Lenguajes de programación y fases del desarrollo

1. Lenguajes de programación:
2. Primer lenguaje de programación (1843): Ada Lovelace desarrolla el primer lenguaje de programación cuando escribió un algoritmo para la máquina analítica de Charles Babbage.
3. Primera generación (1940): Código máquina y Ensambladores  
   La primera generación de lenguajes de programación consistía enteramente de una secuencia de 0s y 1s que los controles de la computadora interpreta como instrucciones, eléctricamente.  
   Representan los primeros lenguajes de programación que la computadora podía entender, son conocidos como lenguaje máquina.
4. Desarrollo del lenguaje ensamblador (1950): Es un lenguaje de programación de bajo nivel, basado en la representación de código de máquina. (binario ) El ensamblador realiza la traducción del código de máquina.
5. Segunda generación (1950): Lenguaje simbólico (ensambladores):  
   Son más fáciles de utilizar que los lenguajes máquina, pero al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador. El lenguaje ensamblador es el primer intento de sustituir el lenguaje maquina por otro más similar a los utilizados por las personas. Este intenta desflexibilizar la representación de los diferentes campos. Esa flexibilidad se consigue no escribiendo los campos en binario y aproximando la escritura al lenguaje. A principios de la década de los 50 y con el fin de facilitar la labor de los programadores, se desarrollaron códigos mercadotécnicos para las operaciones y direcciones simbólicas.
6. Fortran (1957): (“The IBM Mathematical Formula Translating System”): Un lenguaje de programación de alto nivel de propósito general. Para cálculo numérico y científico (como alternativa al lenguaje ensamblador). Es el lenguaje de programación más antiguo que se utiliza hoy en día.
7. Lips (1958): (“List Processor”) Utilizado para la notación matemática y temas de ciencias de la computación.
8. Cobol (1959):   
   (“Common Business-Oriented Language) Usado principalmente para la informática empresarial. Es el primer lenguaje de programación que se demandan por el Departamento de Defensa de EE.UU.

Tercera generación (1959): LENGUAJES DE ALTO NIVEL:

Estos lenguajes son el más utilizado por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensamblador. Un programa escrito en lenguaje de alto nivel es independiente de la máquina (las instrucciones no dependen del diseño del hardware o de una computadora en particular), por lo que estos programas son portables o transportables. Los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras. Son lenguajes de programación en los que las instrucciones enviadas para que el ordenador ejecute ciertas órdenes son similares al lenguaje humano. Dado que el ordenador no es capaz de reconocer estas órdenes, es necesario el uso de un intérprete que traduzca el lenguaje de alto nivel a un lenguaje de bajo nivel que el sistema pueda entender.

1. Basic (1964): (“Beginner’s All-purpose Symbolic Instruction Code”) Diseñado para la simplicidad. Su popularidad explotó a mediados de los años 70 con los computadores personales.
2. Cuarta generación (1967): La Programación Orientación a Objetos (P.O.O.): surge en Noruega en 1967 con un lenguaje llamado Simula 67, desarrollado por Krinsten Nygaard y Ole-Johan Dahl, en el centro de cálculo noruego. Simula 67 introdujo por primera vez los conceptos de clases, corrutinas y subclases (conceptos muy similares a los lenguajes Orientados a Objetos de hoy en día).
3. Pascal (1970): Es un lenguaje de programación desarrollado por el profesor suizo Niklaus Wirth. Su objetivo era crear un lenguaje que facilitara el aprendizaje de programación a sus alumnos, utilizando la programación estructurada y estructuración de datos. Sin embargo con el tiempo su utilización fue de gran uso para herramientas de todo tipo.
4. Quinta generación (1970):   
   La quinta generación de computadoras, también conocida por sus siglas en inglés, FGCS (de Fifth Generation Computer Systems), fue un ambicioso proyecto hecho por Japón a finales de la década de 1970. Su objetivo era el desarrollo de una nueva clase de computadoras que utilizarían técnicas y tecnologías de inteligencia artificial tanto en el plano del hardware como del software, usando el lenguaje PROLOG2 al nivel del lenguaje de máquina y serían capaces de resolver problemas complejos, como la traducción automática de una lengua natural a otra (del japonés al inglés, por ejemplo).
5. C (1972): (Basado en un lenguaje anterior llamado “B”) Creado para sistemas Unix. Muchos de lenguajes de programación más populares del mundo son derivados de él, entre ellos tenemos C#, Java, JavaScript, Perl, PHP y Python.
6. Ada (1980): (En honor a Ada Lovelace): Es derivado del Pascal. Contratado por el Departamento de Defensa de los EE.UU. en 1977 para el desarrollo de sistemas de software grandes.
7. C++ (1983): C++ es un lenguaje de programación diseñado por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos.
8. Objective –C (1983): Es un lenguaje de programación orientado a objetos creado como un superconjunto de C para que implementase un modelo de objetos parecido al de Smalltalk. Originalmente fue creado por Brad Cox y la corporación StepStone en 1980.
9. Perl (1987): Larry Wall comenzó a trabajar en Perl en 1987 mientras trabajaba como programador en Unisys y anunció la versión 1.0 en el grupo de noticias comp.sources.misc el 18 de diciembre de 1987. El lenguaje se expandió rápidamente en los siguientes años. Perl 2, publicado en 1988, aportó un mejor motor de expresiones regulares. Perl 3, publicado en 1989, añadió soporte para datos binarios.
10. Phyton (1991): se remonta hacia finales de los 80s principio de los 90s1 y su implementación comenzó en diciembre de 1992 cuando en Navidad Guido Van Rossum que trabajaba en el (CWI) (un centro de investigación holandés de carácter oficial) decidió empezar el proyecto como un pasatiempo dándole continuidad al lenguaje de programación ABC del que había formado parte del equipo de desarrollo en el CWI,3 dicho lenguaje se enfocaba en ser fácil de usar y aprender manteniendo potencia en su desempeño pero el hardware disponible en la época de su creación hacía difícil su uso y el proyecto no trascendió como se esperaba.
11. Ruby (1993): Ruby es un lenguaje de programación de alto nivel que fue inventado por Yukihiro “Matz” Matsumoto, en Japón, lanzado al mundo en 1995. Ruby es un lenguaje interpretado, de código abierto, y orientado a objetos.
12. Java (1995): El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems.
13. JavaScript (1995): Javascript es un lenguaje de programación interpretado dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Diseñado por: Netscape Communications Corp, Mozilla Fundation.
14. PHP (1995): Creado por Rasmus Lerdorf, la primera encarnación de PHP era un conjunto simple de ficheros binarios Common Gateway Interface (CGI) escritos en el lenguaje de programación C. Originalmente utilizado para rastrear visitas de su currículum online, llamó al conjunto de scripts "Personal Home Page Tools", más frecuentemente referenciado como "PHP Tools".

2. Documentación del desarrollo software: veo importante la documentación de esta por las siguientes causas;

* Usabilidad.
* Correcciones eventuales.
* Mantenimientos futuros.
* Ampliaciones al sistema.

1. 5 fases del desarrollo software:
2. Análisis: Es una de las etapas más importantes ya que hay que comprender completamente el problema a resolver y su contexto. Esto se traduce en un conjunto de requerimientos hechos por el usuario, indicando las distintas funcionalidades que desea que el sistema tenga. Es una etapa crítica en la que hay que invertir el tiempo que sea necesario y en la que se genera un documento de requerimientos.
3. Diseño y arquitectura: En esta etapa se buscan las mejores herramientas desde el punto de vista técnico.
4. Codificación: Reducir un diseño a código puede ser la parte más obvia del [trabajo](https://www.ecured.cu/Trabajo) de ingeniería de software, pero no es necesariamente la porción más larga. La complejidad y la duración de esta etapa está íntimamente ligada al o a los [lenguajes de programación](https://www.ecured.cu/Lenguaje_de_Programaci%C3%B3n) utilizados.
5. Pruebas: Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación. Una [técnica](https://www.ecured.cu/T%C3%A9cnica) de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral, para así llegar al objetivo. Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó, idealmente un [área](https://www.ecured.cu/%C3%81rea) de pruebas; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. En general hay dos grandes formas de organizar un área de pruebas, la primera es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta forma se evalúa que la documentación] entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el software hace las cosas tal y como están descritas. El segundo enfoque es tener un área de pruebas conformada por programadores con experiencia, personas que saben sin mayores indicaciones en qué condiciones puede fallar una aplicación y que pueden poner atención en detalles que personal inexperto no consideraría.
6. Mantenimiento: Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 de toda la ingeniería de software tiene que ver con dar mantenimiento. Una pequeña parte de este trabajo consiste en arreglar errores, o [bugs](https://www.ecured.cu/Bugs). La mayor parte consiste en extender el sistema para hacer nuevas cosas. De manera similar, alrededor de 2/3 de toda la [Ingeniería civil](https://www.ecured.cu/Ingenier%C3%ADa_Civil), [Arquitectura](https://www.ecured.cu/Arquitectura) y trabajo de construcción es dar mantenimiento.

Se puede decir que con la mejora continua garantiza la [calidad](https://www.ecured.cu/Calidad) del [producto](https://www.ecured.cu/Producto), ya que el estarla aplicando día con día es la mejor decisión que puede llegar a tener cualquier empresa, porque de esta manera evita grandes problemas en la elaboración o desarrollo de los productos. Esto es fundamental para todas las empresas ya que se vuelven competitivas, con mayor productividad y eficiencia. No hay que olvidar que la mejora se da porque el cliente es el rey y hay que satisfacer todas y cada una de sus necesidades siempre garantizando la [calidad](https://www.ecured.cu/Calidad).