

Asignatura

Arquitectura del Software



Profesor

Yago Fontenla Seco

{yago.fontenla1@uie.edu}


Unidad I – Fundamentos de Ingeniería del Software

Se introducen los **conceptos clave de la ingeniería y arquitectura** de software, con un enfoque en los fundamentos de la ingeniería de software, la identificación y modelado de requisitos, y la implementación de **metodologías ágiles** como Scrum y Kanban para el desarrollo de software.

Temas

- 1.1. Introducción a la ingeniería y arquitectura de software, conceptos, ejemplos. Lenguaje de programación.
- 1.2. Fundamentos de ingeniería del software. Requisitos y modelado conceptual.
- 1.3. Metodologías Ágiles para el desarrollo de software.

Tema 1.1

Introducción a la ingeniería y arquitectura de
software 

Introducción a la Ingeniería de Software

¿Qué es un “software”?

¿Qué es ingeniería?

¿Qué es arquitectura?

¿Qué es el software?

El **software** es el **conjunto de programas y datos asociados** creados por programadores profesionales para cumplir tareas específicas en sistemas informáticos. Incluye desde **sistemas operativos** que gestionan hardware hasta aplicaciones que permiten realizar tareas como edición de texto, análisis de datos o control de dispositivos.

El software no tiene una forma física, pero es esencial para el funcionamiento de computadoras, dispositivos móviles y sistemas embebidos. A lo largo de su vida, el software se somete a un proceso de desarrollo que involucra su **diseño, codificación, prueba y mantenimiento** continuo, lo que asegura su adaptabilidad y eficiencia a largo plazo.

¿Qué es el software?

El **software** es un conjunto de **programas y datos asociados** que permiten a una computadora realizar tareas específicas. Puede clasificarse en:

- **Software de sistema:** Como sistemas operativos que gestionan recursos de hardware.
- **Software de aplicación:** Aplicaciones diseñadas para realizar tareas específicas para los usuarios.
- **Software embebido:** Programas incorporados en dispositivos de hardware, como microondas o coches.

El software se compone de:

- **Código fuente:** Instrucciones escritas por programadores.
- **Documentación:** Material que describe cómo usar y mantener el software.
- **Datos de configuración:** Configuraciones necesarias para que el software funcione correctamente.

¿Cómo es un buen software?

Características del producto	Descripción
Mantenimiento	El software debe escribirse de tal forma que pueda evolucionar para satisfacer las necesidades cambiantes de los clientes. Éste es un atributo crítico porque el cambio del software es un requerimiento inevitable de un entorno empresarial variable.
Confiabilidad y seguridad	La confiabilidad del software incluye un rango de características que abarcan fiabilidad, seguridad y protección. El software confiable no tiene que causar daño físico ni económico, en caso de falla del sistema. Los usuarios malintencionados no deben tener posibilidad de acceder al sistema o dañarlo.
Eficiencia	El software no tiene que desperdiciar los recursos del sistema, como la memoria y los ciclos del procesador. Por lo tanto, la eficiencia incluye capacidad de respuesta, tiempo de procesamiento, utilización de memoria, etcétera.
Aceptabilidad	El software debe ser aceptable al tipo de usuarios para quienes se diseña. Esto significa que necesita ser comprensible, utilizable y compatible con otros sistemas que ellos usan.

¿Qué es la ingeniería del software?



¿Qué es la Ingeniería del Software?

La **Ingeniería de Software** es una rama de la ingeniería dedicada al **diseño, desarrollo, mantenimiento y gestión de sistemas de software** a gran escala. Esta disciplina abarca **todas las fases del ciclo de vida del software**, desde la especificación inicial de los requisitos hasta su mantenimiento y evolución tras la implementación.

La ingeniería de software se enfoca en la producción de sistemas **robustos, eficientes, fiables y escalables**, utilizando una combinación de principios **científicos, métodos formales** y prácticas de **gestión de proyectos**.

El éxito en la ingeniería de software depende de un enfoque estructurado, que incluye el uso de **herramientas y técnicas especializadas** para garantizar la calidad, seguridad y escalabilidad del producto final.

Objetivos de la Ingeniería de Software

- **Calidad:** Garantizar que el software cumpla con altos estándares de calidad y sea capaz de responder a las necesidades del usuario.
- **Mantenibilidad:** Crear software que pueda adaptarse y evolucionar fácilmente ante cambios en los requisitos o el entorno.
- **Fiabilidad:** Desarrollar sistemas que minimicen los errores y sean capaces de gestionar fallos de manera controlada.
- **Eficiencia:** Optimizar el uso de recursos (memoria, procesamiento) sin comprometer la funcionalidad.
- **Escalabilidad:** Asegurar que el software pueda expandirse o modificarse para manejar mayores volúmenes de trabajo o usuarios.

¿Cómo alcanza la IS estos objetivos?

La **ingeniería de software** aplica una combinación de **disciplinas, prácticas y herramientas** para garantizar que el software cumpla con los más altos estándares de calidad, fiabilidad y eficiencia, minimizando al mismo tiempo costos y riesgos en proyectos complejos.

Incluye diversas disciplinas y subcampos, como el **diseño y arquitectura**, la **gestión de proyectos**, las **pruebas de software**, y el **desarrollo ágil**, entre otros. Su práctica utiliza tanto **herramientas automatizadas** como **técnicas avanzadas** para asegurar la calidad y reducir los costos y riesgos asociados a proyectos grandes y complejos.

Con la combinación de estas **disciplinas, herramientas y técnicas**, la ingeniería de software asegura que los objetivos de **calidad, eficiencia, fiabilidad y escalabilidad** se logren de manera controlada y a un costo óptimo. De esta manera, se reducen los riesgos en proyectos grandes y complejos, entregando software que cumple con las expectativas tanto del equipo de desarrollo como del usuario final.

¿Cuál es el producto final?

El producto final de la ingeniería de software varía según la perspectiva desde la que se mire:

Desde el punto de vista del ingeniero de software: El producto final es el conjunto completo de componentes de software que incluyen:

- Programas o aplicaciones que ejecutan las funcionalidades requeridas.
- Datos y archivos de configuración que permiten que el sistema funcione correctamente.
- Documentación técnica y de usuario, esencial para su mantenimiento y uso.
- Entregables adicionales como pruebas automatizadas, scripts de instalación y manuales de despliegue.

Este enfoque considera que el sistema debe ser escalable, mantenible y modificable para soportar cambios futuros y ser fácilmente adaptable.

¿Cuál es el producto final?

El producto final de la ingeniería de software varía según la perspectiva desde la que se mire:

Desde la perspectiva del usuario: El producto final no es el software en sí mismo, sino la información y los servicios que el software entrega. Para el usuario, lo que importa es:

- La utilidad que el software proporciona en su vida cotidiana o laboral.
- La experiencia de uso: accesibilidad, rapidez, confiabilidad.
- Cómo este software le permite resolver problemas o mejorar procesos, ya sea en el ámbito personal, empresarial o social.

Importancia de la Ingeniería de Software

La **Ingeniería de Software es vital** para la sociedad moderna porque el software está en el centro de casi todas las infraestructuras, industrias y dispositivos que usamos a diario. Sin un enfoque disciplinado y estructurado para su creación, los sistemas informáticos serían inseguros, ineficientes y poco confiables.

- Controla infraestructuras clave como energía, comunicaciones y transporte.
- Está presente en productos electrónicos, desde electrodomésticos hasta satélites.
- La ingeniería de software permite crear sistemas complejos y fiables que afectan cada aspecto de nuestras vidas.

La ingeniería de software es **responsable de garantizar que los sistemas complejos sean confiables, seguros y capaces de evolucionar** con el tiempo. Esto es fundamental en una sociedad donde los fallos de software pueden tener **consecuencias catastróficas**.

Orígenes de la Ingeniería de Software

El término **“Ingeniería de Software”** fue acuñado en **1968** durante una **conferencia de la OTAN** en Garmisch, Alemania. La conferencia se convocó en respuesta a la llamada **“crisis del software”**, una situación en la que los proyectos de software sufrían retrasos significativos, sobrecostos y fallos generalizados.

Crisis del Software: Durante los años 60, la industria del software experimentaba problemas graves:

- Proyectos a gran escala fallaban en cumplir plazos y presupuestos.
- Baja fiabilidad: Muchos sistemas no eran fiables ni mantenibles.
- Creciente complejidad: Los sistemas de software se volvían cada vez más complejos, mientras que las herramientas y métodos para gestionarlos eran rudimentarios.

Orígenes de la Ingeniería de Software

Cambio de enfoque: La conferencia propuso tratar el desarrollo de software como una disciplina de ingeniería formal, aplicando principios de la ingeniería tradicional (como la planificación, el control de calidad y la validación) a la creación de sistemas de software.

Este evento sentó las bases para el desarrollo de metodologías más sistemáticas y controladas para el desarrollo de software:

- El surgimiento de modelos de desarrollo como el modelo en cascada.
- El impulso para crear herramientas de apoyo como los lenguajes de programación estructurada y métricas para medir la calidad del software.

Este enfoque marcó un punto de inflexión en la historia del software, reconociendo que la creación de software no solo debía centrarse en la programación, sino también en la gestión, diseño y mantenimiento a largo plazo de sistemas complejos.

Evolución de la Ingeniería de Software

Décadas de los 70 y 80:

- **Programación estructurada:** Mejora la legibilidad y mantenibilidad del código.
- **Orientación a objetos:** Introducción de lenguajes como **Smalltalk** y **C++** que promueven la reutilización mediante clases y herencia.
- **Herramientas y notaciones:** Primeras herramientas CASE y la estandarización de diagramas para visualizar sistemas complejos.

Décadas recientes:

- **Metodologías ágiles:** Enfoque iterativo con entregas rápidas y respuesta continua a cambios (Scrum, XP).
- **Desarrollo dirigido por pruebas (TDD):** Asegura calidad y minimiza errores desde el inicio del desarrollo.
- **Reutilización de componentes:** Uso de librerías, frameworks y código abierto para acelerar el desarrollo.

Enfoque actual:

- **Sistemas distribuidos:** Arquitecturas de microservicios y nubes permiten escalabilidad y alta disponibilidad.
- **Escalabilidad:** Plataformas como AWS y Google Cloud posibilitan crecimiento flexible y sin interrupciones.

¿Qué es la arquitectura del software?



¿Qué es la Arquitectura de Software?

La **arquitectura de software** es la estructura de un sistema de software, donde se definen sus componentes, sus relaciones y cómo interactúan entre sí. Incluye decisiones sobre:

- **Componentes del sistema:** Unidades de código independientes o módulos.
- **Relaciones entre componentes:** Cómo se comunican los diferentes módulos o capas.
- **Patrones arquitectónicos:** Soluciones recurrentes que se utilizan para estructurar un sistema, como el patrón de capas o el patrón de cliente-servidor.

Es el marco general sobre el cual se desarrollan y organizan los componentes de un software, guiando tanto el desarrollo como la evolución del sistema. Su objetivo es el desarrollo de **software de calidad y mantenible**.

Diferencia entre Ingeniería de Software y Arquitectura de Software

Ingeniería de Software es una disciplina más amplia que cubre todo el ciclo de vida del software, desde su concepción hasta su mantenimiento, incluyendo:

- **Especificación** de requisitos.
- **Diseño, desarrollo y validación** del sistema.
- **Gestión de proyectos** de software y herramientas de desarrollo.

Arquitectura de Software, en cambio, es un subconjunto de la ingeniería de software que se centra en la estructura del sistema y en las decisiones de diseño que guían el desarrollo:

- Se preocupa por la **organización de componentes** y su interacción.
- Involucra la elección de **patrones arquitectónicos** y **atributos de calidad** como rendimiento y escalabilidad.

Diseño vs Arquitectura del software

Existe mucha confusión sobre la diferencia entre diseño de software y arquitectura del software. ¿Qué es diseño? ¿Qué es arquitectura? ¿Cuáles son las diferencias?

The low-level details and the high-level structure are all part of the same whole. They form a continuous fabric that defines the shape of the system. You can't have one without the other; indeed, no clear dividing line separates them. There is simply a continuum of decisions from the highest to the lowest levels.

Clean Architecture – Robert C. Martin

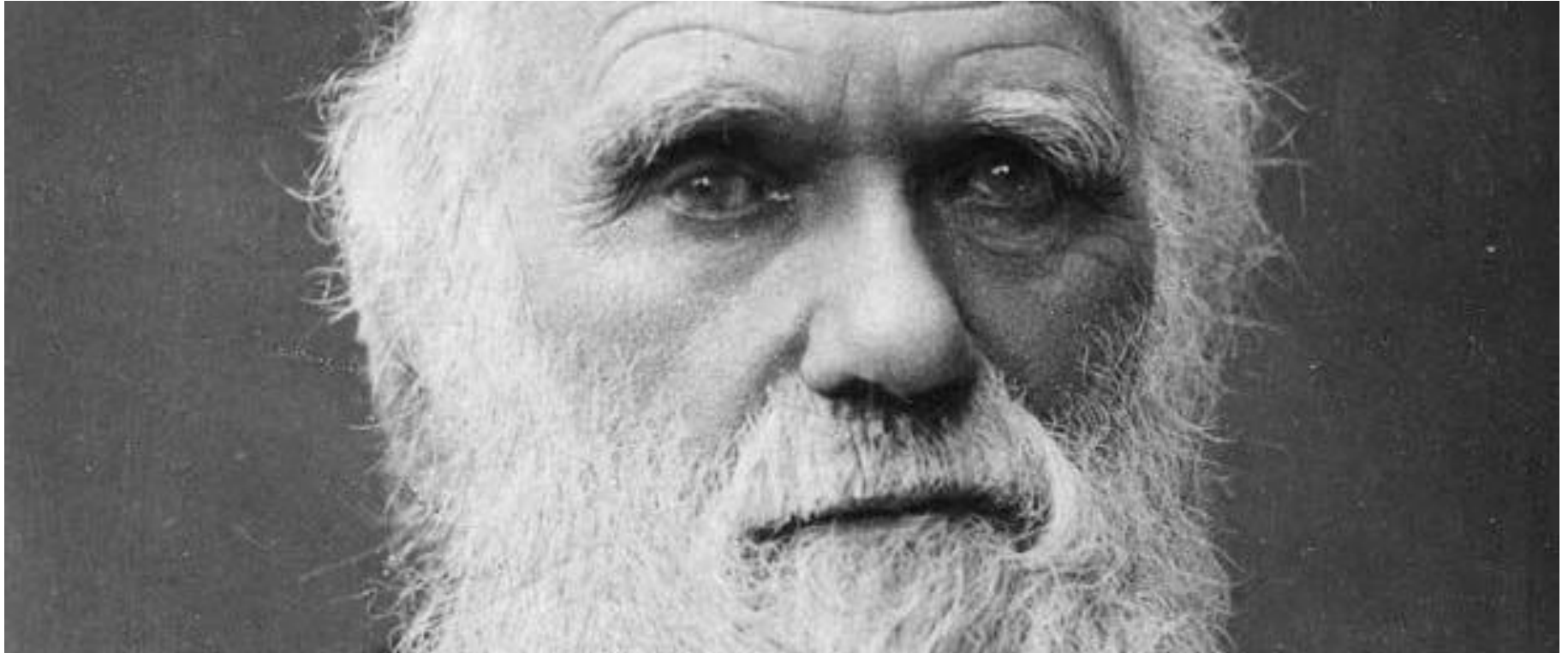
Aspectos clave

Pregunta	Respuesta
¿Qué es software?	Programas de cómputo y documentación asociada. Los productos de software se desarrollan para un cliente en particular o para un mercado en general.
¿Cuáles son los atributos del buen software?	El buen software debe entregar al usuario la funcionalidad y el desempeño requeridos, y debe ser sustentable, confiable y utilizable.
¿Qué es ingeniería de software?	La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software.
¿Cuáles son las actividades fundamentales de la ingeniería de software?	Especificación, desarrollo, validación y evolución del software.
¿Cuál es la diferencia entre ingeniería de software y ciencias de la computación?	Las ciencias de la computación se enfocan en teoría y fundamentos; mientras la ingeniería de software se enfoca en el sentido práctico del desarrollo y en la distribución de software.

Aspectos clave

Pregunta	Respuesta
¿Cuáles son los principales retos que enfrenta la ingeniería de software?	Se enfrentan con una diversidad creciente, demandas por tiempos de distribución limitados y desarrollo de software confiable.
¿Cuáles son los costos de la ingeniería de software?	Aproximadamente 60% de los costos del software son de desarrollo, y 40% de prueba. Para el software elaborado específicamente, los costos de evolución superan con frecuencia los costos de desarrollo.
¿Cuáles son los mejores métodos y técnicas de la ingeniería de software?	Aun cuando todos los proyectos de software deben gestionarse y desarrollarse de manera profesional, existen diferentes técnicas que son adecuadas para distintos tipos de sistema. Por ejemplo, los juegos siempre deben diseñarse usando una serie de prototipos, mientras que los sistemas críticos de control de seguridad requieren de una especificación completa y analizable para su desarrollo. Por lo tanto, no puede decirse que un método sea mejor que otro.

Tipos de software



Clasificación de Sistemas de Software

- **Sistemas embebidos:** Controlan dispositivos de hardware.
- **Sistemas de información:** Gestionan grandes volúmenes de datos.
- **Sistemas en tiempo real:** Responden a eventos en tiempo real.
- **Sistemas de entretenimiento:** Enfocados en la interacción con el usuario.
- **Sistemas distribuidos:** Varias computadoras trabajan juntas para ofrecer una solución unificada

Sistemas embebidos

Los sistemas embebidos son sistemas de software integrados en dispositivos de hardware, diseñados para controlar su funcionamiento. Estos sistemas están presentes en dispositivos cotidianos, desde electrodomésticos hasta automóviles y aviones. Características clave:

- **Función específica:** Diseñados para ejecutar tareas muy concretas dentro de un dispositivo.
- **Alta fiabilidad:** Su operación es crítica, ya que fallos en el software pueden comprometer el dispositivo en sí.
- **Requisitos estrictos de tiempo y recursos:** Usan pocos recursos de memoria y procesamiento y deben funcionar en tiempo real.

Ejemplos: Sistemas de control de frenos ABS en automóviles, controladores de electrodomésticos como microondas, o software de teléfonos móviles.

Sistemas de Información

Los sistemas de información están diseñados para recopilar, almacenar, procesar y distribuir grandes volúmenes de datos. Estos sistemas son fundamentales para empresas y organizaciones, donde la gestión eficiente de la información es crucial para la toma de decisiones y el control operativo.

- Almacenamiento y recuperación de datos: Administran grandes bases de datos para ofrecer información accesible.
- Integración de procesos: Permiten la integración de diversos procesos de negocio.
- Escalabilidad: Pueden expandirse para manejar mayores volúmenes de datos sin perder rendimiento.

Ejemplos: Sistemas ERP (Enterprise Resource Planning), sistemas de gestión de bases de datos, y sistemas CRM (Customer Relationship Management).

Sistemas en tiempo real

Los sistemas en tiempo real están diseñados para responder a eventos o entradas en un tiempo predefinido, lo cual es crucial para muchas aplicaciones de control y monitoreo.

- **Respuestas rápidas:** La latencia debe ser mínima y las respuestas deben darse en un tiempo limitado.
- **Críticos para la seguridad:** Frecuentemente usados en contextos donde el error puede tener graves consecuencias.
- **Control de procesos físicos:** Monitorean y controlan procesos físicos en tiempo real.

Ejemplos: Sistemas de control de tráfico aéreo, sistemas de monitoreo de signos vitales en hospitales, o sistemas de control industrial automatizado.

Sistemas de entretenimiento

Los sistemas de entretenimiento están diseñados principalmente para interactuar con los usuarios, ofreciendo experiencias visuales, auditivas o táctiles. En esta categoría entran los juegos y aplicaciones interactivas, donde la experiencia del usuario es el factor clave.

- Interacción constante con el usuario: El diseño y la fluidez de la interfaz son cruciales.
- Alta demanda gráfica y de rendimiento: Especialmente en los videojuegos, que requieren renderización de gráficos en tiempo real.
- Experiencia inmersiva: La respuesta rápida del sistema y la calidad de la interacción definen la satisfacción del usuario.

Ejemplos: Videojuegos, plataformas de streaming de música o video, aplicaciones de realidad aumentada.

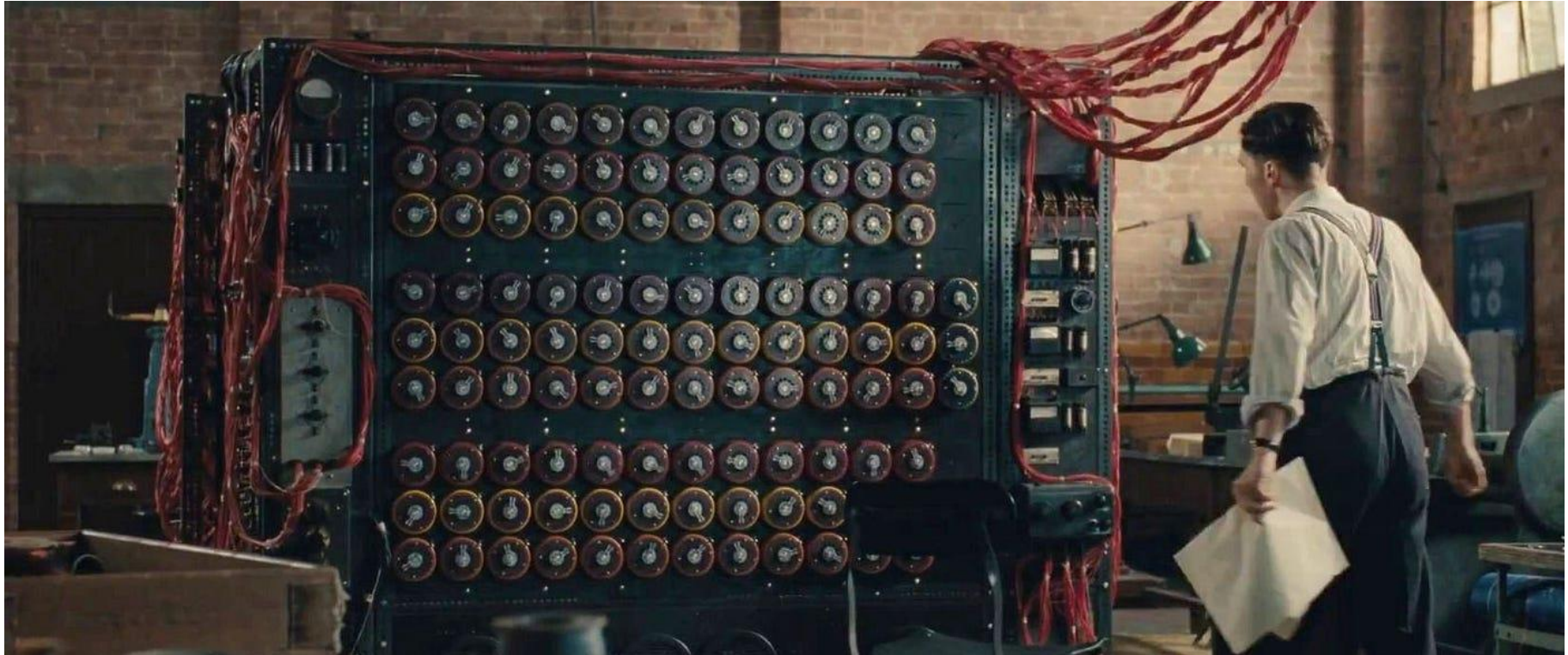
Sistemas distribuidos

Los sistemas distribuidos son aquellos en los que varias computadoras, a menudo geográficamente dispersas, trabajan juntas para ofrecer una solución unificada. Estos sistemas son clave en aplicaciones que requieren escalabilidad y alta disponibilidad.

- **Distribución de tareas:** Las tareas se distribuyen entre múltiples nodos para maximizar la eficiencia y el rendimiento.
- **Tolerantes a fallos:** La redundancia de sistemas permite mantener la operación aun cuando algunos nodos fallen.
- **Comunicaciones a través de redes:** La coordinación entre nodos se realiza a través de redes locales o Internet.

Ejemplos: Sistemas de computación en la nube, redes de contenido distribuido (CDN), o servicios web distribuidos.

Paradigmas y lenguajes de programación



Paradigmas de programación

La **arquitectura** del software comienza por el **código**, así que comenzaremos estudiando lo que sabemos sobre código.

Un **paradigma de programación** es un enfoque o estilo que guía la forma en que los programadores escriben y organizan el código.

Los paradigmas son formas de programar, no vinculadas a un lenguaje y no son mutuamente excluyentes; muchos lenguajes de programación modernos admiten múltiples paradigmas.

Un paradigma indica qué estructuras de código usar y cuando usarlas.

Paradigmas de programación

- **Programación Estructurada**
 - **Descripción:** Se basa en dar órdenes secuenciales al ordenador para que realice acciones específicas.
 - **Características:** Uso de variables, estructuras de control (bucles, condicionales) y asignación de estado.
 - **Ejemplo:** C, Python (imperativo).
- **Programación Orientada a Objetos (POO)**
 - **Descripción:** Organiza el código en objetos que contienen tanto datos (atributos) como comportamiento (métodos).
 - **Características:** Encapsulamiento, herencia, polimorfismo y reutilización de código.
 - **Ejemplo:** Java, C++, Python.
- **Programación Funcional**
- **Programación Declarativa**
- **Programación Lógica**
- **Programación Concurrente**

Lenguajes de Programación

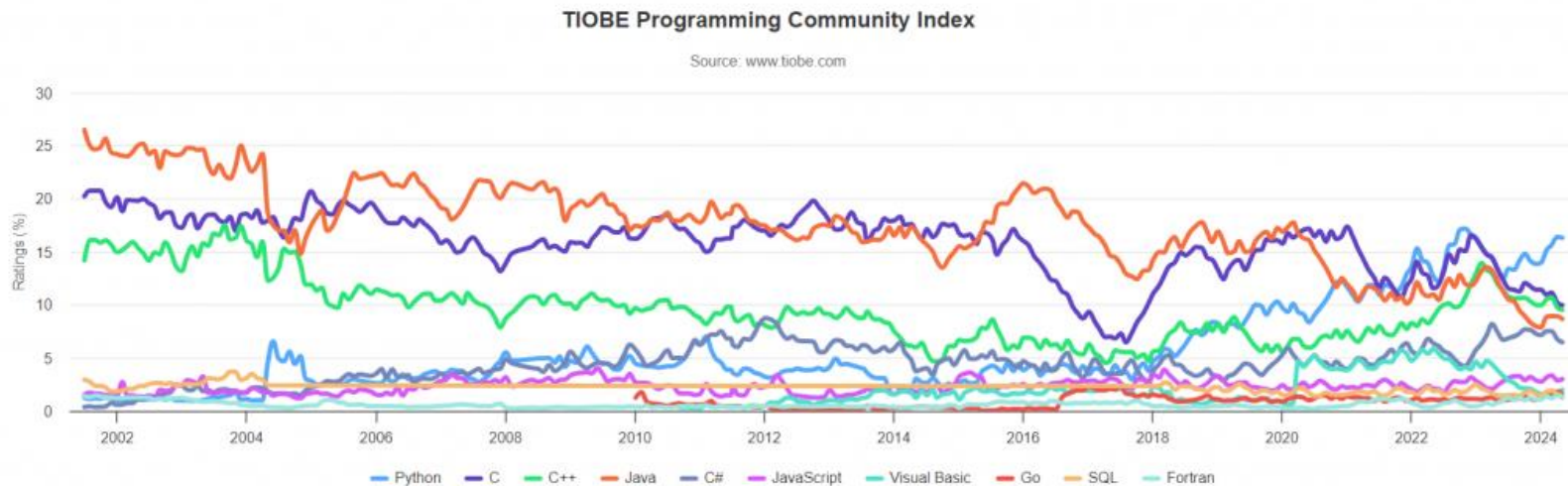
La elección del lenguaje puede afectar la flexibilidad y el rendimiento del sistema.

- **Lenguajes orientados a objetos (Java, C++, Python):** Adecuados para arquitecturas modulares y reutilizables.
- **Lenguajes funcionales (Haskell, Lisp):** Enfoque en la evaluación de funciones matemáticas.
- **Lenguajes de scripting (Python, JavaScript):** Uso frecuente en prototipos rápidos y sistemas web.

Lenguajes de programación



Lenguajes de programación



Asignatura

Arquitectura del Software

Profesor

Yago Fontenla Seco

{yago.fontenla1@uie.edu}