Tema 1 SLDC

1. El concepto SLDC y los Grafos – IA generativa Kimi K2

1.1 ¿Qué es un grafo? despliega la expresión matemática que define un grafo

Un grafo (o gráfica) es una estructura matemática que modela relaciones entre objetos. Está formado por un conjunto de vértices (también llamados nodos) y un conjunto de aristas (o aristas), que representan conexiones entre pares de vértices.

Definición matemática formal

Un **grafo** se define como una tupla:

$$G = (V, E)$$

Donde:

- V es un conjunto finito y no vacío de **vértices** (o nodos),
- $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ es un conjunto de **aristas**, donde cada arista es un par no ordenado de vértices distintos.

> Nota: Esta definición corresponde a un **grafo simple no dirigido**. Existen variantes como grafos dirigidos, multigrafos o grafos ponderados, que tienen definiciones ligeramente distintas.

1.2 ¿Qué es la notación .dot y para qué sirve?

La **notación DOT** es un lenguaje de descripción de grafos creado por el proyecto **Graphviz** (Graph Visualization Software). Sirve para **definir de forma textual la estructura de un grafo** (nodos y aristas) de manera que un programa pueda **dibujarlo automáticamente**.

¿Qué es?

Un archivo `.dot` es simplemente un archivo de texto plano que contiene una descripción declarativa de un grafo usando una sintaxis muy sencilla.

¿Para qué sirve?

- **Generar visualizaciones automáticas** de grafos (diagramas de redes, autómatas, árboles, dependencias, etc.).
- **Intercambio estándar** entre programas que producen o consumen grafos.
- **Prototipado rápido**: escribes diez líneas, ejecutas `dot -Tpng archivo.dot -o salida.png` y obtienes una imagen lista para usar.

Ejemplo mínimo

```
graph G {
    A -- B;
    B -- C;
    C -- A;
}
```

Al procesarlo con Graphviz (`dot -Tpng ejemplo.dot -o ejemplo.png`) se obtiene un triángulo visual.

Variantes

- `graph` → grafo **no dirigido** (aristas con `--`).
- `digraph` → grafo **dirigido** (aristas con `->`).

En resumen, DOT es el "HTML de los grafos": un texto corto que cualquier herramienta Graphviz puede convertir en un dibujo profesional sin que tengas que posicionar nada a mano.

1.3 Cuáles son las direcciones en línea que pueden generar gráficos a partir de notación .dot

Las páginas que **directamente generan una imagen a partir de código DOT** (sin instalar nada) son:
1.**Dreampuf Graphviz Online**
https://dreampuf.github.io/GraphvizOnline
Editor web que compila Graphviz en el navegador; pega tu código, elige formato (SVG/PNG/PDF) y descarga la imagen al instante

.

2.**Graphviz Online Converter (Convertio)**
https://convertio.co/dot-jpg/
Subes un archivo `.dot` y te devuelve JPG, PNG, SVG, PDF, etc. No requiere registro para archivos pequeños

.

3.**Edotor.net**
https://edotor.net
Editor coloreado, preview en vivo y botón de descarga en PNG/SVG; ideal para probar rápidamente pequeños diagramas.
4.**Graphviz Cloud para Confluence** (si usas Confluence)
Añade el macro **Graphviz Cloud** y escribe el código DOT dentro de la página; la imagen se renderiza automáticamente

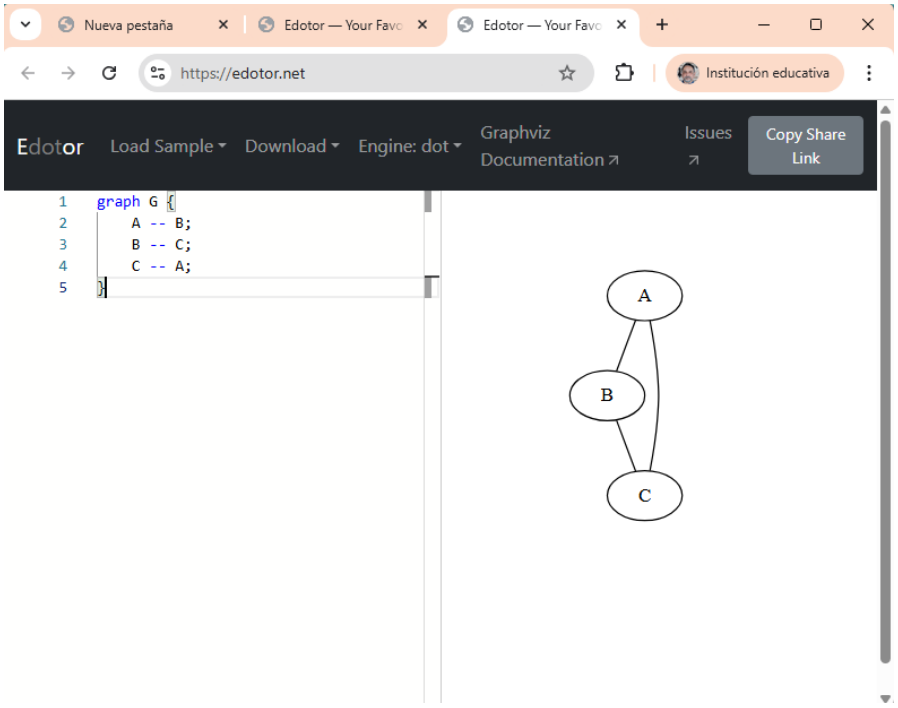
.

5.**VS Code + extensión "Graphviz Interactive Preview"**
Aunque no es "en línea" estrictamente, con la extensión instalada basta con hacer clic derecho → *Preview Graphviz* para ver y guardar la imagen sin salir del editor

.

Con cualquiera de los tres primeros enlaces tienes un generador **100 % web** y gratuito.

1.4 Pre-visualizar el ejemplo mínimo anterior de grafo del punto 1.2. Utilizar la herramienta en https://edotor.net



1.5 ¿Cuál es el ciclo de vida del desarrollo de software?

El ciclo de vida del desarrollo de software (SDLC, por sus siglas en inglés) es un marco de trabajo que define las fases que atraviesa un producto de software desde su concepción hasta su retiro. Aunque existen varios modelos (cascada, ágil, iterativo, etc.), las fases esenciales son comunes:

1.**Planificación**

- Definir el alcance, objetivos, cronograma, presupuesto y recursos.
- Análisis de viabilidad técnica y económica.
- Identificación de riesgos iniciales.

2.**Análisis de requisitos**

- Recopilar y documentar funcionalidades que el sistema debe cumplir (requisitos funcionales y no funcionales).
- Validar con los stakeholders que los requisitos son correctos, completos y alcanzables.

3.**Diseño**

- Arquitectura de alto nivel: componentes, tecnologías, patrones, infraestructura.
- Diseño de bajo nivel: estructuras de datos, interfaces, bases de datos, prototipos de UI/UX.
- Definición de estándares de codificación y seguridad.

4.**Implementación (codificación)**

- Desarrollo del código fuente siguiendo el diseño aprobado.
- Pruebas unitarias por los propios desarrolladores.

- Integración continua y control de versiones.

5.**Pruebas (testing)**

- Pruebas de integración, sistema, rendimiento, seguridad, aceptación del usuario (UAT).
- Gestión de defectos: reporte, priorización, corrección y re-test.
- Validación de que el producto cumple los requisitos.

6.**Despliegue**

- Instalación del software en el entorno de producción.
- Plan de rollback en caso de fallos.
- Capacitación de usuarios finales y soporte inicial.

7.**Operación y mantenimiento**

- Corrección de errores no detectados (correctivo).
- Mejoras de rendimiento o pequeñas funcionalidades (perfectivo).
- Adaptación a nuevos requisitos del entorno (adaptativo).
- Monitoreo continuo, backups, seguridad y parches.

8.**Retiro (fin del ciclo)**

- Desactivación planificada cuando el sistema ya no es viable.
- Migración de datos y transición a un nuevo sistema.
- Archivado de documentación y auditoría final.

Nota: *En modelos ágiles (Scrum, Kanban) estas fases no son secuenciales únicas; se repiten en iteraciones cortas (sprints), priorizando entregas incrementales de valor.*

1.6 Construir un diagrama de SDLC en .dot

A continuación tienes un archivo `.dot` (Graphviz) que representa el flujo clásico del SDLC. Puedes compilarlo con `dot -Tpng sdlc.dot -o sdlc.png` (o cualquier otro formato que prefieras).

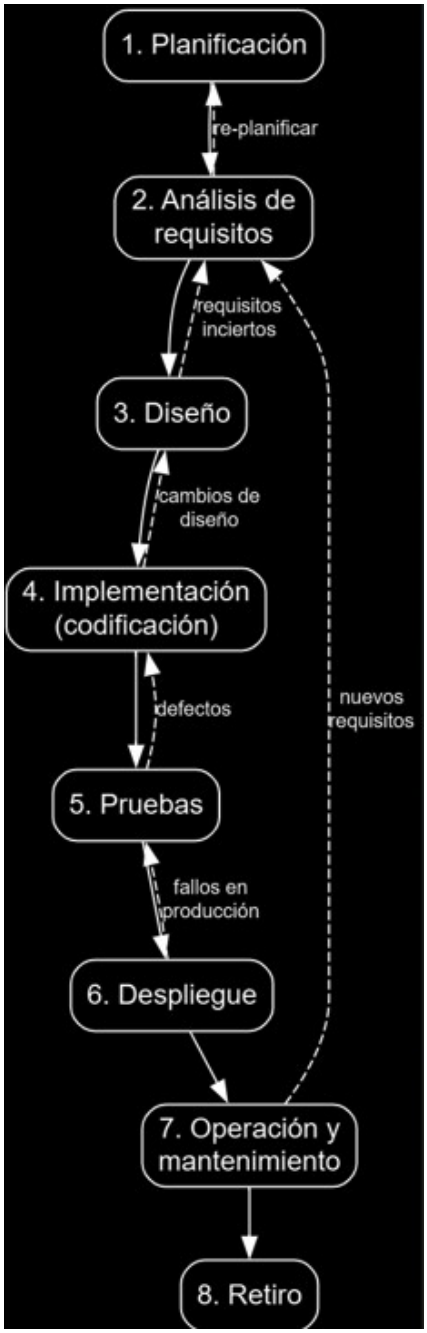
```
digraph SDLC {
    rankdir=TB;
    node [shape=box, style=rounded, fontname="Helvetica"];
    edge [fontname="Helvetica", fontsize=10];

    // Nodos
    plan         [label="1. Planificación"];
    analysis     [label="2. Análisis de\nrequisitos"];
    design       [label="3. Diseño"];
    implement    [label="4. Implementación\n(codificación)"];
    test         [label="5. Pruebas"];
    deploy       [label="6. Despliegue"];
    maintain     [label="7. Operación y\nmantenimiento"];
    retire       [label="8. Retiro"];

    // Flujo principal
    plan -> analysis -> design -> implement -> test -> deploy -> maintain ->
retire;

    // Bucles típicos (feedback)
    analysis -> plan [label="re-planificar", style=dashed];
    design     -> analysis [label="requisitos\ninciertos", style=dashed];
```

```
    implement  -> design   [label="cambios de\ndiseño", style=dashed];
    test       -> implement [label="defectos", style=dashed];
    deploy     -> test      [label="fallos en\nproducción", style=dashed];
    maintain   -> analysis  [label="nuevos\nrequisitos", style=dashed];
}
```

Guarda el bloque anterior en un archivo llamado `sdlc.dot` y genera la imagen con:

bash
```
dot -Tpng sdlc.dot -o sdlc.png
```

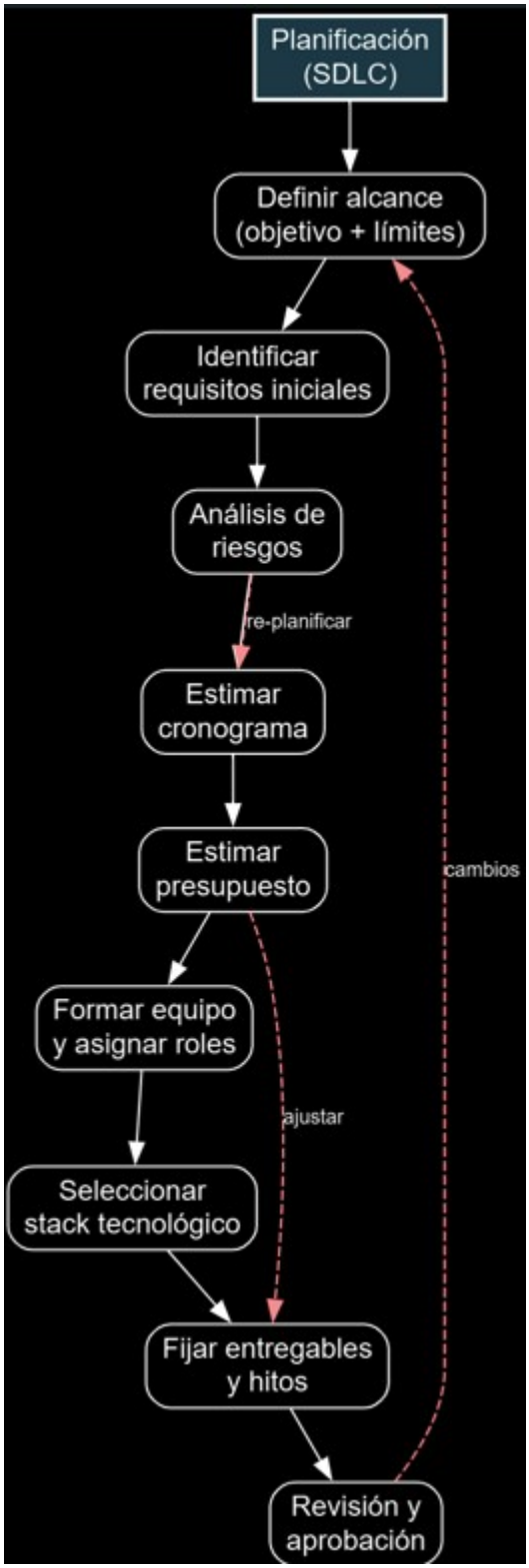
Obtendrás un diagrama claro con las 8 fases y los bucles de retroalimentación más habituales.



1.7 Dibujar la etapa de Planificación usando .dot

A continuación tienes un pequeño archivo `.dot` que ilustra **solo la etapa de Planificación** con los elementos y decisiones típicas que se toman en ella. Puedes copiar/pegar el código en https://edotor.net o en cualquier visor Graphviz para ver el gráfico.

```
digraph Planificacion {
    rankdir=TB;
    node [shape=box, style=rounded, fontname="Helvetica"];
    edge [fontname="Helvetica", fontsize=10];

    /* Nodo principal */
    plan [label="Planificación", shape=box, style=filled, fillcolor=lightblue,
penwidth=2];

    /* Sub-tareas / actividades */
    scope   [label="Definir alcance"];
    obj     [label="Establecer objetivos"];
    time    [label="Cronograma"];
    budget  [label="Presupuesto"];
    risk    [label="Identificar riesgos"];
    team    [label="Asignar recursos\nequipo"];
    tools   [label="Seleccionar herramientas\ny tecnologías"];

    /* Conexiones */
    plan -> scope;
    plan -> obj;
    plan -> time;
    plan -> budget;
    plan -> risk;
    plan -> team;
    plan -> tools;

    /* Estilo opcional para que quede compacto */
    {rank=same; scope obj time budget}
    {rank=same; risk team tools}
}
```


Planificación (SDLC)
Definir alcance (objetivo + límites)
Identificar requisitos iniciales
Análisis de riesgos
re-planificar
Estimar cronograma
Estimar presupuesto
cambios
Formar equipo y asignar roles
ajustar
Seleccionar stack tecnológico
Fijar entregables y hitos
Revisión y aprobación