**1. Necesidad de una metodología**

Antes de los años 70, los sistemas informáticos eran simples y se desarrollaban con técnicas rudimentarias como el método *code & fix*. Este método consistía en:

* Recopilar requerimientos básicos y ambiguos.
* Codificar rápidamente sin análisis previo.
* Corregir sobre la marcha los errores encontrados, tanto lógicos como de requerimientos.

**Problemas de este enfoque:**

* Falta de supervisión, planificación y administración.
* Incremento en costos y tiempos de desarrollo para proyectos complejos.
* Código de baja calidad y difícil mantenimiento.

**Ventajas limitadas:**

* Puede ser útil para proyectos muy pequeños, desarrollados por uno o dos programadores, donde no se justifica invertir en análisis exhaustivo.

El aumento en la complejidad de los programas en los años 70 marcó la necesidad de metodologías sistemáticas para abordar el desarrollo de software de manera más estructurada, evitando los errores y sobrecostos del enfoque *code & fix*.

**2. Definición de metodología**

Una metodología para el desarrollo de software es un conjunto de procesos organizados que guía desde la concepción hasta el mantenimiento de un software. Su propósito es dividir grandes proyectos en etapas manejables y garantizar el éxito mediante:

* **Entradas y salidas claras en cada etapa**: ¿Qué se necesita para iniciar? ¿Qué debe entregarse al final?
* **Normas y procesos definidos**: Cómo realizar las tareas, coordinar equipos y gestionar recursos.
* **Control y seguimiento**: Asegura que el proyecto se desarrolle según lo planeado.

**Ciclo de vida según ISO 12207**: Define al ciclo de vida del software como un marco de actividades desde la definición inicial hasta el retiro del producto.

**Etapas clave del ciclo de vida del software:**

1. **Planificación**:
   * Establecer objetivos del proyecto.
   * Analizar recursos económicos, temporales y humanos.
2. **Implementación**:
   * Realizar actividades prácticas para desarrollar el software.
   * Codificar las funcionalidades definidas en las etapas anteriores.
3. **Puesta en producción**:
   * Validar que el producto cumpla con los requerimientos iniciales.
   * Resolver problemas detectados en la etapa de implementación.

Otras etapas adicionales:

* **Inicio**: Define los objetivos y características del proyecto.
* **Control en producción**: Evalúa la calidad del producto tras su puesta en marcha, corrigiendo desviaciones menores.

**3. Objetivos de cada etapa**

En cada etapa se llevan a cabo actividades específicas para asegurar el progreso del proyecto:

1. **Expresión de necesidades**:
   * Se elabora un documento con los requerimientos funcionales y no funcionales del cliente. Se define el *qué*, no el *cómo*.
2. **Especificaciones**:
   * Formalización de los requerimientos. Este documento será la base para el análisis técnico.
3. **Análisis**:
   * Estudio de la estructura, funcionalidades y elementos del sistema.
   * Define claramente el producto a construir.
4. **Diseño**:
   * Determinar *cómo* se construirá el sistema: lenguajes, bases de datos y estructuras.
5. **Implementación**:
   * Codificación del sistema según lo planeado.
   * Enfoque en transformar el diseño en un producto funcional.
6. **Debugging**:
   * Detección y corrección de errores en el sistema.
   * Garantiza que el programa funcione correctamente a nivel técnico.
7. **Validación**:
   * Comprobar que el sistema cumple con los requerimientos del cliente.
8. **Evolución**:
   * Mantenimiento y mejora del sistema para corregir errores residuales y añadir nuevas funcionalidades.

**4. Clasificación de metodologías**

Existen dos enfoques principales en el desarrollo de software:

1. **Metodología estructurada**:
   * Centrada en descomponer el sistema en procesos y módulos pequeños.
   * Adecuada para resolver problemas mediante análisis lógico.
2. **Metodología orientada a objetos**:
   * Basa el sistema en objetos independientes con atributos y métodos.
   * Favorece la reutilización, flexibilidad y mantenimiento.

Ambas metodologías se adaptan a diferentes tipos de proyectos según sus necesidades y restricciones.

**5. Modelos de ciclo de vida**

Cada modelo se adapta a proyectos con características específicas. A continuación, se detalla cada uno:

**a) Ciclo de vida lineal**

* **Características**:
  + Divide el desarrollo en etapas secuenciales y no retroalimentadas.
  + Cada etapa se completa antes de pasar a la siguiente.
* **Ventajas**:
  + Sencillez en planificación y gestión.
  + Adecuado para proyectos pequeños y simples.
* **Desventajas**:
  + No apto para proyectos donde surjan cambios o retrocesos.
  + Costoso retomar etapas previas para corregir errores.

**b) Ciclo de vida en cascada puro**

* **Características**:
  + Propone revisiones al final de cada etapa antes de avanzar.
  + Acepta iteraciones limitadas.
* **Ventajas**:
  + Elevada calidad del producto final.
  + Metodología clara y estructurada.
* **Desventajas**:
  + Requiere conocer todos los requerimientos al inicio.
  + Costoso y lento si hay que retroceder etapas.

**c) Ciclo de vida en V**

* Similar al cascada puro, pero introduce validaciones cruzadas entre etapas (análisis-diseño y diseño-debugging).
* Aumenta la confiabilidad del producto.

**d) Ciclo de vida iterativo**

* **Características**:
  + Repite ciclos pequeños de análisis, diseño, implementación y pruebas.
  + El cliente evalúa el producto tras cada iteración.
* **Ventajas**:
  + Mayor adaptabilidad a cambios en requerimientos.
  + Proporciona versiones funcionales parciales al cliente.
* **Desventajas**:
  + Puede ser difícil gestionar múltiples iteraciones.

**e) Ciclo de vida por prototipos**

* **Características**:
  + Se desarrollan prototipos para validar requerimientos antes del producto final.
  + Útil en proyectos con tecnologías nuevas o alta incertidumbre.
* **Desventajas**:
  + Costos elevados y gestión complicada.

**f) Ciclo de vida incremental**

* Construye el sistema por módulos funcionales independientes.
* Permite entregas graduales al cliente.

**g) Ciclo de vida en espiral**

* Combina prototipado e incremental, con un enfoque en análisis de riesgos.
* Cada iteración incluye planificación, análisis de riesgos, desarrollo y evaluación.
* Ideal para proyectos grandes con alta incertidumbre inicial.

**h) Ciclo de vida orientado a objetos**

* Basado en la abstracción de requerimientos en objetos, favoreciendo modularidad y reutilización.

**6. Conclusión**

No hay un modelo ideal universal; la elección depende del tipo de proyecto, la claridad de los requerimientos y el entorno de desarrollo. Factores clave:

* Complejidad del problema.
* Disponibilidad de tiempo y recursos.
* Necesidad de entregas parciales.