Tipos de Metodologías de Desarrollo de Software

1. ¿Qué es una Metodología de Desarrollo de Software?

Definición:

Una **metodología de desarrollo de software** es un conjunto de principios, prácticas, procesos, reglas y herramientas que orientan la planificación, ejecución y entrega de un proyecto de software.

Sirve como **una guía estructurada** que permite al equipo de desarrollo avanzar desde la concepción de la idea hasta la entrega del producto final, pasando por etapas como análisis, diseño, programación, pruebas e implementación.

¿Qué incluye una metodología?

- Fases del ciclo de vida del desarrollo.
- Roles y responsabilidades (quién hace qué).
- Normas de documentación.
- Gestión de tareas y tiempos.
- Prácticas de control de calidad.
- Estrategias de comunicación y retroalimentación.

¿Por qué usar una metodología?

Aplicar una metodología **no es opcional** en proyectos medianos o grandes. Es clave para asegurar que:

a. El proyecto se mantenga organizado

Sin una metodología, las tareas pueden ser caóticas, se solapan o se omiten partes clave del desarrollo.

Ejemplo: Sin planificación, un desarrollador puede comenzar a programar sin saber qué necesita realmente el cliente, lo que lleva a retrabajo.

b. Se cumplan los plazos y presupuestos

Las metodologías permiten asignar tiempos, recursos y establecer entregas progresivas.

Ejemplo: En Scrum, se trabaja por sprints, lo que permite controlar cuánto se ha avanzado cada 2 semanas.

c. Haya una mejor comunicación en el equipo

Cada miembro sabe qué debe hacer, en qué etapa está el proyecto y a quién acudir ante un problema.

Ejemplo: En una metodología ágil, como Scrum, hay reuniones diarias para sincronizar al equipo.

d. Se pueda escalar y mantener el sistema

Un software desarrollado con buenas prácticas es más fácil de actualizar, corregir o ampliar.

Ejemplo: Si un sistema está bien documentado y sigue etapas claras, otro equipo puede retomarlo en el futuro sin empezar de cero.

d. El cliente reciba lo que realmente necesita

Las metodologías fomentan la validación continua con el cliente (especialmente en las ágiles), evitando errores de interpretación.

Elaborado por Pablo Ochoa

¿Qué pasa si no usamos ninguna metodología?

- Confusión de roles.
- Falta de planificación y control.
- Cambios desordenados y fuera de tiempo.
- Producto final con fallas o que no satisface al cliente.
- Mayor costo y retrabajo.

2. Características Generales de una Metodología

Una metodología de desarrollo no es solo una lista de pasos, sino una **forma organizada de trabajar** que tiene ciertas **características clave**. Estas características ayudan a los equipos a mantener el enfoque, la calidad y la coordinación durante todo el ciclo de vida del desarrollo del software.

Principales Características

a. Estructura Definida por Fases

Las metodologías se organizan en **etapas o fases específicas** que guían el desarrollo del software de principio a fin.

Ejemplos de fases comunes:

- Análisis de requerimientos
- Diseño del sistema
- Codificación (programación)
- Pruebas
- Implementación
- Mantenimiento

  Ejemplo: En el modelo en cascada, estas fases son secuenciales; en metodologías ágiles, como Scrum, se repiten en ciclos cortos llamados sprints.

b. Planificación y Gestión del Proyecto

Toda metodología incluye herramientas o estrategias para:

- Estimar **tiempos** y **costos**.
- Asignar **recursos humanos** y materiales.
- Identificar **riesgos** y prever problemas.

  Objetivo: Asegurar que el proyecto no se salga de presupuesto ni se retrase innecesariamente.

c. Documentación

Dependiendo del tipo de metodología, la documentación puede ser:

- **Extensa y detallada** (en metodologías tradicionales como Cascada).
- **Ligera y flexible** (en metodologías ágiles como Kanban o Scrum).

La documentación permite:

- Rastrear decisiones tomadas.
- Facilitar el mantenimiento o traspaso del proyecto.
- Servir como referencia en futuras versiones.

d. Control y Seguimiento del Progreso

Permite:

- Medir cuánto se ha avanzado.
- Detectar problemas a tiempo.
- Evaluar si se está cumpliendo el cronograma.

Ejemplo: Scrum utiliza tableros y reuniones diarias (Daily Standup) para monitorear el progreso.

e. Gestión de Cambios

Una metodología bien diseñada incluye formas de manejar cambios en los requisitos del cliente.

- **Tradicionales**: Cambiar algo implica rehacer fases completas.
- **Ágiles**: Aceptan el cambio como parte del proceso, adaptando el trabajo en tiempo real.

f. Roles y Responsabilidades Claras

Cada integrante del equipo sabe qué debe hacer y en qué momento intervenir.

Ejemplo: En Scrum, los roles están claramente definidos: **Product Owner**, **Scrum Master** y **Development Team**.

g. Participación del Cliente o Usuario

- En metodologías **ágiles**, el cliente participa activamente durante todo el proyecto.
- En metodologías **tradicionales**, participa más al principio (definiendo requisitos) y al final (recibiendo el producto).

Esto permite validar constantemente si lo que se está desarrollando es lo que realmente necesita el cliente.

h. Calidad y Pruebas Incorporadas

Las metodologías consideran pruebas para verificar que:

- El software funciona correctamente (funcionalidad).
- No tiene errores graves (estabilidad).
- Cumple los requisitos acordados (calidad).

Algunas metodologías, como **XP (Extreme Programming)**, dan especial importancia a las pruebas automáticas y continuas.

3. Tipos de Metodologías(Desarrollo Secuencial)

Las metodologías tradicionales siguen un enfoque **estructurado y secuencial**. Son ideales cuando los requisitos están **claramente definidos desde el inicio** y no se espera que cambien.

I. Modelo en Cascada (Waterfall)

Descripción:

Es uno de los modelos más antiguos. El desarrollo del software se realiza en **etapas lineales y secuenciales**. Cada fase debe completarse **por completo** antes de pasar a la siguiente.

Fases típicas:

1. Recolección y análisis de requerimientos
2. Diseño del sistema
3. Implementación (codificación)
4. Pruebas (testing)
5. Implementación en producción
6. Mantenimiento

Características:

- Cada fase tiene una **salida clara y documentada**.
- El cliente se involucra **solo al principio y al final**.
- No permite regresar fácilmente a fases anteriores.

Ventajas:

- Fácil de entender y aplicar.
- Buena documentación.
- Útil en proyectos bien definidos y con bajo nivel de cambio.

Desventajas:

- Muy rígido ante cambios.
- Problemas se detectan tarde (en la etapa de pruebas).
- No se adapta bien a proyectos donde los requisitos evolucionan.

¿Cuándo usarlo?

- Proyectos de corta duración.
- Cuando los requisitos están bien definidos desde el principio.
- Aplicaciones internas en entornos muy controlados.

Ejemplo:

Desarrollo de un sistema de nómina en una institución gubernamental, con reglas estrictas y poca posibilidad de cambios.

II. Modelo V (Verificación y Validación)

Descripción:

Es una **evolución del modelo en cascada**, donde cada fase de desarrollo tiene una **fase de pruebas asociada**. Se representa en forma de “V” para mostrar la relación entre las etapas de construcción y sus respectivas validaciones.

Elaborado por Pablo Ochoa

Fases:

- Lado izquierdo de la "V" (desarrollo):
  Requisitos → Diseño funcional → Diseño técnico → Codificación
- Lado derecho de la "V" (pruebas):
  Pruebas unitarias ← Pruebas de integración ← Pruebas de sistema ← Pruebas de aceptación

Características:

- Enfatiza la **verificación temprana** de cada fase.
- Control de calidad fuerte desde el diseño.
- El diseño y pruebas se planifican **desde el inicio**.

Ventajas:

- Detección temprana de errores.
- Mejora la calidad general del producto.
- Ideal para sistemas críticos.

Desventajas:

- Requiere **mucha planificación inicial**.
- Al igual que el cascada, es **poco flexible** a cambios.

¿Cuándo usarlo?

- Sistemas que demandan alta fiabilidad (como software médico o aeroespacial).
- Proyectos donde las fallas pueden tener graves consecuencias.

Ejemplo:

Sistema de control de un equipo médico o de aviónica, donde cada módulo debe estar perfectamente validado antes de usarse.

III. Modelo en Espiral

Descripción:

Combina las características del modelo cascada con **iteraciones cíclicas** que permiten gestionar riesgos. Cada “vuelta” de la espiral representa una versión más avanzada del proyecto, evaluando los riesgos en cada etapa.

Etapas de cada ciclo:

1. Identificación de objetivos y requisitos.
2. Evaluación de riesgos y planificación.
3. Desarrollo y pruebas.
4. Revisión con el cliente y planificación del siguiente ciclo.

Características:

- Centrado en la **gestión de riesgos**.
- Permite revisiones y mejoras continuas.
- Ideal para proyectos **grandes y complejos**.

Ventajas:

- Flexible a los cambios.
- Permite desarrollar versiones incrementales.
- Enfocado en reducir errores costosos mediante análisis de riesgos.

Desventajas:

- Difícil de aplicar en proyectos pequeños.
- Requiere experiencia en análisis y gestión de riesgos.
- Mayor tiempo y costo en planificación.

¿Cuándo usarlo?

- Proyectos a largo plazo y con alta incertidumbre.
- Cuando se requiere validación progresiva del cliente.

Ejemplo:

Sistema ERP modular para una gran corporación, con funcionalidades que se agregan por etapas y con validación constante.

Conclusión

Modelo	Tipo	Ideal para…	Flexibilidad	Foco principal
Cascada	Secuencial	Proyectos simples y con requisitos estables	Baja	Estructura y control
Modelo V	Secuencial	Sistemas críticos donde se requiere alta verificación	Baja	Calidad y validación
Espiral	Iterativo	Proyectos grandes, complejos y con alto riesgo	Alta	Gestión de riesgos y evolución

B. METODOLOGÍAS ÁGILES

Son metodologías **iterativas, incrementales y colaborativas** que permiten adaptarse rápidamente a los cambios. Fomentan la **entrega continua de valor** al cliente.

I. Scrum

Descripción:

Scrum es un marco de trabajo ágil basado en **iteraciones llamadas Sprints** (1 a 4 semanas), en las que se entrega una parte funcional del producto.

Roles:

- **Product Owner**: Define y prioriza las funcionalidades.
- **Scrum Master**: Facilita el proceso y remueve obstáculos.
- **Equipo de desarrollo**: Multidisciplinario y autogestionado.

Elementos clave:

- **Sprint**: Ciclo corto de desarrollo.
- **Daily Scrum**: Reunión diaria de 15 minutos.

- **Sprint Review y Retrospective**: Revisión y mejora continua.
- **Product Backlog**: Lista priorizada de tareas.
- **Sprint Backlog**: Tareas seleccionadas para el sprint actual.

Ventajas:

- Adaptable a cambios frecuentes.
- Alta participación del cliente.
- Fomenta el trabajo colaborativo.

Desventajas:

- Puede volverse caótico sin disciplina.
- No apto para equipos poco autónomos.
- Requiere compromiso constante del cliente.

Ejemplo:

Desarrollo de una app móvil para una startup, donde las prioridades cambian rápidamente según el feedback del mercado.

II. Kanban

Descripción:

Kanban es un método visual de gestión del trabajo que usa un **tablero dividido en columnas** (por ejemplo: Por hacer, En proceso, Hecho). En lugar de iteraciones fijas, se enfoca en el **flujo continuo**.

Elementos clave:

- **Tablero Kanban**: Muestra el estado de las tareas.
- **Límite de trabajo en progreso (WIP)**: Controla cuántas tareas se pueden hacer al mismo tiempo.
- **Mejora continua**: Identificar cuellos de botella y optimizar el flujo.

Ventajas:

- Visualiza fácilmente el estado del proyecto.
- Flexible ante cambios.
- Promueve la eficiencia y el flujo de trabajo.

Desventajas:

- Menos estructura que Scrum.
- Puede ser difícil de escalar a proyectos grandes.
- No incluye roles definidos ni reuniones obligatorias.

Ejemplo:

Mantenimiento de una plataforma web, donde las tareas llegan de forma continua y deben resolverse en cualquier momento.

III. Extreme Programming (XP)

Descripción:

XP es una metodología ágil enfocada en **la calidad del software y la excelencia técnica**. Promueve la entrega frecuente de versiones funcionales y prácticas de programación avanzadas.

Prácticas clave:

- **Programación en pareja (Pair Programming)**.
- **Desarrollo guiado por pruebas (TDD)**.
- **Integración continua**.
- **Pequeñas entregas frecuentes**.
- **Refactorización constante**.

Ventajas:

- Código de alta calidad.
- Facilita el aprendizaje y revisión entre programadores.
- Reducción de errores mediante pruebas automatizadas.

Desventajas:

- Requiere alto compromiso técnico.
- No es adecuada para equipos sin experiencia en buenas prácticas de programación.
- Puede ser costosa por el tiempo dedicado a pruebas y revisiones.

Ejemplo:

Desarrollo de software financiero donde la calidad del código y la confiabilidad son críticas.

IV. Lean Software Development

Descripción:

Adaptación de los principios Lean de manufactura al desarrollo de software. Su enfoque es **reducir el desperdicio**, **mejorar continuamente** y **entregar valor al cliente lo antes posible**.

Principios clave:

1. Eliminar desperdicios.
2. Ampliar el aprendizaje.
3. Decidir lo más tarde posible (evitar suposiciones).
4. Entregar lo antes posible.
5. Empoderar al equipo.
6. Construir calidad desde el inicio.
7. Ver todo el proceso como un sistema optimizable.

Ventajas:

- Alta eficiencia operativa.

- Minimiza recursos innecesarios.
- Ideal para entornos con pocos recursos o necesidad de optimización.

Desventajas:

- Difícil de aplicar si no se identifican bien los “desperdicios”.
- Requiere mucha disciplina y análisis constante del proceso.

Ejemplo:

Empresa que desea optimizar su proceso de desarrollo interno reduciendo tiempos de espera, retrabajo y reuniones innecesarias.

Comparativa rápida entre metodologías ágiles

Criterio	Scrum	Kanban	XP	Lean
Enfoque	Iteraciones (Sprints)	Flujo continuo	Calidad técnica	Eliminación de desperdicio
Estructura	Alta	Baja	Alta	Moderada
Ideal para…	Proyectos con entregas frecuentes	Soporte o mantenimiento	Equipos técnicos experimentados	Procesos ineficientes
Participación del cliente	Constante	Variable	Muy activa	Moderada
Cambio de requisitos	Alto	Alto	Alto	Alto
Curva de aprendizaje	Moderada	Baja	Alta	Alta

Nota:

Las metodologías ágiles permiten adaptarse mejor a entornos cambiantes y centrarse en la entrega de valor al cliente. Cada metodología tiene sus fortalezas y es importante elegir la adecuada según el contexto del proyecto, la madurez del equipo y las expectativas del cliente.

¿Cómo elegir la metodología adecuada?

Factor	Tradicionales	Ágiles
Requisitos fijos	Sí	No siempre
Cambios frecuentes	Difícil de manejar	Se adapta rápido
Proyectos grandes y críticos	Ideal	Requiere adaptación
Tiempo y costo fijo	Planificable	Variable
Participación del cliente	Limitada	Activa
Entregas rápidas	Solo al final	Frecuentes

En resumen:

- Las metodologías **tradicionales** ofrecen control y planificación detallada, pero poca flexibilidad.
- Las metodologías **ágiles** son ideales para contextos cambiantes y donde el cliente necesita ver avances continuos.
- Algunas organizaciones usan **metodologías híbridas**, combinando lo mejor de ambos mundos.

Elaborado por Pablo Ochoa

4. Comparativa entre Metodologías

Las metodologías de desarrollo no son universales ni mejores unas que otras por sí mismas. Cada una se adapta mejor según el **tipo de proyecto, equipo, cliente y contexto organizacional**.

A continuación comparamos las **metodologías tradicionales** y **ágiles** en función de distintos aspectos:

Tabla Comparativa General

Criterio	Metodologías Tradicionales (Ej: Cascada, V, Espiral)	Metodologías Ágiles (Ej: Scrum, Kanban, XP)
Enfoque principal	Planificación y control rígido	Flexibilidad, colaboración y entrega continua
Gestión del tiempo	Lineal, por fases definidas	Iterativa, por ciclos cortos (sprints o flujos continuos)
Gestión de requisitos	Fijos al inicio del proyecto	Evolutivos, se adaptan durante el proceso
Documentación	Extensa y formal	Mínima pero funcional
Participación del cliente	Solo al inicio y al final	Constante durante todo el proyecto
Entrega del producto	Al final del ciclo	Frecuente, por versiones parciales
Adaptación a cambios	Muy baja, implica retrabajo	Alta, los cambios son parte del proceso
Control de calidad	Etapa posterior a desarrollo	Calidad integrada durante todo el proceso
Roles definidos	Generalmente jerárquicos	Colaborativos y autogestionados
Escenarios ideales	Proyectos estables y con requisitos bien definidos	Proyectos cambiantes, clientes activos y necesidades abiertas

Comparativa Específica entre Cascada vs Scrum

Aspecto	Cascada	Scrum
Tipo de metodología	Tradicional	Ágil
Fases	Secuenciales	Iterativas (sprints)
Cambios en requisitos	Costosos y difíciles	Se esperan y se adaptan
Entregas	Solo al final	Al final de cada sprint (ej: cada 2 semanas)
Cliente	Participa al principio y final	Participa constantemente
Equipo de trabajo	Jerárquico, roles separados	Multidisciplinario, autoorganizado
Documentación	Muy detallada	Ligera, solo la necesaria

Análisis según diferentes criterios

a. Contexto del Proyecto

- **Proyectos críticos o normados** (banca, salud, gobierno):
  Se prefiere un modelo tradicional como **Cascada o V**, que permita auditorías y documentación estricta.
- **Startups, desarrollo web o apps móviles**:
  Se prefiere **Scrum, Kanban o XP**, donde se necesita rapidez y adaptación constante.

b. Tipo de Cliente

- Cliente sin mucho tiempo o con requerimientos fijos:
  Mejor usar un modelo tradicional.
- Cliente muy involucrado, con retroalimentación constante:
  Mejor adoptar una metodología ágil.

c. Tamaño del Equipo

- Equipos grandes y estructurados:
  Tradicional o híbrido (por control de roles).
- Equipos pequeños o medianos, multidisciplinarios:
  Ágil, por su enfoque colaborativo.

d. Riesgo y Complejidad

- Si el **riesgo técnico es alto** (por ejemplo, sistemas aeroespaciales):
  Usar **modelo espiral**, que permite evaluar riesgos de forma iterativa.
- Si hay **incertidumbre en requisitos o tecnología**:
  Usar una ágil como **Scrum**, que es más adaptable.

¿Y las metodologías híbridas?

Muchas organizaciones combinan elementos de metodologías tradicionales y ágiles, adaptándolas a su realidad.

> *Ejemplo: Modelo "Agile-Waterfall"*
> Planificación y análisis al estilo cascada, pero desarrollo y pruebas en ciclos ágiles.

> ❝ No existe una metodología "mejor" para todos los proyectos, sino una más **adecuada** según el contexto, los recursos, el cliente y los objetivos del desarrollo. ❞