

Tema 1: Introducción a la Ingeniería del Software

BLOQUE I: INTRODUCCIÓN Y PARADIGMAS
DE DESARROLLO EN INGENIERÍA DEL SOFTWARE

Ingeniería del Software
Grado en Ingeniería Informática
Curso 2024/2025

David Cáceres Gómez



Índice

1. ¿Qué es el Software?	2
1.1. El desarrollo software	2
1.2. Ámbito de la ingeniería del software	4
2. Evolución Histórica	6
3. Naturaleza y Problemas del Desarrollo de Software	9
3.1. La crisis del software	9
3.2. Curva de fallos hardware y software	10
3.3. Mitos del software	11
4. La Ingeniería del Software	12
4.1. Definición de ingeniería del software	12
4.2. Aspectos esenciales	13
4.3. La Ingeniería del Software	13
4.4. Estandarización de la IS	14
4.4.1. Entidades de estandarización	14
4.4.2. Estándares y necesidades de la IS	15
5. Conceptos de la Ingeniería del Software	16
6. Principios de la Ingeniería del Software	18
Referencias	20

1. ¿Qué es el Software?

Antes de empezar con el estudio de la ingeniería del software, debemos ponernos en contexto y definir qué es el software, qué actividades relacionadas con el software cubre la ingeniería del software y cómo hemos llegado, como industria, a la conclusión de que la ingeniería es el enfoque adecuado para estas actividades.

Software

Denominamos *software* a todo aquello intangible (no físico) que hay en un ordenador, incluyendo el conjunto de programas informáticos que indican la secuencia de instrucciones que un ordenador debe ejecutar durante su funcionamiento (también denominado código) y el resto de los datos que este ordenador manipula y almacena [1].

Hardware

Por oposición, denominamos *hardware* al conjunto de componentes físicos de un ordenador. Este hardware ofrece una serie de instrucciones que el ordenador es capaz de ejecutar cuando ejecuta un programa [1].

1.1. El desarrollo software

El desarrollo de software es el acto de producir o crear software.

A pesar de que el desarrollo de software incluye la programación (la creación del código fuente), usamos el término **desarrollo de software** de una manera más amplia para referirnos al conjunto de actividades que nos llevan desde una determinada idea sobre lo que queremos hasta el resultado final del software.

Actividades del desarrollo:

- Programación
- Compilación
- El estudio y la documentación de las necesidades de los usuarios
- El mantenimiento del software una vez se empieza a usar
- La coordinación del trabajo en equipo de las diferentes personas que intervienen en el desarrollo

- La redacción de manuales y ayudas de uso para los usuarios
- Verificación
- etc.

Como en otras áreas, al crecer la complejidad, y el coste e impacto del software, se empiezan a aplicar **técnicas de ingeniería** al desarrollo software.

El software presenta una serie de peculiaridades (en comparación con otros *productos*):

- El producto software es enteramente **conceptual**.
- No tiene propiedades físicas como peso, color o voltaje, y, en consecuencia no está sujeto a leyes físicas o eléctricas.
- Su naturaleza conceptual crea una **distancia intelectual** entre el software y el problema que el software resuelve.
- Difícil para una persona que entiende el problema entender el sistema software que lo resuelve.
- Para probar es necesario disponer de un sistema físico.
- El mantenimiento no es sólo una substitución de componentes.

Hoy en día, aún existe debate sobre si el desarrollo de software es un arte/arte-sanía o una disciplina de ingeniería. Obviamente asumiremos lo segundo. En todo caso, **desarrollar software de calidad** con el mínimo coste posible ha resultado ser una **actividad**, en general, muy **compleja**.



Figura 1: Evolución del Software hacia la Ingeniería [2]

1.2. Ámbito de la ingeniería del software

Inicio de la informática

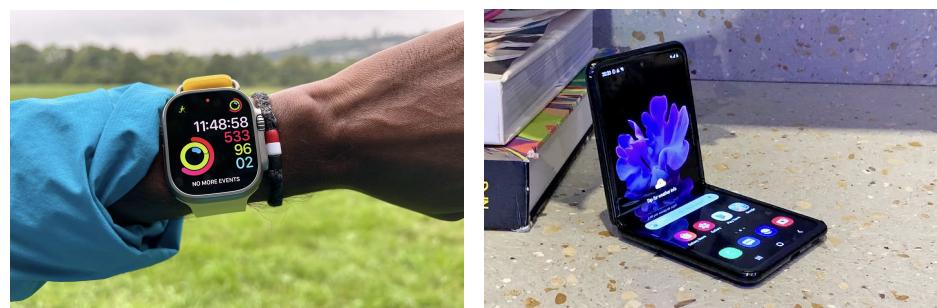
Se crearon grandes ordenadores como el ENIAC dedicados a cálculos matemáticos (computadoras).



Figura 2: Primer ordenador del mundo: el ENIAC [3]

Actualidad

En estos tiempos, hay una tendencia de dispositivos electrónicos cada vez más pequeños y con más potencia (ordenadores, tablets, relojes...)



(a) Reloj inteligente [4]

(b) Móvil plegable [5]

Figura 3: Dispositivos de la actualidad

Según el ámbito las **diferencias** son **abismales**:

- Red social: actualizar el software consiste en desplegar una nueva versión en una red de servidores.
- Sistema de navegación de un coche: actualizar el software requiere el coste de llevar todos los coches a talleres.

A grandes rasgos, estos son ámbitos del software, que puede pertenecer a más de una categoría:

- **Software de sistemas:** Dan servicio a otros programas. **No** tienen un **propósito específico**. Por ejemplo: un servidor de bases de datos no se diseña pensando si almacenará los resultados de la Quiniela o los datos de un hospital. Son programas escritos para dar servicio a otros programas, como los sistemas operativos o los compiladores. Este tipo de programas suelen interactuar directamente con el hardware, de manera que sus usuarios no son los usuarios finales que usan el ordenador, sino otros programadores.
- **Software de aplicación:** Son programas independientes que resuelven una necesidad **específica**, normalmente de una organización. Por ejemplo, sistema de citas de un centro de salud. Pueden ser desarrollados a medida (para un único cliente) o como software de propósito general (se intentan cubrir las necesidades de varios clientes y es habitual que éstos utilicen sólo un subconjunto de la funcionalidad total).
- **Software científico y de ingeniería:** Muy enfocados al cálculo y a la simulación, se caracterizan por la utilización de algoritmos y modelos matemáticos complejos.
- **Software empotrado:** Forma parte de algún dispositivo. Ejemplos: sistemas de frenado, lavadoras...
- **Aplicaciones web:** Se caracterizan por unificar fuentes de datos y diferentes servicios en entornos altamente distribuidos. Es en este ámbito donde las metodologías ágiles son muy populares.
- **Software de inteligencia artificial (IA):** Es un programa que realiza tareas que requieren inteligencia humana, como el aprendizaje, la toma de decisiones y el procesamiento del lenguaje. Utiliza algoritmos como el aprendizaje automático y el procesamiento de datos para mejorar su rendimiento. Sus aplicaciones incluyen asistentes virtuales, diagnósticos médicos, y vehículos autónomos, entre otros.

En general, las técnicas de ingeniería del software están enfocadas al desarrollo de **software de aplicación**, y en concreto a **sistemas de información**.

Sistemas de información

Un sistema de información es un conjunto de elementos (personas, datos, técnicas y recursos materiales, fundamentalmente informáticos) orientados al tratamiento y administración de datos e información, organizados y listos para su uso posterior, generados para cubrir una necesidad o un objetivo.

El software para sistemas de información es un tipo de software que gestiona una cierta información mediante un sistema gestor de bases de datos y soporta una serie de actividades humanas dentro del contexto de un sistema de información.

2. Evolución Histórica

Contexto histórico años 60-70:

- Pocas computadoras:
 - Grandes computadoras o *mainframes*
 - Muy pocos y muy caros
 - Poca variedad de software → muy específico y desarrollado por el fabricante. Obviamente era difícil programar por libre.
- Se desarrolla software **artesanal**:
 - El negocio estaba en el hardware, no se le prestaba demasiada atención al software
 - Se dispone del código fuente y los desarrolladores de software compartían libremente sus programas unos con otros con ánimo constructivo (más adelante en la década de los 80 esto sería uno de los detonantes para el movimiento del **software libre**)
 - El desarrollo de software no se gestionaba según una planificación y era prácticamente imposible predecir los costes y el tiempo de desarrollo. Pero ya se aplicaban algunas técnicas de reutilización, como la programación modular.

Originalmente, el software era un producto gratuito que se incluía al comprar hardware y era desarrollado, principalmente, por las compañías fabricantes del hardware. Unas pocas compañías desarrollaban software a medida, pero no existía el concepto de software empaquetado, como producto.

En los primeros años estaban enfocados en desarrollar el hardware y reducir los costes de procesamiento y almacenamiento.

Algunas **referencias útiles** [6] para comprender cuáles eran los conocimientos establecidos para el desarrollo de software en **1968** son:

- 1962 primer algoritmo para búsquedas binarias. En 1962 se publicó el primer algoritmo para búsquedas binarias.
- 1966 fundación para la eliminación de “GoTo” y la creación de la programación estructurada. C. Böhm y G. Jacopini publicaron en 1966 el documento que creaba una fundación para la eliminación de “GoTo” y la creación de la programación estructurada.
- En 1968 los programadores se debatían entre el uso de la sentencia GoTo, y la nueva idea de programación estructurada; ese era el caldo de cultivo en el que Edsger Dijkstra escribió su famosa carta “GoTo Statement Considered Harmful” en 1968.
- 1974 Primera publicación sobre programación estructurada, por Larry Constantine, Glenford Myers y Wayne Stevens.
- 1976 primer libro sobre métrica de software por Tom Gilb.
- 1976 primer libro sobre análisis de requisitos.

Contexto histórico década de los 80:

- Desarrollo de la microelectrónica
- Mayor potencia de cálculo y reducción de costes
- Aparecen lenguajes con nociones de programación estructurada
- Se comienza a dar importancia al diseño (herramientas CASE) y los costes
- Aparece el movimiento del **software libre**, una iniciativa social y filosófica que promueve la libertad de los usuarios de computadoras para ejecutar, estudiar, compartir y modificar el software. Está basado en la idea de que el software debe considerarse un bien público, accesible y controlado por sus usuarios, en lugar de estar restringido por licencias propietarias.

En esta década, **aumentan** los problemas del desarrollo de software con la sobreexploitación del potencial del hardware y la incapacidad de atender a la demanda y de mantener el software existente.

Contexto histórico década de los 90:

- Se impone el desarrollo orientado a objetos (nace Java)
- Creciente atención a la arquitectura del software (componentes)

- Creciente uso de arquitecturas distribuidas (cliente/servidor)
- Primeros sistemas de control de versiones (CVS)

Contexto histórico década de los 2000:

- Aparición de los entornos de desarrollo integrados (IDEs)
- Aunque se impone UML, decaen las herramientas CASE
- Nuevos lenguajes de programación: C#, Scala, Go...

Contexto histórico década de los 2010:

- Auge de los sistemas web y el software como servicio
- Metodologías ágiles de desarrollo
- Irrumpe la industria móvil
- Crecimiento de los sistemas empresariales y gubernamentales

La ingeniería del software en la actualidad:

- Heterogeneidad de sistemas: web, apps, sistemas ciber-físicos, videojuegos, etc.
- Volatilidad de las tecnologías (mayor competencia)
- La reutilización cobra especial importancia
- Evolución constante de los lenguajes (se han creado cerca de 9000)
- Escalabilidad y disponibilidad global
- Herramientas de soporte y automatización en muchas de sus fases:
 - *Frameworks* de desarrollo
 - Desarrollo continuo: CI/CD (*Continuous integration / Continuous deployment*)
 - Automatización de pruebas (*Selenium, Cypress*)
 - Asistentes inteligentes de programación (*Copilot, Ponicode*)

3. Naturaleza y Problemas del Desarrollo de Software

El software, como un elemento lógico, se desarrolla en lugar de fabricarse, ya que su creación implica un proceso de diseño y programación, no una producción en serie. La calidad del software depende del diseño y no de la materia prima, siendo los costos más importantes aquellos relacionados con el desarrollo y mantenimiento, como la mano de obra y gestión. A lo largo del tiempo, el software tiende a “deteriorarse” debido a cambios y actualizaciones constantes que pueden introducir errores o complejidad. Al inicio de su historia, el software se desarrollaba a medida, ya que no existían componentes reutilizables ni herramientas de desarrollo estandarizadas.

3.1. La crisis del software

Este problema se identificó por primera vez en 1968, año en el que la organización OTAN (Organización del Tratado del Atlántico Norte) desarrolló la primera conferencia sobre desarrollo de software, y en la que se acuñaron los términos “crisis del software” para definir a los problemas que surgían en el desarrollo de sistemas software, e “ingeniería del software” para describir el conjunto de conocimientos que existían en aquel estado inicial.

La **crisis del software** se refiere a la dificultad en escribir programas libres de defectos, fácilmente comprensibles, y que sean verificables.

Aparecieron varios síntomas de crisis en el desarrollo de software:

- **Productividad:** de los desarrolladores baja en relación a la demanda.
- **Expectativas:** los sistemas no responden a las expectativas
- **Fiabilidad:** Los programas fallan a menudo
- **Calidad:** No es adecuada
- **Costes:** Difíciles de predecir, a menudo sobrepasan lo esperado
- **Mantenimiento:** Modificación del software costosa y compleja
- **Plazos:** no se cumplen
- **Portabilidad:** Difícil cambiar de plataforma
- **Eficiencia:** No hay aprovechamiento óptimo de recursos

Como consecuencia, la productividad y la calidad del software disminuyeron. Como solución, en la conferencia de la OTAN se propuso aplicar principios de ingeniería en la construcción de sistemas informáticos.

3.2. Curva de fallos hardware y software

En la Figura 4a se puede ver la “curva en U” del hardware, que muestra una alta tasa de fallos al principio de su vida debido a defectos de diseño o manufactura. Esta tasa disminuye a un nivel estable y bajo una vez corregidos los defectos iniciales, pero con el tiempo aumenta nuevamente debido al desgaste acumulativo causado por factores como suciedad, vibración y temperaturas extremas. En resumen, el hardware se desgasta con el tiempo, lo que provoca un aumento gradual en la tasa de fallas.

Por otro lado, la Figura 4b representa la “curva idealizada” del software, donde las tasas de fallos son inicialmente altas debido a defectos ocultos que se corigen, estabilizándose luego en un nivel bajo. Sin embargo, la “curva real” del software muestra que la tasa de fallos puede aumentar abruptamente cada vez que se realizan cambios o actualizaciones, ya que el software no se desgasta físicamente, pero sí se deteriora con el tiempo debido a la introducción de nuevos errores. A diferencia del hardware, no hay refacciones para el software, haciendo que su mantenimiento sea más complejo.

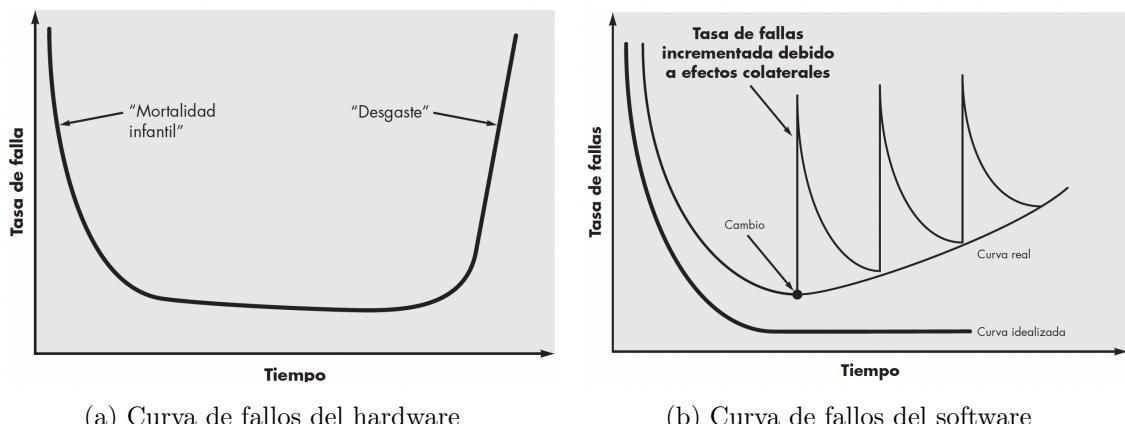


Figura 4: Curva de fallos [7]

3.3. Mitos del software

Mitos de los gestores:

- **Uso de estándares de otras ingenierías:** Los gestores a menudo creen que las metodologías y estándares de otras disciplinas de ingeniería (como la construcción o la manufactura) son aplicables al desarrollo de software de manera directa. Sin embargo, el software es un elemento lógico, no físico, y su desarrollo sigue ciclos más flexibles e iterativos, por lo que necesita métodos específicos.
- **Uso de herramientas:** Los gestores a menudo creen que el simple uso de herramientas de software (como gestores de proyectos, entornos de desarrollo integrados, o metodologías ágiles) por sí solo garantizará el éxito del proyecto. Sin embargo, las herramientas son solo tan eficaces como el equipo que las utiliza y deben estar integradas en una planificación adecuada. El éxito depende más de la correcta implementación de las metodologías y de la capacidad de los equipos para aprovechar las herramientas, no simplemente de tenerlas disponibles.
- **Mala planificación:** un aumento de programadores acelera el proyecto; Existe la creencia errónea de que añadir más programadores a un proyecto atrasado acelerará su finalización. En realidad, esto suele generar retrasos adicionales debido al tiempo necesario para integrar a los nuevos miembros y coordinar su trabajo con el equipo existente.

Mitos de los desarrolladores:

- **Programa funcionando = fin del trabajo;** Muchos desarrolladores piensan que una vez que el programa funciona, el trabajo ha terminado. Sin embargo, el software necesita pruebas exhaustivas, mantenimiento, optimización y documentación, lo que significa que su desarrollo continúa incluso después de que el programa es funcional.
- **Calidad = el programa se ejecuta sin errores;** Otro mito común es creer que un software de calidad es aquel que no presenta errores durante su ejecución. Sin embargo, la verdadera calidad del software incluye aspectos como la facilidad de uso, la seguridad, la escalabilidad y la capacidad de ser mantenido y modificado eficientemente.
- **Entrega al cliente = programa funcionando;** Muchos desarrolladores asumen que entregar el programa al cliente es sinónimo de que está completamente operativo. Pero la entrega es solo una fase; se requiere capacitación, pruebas adicionales, ajustes, y soporte posterior para asegurar que el software cumpla todas las expectativas del cliente.

Mitos del cliente:

- **Requisitos establecidos como una declaración general de objetivos:** Los clientes suelen creer que una descripción general de sus objetivos es suficiente para que los desarrolladores entiendan y construyan el software adecuado. Sin embargo, los requisitos deben ser específicos y detallados para evitar malentendidos y garantizar que el software cumpla exactamente con sus necesidades.
- **Flexibilidad del software ante los cambios:** Los clientes también tienden a suponer que el software puede ser modificado fácilmente en cualquier etapa del proyecto. Aunque el software es flexible, los cambios significativos en fases avanzadas pueden ser costosos y difíciles de implementar sin afectar la calidad y los plazos de entrega.

4. La Ingeniería del Software

4.1. Definición de ingeniería del software

Definición original

“Establecimiento y uso de principios de ingeniería para obtener software económico que trabaje de forma eficiente en máquinas reales”. Fritz Bauer, 1968 (conferencia OTAN).

Otras definiciones

- “Ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y el estudio de estos enfoques, es decir, la aplicación de la ingeniería al software. Es la aplicación de la ingeniería al software, ya que integra matemáticas, ciencias de la computación y prácticas cuyos orígenes se encuentran en la ingeniería” [8].
- “Disciplina para producir software de calidad desarrollado sobre las agendas y costes previstos y satisfaciendo los requisitos” [9].
- “Disciplina de ingeniería que se ocupa de todos los aspectos de la producción de software, desde las primeras fases de especificación del sistema hasta el mantenimiento del sistema después de su uso” [10].

4.2. Aspectos esenciales

Establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico, fiable, eficiente y que satisfaga las necesidades del usuario.

Disciplina que comprende **todos los aspectos de la producción de software**, desde las etapas iniciales hasta el mantenimiento:

- “disciplina de ingeniería”: aplicación de teorías, métodos y herramientas para solucionar problemas, y teniendo en cuenta restricciones financieras y organizativas.
- “todos los aspectos de producción”: comprende procesos técnicos del desarrollo y actividades como la administración de proyectos, desarrollo de herramientas, métodos y teorías.

No solo es programar, engloba planificar, modelar, diseñar, probar, validar, corregir, mejorar, desplegar, etc. Se encuentra orientada a la solución de problemas y a la adquisición de conocimiento.

4.3. La Ingeniería del Software

Desde 1968 hasta la fecha han sido muchos los esfuerzos realizados por los departamentos de informática de las universidades, y por organismos de estandarización para identificar las causas del problema y definir pautas estándar para la producción y mantenimiento del software.

Los esfuerzos se han encaminado en tres direcciones principales:

- Identificación de los factores clave que determinan la calidad del software.
- Identificación de los procesos necesarios para producir y mantener software.
- Acotación, estructuración y desarrollo de la base de conocimiento necesaria para la producción y mantenimiento de software.

El resultado ha sido la necesidad de profesionalizar el desarrollo, mantenimiento y operación de los sistemas de software, introduciendo métodos y formas de trabajo sistemáticos, disciplinados y cuantificables.

¿Qué es la Ingeniería del Software?

La IS es aplicar el sentido común al desarrollo de sistemas software.

4.4. Estandarización de la IS

4.4.1. Entidades de estandarización

Desde la identificación del fenómeno “crisis del software” se han ido generando normas y estándares. Se consideran como entidades de mayor reconocimiento internacional, por sus trabajos y esfuerzos realizados para la normalización, y reconocimiento de la Ingeniería del Software a: ISO, IEEE-Computer Society y SEI.

Organización Internacional para la Estandarización (ISO)

ISO fue fundada en 1947 y actualmente son miembros 172 países [11]. En 1987, ISO y la Comisión Internacional Electrotécnica (IEC), establecieron un Comité Internacional (JTC1) para las Tecnologías de la Información. La misión del JTC1 es la “estandarización en el campo de los sistemas de tecnologías de la información”, incluyendo microprocesadores y equipos.

Los estándares o instrucciones técnicas más importantes para la Ingeniería del Software:

- ISO/IEC 12207: Information Technology Software Life Cycle Processes.
- ISO/IEC TR 15504: Software Process Improvement Capability Determination.
- ISO 29119: Software Testing.
- ISO/IEC/IEEE 90003 2018 Software engineering Guidelines for the application of ISO 9001 2015 to computer software.

IEEE Computer Society

IEEE es el Instituto de Ingenieros en electricidad y electrónica (*Institute of Electrical and Electronics Engineers*) [12]. Su misión es preservar, investigar y promover la información de las tecnologías eléctricas y electrónicas.

Surgió en 1963 con la fusión del AIEE (Instituto Americano de Ingenieros Eléctricos) y el Instituto de Ingenieros de Radio (IRE). La IEEE Computer Society es una

sociedad integrada en IEEE, formada en la actualidad por más de 100.000 miembros en todo el mundo.

Su finalidad es avanzar en la teoría, práctica y aplicación de las tecnologías de la información. Realiza conferencias, publicaciones, cursos de formación, y desarrolla estándares.

IEEE ha desarrollado estándares para todas las áreas de Ingeniería del Software. Algunos de ellos, correspondientes a las principales áreas específicas de la Ingeniería del Software son:

- IEEE Std. 830 Prácticas recomendadas para las especificaciones de software
- IEEE Std. 1362 Guía para la especificación del documento de requisitos ConOps
- IEEE Std. 1063 Estándar para la documentación de usuario de software
- IEEE Std. 1012 Estándar para la verificación y validación de software
- IEEE Std. 1219 Estándar para el mantenimiento del software

Instituto de Ingeniería del Software (SEI)

Integrado en la Universidad Carnegie Mellon, los trabajos y aportaciones realizadas por el Instituto de Ingeniería del Software [13] son también referente mundial de primer orden, siendo la aportación más significativa los modelos de madurez de las capacidades: CMM y CMMI; que en sus casi 15 años de implantación efectiva en entornos de producción de software han demostrado su efectividad en las dos finalidades que cubren: como marco de referencia para mejora de procesos, y como criterio de evaluación para determinar la madurez, y por tanto fiabilidad de resultados previsibles de una organización de software.

4.4.2. Estándares y necesidades de la IS

- **Definirse a sí misma:** ¿Cuáles son las áreas de conocimiento que la comprenden? → SWEBOK: Software Engineering Body of knowledge
- **Definir los procesos que intervienen en el desarrollo, mantenimiento y operación del software** → ISO/IEC 12207: Procesos del ciclo de vida del software.

- De las mejores prácticas, extraer modelos de cómo ejecutar esos procesos para evitar los problemas de la “crisis del software” → CM-M/CMMI, ISO/IEC TR 15504
- Definir estándares menores para dibujar criterios unificadores en requisitos, pruebas, gestión de la configuración, etc. → IEEE 830-IEEE 1362, ISO/IEC 14764, UML...

5. Conceptos de la Ingeniería del Software

Papel o Rol

Un rol es un conjunto de responsabilidades dentro del proyecto o sistema. Está vinculado a un conjunto de tareas específicas y se asigna a un participante, aunque un mismo participante puede asumir varios roles simultáneamente.

Roles en IS

Los participantes son todas las personas involucradas en el proyecto:

- **Cliente:** encarga y paga el sistema.
- **Equipo de desarrollo:** construyen el sistema (analistas de negocio, arquitectos software, programadores, probadores...).
- **Gerente o director del proyecto:** planifica y calcula el presupuesto, coordina a los desarrolladores y cliente.
- **Usuarios finales:** los que van a utilizar el sistema.

¿Qué hace un ingeniero de software?

Un ingeniero de software se encarga de diseñar, desarrollar, probar y mantener aplicaciones o sistemas de software. Trabaja en todas las etapas del ciclo de vida del software, desde la planificación y el análisis de requisitos hasta la implementación y el mantenimiento. Su trabajo incluye escribir código, depurar programas, optimizar el rendimiento del software y colaborar con otros equipos para garantizar que el software cumpla con las especificaciones del cliente o del proyecto. Además, se asegura de que el software sea escalable, seguro y eficiente.

¿Qué habilidades necesita un ingeniero de software?

Un ingeniero de software necesita una sólida comprensión de lenguajes de programación, algoritmos y estructuras de datos, así como experiencia con bases de datos y metodologías de desarrollo. Debe ser competente en el uso de herramientas de control de versiones y pruebas de software, además de tener habilidades en diseño y arquitectura de software. Es crucial que sea capaz de resolver problemas, comunicar ideas claramente, gestionar su tiempo eficazmente, colaborar bien en equipo y adaptarse a nuevas tecnologías y cambios en el proyecto.

Otros conceptos

■ Sistemas y modelos

- **Sistema:** realidad subyacente
- **Modelo:** cualquier abstracción de la realidad

■ Productos de trabajo o **Entregable**

- Artefacto o elemento que se produce durante el desarrollo (documento, fragmento de software, ...).
- Dos tipos:
 - Producto de trabajo interno: producto para el consumo interno del proyecto (por ejemplo, una revisión de la estructura de la base de datos, resultados de pruebas para el gerente, ...).
 - Entrega: producto de trabajo para un cliente (especificación de requisitos, manual de usuario, producto final, ...).

■ Actividades (o fases):

conjunto de tareas que se realiza con un propósito específico (obtención de requisitos, entrega, administración,...) que pueden componerse de otras actividades.

■ Tareas:

unidad elemental de trabajo que puede ser administrada; consumen recursos, dan como resultado productos de trabajo y dependen de productos de trabajo producidos por otras tareas.

■ Recursos:

bienes que se utilizan para realizar el trabajo:

- Tiempo, equipamiento y recursos humanos.
- Al planificar, el gerente divide el trabajo en tareas y les asigna recursos.

■ Objetivos:

- Principios de alto nivel que se utilizan para guiar el proyecto.

- Definen los atributos realmente importantes del sistema (seguridad, fiabilidad, ...).
- A veces hay conflicto entre objetivos (por ejemplo, seguridad y bajo coste) que aumentan la complejidad del proyecto.

■ **Requerimientos:**

- Características que debe tener el sistema.
- **Requerimiento funcional:** área de funcionalidad que debe soportar el sistema (por ejemplo, proporcionar billetes de tren).
- **Requerimiento no funcional:** restricción que se establece sobre el funcionamiento del sistema (por ejemplo, proporcionar billetes de tren en menos de un segundo).
- **Otras restricciones:** por ejemplo, utilización de un determinado lenguaje, de una determinada plataforma o de un sistema antiguo que el cliente no quiere retirar.
- **Notaciones:** conjunto de reglas gráficas o de texto para representar un modelo (UML, *Unified Modelling Language*), es una notación gráfica orientada a objetos para representar modelos).
- **Métodos:** técnica repetible para resolver un problema específico. Por ejemplo:
 - un algoritmo de ordenación es un método para ordenar elementos en una lista.
 - la administración de la configuración es un método para el seguimiento de los cambios.
- **Metodologías:** colección de métodos para la resolución de una clase de problemas (OMT, metodología de Booch, Catalysis, Proceso Unificado de Desarrollo...).

6. Principios de la Ingeniería del Software

Los principios de la Ingeniería del Software [2] son los siguientes:

- Haz de la calidad la razón de trabajar.
- Una buena gestión es más importante que una buena tecnología.
- Las personas y el tiempo no son intercambiables.

- Seleccionar el modelo de ciclo de vida adecuado.
- Entregar productos al usuario lo más pronto posible.
- Determinar y acotar el problema antes de escribir los requisitos.
- Realizar un diseño.
- Minimizar la distancia intelectual.
- Documentar.
- Las técnicas son anteriores a las herramientas.
- Primero hazlo correcto, luego hazlo rápido.
- Probar, probar y probar (incluye inspecciones).
- Introducir las mejoras y modificaciones con cuidado.
- Asume responsabilidades.
- La entropía del software es creciente.
- La gente es la clave del éxito.
- Nunca dejes que tu jefe o cliente te convenza para hacer mal un trabajo.
- La gente necesita sentir que su trabajo es apreciado.
- La educación continua es responsabilidad de cada miembro del equipo.
- El compromiso del cliente es el factor más crítico en la calidad del software.
- Tu mejor desafío es compartir la visión del producto con el cliente.
- La mejora continua de tu proceso de desarrollo de software es posible y esencial.
- Tener procedimientos escritos de desarrollo de software puede ayudar a crear una cultura compartida de buenas prácticas.
- La calidad es el principal objetivo; la productividad a largo plazo es una consecuencia de alta calidad.
- Haz que los errores los encuentre un colaborador y no un cliente.
- Una clave en la calidad en el desarrollo de software es realizar iteraciones en todas las fases de desarrollo.
- La gestión de errores y solicitud de cambios es esencial para controlar calidad y el mantenimiento.

- Si mides lo que haces, puedes aprender a hacerlo mejor.
- Haz lo que tenga sentido, no recurras a los dogmas.
- No se puede cambiar todo de una vez. Identifica los cambios que se traduzcan en los mayores beneficios, y comienza a implementarlos.

Referencias

- [1] J. Pradel Miquel and J. Raya Martos, Introducción a la ingeniería del software. Asignatura Ingeniería de ingeniería del Software, Universitat Oberta de Catalunya, 2010.
- [2] Irene T. Luque Ruiz, Apuntes de la Asignatura Ingeniería del Software (2013).
- [3] En 1946 fue presentado al público, en Philadelphia, el que se tiene como primer ordenador del mundo: el ENIAC, <https://www.prensaescuela.es/>.
- [4] Apple Watch Ultra, <https://www.gq.com.mx/relojes/articulo/apple-watch-ultra-2022-resena>.
- [5] Samsung Galaxy Z Flip, <https://www.xataka.com/analisis/samsung-galaxy-z-flip-primeras-impresiones-opiniones-video-fotos>.
- [6] J. Palacio and C. Ruata, Scrum Manager Gestión de Proyectos, http://www.scrummanager.net/bok/index.php?title>Main_Page (2011).
- [7] Roger S. Pressman, Ingeniería del Software, Un Enfoque Práctico, 9th Edition, Mc Graw Hill, 2021.
- [8] A. Abran, P. Bourque, R. Dupuis, J.W. Moore, and L. L. Tripp, Guide to the Software Engineering Body of Knowledge - SWEBOK, IEEE Press, Piscataway, 2004.
- [9] Stephen R. Schach, Software engineering, 1990.
- [10] I. Sommerville, Ingeniería del Software, 9th Edition, Pearson, 2011.
- [11] Organización Internacional para la Estandarización (ISO), www.iso.org.
- [12] IEEE Computer Society, www.computer.org.
- [13] Instituto de Ingeniería del software (SEI), www.sei.cmu.edu/.