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.

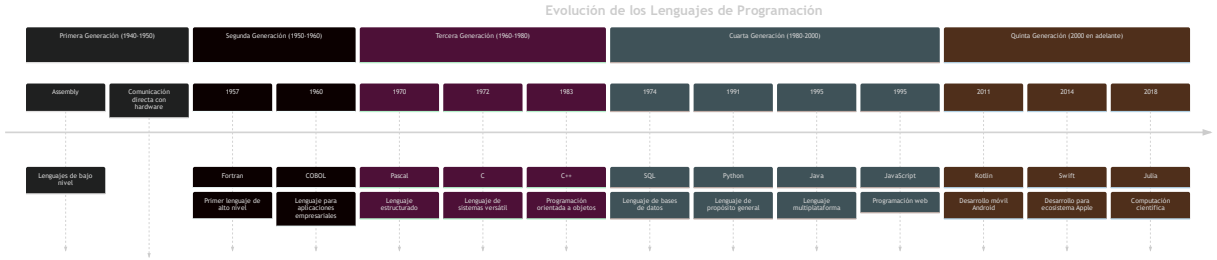
Tercera generación (1960-1980):
Surgieron los lenguajes estructurados y los orientados a objetos, que hicieron más eficiente el desarrollo de software y facilitaron el mantenimiento de los programas.
Ejemplos: C (1972) y Pascal (1970).

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

HERRAMIENTAS USADAS EN PROGRAMACIÓN Y SU ANÁLISIS

La programación hoy en día usa varias herramientas que ayudan en el desarrollo, gestión y verificación de software. A continuación, se revisan algunas de las más comunes, resaltando sus características, pros y contras:

IDEs (Entornos de Desarrollo Integrado):

Visual Studio Code:

Características: Este IDE funciona en múltiples plataformas y soporta lenguajes como JavaScript, Python y C#. Tiene un fuerte sistema de extensiones y un depurador integrado.

Ventajas: Tiene una gran comunidad de apoyo, es ligero y se adapta a diversos proyectos gracias a sus extensiones.
Limitaciones: Aunque es versátil, le faltan funciones avanzadas para proyectos grandes sin usar complementos adicionales.

Eclipse:

Características: Reconocido especialmente en proyectos Java, Eclipse soporta herramientas como Maven y Git.
Ventajas: Es excelente para desarrollos empresariales en Java, incluyendo muchas herramientas preconfiguradas.
Limitaciones: Puede ser más lento que otros IDEs más livianos.

Sistemas de control de versiones:

Git:
Características: Es un sistema distribuido que gestiona versiones, mantiene un historial de cambios y permite trabajo en conjunto.
Ventajas: Es esencial en proyectos actuales y funciona con plataformas como GitHub, GitLab y Bitbucket.
Limitaciones: Puede ser complicado para principiantes debido a su curva de aprendizaje.

Herramientas de documentación:

Javadoc:

Características: Esta herramienta genera documentación técnica a partir de los comentarios en el código, dirigida a Java.
Ventajas: Facilita la documentación al integrarse en el flujo de trabajo y estandarizar el formato.

Markdown:

Características: Es un lenguaje simple ideal para crear documentos legibles.
Ventajas: Su uso en GitHub le da popularidad para documentar proyectos.
Pruebas y validación:

JUnit:

Características: Es un marco popular para pruebas unitarias en Java.
Ventajas: Asegura la calidad de los módulos antes de unirlos al sistema completo.
Selenium:

Características: Herramienta centrada en la automatización de pruebas para interfaces de usuario en aplicaciones web.
Ventajas: Garantiza que la funcionalidad y la experiencia del usuario sean buenas en proyectos web.

04

DOCUMENTACIÓN DE FASES DEL DESARROLLO DE SOFTWARE

Registrar cada etapa del desarrollo es clave para que el proyecto sea entendible y fácil de mantener. Esto también mejora el trabajo en equipo y hace más simples futuras extensiones. Aquí se indican los documentos generados en cada fase:

1. Análisis de requerimientos:
Documento creado: Especificación de Requisitos del Software (SRS).
Contenido:
Objetivos funcionales y no funcionales.
Restricciones técnicas del sistema.
Ejemplo: "El sistema debe permitir el inicio de sesión con OAuth2".

2. Diseño del sistema:
Documento creado: Diseño de Arquitectura de Software.
Contenido:
Diagramas UML como casos de uso, diagramas de clases y de secuencia.
Esquema de bases de datos.
Herramienta recomendada: Lucidchart para diagramas claros y colaboración.

3. Implementación (Desarrollo):
Documento creado: Manual para Desarrolladores.
Contenido:
Instrucciones para configurar el entorno de desarrollo.
Normas de codificación y estructura de carpetas del proyecto.
Ejemplo: "Los controladores deben estar en /src/controllers".

4. Pruebas:
Documento creado: Informe de Pruebas.
Contenido:

Casos de prueba desarrollados para validar funcionalidades.
Resultados de pruebas unitarias y de integración.
Herramienta recomendada: TestRail, que organiza las pruebas de forma eficiente.

5. Mantenimiento:
Documento creado: Registro de Cambios (Changelog).
Contenido:
Funcionalidades Nuevas correcciones y mejoras. Ejemplo: "Versión 1.0.2: Error en el registro de usuarios fue corregido".

05

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.

0 6

R E F E R E N C I A S

https://www.freecodecamp.org/news/what-is-computer-programming-defining-software-development/

https://www.linkedin.com/pulse/la-importancia-de-documentaci%C3%B3n-en-los-proyectos-software-/

https://itequia.com/es/la-documentacion-el-secreto-para-optimizar-procesos-y-evitar-errores-costosos/

https://www.youtube.com/watch?v=t4pbPOEzEiE

https://merida.anahuac.mx/tecnia/blog/herramientas-para-programadores-web

https://www.hostinger.es/tutoriales/herramientas-de-programacion

https://www.tokioschool.com/noticias/herramientas-programacion/

https://asana.com/es/resources/process-documentation

https://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software

https://www.youtube.com/watch?v=s5ABwHaN7as

https://www.youtube.com/watch?v=Cnheob-ohtE