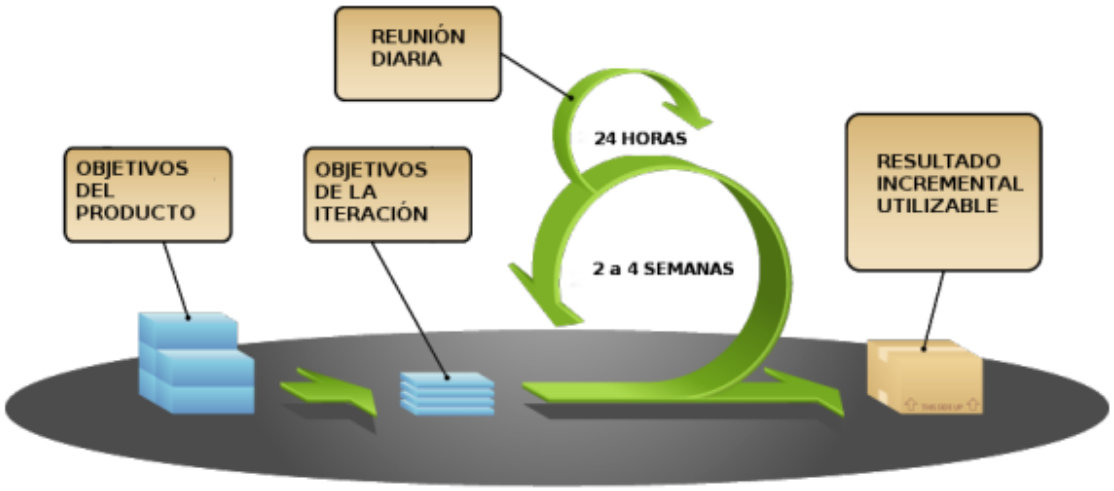
Pizarra kanban



SCRUM

Modelo de desarrollo incremental

- Iteraciones (sprint) regulares cada 2 a 4 semanas
- Al principio de cada iteración se establecen sus objetivos priorizados (sprint backlog)
- Al finalizar cada iteración se obtiene una entrega parcial utilizable por el cliente
- Existen reuniones diarias para tratar la marcha del sprint



ROLES PRINCIPALES

Product Owner: Es "la voz del cliente". Define criterios de aceptación y asegura de que se cumplan

Scrum Master: Asegura que se sigue la metodología Scrum. Motiva y facilita el trabajo del equipo

Team de desarrollo: Equipo de desarrollo auto-organizado y multifuncional. Entre 6 y 10 miembros

ARTEFACTOS

Product Backlog: Lista ordenada con los requisitos del producto

Sprint Backlog: Lista de requisitos sacados del backlog para su desarrollo durante el sprint

Incremento: Estado del producto después de cada sprint

EVENTOS

- **Sprint**: Evento principal, que contiene al resto de eventos. Duración máxima: 1 mes
- **Sprint Planning**: Reunión inicial donde se planifica el trabajo del Sprint. Duración máxima: 8 horas
- **Daily Scrum**: Reunión diaria de puesta en común sobre la marcha del sprint. Duración máxima: 15 minutos
- **Scrum Review**: Reunión final para evaluar el incremento obtenido. Duración máxima: 4 horas.
- **Scrum Retrospective**: Reunión final para evaluar la correcta aplicación de la metodología Scrum. Duración máxima: 3 horas

PROGRAMACIÓN EXTREMA

- Simplicidad
- Comunicación
- Retroalimentación
- Valentía o coraje
- Respeto o humildad

CARACTERÍSTICAS

- Diseño sencillo
- Pequeñas mejoras continuas
- Pruebas y refactorización
- Integración continua
- Programación por parejas
- El cliente se integra en el equipo de desarrollo
- Propiedad del código compartida
- Estándares de codificación
- 40 horas semanales

LENGUAJE DE PROGRAMACIÓN

Obtención de código ejecutable

Para obtener código binario ejecutable tenemos 2 opciones:

- Compilar
- Interpretar

Proceso de compilación/interpretación

La compilación/interpretación del código fuente se lleva a cabo en dos fases:

1. Análisis léxico
2. Análisis sintactico

- Si no existen errores, se genera el código objeto correspondiente
- Un código fuente correctamente escrito no significa que funcione según lo deseado
- No se realiza un análisis semántico

Lenguajes compilados

Ejemplos: C, C++

- **Principal ventaja**: Ejecución muy eficiente
- **Principal desventaja**: Es necesario compilar cada vez que el código fuente es modificado

Lenguajes interpretados

Ejemplos: PHP, Javascript

- **Principal ventaja**: El código fuente se interpreta directamente
- **Principal desventaja:** Ejecución menos eficiente

JAVA

Lenguajes compilados e interpretados.

- El código fuente Java se compila y se obtiene un código binario intermedio denominado bytecode
- Puede considerarse código objeto pero destinado a la máquina virtual de Java en lugar de código objeto nativo
- Después este bytecode se interpreta para ejecutarlo

Ventajas:

- Estructurado y Orientado a Objetos
- Relativamente fácil de aprender
- Buena documentación y base de usuarios

Desventajas:

- Menos eficiente que los lenguajes compilados +

Tipos

• Según la forma en la que operan:

- Declarativos: indicamos el resultado a obtener sin especificar los pasos.
- Imperativos: indicamos los pasos a seguir para obtener un resultado.

Tipos de lenguajes declarativos:

- Lógicos: Utilizan reglas. Ej: Prolog
- Funcionales: Utilizan funciones. Ej: Lisp, Haskell
- Algebraicos: Utilizan sentencias. Ej: SQL

• Normalmente son lenguajes interpretados.

• Tipos de lenguajes imperativos:

- Estructurados: C
- Orientados a objetos: Java
- Multiparadigma: C++, Javascript

• Los lenguajes orientados a objetos son también lenguajes estructurados.

• Muchos de estos lenguajes son compilados.

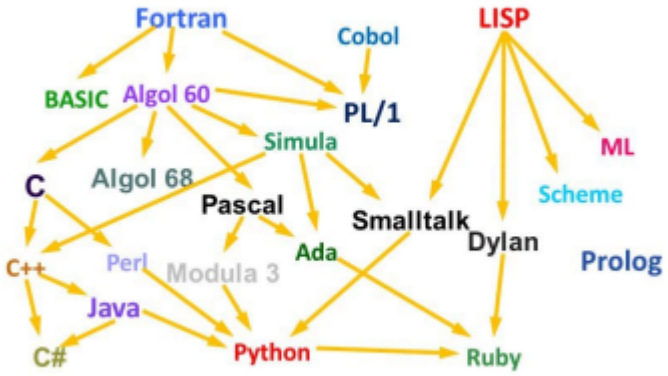
• Tipos de lenguajes según nivel de abstracción:

- Bajo nivel: ensamblador
- Medio nivel: C
- Alto nivel: C++, Java

Evolución

- Código binario
- Ensamblador
- Lenguajes estructurados
- Lenguajes orientados a objetos

Historia



Criterios para la selección de un lenguaje

- Campo de aplicación
- Experiencia previa
- Herramientas de desarrollo
- Documentación disponible
- Base de usuarios
- Reusabilidad
- Transportabilidad
- Imposición del cliente