# ***INGENIERÍA DEL SOFTWARE***

**Introducción a la Ingeniería del Software: ¿Qué es?**

La **Ingeniería del Software** es una disciplina de la informática que se encarga de la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software. Su objetivo principal es producir software de alta calidad que cumpla con los requisitos del usuario, sea confiable, eficiente y mantenible a lo largo del tiempo.

**¿Por qué surge la Ingeniería del Software?**

- A medida que el software se ha vuelto más complejo y crucial en la mayoría de las áreas de la sociedad, surgió la necesidad de aplicar principios y prácticas bien estructuradas para evitar problemas de desarrollo como la entrega tardía, costos excesivos o software que no satisface las necesidades del cliente.

La Ingeniería del Software proporciona **metodologías**, **herramientas** y **prácticas** que permiten gestionar la complejidad de los sistemas de software, centrándose en aspectos como:

- **Especificación de requisitos**: definir qué debe hacer el software.
- **Diseño**: estructurar la solución del problema.
- **Desarrollo**: escribir el código y construir el sistema.
- **Verificación y pruebas**: garantizar que el software funcione correctamente.
- **Mantenimiento**: actualizar y modificar el software a lo largo del tiempo.

**Estado Actual y Antecedentes: La Crisis del Software**

La **Crisis del Software** es un concepto que surgió en la década de los 60 y 70, cuando la demanda de software creció exponencialmente debido al auge de la informática, mientras que la capacidad de las organizaciones para producir software de calidad no podía seguir el ritmo. Los problemas más destacados que caracterizan esta crisis son:

1. **Proyectos fuera de control**:
    - Muchos proyectos de software no cumplían con los plazos de entrega o se excedían enormemente en el presupuesto.
2. **Software de mala calidad**:
    - El software que se entregaba solía tener errores, ser ineficiente y difícil de mantener. Esto resultaba en productos que no satisfacían las necesidades del usuario o que requerían grandes esfuerzos para corregirse y mejorarse.
3. **Escalabilidad y complejidad**:
    - A medida que los sistemas se volvían más grandes y complejos, los enfoques tradicionales para desarrollar software (que en muchos casos se centraban en la programación individual) ya no eran suficientes.
4. **Escasez de profesionales capacitados**:
    - El rápido crecimiento de la demanda de software no iba acompañado de suficientes ingenieros bien preparados para abordar estos desafíos.

**Estado actual**:

- Hoy en día, aunque algunos de los problemas de la Crisis del Software aún persisten (como los proyectos fallidos o las dificultades en el mantenimiento), la Ingeniería del Software ha avanzado enormemente en cuanto a metodologías, herramientas y prácticas. La adopción de metodologías ágiles, el uso de patrones de diseño, la automatización de pruebas y el uso de herramientas modernas de gestión de proyectos han contribuido a mitigar algunos de los problemas que impulsaron la Crisis del Software.

**Disciplinas que Conforman la Ingeniería de Software**

La Ingeniería del Software abarca un conjunto de disciplinas interrelacionadas, cada una de las cuales se centra en una parte específica del ciclo de vida del software o en algún aspecto técnico clave. Entre las principales disciplinas se encuentran:

1. **Análisis de Requisitos**:
    - Implica la identificación y documentación de las necesidades y expectativas del cliente y los usuarios del sistema. Se trata de entender exactamente qué debe hacer el software antes de comenzar a desarrollarlo.
    - Herramientas y técnicas comunes: entrevistas, casos de uso, historias de usuario.
2. **Diseño de Software**:
    - Se refiere a la creación de la arquitectura y estructura del sistema de software, descomponiéndolo en componentes que colaboran entre sí para cumplir con los requisitos. El diseño puede dividirse en:
        - **Diseño de alto nivel** (o arquitectura): cómo se estructura el sistema en módulos o componentes grandes.
        - **Diseño detallado**: diseño de los módulos y sus interacciones.
    - Ejemplos: arquitectura orientada a microservicios, patrones de diseño como el MVC (Modelo-Vista-Controlador).
3. **Desarrollo o Programación**:
    - Esta disciplina se enfoca en la escritura del código fuente del software, implementando los algoritmos y estructuras de datos necesarios para que el sistema funcione. El desarrollo puede hacerse utilizando distintos lenguajes de programación y paradigmas, como programación orientada a objetos, funcional, entre otros.
4. **Pruebas de Software**:
    - Consiste en verificar y validar que el software cumple con los requisitos especificados y que está libre de errores o "bugs". Existen diferentes niveles de pruebas, como pruebas unitarias, de integración y de aceptación, que se aplican en distintas etapas del desarrollo.
    - Técnicas de pruebas: pruebas automáticas, pruebas manuales, pruebas de caja negra y pruebas de caja blanca.
5. **Mantenimiento de Software**:
    - Esta disciplina se refiere a las actividades necesarias para corregir, mejorar y actualizar el software después de que ha sido entregado y está en uso. El mantenimiento es crucial, ya que el software rara vez es estático y necesita adaptarse a nuevos requerimientos o tecnologías.
    - Tipos de mantenimiento: correctivo, adaptativo, perfectivo y preventivo.
6. **Gestión de Proyectos de Software**:
    - Se centra en la planificación, organización y control de los recursos necesarios para completar un proyecto de software a tiempo y dentro del presupuesto. Esto incluye la gestión de equipos, la estimación de tiempos y costos, la gestión de riesgos, entre otros.
    - Metodologías comunes: Scrum, Kanban, PMBOK.
7. **Ingeniería de Calidad del Software**:
    - Abarca las prácticas necesarias para garantizar que el software producido sea de alta calidad. Esto incluye la definición de estándares de calidad, revisiones de código, auditorías y pruebas rigurosas.
    - Enfoques comunes: mejora continua, modelos de calidad como ISO/IEC 25010.
8. **Ingeniería de Procesos de Software**:

- Esta disciplina estudia y mejora los procesos utilizados en el desarrollo de software. El objetivo es optimizar la eficiencia y calidad de dichos procesos, utilizando modelos de referencia como CMMI (Capability Maturity Model Integration) o ISO 9001.

**Ejemplos de Grandes Proyectos de Software Fallidos y Exitosos**

**Proyectos Fallidos**

1. **Proyecto del Sistema de Control Aéreo de Denver (1994)**
   - **Descripción**: El aeropuerto internacional de Denver intentó implementar un sofisticado sistema automatizado para el manejo de equipaje y control aéreo.
   - **Problemas**: El sistema era demasiado ambicioso, con requisitos complejos y tecnologías inadecuadas. El desarrollo se retrasó, los costos se dispararon y el proyecto terminó siendo cancelado.
   - **Conclusión**: La mala gestión de requisitos y el exceso de confianza en la tecnología sin una planificación sólida llevaron a su fracaso. Fue un ejemplo clásico de la "Crisis del Software" en la práctica.
2. **Proyecto del Sistema de Salud "Care.data" del NHS (Reino Unido, 2016)**
   - **Descripción**: Un proyecto para recolectar y compartir datos de pacientes del Sistema Nacional de Salud (NHS) del Reino Unido con el fin de mejorar los servicios de salud.
   - **Problemas**: Hubo una mala comunicación con los pacientes, quienes no comprendían cómo se utilizarían sus datos, lo que llevó a una fuerte reacción pública y preocupaciones por la privacidad. Finalmente, el proyecto fue cancelado.
   - **Conclusión**: Un mal manejo de la relación con los stakeholders, junto con la falta de claridad en la gestión de datos, llevó al fracaso de este proyecto.
3. **Proyecto de Integración del Software de Defensa del Pentágono (F-35 Joint Strike Fighter)**
   - **Descripción**: El programa F-35 implicaba el desarrollo de software avanzado para aviones de combate. El software tenía como objetivo integrar diferentes sistemas de armas y navegación en una sola plataforma.
   - **Problemas**: El desarrollo fue plagado de sobrecostos, retrasos y defectos en el software. Al ser un sistema altamente complejo, el código fue difícil de mantener, y las pruebas resultaron insuficientes.
   - **Conclusión**: La complejidad del proyecto, junto con problemas de gestión, hicieron que el sistema tuviera dificultades para cumplir con los plazos y el presupuesto previsto.

**Proyectos Exitosos**

1. **Sistema Operativo Linux**
   - **Descripción**: Linux es un sistema operativo de código abierto que comenzó como un proyecto personal de Linus Torvalds y ha evolucionado hasta convertirse en un sistema utilizado en servidores, dispositivos móviles y sistemas embebidos.
   - **Éxito**: Gracias a la colaboración de una vasta comunidad de desarrolladores y un modelo de desarrollo basado en la transparencia, Linux se ha convertido en un pilar de la informática moderna.
   - **Conclusión**: El enfoque colaborativo y el uso de prácticas ágiles y distribuidas ayudaron a crear un software altamente exitoso.
2. **Proyecto Apache HTTP Server**
   - **Descripción**: Apache es uno de los servidores web más utilizados en el mundo. Iniciado como un esfuerzo colaborativo de código abierto, ha proporcionado una plataforma estable y confiable para la mayoría de los sitios web.

- **Éxito**: Su modelo de desarrollo abierto y modular, junto con una comunidad dedicada, ha permitido una rápida innovación y una gran adopción.
- **Conclusión**: Un proyecto basado en prácticas ágiles y una buena gestión de la comunidad fue clave para su éxito.

3. **Sistema de Reservas Aéreas SABRE**
   - **Descripción**: SABRE, desarrollado en la década de 1960 por IBM y American Airlines, es uno de los primeros sistemas de reservas electrónicas en tiempo real. Ha sido clave para la gestión de reservas de vuelos y sigue siendo un sistema fundamental para las aerolíneas hoy en día.
   - **Éxito**: Fue pionero en el uso de sistemas distribuidos y redes globales, estableciendo un estándar para la industria aérea.
   - **Conclusión**: Un enfoque innovador, junto con la capacidad de adaptarse y evolucionar, permitió a SABRE convertirse en un proyecto exitoso y duradero.

---

**Ciclos de Vida (Modelos de Proceso) y su Influencia en la Administración de Proyectos de Software**

Los **ciclos de vida del software** son los modelos que definen las etapas por las que pasa un proyecto de software desde su concepción hasta su finalización. Estos modelos influyen en cómo se gestiona un proyecto, ya que determinan la secuencia de actividades y cómo interactúan entre sí. Existen varios modelos de ciclo de vida, algunos de los cuales son más rígidos y otros más flexibles, afectando directamente la planificación, ejecución y control del proyecto.

1. **Modelo en Cascada**:
   - **Descripción**: Este modelo sigue un enfoque secuencial, donde cada fase del ciclo de vida (requisitos, diseño, desarrollo, pruebas, implementación) se completa antes de pasar a la siguiente.
   - **Impacto en la administración de proyectos**:
     - Es fácil de gestionar debido a su estructura lineal y clara, pero es inflexible si los requisitos cambian.
     - Es ideal para proyectos con requisitos muy bien definidos desde el inicio, pero arriesgado en proyectos donde los requisitos pueden evolucionar.
2. **Modelo Iterativo e Incremental**:
   - **Descripción**: En lugar de desarrollar todo el software de una vez, el proyecto se divide en pequeños incrementos. Cada incremento agrega funcionalidad al sistema y puede ser revisado y ajustado antes del próximo ciclo.
   - **Impacto en la administración de proyectos**:
     - Permite gestionar cambios en los requisitos, ya que en cada iteración se pueden hacer ajustes.
     - Mejora la satisfacción del cliente al entregar partes del software funcionales de manera temprana y frecuente.
3. **Modelo Ágil**:
   - **Descripción**: Basado en metodologías ágiles como Scrum y Kanban, este modelo favorece la entrega rápida de software funcional, la colaboración continua con el cliente y la adaptabilidad al cambio.
   - **Impacto en la administración de proyectos**:
     - Requiere una administración de proyectos más dinámica, con entregas rápidas y revisiones continuas.

        - La flexibilidad y capacidad de adaptación lo hacen ideal para proyectos donde los requisitos no están completamente definidos desde el inicio.

4. **Modelo Espiral**:
    - **Descripción**: Combina elementos del modelo en cascada y el enfoque iterativo, con un fuerte enfoque en la gestión de riesgos. Se realiza un desarrollo en ciclos, donde en cada ciclo se identifican y mitigan riesgos.
    - **Impacto en la administración de proyectos**:
        - Es útil para proyectos grandes y complejos, ya que permite una evaluación continua de riesgos y ajustes en función de estos.
        - La gestión de riesgos es clave, lo que requiere una administración de proyectos sólida y proactiva.

---

**Procesos de Desarrollo Empíricos vs. Definidos**

Los **procesos de desarrollo empíricos** y **definidos** representan dos enfoques opuestos en la forma en que se gestiona y controla el desarrollo de software:

1. **Procesos Definidos**:
    - **Características**: En un proceso definido, las actividades y los resultados son predecibles y repetibles. Se basa en planes detallados y especificaciones que deben cumplirse estrictamente.
    - **Ventajas**:
        - Alta previsibilidad: los pasos a seguir y los resultados son claros desde el inicio.
        - Ideal para proyectos con pocos cambios o que requieren alta precisión en su ejecución.
    - **Desventajas**:
        - Menos flexibilidad para adaptarse a cambios imprevistos.
        - Puede ser ineficaz en proyectos donde los requisitos no están claramente definidos.
2. **Procesos Empíricos**:
    - **Características**: En este enfoque, se asume que el desarrollo del software es inherentemente impredecible. Las decisiones se basan en la observación y la retroalimentación continua. Un ejemplo es el desarrollo ágil.
    - **Ventajas**:
        - Alta adaptabilidad: permite cambios en los requisitos y ajustes en el proceso sobre la marcha.
        - Fomenta la colaboración y la entrega rápida de valor.
    - **Desventajas**:
        - Más difícil de gestionar en grandes proyectos o en entornos altamente regulados.
        - Requiere una gestión más activa y la capacidad de adaptarse rápidamente a nuevos descubrimientos.

**Ventajas y Desventajas de los Ciclos de Vida (Modelos de Proceso)**

Cada ciclo de vida del desarrollo de software ofrece un enfoque diferente para gestionar un proyecto, lo que tiene implicaciones en términos de flexibilidad, control, costos y capacidad para adaptarse a cambios. A continuación se detallan las **ventajas** y **desventajas** de los modelos más comunes:

**1. Modelo en Cascada**

- **Ventajas**:
    - Claridad en las fases: Las etapas del proyecto están bien definidas (análisis, diseño, implementación, pruebas, etc.).
    - Fácil de gestionar: Es simple y fácil de seguir, ideal para equipos sin experiencia en la gestión de proyectos.
    - Documentación exhaustiva: Cada fase genera una documentación completa que facilita la revisión y auditoría.
- **Desventajas**:
    - Falta de flexibilidad: Los cambios en los requisitos son difíciles de incorporar una vez que el proyecto avanza.
    - Problemas en la detección temprana de errores: Los errores se detectan tarde en el ciclo, en la fase de pruebas.
    - Difícil de adaptar a proyectos con requisitos cambiantes o poco definidos al inicio.
    - Alta posibilidad de riesgos si los requisitos iniciales no están bien entendidos.

**2. Modelo Iterativo e Incremental**

- **Ventajas**:
    - Flexibilidad: Permite realizar cambios a lo largo del ciclo de vida del proyecto.
    - Entrega temprana de partes funcionales: Se entregan incrementos pequeños y funcionales del software, lo que mejora la retroalimentación del cliente.
    - Reducción del riesgo: Al entregar incrementos, se pueden abordar problemas y riesgos más rápidamente.
- **Desventajas**:
    - Mayor necesidad de gestión: Al ser iterativo, requiere una planificación y seguimiento constante.
    - Posible acumulación de deuda técnica: Si no se gestionan bien las iteraciones, puede aumentar la complejidad del sistema con cada ciclo.
    - Requiere un mayor compromiso del cliente y retroalimentación continua.

**3. Modelo Ágil**

- **Ventajas**:
    - Adaptabilidad: Responde rápidamente a los cambios en los requisitos del proyecto.
    - Mejora continua: La retroalimentación constante permite mejorar el producto de forma iterativa.
    - Satisfacción del cliente: Los entregables son funcionales y se entregan rápidamente, lo que mejora la comunicación y la satisfacción del cliente.
    - Enfoque en las personas y la colaboración: Fomenta un trabajo más colaborativo entre los equipos.
- **Desventajas**:
    - Dificultad en grandes proyectos: La escalabilidad puede ser un reto en proyectos grandes o con muchos equipos.
    - Falta de documentación: Al priorizar la funcionalidad, la documentación puede ser insuficiente.
    - Menos adecuado para proyectos con plazos o presupuestos rígidos: Si no se gestiona bien, la flexibilidad puede llevar a desviaciones de tiempo o costo.

**4. Modelo Espiral**

- **Ventajas**:
    - Gestión de riesgos: En cada ciclo, se evalúan y mitigan los riesgos, lo que mejora la probabilidad de éxito del proyecto.
    - Flexibilidad: Combina lo mejor de los modelos en cascada e iterativo, lo que permite ajustar los requisitos y los resultados a lo largo del tiempo.
    - Adecuado para proyectos grandes y complejos: Es ideal cuando se necesita una gestión cuidadosa de riesgos y una validación constante.
- **Desventajas**:
    - Complejidad: Es más complejo de gestionar, especialmente en cuanto a la evaluación de riesgos y la planificación de múltiples ciclos.
    - Costoso: Requiere más tiempo y recursos para realizar evaluaciones de riesgos y revisiones continuas.
    - Difícil de gestionar sin experiencia: Necesita de equipos altamente capacitados para implementar adecuadamente el enfoque.

## Resumen de Criterios de Selección

| Criterio | Modelo en Cascada | Modelo Iterativo/Incremental | Modelo Ágil | Modelo Espiral |
|---|---|---|---|---|
| Claridad de Requisitos | Claros y Estables | No totalmente claros | No definidos / Cambiantes | Parcialmente claros |
| Tamaño del Proyecto | Pequeño a mediano | Mediano a grande | Pequeño a mediano | Grande y complejo |
| Flexibilidad | Baja | Moderada | Alta | Moderada a alta |
| Plazos y Presupuesto | Rígidos | Flexibles | Flexibles | Rígidos o flexibles |
| Gestión de Riesgos | Baja | Moderada | Baja | Alta |
| Participación del Cliente | Baja a moderada | Moderada a alta | Alta | Moderada |
| Entregas Frecuentes | No | Sí | Sí | Moderado |
| Recursos y Capacidades del Equipo | Menos intensivo | Moderado | Requiere equipo ágil y flexible | Requiere equipo experimentado |