**Hardware**: Es la parte física del ordenador (todos los elementos que lo componen).

**Software**: Es la parte lógica del ordenador, el código informático (programas).

**El internet de las cosas (internet of things IoT):** Cualquier dispositivo hoy en día es capaz de conectarse a internet, electrodomésticos, móviles, relojes etc.

**Arquitectura de Von Neumann**: Consiste en que todos los dispositivos cuentan con periféricos de E/S, la CPU, memoria central (registros, caché, RAM y ROM), un dispositivo de conexión en red (en la mayoría de los casos) y memorias secundarias (discos duros, pen drives, DVD etc).

**Tipos de software:**

**1- Sistema Operativo o Software básico (Operating system):** Maneja el hardware a bajo nivel y debemos tenerlo instalado para que el resto de programas puedan ejecutarse.

**2- Software para el desarrollo de programación (CASE Tools, Computer Aided Software Engineering Tools):** Herramientas que permiten crear programas. Escritura y depuración de código.(Visual Studio Code, Eclipse, Notion).

**3- Aplicaciones informáticas (Computer or Mobile applications):** Programas que tienen una utilidad funcional más o menos específica para el usuario final.

**3.1- De escritorio:** Programas normales que se instalan y configuran en el equipo y que no necesitan internet para funcionar.

**3.2- Para Web:** Se necesita de un navegador para poder usarlas ya que no se ejecuta ni se instala en el equipo, si no que esto ocurre en un servidor de internet.

**3.3- Multiplataforma:** Son pequeñas aplicaciones optimizadas para dispositivos más limitados, como smartphones, tablets, etc.

**3.3.1- Nativas:** Utilizan el lenguaje nativo del dispositivo y pueden ser publicadas en las tiendas de apps des estos dispositivos.

**3.3.2- WebApp:** Son páginas web adaptadas para que se vean correctamente en estos dispositivos y tratan de emular el funcionamiento de una App Nativas(son independientes del lenguaje nativo).

**3.3.3- Híbridas:** Combinación de las anteriores. Esta encapsula una Web dentro de un WebView nativo, por lo que realmente es una WebApp dentro de un navegador.

**Programa informático (Computer program):** Conjunto de instrucciones ordenadas (algoritmos), codificado en un lenguaje entendible y ejecutable por el ordenador y que ofrece alguna función.

**Algoritmo (Algorithm):** Es un conjunto de órdenes secuenciales que describen la resolución de un problema.

\*A los programas se les añade una interfaz (UX User eXperrience), además de que los programas cuentan con una lógica interna (back-end) y un diseño de interfaz (front-end).

**\*Aplicación(Application):** Es software complejo, formado por uno o más programas que junto a la documentación del mismo y de los archivos necesarios para su funcionamiento crean una herramienta de trabajo que soluciona un problema de manejo de información. (la diferencia con un programa es que este podría ser un único código más sencillo y aislado).

**\*Lenguajes de programación(Programing languages):** Conjunto de símbolos y caracteres combinados entre sí de acuerdo a una sintaxis definida, que se traducen a señales eléctricas 0s y 1s. Permite la transmisión de instrucciones entre la persona y el hardware, aunque es muy exigente y preciso (sin ambigüedad).

**Tipos de lenguajes:**

**1- Lenguaje de bajo nivel (Low-level language):** Depende de la arquitectura de cada máquina.

**1.1- Lenguaje máquina (Machine language):** Se realiza mediante 0s y 1s (no se utiliza).

**1.2- Lenguaje Ensamblador(Assembly language):** Las cadenas binarias se sustituyen por nombres simbólicos que luego deben ser traducidos por 0s y 1s mediante un software ensamblador.

**2- Lenguaje de alto nivel (High-level language):** Es el más cercano al lenguaje humano, pero con restricciones sintácticas (es imprescindible usar un software traductor para traducir el código fuente (Source code) al código máquina).

**Tipos de traductores:**

1. **Intérpretes(Interpreters):** Traduce el código fuente instrucción por instrucción y ejecuta cada instrucción una vez traducida. JavaScript, php, Phython etc

**\*Ventajas:** Flexibilidad en la fase de depuración, en el manejo de estructuras de datos y sintaxis (menos exigentes sintácticamente), como el código fuente siempre está disponible y no dependen de una arquitectura hardware particular este es portable.

**\*Inconvenientes:** Su ejecución es lenta al ser secuencial y como siempre está visible es difícil proteger este con copyright (a no ser que resida y se ejecuta en un servidor)

1. **Compiladores (Compilers):** Traduce el código fuente a un lenguaje máquina llamado programa objeto (object program) el cual no es ejecutable ya que solo es la traducción por lo que se necesita otro software (montador o enlazador / Linker). C, C++ y VisualBasic.

**Ventajas:** Su código no es visible y la ejecución del programa final es rápida.

**Inconvenientes:** No es tan flexible en la depuración ni sintácticamente.

**Fases de compilación:**

**1º** Se escribe el código fuente en un editor y se almacena.

**\*2º** Se traduce/compila el código completo.

**3º** Se verifica que no hay errores y los muestra si los hay.

**4º** Si está libre de errores se obtiene el programa objeto.

**5º** Montaje mediante un enlazador y se obtiene el ejecutable.

**6º** Se ejecuta el programa.

**\***Ya sea interpretado o compilado, todos se trabajan en un **IDE** (Integrated Development Environment o Entorno de Desarrollo Integrado) como Visual Studio Code o Sublime text.

Patchs -> Parche y Deployment -> Despliegue

Ingenieros de QA (Quality Assurance Engineers) -> testers

**El Caso Java:**

Este no es interpretado ni compilado, sino que su traducción se realiza en 2 fases:

**1º** Se compila el código fuente a un código intermedio (Bytecode), el cual solo se ejecuta en JVM o Java Virtual Machine.

\***2º** El Bytecode se interpreta y ejecuta para el código máquina de la máquina particular que usemos, la cual previamente tenía instalado su versión correspondiente de JVM.

\*Esta forma de traducción permite la portabilidad casi total de un mismo Bytecode (“Write one, run anywhere”), además de que el código no se encuentre a la vista.

**Generaciones de lenguajes y máquinas:**

**1º Generación (1GL) años 40 –50:** Su lenguaje era en 0s y 1s, luego aparecen los lenguajes ensambladores (assembly languages) que describían operaciones y datos mediante símbolos y por lo tanto aparecen los primeros traductores.

**2º Generación (2GL) Finales de los 50:** Aparecen lenguajes de alto nivel no estructurados y la traducción se vuelve más compleja. (Fortran, Cobol y Basic).

**3º Generación (3GL) Años 70:** Aparecen lenguajes de alto nivel estructurados (Structured High-level language) tras la 1º crisis del software. Pascal, C y lenguajes de lógica, inteligencia artificial y orientado a objetos (LOO Object Oriented Language). También aparece el modelo relacional de BD.

**4º Generación (4GL) Años 80:** Avances en la sintaxis, que cada vez era más natural. Aparece SQL y herramientas CASE, C++, Java y lenguajes de programación visual (Visual Programming language), Visual Basic y Visual C. En esta época, también se avanza en la informática personal.

**5º Generación:** Lenguajes con algoritmos que intentan imitar el racionamiento humano. Inteligencia artificial (AI Artificial Inteligence) y redes neuronales (Neural Networks).

**Los lenguajes de programación están constituidos por:**

**1 Alfabeto (Alphabet):** Los símbolos permitidos.

**2 Sintaxis (Sintax):** Normas de construcción con símbolos permitidas

**3 Semantica (Semantic):** Significado de las construcciones para hacer acciones válidas.

**Clasificación de lenguajes de alto nivel:**

1. **Modulares y Estructurados (Modular & Structured):** Solo permite el uso de 3 tipos de sentencias o estructuras de control: secuenciales, condicionales (if) y bucles (sequiencies, selections & loops). Esta evoluciona a la programación modular que divide el programa en módulos (Divide & conquer).
2. **Orientado a Objetos(Object Oriented OOP):** Java, C++ etc. No es una secuencia de pasos, sino de la descripción de un conjunto de objetos que representan una modelización del mundo real, cada uno con unas características y comportamientos que colaboran e interactúan entre sí para realizar acciones.
3. **Programación Visual (Visual programming):** Está basado en las anteriores, pero permite programar interactuando con un entorno gráfico, arrastrando y pegando componentes (en segundo plano se genera el código correspondiente automáticamente).

**Ingeniería del software (Software Engineering), solución a la 1º crisis del software:** Se establecen estándares y requisitos metodológicos y estructurados, para obtener software de calidad.

**Conocimientos de un ingeniero de software según SWEBOK (Software Engineering Body of Knowledge):**

1. **Requerimientos (Requirements):** Necesidades y restricciones que debe tener el producto para solucionar un problema (comunicación clara con el cliente).
2. **Diseño (Design):** Modelos/planos para la construcción del software.
   1. **Arquitectónico:** Identifica los componentes en el nivel más alto de abstracción.
   2. **Detallado:** Descripción detallado de cada componente del diseño arquitectónico.
3. **Construcción(Construction):** La escritura del código.
4. **Pruebas de software (software testing):** Poner a prueba el software para detectar fallos de funcionalidad, eficiencia, etc.
5. **Calidad del software (Software quiality):** Ponen a prueba el software terminado con pruebas de seguridad, carga, estrés, etc.
6. **Mantenimiento(Software maintenance):** Modificación de software liberado (released), para mejorar, corregir fallos, etc.
7. **Gestión de la configuración(Configuration Management):** Control de los cambios realizados en el producto obtenido durante cualquiera de las etapas del desarrollo (manteniendo una versión estable). Incluye, código fuente, código ejecutable, documentación..
8. **Gestión de la ingeniería de proyectos (Project Management):** Actividades administrativas que aseguran que el desarrollo y mantenimiento se realiza de forma adecuada.
9. **Manejo de herramientas y métodos de ingeniería de software (tools and methods):** Las herramientas automatizan tareas y los métodos sistematizan actividades para aumentar las probabilidades de éxito.
10. **Funcionamiento de computación, matemáticos y de ingeniería general (Mathematic, Engineering and computing foundations)**

**Ciclo de la vida del software (Software lifecycle):** Son cada uno de los pasos (procesos o etapas) que transcurren desde que se tiene la idea de un producto hasta que este deja de estar disponible. Las etapas son:

1 **Analisis(Analysis):** Los requerimientos de los usuarios.

**2 Diseño(Design):** De la estructura de datos, la arquitectura del software, la interfaz de usuario y los procedimientos. También se elige el lenguaje.

**3 Codificación(Codification or construction):** Escritura del código.

**4 Pruebas(Testing):** Comprobación del correcto funcionamiento del software.

**5 Mantenimiento(Maintenance) o soporte (software support):** Despliegue continuo de nuevas modificaciones y funcionalidades.

\*En cada una de las etapas se debe realizar la generación de documentación.

**Modelos de ciclos de vida del software**

**1-Modelo en Cascada o lineal (Waterfall lifecycle):** Consiste en recorrer de forma secuencial los pasos anteriores, cuando se termina uno se pasa al siguiente. No se usa.

**2-Modelo en cascada con realimentación(Waterfall lifecycle with feedback):** Igual a la anterior pero se puede volver atrás para modificar o corregir aspectos.

**3-Modelos Evolutivos:**

**3.1-Modelo Iterativo e Incremental (Iterative and incremental lifecycle):** Basado en el de cascada con realimentación donde las fases se van repitiendo y refinando. Es incremental porque el producto se divide en partes funcionales abordables que se van agregando en cada vuelta y es interativo porque a cada iteración se revisa, evalúa y mejora el producto completo.

**3.2-Modelo en Espiral(Spiral lifecycle model):** Se va construyendo en forma de versiones una detrás de otra mejorando en cada una, siendo la primera la de riesgos más asumibles o mínimo producto viable (MVP).

**Ventajas:** El desarrollador comprende mejor el producto y reacciona mejor a los riesgos.

**Inconvenientes:** No se sabe cuándo acabará el proceso, ni lo que acabará costando.

**4-Modelo Agile:** Es una filosofía de trabajo donde las personas experimentadas proponen cambios, el cliente es un miembro más del equipo y se valora más la capacidad de respuesta al cambio que planes de seguimiento pre-establecidos.

**Ventajas:** Flexibilidad extrema, tiempo record en tener una versión funcional del producto y realimentación continua del cliente.

**Inconvenientes:** No hay certeza en la planificación del proyecto, ni en los plazos de entrega, ni en los presupuestos y el cliente debe gastar tiempo en reuniones y habrá ocasiones donde habrá que formarlo.

**\*\*Metodología de Desarrollo de Software: Esta** Basado en los ciclos de vida. Utilizando una serie de herramientas determinadas y best practices establece las técnicas para organizar el desarrollo, organiza de forma clara el trabajo a realizar e indica detalles del proceso y documentos necesarios.

**\*UML(Unified Modeling Language):** Es un lenguaje gráfico. Una forma estandarizada de representar las partes del desarrollo de software, mediante diagramas gráficos.

**Licencia de software(Software licence):** Es un contrato que se establece entre el desarrollador de software sometido a propiedad intelectual y derechos de autor con el usuario, donde se definen los derechos y deberes de ambas partes.

**Software libre(free software):** Es aquel donde el autor cede los siguiente derechos (lo creo Richard Stallman):

1 Libertad de usar el programa con cualquier fin.

2 Libertad de estudiar cómo funciona el programa y de adaptar su código o mejorarlo (acceder al código fuente).

3 Libertad de distribuir copias a otros usuarios (incluso si se ha modificado su código).

4 Libertad de mejorar el programa.

La licencia GPL (GNU General Public Licence): es la licencia más completa que se puede dar en software, permite usar y modificar el programa con obligación de publicar estas. A partir de esta licencia han aparecido otras variantes como la EUPL (European Union PL).

**Copyright** -> lo prohíbe todo **Copyleft** -> lo permite todo.

**Software propietario (propietary software):** Se distribuye como bytecode y su código fuente no es visible y prohíbe al usuario la distribución, modificación, copia, usos simultáneos etc.

**Software de dominio público (public domain software):** No tiene ningún tipo de licencia o no se puede determinar porque se desconoce el autor.

**Licencias Creative Commons:** Es una organización que se dedica a publicar distintas licencias de autor gratuitas con distintos tipos de restricciones, estas usan símbolos visuales que explican los detalles de cada licencia.

**La C al revés permite la distribución obligando poner las mismas restricciones cc y el = no permite realizar modificaciones al producto.**

**Máquina virtual (Virtual machines):** Es un software que simula un SO determinado y que se ejecuta “sobre” el SO real que lo comunica con el hardware.

Tipos: 1 de **Sistemas** como virtual box (SO dentro de un SO real).

2 De **procesos o aplicación** como la máquina virtual de java (permite ejecutar aplicaciones en distintos tipos de hardware sin modificarlas).

Proxmox es un software de MV pero en una nube local.

**\*Entornos de ejecución (Execution environments):** Es un intermediario entre el código fuente y el SO está formado por la MV y una serie de API’s (bibliotecas) compatibles que permiten que se ejecute una aplicación.

**\*Contenedor Software (Software conteiners):** Es un paquete de elementos que además de la aplicación contiene sus dependencias, librerías y configuración necesarias (Solo incluye lo imprescindible para la portabilidad del software). Docker. Es como una vm de aplicación.

**\*Frameworks (plataforma o entorno de trabajo):** Establece una estructura de módulos (bibliotecas) que sirven como base para la organización y el desarrollo de software en un determinado lenguaje.

**Computación en la nube (Cloud Computing):** Capacidad de ejecución de aplicaciones y almacenamiento de datos en internet (servidor).

**Tipos:**

**Saas (Software-as-a-service):** Son aquellos a los que se accede a través de un navegador.

**Paas(Platform-as-a-Service):** Desarrollo de aplicaciones que se ejecutan en la nube, sin preocuparnos de la infraestructura.

**IaaS(Infraestructure-as-a-Service):** Tenemos más control que con PaaS pero tenemos que encargarnos de la infraestructura. Ej Amazon Web Service (AWS).

**Devops (Development-Operation):** Es un forma agile de organización en conjunto entre los equipos Dev-elopment (desarrollo de aplicaciones) y Op-eration (administrar y explotar las aplicaciones).