|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **1** |
|  | TEMA |

Introducción a la ingeniería del *software* y al modelado

[1.1] La naturaleza del *software*

[1.2] La naturaleza única de las *webapps*

[1.3] Ingeniería del *software*

[1.4] El proceso del *software*

[1.5] La práctica de la ingeniería del *software*

[1.6] Mitos del *software*

[1.7] Cómo comienza todo

[1.8] Conceptos orientados a objetos

[1.9] Introducción a UML

[1.10] Referencias

Esquema



Ideas clave

1.1. La naturaleza del *software*



Las tres principales diferencias que presenta el *software* respecto al *hardware* son: se desarrolla o modifica con intelecto, no se 'desgasta' y se construye para uso individualizado.

El *software* tiene **grandes dominios de aplicación** como sistemas, aplicación, ingeniería, incrustado, línea de productos, aplicaciones web e inteligencia artificial. Y también **grandes desafíos** como la computación en un mundo abierto, la construcción de redes o el código fuente abierta.

El ***software* heredado** es el **desarrollado en décadas pasadas** (años 60, 70, 80), que normalmente se van modificando para adaptarse a las nuevas necesidades tecnológicas o del negocio, que en muchas ocasiones contiene funciones básicas para el negocio de la empresa, pero que también suele caracterizarse por su mala calidad.

Si este *software* funciona correctamente y satisface las necesidades del usuario, no habría que tocarlo, pero si requiere adaptación tecnológica o surgen nuevos requisitos funcionales, deberá hacerse **reingeniería del sistema heredado**.

1.2. La naturaleza única de las *webapps*

Las **aplicaciones web** presentan **características diferenciadoras** con el resto del *software*: hacen un **uso intensivo de las redes**, trabajan en **concurrencia**, el **rendimiento y la disponibilidad son críticos**, están orientadas a **datos**, el **contenido** y la **estética** son muy importantes, están en **evolución continua** con plazos cortos de desarrollo y es fundamental garantizar su seguridad.

1.3. Ingeniería del *software*

Antes de afrontar un desarrollo *software* hay que tener en cuenta que se debe **entender el problema a resolver previamente**, el diseño previo del *software* es crucial, debe tener alta calidad y un mantenimiento posterior sencillo. Para llevarlo a cabo es necesaria la ingeniería de *software*.



1.4. El proceso del *software*

«Un proceso define quién hace qué, cuándo y cómo, para alcanzar cierto objetivo». (I. Jacobson, G. Booch y J. Rumbaugh).

La estructura de proceso del *software* está compuesta por **cinco actividades principales** que la mayoría de las veces se aplican de forma iterativa:



Estas actividades se complementan por actividades comunes que se aplican a lo largo de todo el proyecto: seguimiento y control, gestión del riesgo, aseguramiento de la calidad, revisiones técnicas, medición, gestión de la configuración, gestión de la reutilización y preparación y producción del producto del trabajo.

1.5. La práctica de la ingeniería del *software*

Para resolver un problema en general y también aplicable a la ingeniería de *software* en particular se debe entender el problema, planificar la solución, ejecutar el plan y examinar el resultado.

Los siete principios de la ingeniería de *software*, como propuso David Hooker, son:



1.6. Mitos del *software*

Los siguientes son falsos mitos del *software* que hay que evitar, tal como señala Pressman (2010):

* **Mitos de la administración:**
* Tenemos un libro lleno de estándares y procedimientos para elaborar *software*.
* Si nos atrasamos, podemos agregar más programadores y ponernos al corriente.
* Si decido subcontratar el proyecto de *software* a un tercero, puedo descansar y dejar que esa compañía lo elabore.
* **Mitos del cliente:**
* Para comenzar a escribir programas, es suficiente el enunciado general de los objetivos y entraremos en detalles más adelante.
* Los requisitos del *software* cambian continuamente, pero el cambio se asimila con facilidad debido a que el *software* es flexible.
* **Mitos del profesional:**
* Una vez escrito el programa y funcionando, nuestro trabajo ha terminado.
* Hasta que no se ejecute el programa, no hay manera de evaluar su calidad.
* El único producto resultante del trabajo que se entrega en un proyecto exitoso es el programa que funciona.
* La ingeniería de *software* hará que generemos documentación voluminosa e innecesaria, e invariablemente nos retrasará.

1.7. Cómo comienza todo

Los proyectos de *software* comienzan por la **aparición de una necesidad**: ampliación de funcionalidades o corrección de defectos en una aplicación existente, adaptar un sistema heredado a un nuevo entorno o la necesidad de crear un nuevo producto/servicio/sistema.

1.8. Conceptos orientados a objetos

Una **clase** es una estructura que encapsula datos (**atributos**) y operaciones (**métodos**), y un **objeto** es una instancia concreta de una clase.



En un programa orientado a objetos los diferentes objetos interactúan entre ellos mediante los **mensajes**, que son llamadas desde uno objeto emisor para que se ejecute el código que contiene un método del objeto receptor. Un buen diseño de una clase debe garantizar el encapsulamiento de sus atributos y métodos, la alta cohesión y el bajo acoplamiento.

1.9. Introducción a UML

El Lenguaje de Modelado Unificado (UML) es un lenguaje de diseño para modelar *software* mediante diferentes tipos de diagramas. Los principales son:

* **Diagrama de clases**: proporciona elementos para modelar las clases (con sus atributos y métodos) y las relaciones y asociaciones entre ellas.
* **Diagrama de implementación**: permite mostrar la distribución física de un sistema *software* en las plataformas de *hardware* y entornos de ejecución.
* **Diagramas de casos de uso**: representan la interacción del usuario con las diferentes funciones del sistema.
* **Diagramas de secuencia**: muestra la comunicación dinámica entre los diferentes objetos para llevar a cabo una tarea, describiendo el orden de ejecución.
* **Diagramas de comunicación**: representan como los diagramas de secuencia la comunicación entre los objetos para realizar una tarea, enfatizando en las relaciones entre las clases y objetos.
* **Diagramas de actividad**: muestran el comportamiento dinámico del sistema mediante el flujo de control de las acciones que debe realizar.
* **Diagramas de estado**: modelan los estados por los que pasa un objeto y las acciones que desencadenan la transición de estados.

1.10. Referencias

Hooker, D. (1996). *Seven Principles of Software Development*. Recuperado de: [http://c2.com/cgi/wiki?SevenPrinciplesOf*Software*Development](http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment)

Jacobson, I., Booch, G., Rumbaugh, J. (2013). *El proceso unificado de desarrollo de software*. Madrid: Addison Wesley.

Pressman, R. (2010). *Ingeniería del software* (7ª ed.). México: McGrawHill.

Lo + recomendado

No dejes de leer…

**Visión general de la ingeniería del *software***

En este artículo de Wikipedia se aporta una muy buena visión general de la ingeniería del *software*.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:

[http://en.wikipedia.org/wiki/*Software*\_engineering](http://en.wikipedia.org/wiki/Software_engineering)

**Historia de la ingeniería del *software* y crisis del *software***

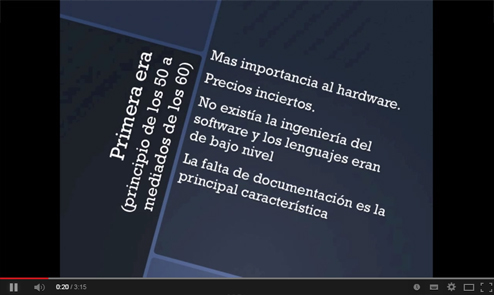
El artículo repasa la historia del *software*, la crisis del *software* y el subsiguiente surgimiento de la disciplina de la ingeniería del *software*.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:

[http://en.wikipedia.org/wiki/History\_of\_*software*\_engineering](http://en.wikipedia.org/wiki/History_of_software_engineering)

No dejes de ver…

**La crisis del *software***



Inicialmente el *software* se desarrollaba ad hoc, sin ninguna planificación ni modelado previos. Es como si quisiéramos construir un rascacielos sin ningún plano de ingeniería civil. Con el tiempo, esto originó la llamada «Crisis del *software*», motivada por costes de mantenimiento cada vez más altos debido a la mala calidad del *software* desarrollado sin ningún modelo ni planificación. Debido a esta crisis se va desarrollando la disciplina de la ingeniería del *software*. El siguiente vídeo profundiza en este concepto.

Accede a los vídeos desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=miGf6iZ0GZY>

+ Información

A fondo

**El Lenguaje Unificado de Modelado**

Jacobson, I., Booch, G., Rumbaugh, J. (2013). *El proceso unificado de desarrollo de software*. Madrid: Addison Wesley.

El Lenguaje Unificado de Modelado (UML) no es un mero lenguaje gráfico de modelado de *software*. Tuvo y tiene una importancia crucial, ya que la ingeniería del *software* actual se ha desarrollado y se extendido gracias a UML. Este libro está escrito por los tres inventores de UML y contiene muchos ejemplos de modelado para ilustrar el uso y la aplicación de UML. Es altamente recomendable para todo Ingeniero Informático tener este libro y leerlo en su totalidad (además es muy ameno), pero aquí recomiendo especialmente leer los primeros capítulos, en donde se explica lo que es la ingeniería del *software*, su proceso, sus métodos y su utilidad. Y todo ello explicado mediante ejemplos amenos e incluso divertidos.



Enlaces relacionados

*Software* Engineering Institute

Página web de una de las escuelas especializadas en Ingeniería del *Software* más prestigiosas del mundo.



Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<http://www.sei.cmu.edu/>

Recursos externos

Mapas conceptuales

Para elaborar mapas conceptuales (muy útiles para estudiar contenidos teóricos), podemos usar el programa CMapTools.



Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<http://cmap.ihmc.us>

Actividades

Foro #1: Fundamentos de Ingeniería de Software

**Descripción del foro**

En este foro se profundizará sobre los conceptos trabajados en la unidad sobre Ingeniería de Software, el paradigma orientado a objetos y UML

Para ello, con base a la presentación sobre esta unidad, el profesor debe proponer un topico de discusión sobre la relación entre la Ingeniería de Software, el paradigma de programación orientado a objetos y UML.

Los alumnos deberán argumentar su respuesta desde la relación entre los conceptos de:

* Fundamentos de Ingeniería de Software.
* Conceptos del paradigma de programación orientado a objetos
* Lengueje de modelado unificado UML

Test

**1.** Indica cuáles son las capas de la ingeniería del *software*:

A. Proceso, actividad, acción y tarea.

B. Compromiso con la calidad, proceso, métodos y herramientas.

C. Métodos, actividades y etapas.

D. Ninguna de las anteriores.

**2.** ¿Cuál es el principal problema del *software* heredado, según la perspectiva de la ingeniería del *software*?

A. Que está tecnológicamente obsoleto.

B. Que ya no cumple con sus requisitos.

C. Que generalmente es de mala calidad y por lo tanto cuesta mucho mantenerlo.

D. Todas son correctas.

**3.** ¿Cuál es el dominio de aplicación del *software* que hace uso de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o con el análisis directo?

A. *Software* de ingeniería y ciencias.

B. *Software* incrustado.

C. Aplicaciones web.

D. *Software* de inteligencia artificial.

**4.** Señala la respuesta incorrecta:

A. El *software* siempre ha estado basado en componentes.

B. Idealmente el *software* debería estar basado en componentes.

C. La ingeniería mecánica está basada en componentes.

D. La ingeniería del *hardware* está basada en componentes.

**5.** Señala la característica de las *webapps*:

A. Concurrencia.

B. Rendimiento.

C. Uso intensivo de redes.

D. Todas son correctas.

**6.** Señala la característica deseable que debe tener un buen *software*:

A. Alta calidad.

B. Facilidad de mantenimiento.

C. Entender bien el problema antes de desarrollar el *software.*

D. Todas son correctas.

**7.** ¿Cuál es la primera actividad en la estructura de proceso general para la ingeniería del *software*?

A. Modelado.

B. Construcción.

C. Planeación.

D. Comunicación.

**8.** Señala la actividad sombrilla:

A. Administración del riesgo.

B. Despliegue.

C. Construcción.

D. Modelado.

**9.** ¿Cuál es el diagrama UML que muestra distribución física de un sistema *software* en las plataformas de hardware y entornos de ejecución?

A. Diagrama de flujo.

B. Diagrama de actividad.

C. Diagrama de implementación.

D. Diagrama de clases.

**10.** ¿Cuál es el diagrama UML que modela la interacción del usuario con las diferentes funciones del sistema?

A. Diagrama de clases.

B. Diagrama de casos de uso.

C. Diagrama de actividad.

D. Diagrama de flujo.