Metodologías de desarrollo

Ingeniería del Software para Inteligencia Artificial

Metodologías de Desarrollo de Software

- Son conjuntos estructurados de principios, prácticas, procesos y reglas que guían la planificación, ejecución, control y entrega de proyectos de software.
- Su objetivo es asegurar que el software se desarrolle de manera eficiente, predecible y con calidad, facilitando la colaboración entre equipos y alineando el producto final con las necesidades del cliente o usuario.

Tipos de Metodologías de Desarrollo

- **Metodologías Tradicionales**
  - También conocidas como **en cascada**
  - Siguen un **enfoque secuencial** donde cada fase del proyecto debe completarse antes de pasar a la siguiente.
- **Metodologías Ágiles**
  - Basadas en ciclos iterativos e incrementales.
  - Fomentan la colaboración continua con el cliente y la capacidad de adaptarse a cambios durante el desarrollo.

Modelo en cascada (Waterfall)

Modelo en cascada (Waterfall)

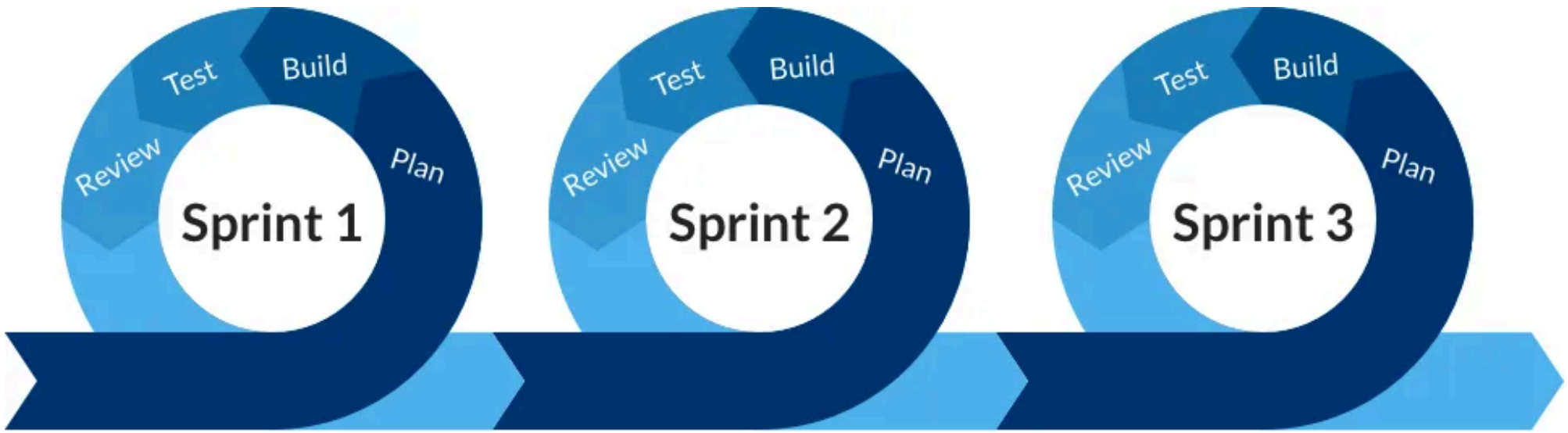
Ventajas:

- Claridad en los entregables por fase.
- Adecuado para proyectos con requisitos bien definidos.

Desventajas:

- Poco flexible ante cambios.
- Riesgo de detectar errores tarde.

Metodologías Ágiles

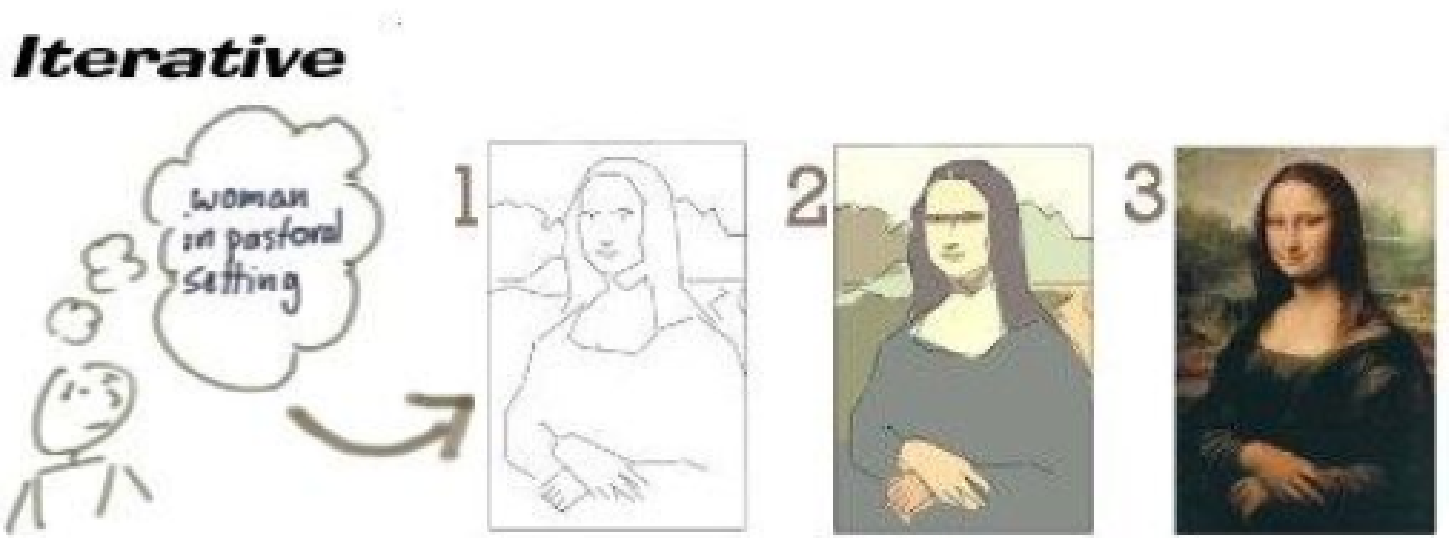


Metodologías Ágiles

Principios clave:

- Basadas en ciclos cortos (**iterativas**).
- Las entregas son **incrementales**.
- Entregas frecuentes de software funcional.
- Equipos autoorganizados y multidisciplinarios.
- Feedback continuo.
- Mejora continua.

Metodologías Ágiles



Metodologías Ágiles

Ventajas:

- Flexibilidad y adaptabilidad.
- Mayor satisfacción del cliente.
- Reducción del tiempo hasta tener valor usable.

Desventajas:

- Requiere compromiso constante del equipo y cliente.
- Puede ser difícil de escalar sin una buena organización.

Metodologías Ágiles

Ejemplos:

- Scrum
- Kanban
- Extreme Programming (XP)
- SAFe (Scaled Agile Framework)

Metodologías Híbridas

Combinan elementos de metodologías tradicionales y ágiles para adaptarse a contextos específicos.

Ejemplo:

- **Agile-Waterfall (o Water-Scrum-Fall)**: requisitos y planificación inicial tradicionales, pero desarrollo ágil.

Elección de la Metodología

La elección de una metodología depende de varios factores:

- Tamaño y complejidad del proyecto.
- Nivel de incertidumbre en los requisitos.
- Cultura organizacional.
- Nivel de experiencia del equipo.
- Necesidad de cumplimiento normativo o regulaciones.

Metodologías en Proyectos IA

Se suelen aplicar metodologías **iterativas e híbridas**, ya que el desarrollo de modelos y la preparación de datos implican exploración, experimentación y ajustes continuos.

- Agile + Data Science
- CRISP-DM

Agile + Data Science

Se adapta el enfoque ágil (Scrum/Kanban) para tareas como:

- Experimentación con algoritmos
- Análisis exploratorio
- Validación iterativa con usuarios y métricas

CRISP-DM

CRISP-DM

Combinación de enfoques ágiles con etapas específicas de ciencia de datos

- Recolección de datos
- Análisis exploratorio
- Entrenamiento y validación del modelo
- Integración y puesta en producción

Fases de CRISP-DM

1. Comprensión del Negocio (*Business Understanding*)

- Identificar objetivos del negocio.
- Traducirlos a objetivos de análisis de datos.
- Definir criterios de éxito.

> *Ejemplo:* reducir la rotación de clientes → predecir la probabilidad de cancelación.

2. Comprensión de los Datos (*Data Understanding*)

- Recolectar datos iniciales.
- Realizar análisis exploratorio.
- Identificar problemas de calidad o inconsistencias.

> *Herramientas típicas:* estadísticas descriptivas, visualizaciones, detección de valores atípicos.

3. Preparación de los Datos (*Data Preparation*)

- Limpiar, transformar y seleccionar los datos relevantes.
- Crear nuevas variables si es necesario.
- Construir el dataset final para modelar.

> *Suele consumir el 60-80% del tiempo del proyecto.*

4. Modelado (*Modeling*)

- Seleccionar técnicas de modelado (clasificación, regresión, clustering…).
- Entrenar y ajustar modelos.
- Evaluar mediante validación cruzada o división train/test.

> *Ejemplo:* árboles de decisión, redes neuronales, random forest, etc.

5. Evaluación (*Evaluation*)

- Verificar que el modelo cumple los objetivos del negocio.
- Analizar resultados más allá de las métricas técnicas.
- Validar que no haya errores o interpretaciones erróneas.

> *¿El modelo es útil, confiable, explicable, justo?*

6. Despliegue (*Deployment*)

- Integrar la solución en el entorno del usuario.
- Automatizar procesos si es necesario.
- Capacitar usuarios y documentar resultados.
- Establecer monitoreo y mantenimiento del modelo.

> *Ejemplos: APIs, dashboards, informes, sistemas de recomendación.*

Enfoque Iterativo

CRISP-DM **no es lineal**. A lo largo del proyecto, es común volver a fases anteriores:

- Si un modelo no funciona bien, puede que haya que regresar a **Preparación de datos**.
- Si los datos no resuelven el problema, se puede redefinir el objetivo en **Comprensión del negocio**.

Ventajas

- Flexible y adaptable.
- Independiente de herramientas o plataformas.
- Fomenta la alineación entre ciencia de datos y negocio.
- Enfocado en valor práctico, no solo técnico.

Limitaciones

- No cubre directamente prácticas modernas como:
  - MLOps (reentrenamiento automático, versionado de modelos).
  - Control de versiones de datos.
  - Evaluación ética o legal.

> *Se puede complementar con herramientas modernas como MLflow, DVC, Kubeflow, etc.*

CRISP-DM vs. Metodologías Ágiles

Aspecto	CRISP-DM	Agile + Data Science
Enfoque	Proceso estructurado iterativo	Ciclos rápidos y adaptativos
Foco principal	Solución analítica robusta	Entregas rápidas y validación
Ideal para…	Exploración y prototipado	Proyectos en producción continua
Limitaciones	Poca flexibilidad en cambios	Poca estructura en proyectos

Captura de Requisitos

Captura de Requisitos

- La **captura de requisitos** es una fase crítica en cualquier proyecto de desarrollo de software.
- En el contexto de la **inteligencia artificial (IA)**, este proceso presenta desafíos adicionales debido a la naturaleza experimental, probabilística y basada en datos de estos sistemas.

Requisitos Funcionales vs. No Funcionales

Requisitos Basados en Datos

Los sistemas de IA dependen fuertemente de los datos, por lo tanto, se deben capturar requisitos como:

- Volumen y disponibilidad de los datos.
- Calidad, variedad y relevancia.
- Necesidades de etiquetado manual.
- Requisitos de anonimización y privacidad.
- Detección y mitigación de sesgos.

Expectativas vs. Realidad

Es habitual que los stakeholders tengan una visión poco realista de lo que la IA puede lograr.

Es crucial:

- Definir niveles aceptables de precisión y error.
- Delimitar claramente los casos en los que el modelo puede fallar.
- Comunicar riesgos e incertidumbres inherentes al sistema.

Explicabilidad y Transparencia

En dominios críticos (salud, finanzas, legal, etc.):

- Se deben definir requisitos para la interpretabilidad del modelo.
- Especificar si se necesitan explicaciones comprensibles para humanos.
- Considerar la trazabilidad de decisiones del sistema.

Validación y Métricas

Es fundamental establecer desde el inicio:

- Las métricas que se usarán (accuracy, precision, recall, F1…).
- Estrategias de validación (cross-validation, test sets, etc.).
- Límites mínimos aceptables de desempeño.

Interacción Humano-IA

Cuando la IA **asiste** (y no reemplaza) a usuarios humanos:

- Definir cómo será la interacción.
- Qué control o supervisión tendrá el usuario final.
- Cómo se mostrarán resultados, recomendaciones o alertas.

Buenas Prácticas

Diseño de Software

Diseño de Software

- El **diseño de software** en proyectos de IA no se limita a definir clases, interfaces o bases de datos.
- Debe considerar aspectos como:
  - Experimentación con modelos
  - Gestión de datos
  - Trazabilidad
  - Integración del modelo en sistemas productivos
- Desde un punto de vista metodológico, el diseño se ve influenciado por la necesidad de trabajar de forma **iterativa, modular y orientada a datos**.

Diseño Clásico vs. Diseño Sistemas IA

Aspecto	Diseño Clásico	Diseño en IA
Requisitos	Estáticos y bien definidos	Evolutivos, dependientes de los datos
Componentes principales	Módulos, clases, interfaces	Pipelines de datos, modelos, experimentos
Validación	Tests unitarios	Validación con métricas estadísticas
Proceso de desarrollo	Determinista	Experimental e iterativo

Diseño Modular e Iterativo

- **Separación de responsabilidades**: dividir el sistema en componentes como:
  - Ingesta y procesamiento de datos
  - Entrenamiento y validación del modelo
  - Persistencia de modelos y métricas
  - Exposición del modelo (API, microservicio)
  - Interfaz de usuario (si aplica)