Lenguajes de

programación - EIN080B

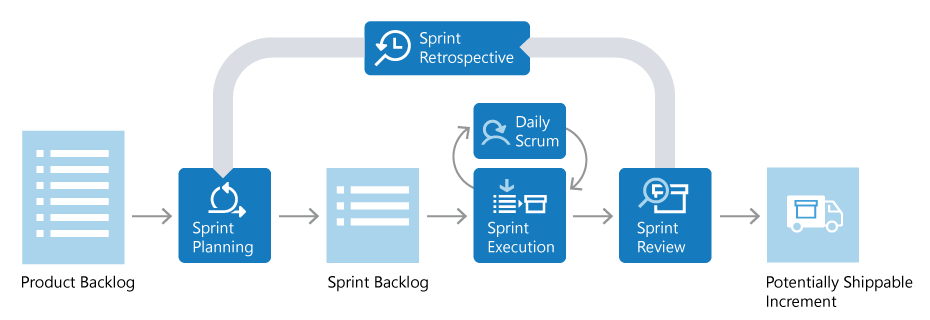
**Metodología de la ingeniería aplicada a proyectos:**

1. Necesidad/Idea.
2. Levantamiento de requerimientos.
3. Análisis.
4. Diseño.
   1. Palabras.
   2. Diagramas UML.\*
5. Construcción ➡️ Programación.
6. Pruebas.
   1. Testeo con usuarios.
   2. Testeo con desarrolladores.
   3. Testeo con otros software.
   4. Testeo con especialistas en el área de aplicación.
   5. Pruebas de estrés (capacidad de conexiones simultáneas).
7. Implementación.
   1. En periodo, contexto y ambiente seguro.
8. Mantención.

**Metodologías ágiles:**

Trabajo colaborativo entre buenos profesionales con mucha experiencia (expertos en 2 disciplinas y buenos en las demás). “Divide y vencerás”.

* XP: Xtreme Programming.
* Design Thinking.
* Lean.
* Kanban.
* SCRUM (la más utilizada):



***“Es imposible ser experto en informática, porque al momento en que aprendiste algo, eso ya se actualizó.”***

**Buen código:**

* Evaluable: *asegurar*  que el código funciona.
* Mantenible: *mantener* el código funcionando.
* Extensible: *agregar* nuevas funcionalidades.
* Flexible: *adoptar* nuevos requisitos.
* DOCUMENTACIÓN, Modularidad, aislamiento, DRY (don’t repeat yourself).

**History of Computer**

- Máquina analítica de Charles Babbage.

- EDVAC y la arquitectura de computadores, John Von Neumann.

- Evolución de los computadores.

- Telefonía móvil (computadores reducidos).

**Algoritmos**

- Definición: Secuencia de instrucciones **no-ambiguas** para resolver un problema y/o ejecutar una determinada acción.

- Al-Juarismi (el señor que aparece en la portada de BALDOR), fue el matemático que desarrolló el primer algoritmo (750-850 DC).

- Ada Lovelace, creadora del primer programa informático, el salto condicional y algoritmo secuencial (siglo 19).

- Algoritmo de Akiyama-Tanigawa: actualización del algoritmo para los números de Bernoulli.

**Futuro de la programación**

- Transhumanismo.

- Programación de seres vivos mediante implementación de hardware en ratones.

- Edición de material genético.

- Computadores cuánticos.

**Evolución del lenguaje de programación:**

* Lenguajes de máquina: primeros programas; Babbage (1837), Turing (1936), Zuse (1941), ENIAC (1946).
* Lenguaje de ensamblado: código simbólico y herramientas de software; IBM (1954).
* Primeros lenguajes de alto nivel: lenguajes independiente de la máquina; FORTRAN (1957), LISP (1958), COBOL (1959).
* Lenguajes estructurados: ALGOL\* (1960), ALGOL 68, PASCAL (1970), C\*\* (1972) y ADA (1979).
* Lenguajes orientados a objeto: Simula (1967), Smalltalk\*\*\* (1980), C++ (1983), Eiffel (1986), Java (1995).
* Lenguajes de Scripting\*\*\*\*: JCL (1964), RUNCOM (1964), SH (1971), GREP (1973), AWK (1977), PERL (1887), TCL (1988), Python (1991), Javascript (1995), PHP (1995), Ruby (1995).

\*Lenguaje elegante, que se construye con un sustento teórico más fuerte.

\*\*Se crea el sistema operativo UNIX.

\*\*\*Se considera un lenguaje puro, es decir, que no utiliza partes de otros tipos de lenguaje para funcionar.

\*\*\*\*No compilado, ”Interpretable”.

**Paradigmas de programación.**

En ciencia, es el método mediante el cuál se resuelve un problema. En programación específicamente, hace referencia a las estructuras bajo las cuales se desarrolla cada lenguaje de programación.

1. ORIENTADO A OBJETOS **(POO)**: Se crean los tipos de datos abstractos, y de aquí evoluciona este lenguaje hacia la manipulación de conjunto de objetos (piezas) que interactúan controladamente intercambiando mensajes. Suelen extenderse del paradigma imperativo. Ejemplos: Smalltalk, Python, C++ y **Java**.
2. IMPERATIVO/SECUENCIAL: Basado en la Máquina de Von Neumann. Ejecución secuencial, variables de memoria, asignación y entrada/salida. Procedimientos típicos. Dominan por mejor desempeño. Ejemplos: Fortran, Algol, Pascal y **C**.
3. DECLARATIVO\*: Se declara lo que se quiere hacer, pero no cómo hacerlo. Es más abstracto al no especificar un algoritmo. Dentro de este derivan:
   1. FUNCIONAL: Basado en cálculo Lambda, usa **funciones** y recursión. Ejemplos: LISP, **Scheme**, Haskell.
   2. LÓGICO: Basado en cálculo de predicados (lógica simbólica). Está fundamentalmente basado en reglas. Ejemplo: **PROLOG**.

\*No existen las variables globales y prácticamente no se utilizan las locales ni de ningún tipo, así que no tiene efectos colaterales.

Otros paradigmas:

* PROGRAMACIÓN BASADO EN EVENTOS: interfaces gráficas, manejo de interrupciones, sistema de sensores.
* PROGRAMACIÓN CONCURRENTE: Conjuntos de procesos cooperativos que se pueden ejecutar en paralelo. Sistemas operativos, sistemas distribuidos.
* PROGRAMACIÓN VISUAL: Manipula objetos gráficos y se integra con otros lenguajes. Scratch, Ingeniería de software, Visual Lab.

**Abstracciones:**

* DATOS PRIMITIVOS: los datos básicos que trae el computador (ej: enteros y reales, caracteres).
* DATOS SIMPLES: datos **no** estructurados, que pueden ser primitivos o definidos por el usuarios en base a uno primitivo.
* DATOS ESTRUCTURADOS: permiten agrupar/componer conjuntos de datos en una unidad (ej: arreglo, registro, archivo de texto). Define nuevo tipo de datos.
* SENTENCIAS: abstraen un conjunto de instrucciones.
* ESTRUCTURAS DE CONTROL: secuenciación, condición y repetición.
* ABSTRACCIÓN PROCEDURAL: permite invocar un procedimiento con un nombre y parámetros.

**Modelos de implementación:**

* COMPILACIÓN: se traduce a lenguaje de máquina para su posterior ejecución (interpretación directa por el procesador). Ejemplo: C, C++.
* INTERPRETACIÓN: una máquina virtual interpreta directamente el código fuente durante la ejecución. LISP, Python.
* SISTEMA HÍBRIDO:

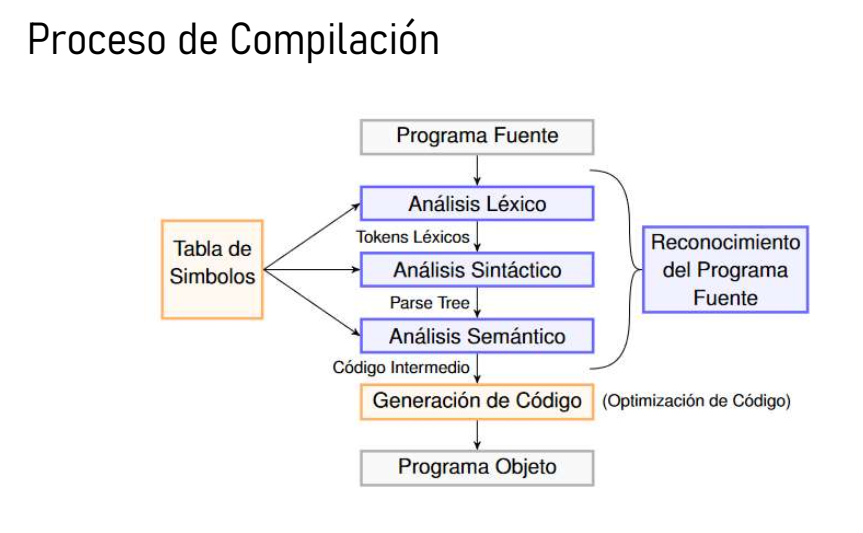
**Sintaxis:** Descripción precisa de todos los programas **gramaticalmente correctos.**

* LÉXICA: Define reglas para símbolos básicos o palabras denominadas *tokens* (ej: identificadores, literales, operadores y puntuación).
* CONCRETA: Representación real de un programa usando símbolos del alfabeto.
* ABSTRACTA: Lleva sólo la información esencial del programa (ej: elimina puntuación). Útil para la traducción posterior y optimización en la generación del código.

**Especificación formal de la sintaxis:**

* RECONOCIMIENTO: Reconoce si una cadena de entrada pertenece al lenguaje.
* GENERACIÓN: Genera todas las cadenas posibles pertenecientes al lenguaje.

**Semántica:** Se refiere al significado del lenguaje, no a la forma (sintaxis). Provee reglas para interpretar la sintaxis, estableciendo restricciones para la interpretación de lo declarado.



**Jerarquía de Chomsky:** Clasificación de gramáticas formales en:

* Tipo 0: lenguajes Recursivamente Enumerables (la categoría más general que puede ser reconocida).
* Tipo 1: lenguajes Sensibles al Contexto.
* Tipo 2: lenguajes Independientes de Contexto (basado en una gramática libre de contexto). Se compone de variables o símbolo no terminal (V), alfabeto o símbolo terminal (Σ), reglas de producción (P) y punto de partida (S).
* Tipo 3: lenguajes Regulares (basado en gramática regular y que se puede obtener mediante expresiones regulares).

**Backus-Naur Form (BNF):** Es un metalenguaje que permite especificar formalmente gramática libres de contexto.

**LIGADO DE TIPO DE DATOS:** Existen de clasificación estática, cuya declaración puede ser explícita o implícita, y se aplica durante la compilación. La clasificación dinámica es aquella que se genera en la ejecución.

**LIGADO DE MEMORIA:** Un nombre puede ser referido con diferentes direcciones y tiempo de ejecución en un programa. El mismo nombre puede ser usado para referirse a distintos objetos de memoria. Ejemplo: Unión y punteros en C y C++, variables dinámicas en Python.

* Alias = sobrenombre, se utiliza para referirse a un mismo objeto en la memoria pero con variables distintas.
* Sobrecarga = un mismo nombre se refiere a diferentes objetos, por ejemplo el operador suma, métodos, etc.

**TAXONOMÍA DE LA MEMORIA:**

* La memoria estática se asigna en tiempo de compilación y no se reescribe, se utiliza de forma más compacta.
* Memoria dinámica:
  + Stack *(“pila”)*: Se utiliza la memoria como si fuera una pila, cuando una variable es llamada, “apila” la memoria necesario para su utilización sobre la memoria estática/principal y la **libera** una vez que terminó de utilizarla.
  + Heap (*“montón”*): Se almacena memoria “por montón”, ya que no se define cuánta memoria se requerirá, sólo se avisa que habrá un requerimiento. A diferencia de la memoria estática y Stack, la memoria Heap no se libera automáticamente, una debe gestionarla.