ENTORNOS DE DESARROLLO

PRUEBA ABIERTA I

ALUMNO CESUR

24/25

Alejandro Muñoz de la Sierra

PROFESOR

Diego Tinedo Rodríguez

0 1

E V O L U C I Ó N D E L O S L E N G U A J E S D E P R O G R A M A C I Ó N Y G R Á F I C O C R O N O L Ó G I C O

Evolución de los lenguajes de programación y gráfico cronológico
A lo largo de los años, los lenguajes de programación han experimentado un desarrollo constante, ajustándose a las necesidades tecnológicas y facilitando el trabajo de los programadores. Este recorrido se puede dividir en varias generaciones, cada una con características distintivas:

Primera generación (1940-1950):
En esta etapa inicial, predominaban los lenguajes ensambladores, diseñados para comunicarse directamente con el hardware. Estos lenguajes utilizaban instrucciones binarias, lo que los hacía complejos pero esenciales para los primeros sistemas informáticos.
Ejemplo: Assembly.

Segunda generación (1950-1960):
Aparecieron los primeros lenguajes de alto nivel, que introdujeron palabras clave parecidas al inglés para simplificar la programación. Esto marcó un gran avance en accesibilidad para los desarrolladores.
Ejemplo destacado: Fortran (1957), utilizado en cálculos científicos y matemáticos.

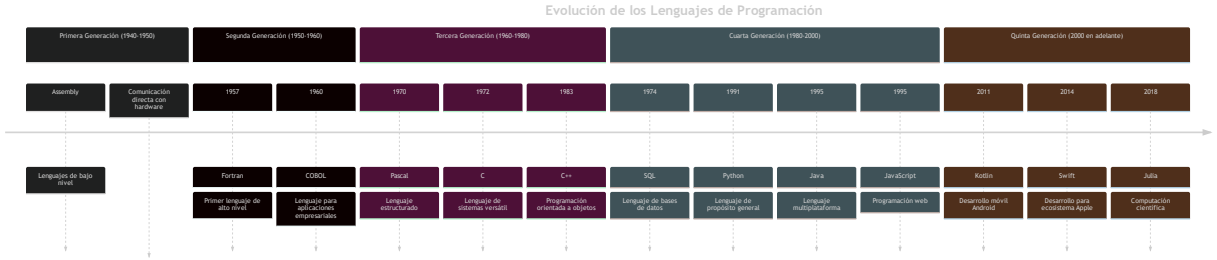
Tercera generación (1960-1980):
Surgieron los lenguajes estructurados y los orientados a objetos, que hicieron más eficiente el desarrollo de software y facilitaron el mantenimiento de los programas.
Ejemplos: C (1972) y Pascal (1970).

Cuarta generación (1980-2000):
Durante esta época, se desarrollaron lenguajes más declarativos y especializados, como los lenguajes de bases de datos o los de scripting. Su diseño se centró en resolver problemas específicos de manera más sencilla.
Ejemplos: SQL (1974) y Python (1991).

Quinta generación (2000 en adelante):
Con un enfoque hacia la inteligencia artificial y el desarrollo móvil, los lenguajes de esta generación priorizan la integración con tecnologías emergentes como el aprendizaje automático y la analítica de datos.
Ejemplos: Kotlin (2011), Swift (2014), y Julia.

Gráfico cronológico:
Para ilustrar esta evolución, diseña una línea del tiempo destacando hitos clave como:

1957: Fortran.
1960: COBOL.
1972: C.
1983: C++.
1991: Python.
1995: Java y JavaScript.
2014: Swift.



0 2

IMPORTANCIA DE DOCUMENTAR TODAS LAS FASES DEL DESARROLLO DEL SOFTWARE

La documentación es el pilar que sostiene el desarrollo de software eficiente, ya que asegura que todos los involucrados puedan entender, mantener y ampliar el sistema con facilidad. Aquí tienes cinco razones fundamentales para documentar cada fase del proyecto:

1. Rastreo y solución de errores:
Una documentación detallada permite identificar rápidamente dónde se originaron los problemas.
Ejemplo práctico: Si un módulo falla, los diagramas y especificaciones pueden indicar en qué parte del sistema ocurrió el error.

2. Facilita la colaboración en equipo:
Sirve como referencia común para todos los desarrolladores, evitando malentendidos y conflictos.
Ejemplo: Un nuevo integrante del equipo puede usar una guía de instalación y manuales de usuario para incorporarse más rápidamente.

3. Mantenimiento y escalabilidad:
Con una documentación adecuada, actualizar o agregar nuevas funcionalidades es mucho más sencillo.
Ejemplo: Un software antiguo puede adaptarse para ser compatible con dispositivos móviles sin necesidad de rediseñarlo desde cero.

4. Cumplimiento de estándares:
Permite seguir normas de calidad, como las certificaciones ISO/IEC 25010, asegurando que el sistema cumpla con requisitos específicos.

Ejemplo: Una auditoría de calidad requerirá documentación precisa para verificar el cumplimiento de normativas.

5. Aprendizaje y mejora continua:
Registrar las decisiones y procesos permite analizar qué funcionó y qué no, mejorando futuros proyectos.
Ejemplo: Documentar un proyecto en el que la falta de pruebas llevó a fallos ayuda a evitar errores similares.

0 3

RESUMEN DE LAS 5 FASES DEL DESARROLLO DE UNA APLICACIÓN DE SOFTWARE

El ciclo de vida del desarrollo de software puede dividirse en cinco etapas clave, cada una con objetivos, actividades y herramientas específicas:

Fase	Objetivo	Herramientas	Salida
Análisis de requerimientos	Identificar qué necesita el cliente.	Entrevistas, encuestas, casos de uso.	Documento de requerimientos funcionales.
Diseño del sistema	Planificar cómo funcionará el software.	Diagramas UML, wireframes.	Especificación técnica y diseño conceptual.
Implementación	Convertir el diseño en código funcional.	IDEs (Eclipse), control de versiones.	Código fuente y pruebas unitarias.
Pruebas	Verificar que el software cumple con los requisitos.	Herramientas de testing, informes.	Informe de errores y correcciones.
Mantenimiento	Garantizar que el software siga siendo útil y actualizado.	Actualizaciones de seguridad.	Software funcional y adaptado.