

4 de junio de 2017

# Ingeniería del Software I

Samuel García  
Ignacio Ballesteros

# Índice general

<b>1. Introducción a la Ingeniería del Software</b>	<b>2</b>
1.1. La crisis del Software . . . . .	2
1.2. Costes en la Ingeniería del Software . . . . .	3
1.3. El Software . . . . .	4
1.4. Procesos en el Software . . . . .	4
1.5. Ciclo de Vida . . . . .	5
<b>2. Ingeniería de Requisitos</b>	<b>7</b>
<b>3. Diseño Estructurado de Alto Nivel</b>	<b>8</b>
<b>4. Objetos</b>	<b>9</b>
4.1. Orientación a objetos . . . . .	9
4.1.1. Enfoque estructurado . . . . .	9
4.1.2. Enfoque orientado a objetos . . . . .	10
<b>5. Arquitectura</b>	<b>12</b>

# Bloque 1

## Introducción a la Ingeniería del Software

**Ingeniería del Software** (*IEEE*) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de *software*.

Las **Ciencias de la Computación** se preocupa de los fundamentos de la teoría; mientras que la Ingeniería del Software de los aspectos prácticos.

La **Ingeniería del Software** estudia los **productos** producidos (ejecutables, módulos, sistemas, librerías...) y los **procesos** usados para producir esos productos.

### 1.1. La crisis del Software

Los principales problemas que hay detrás de la crisis del Software son:

- El incremento en el tamaño y la complejidad
- Los sobrecostos
- Fallos en el diseño
- Mal mantenimiento
- Herramientas no solo de programación.

Pero desde un enfoque más moderno, también se aprecian otros tipos de problemas:

- Falta de robustez en el Software para componentes críticos o de los que somos dependientes.
- Excesiva complejidad en el Software como para entenderlo y comprenderlo.
- Exigencia de cambiar rápidamente.

Frente a esto, desde la Ingeniería del Software se plantea la mejora en las metodologías y el uso de lenguajes de alto nivel (mayor abstracción).

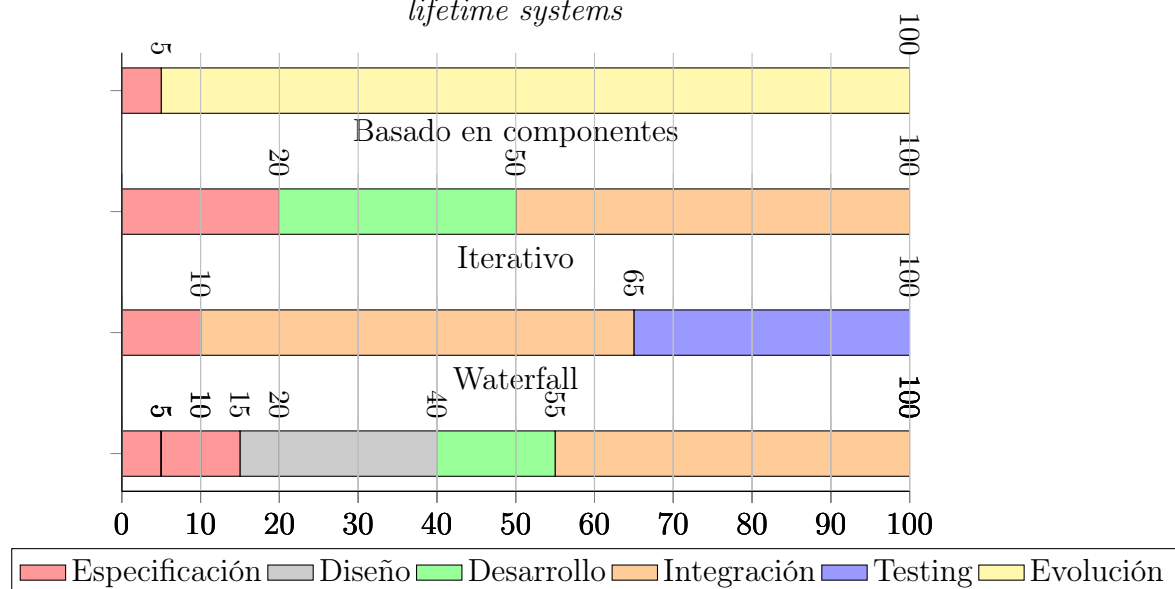
Tras el intento de solucionar estos problemas, el concepto de Ingeniería del Software se redefine en:

*Disciplina que tiene como objetivo la producción de Software libre de errores, sin retrasos y dentro del presupuesto; satisfaciendo las necesidades del cliente*

## 1.2. Costes en la Ingeniería del Software

En cuanto a la realización de productos, aproximadamente el 60 % del coste es el desarrollo, y el 50 % en pruebas (test). Sin embargo, con el uso del software, los gastos de mantenimiento superan a los del desarrollo.

Figura 1.1: Distribución del coste según metodología *lifetime systems*



Cuanto más tarde se encuentran errores en el Software, o más tarde se hace un cambio de requisito, los cambios son más costosos. Comparando el cambio en dos fases del ciclo de vida:

Temprano	Tardío
Cambio en la especificación.	Cambio en la especificación.
	Cambiar el código y la documentación.
	Probar el cambio.
	Testing.
	Instalación del producto en el cliente.

A medida que pasa el tiempo, la cantidad de fallos que aparecen aumentan debido al deterioro.

### 1.3. El Software

Entre las características que se le atribuyen a un *producto Software* encontramos:

- Múltiples **programas**.
- Archivos de **configuración**.
- La **documentación** del sistema.
- La documentación de **uso**.
- Los **datos** del sistema.
- **Actualización** de información.

El *Software* se realiza para **clientes particulares** o para el uso **general**. Esto está también relacionado con que los productos de software sean **genéricos** o **a medida**.

Dependiendo del producto desarrollado, el software puede entrar en categorías como: tiempo real, negocios, científico, embebido, PC, IA, Web...

Los atributos de un **buen Software** varían según las perspectivas:

Usuario	Desarrolador
Exactitud	Consistencia
Confiabilidad	Comprensibilidad
Eficiencia	Capacidad de ser probado
Mantenibilidad	Compacidad
Usabilidad	Compatibilidad
Robustez	

### 1.4. Procesos en el Software

Se entiende por *proceso Software* un conjunto de **actividades y resultados** asociados a la producción de Software.

Este proceso se puede analizar desde diferentes perspectivas:

- Flujo de **trabajo**.
- Flujo de **datos**.
- **Acción**.

Los modelos del *ciclo de vida* del Software especifican las fases del *proceso de Software*. Hemos mencionado ya ejemplos en la sección 1.2. Un modelo está compuesto de:

- Descripción propia.
- Reglas.
- Recomendaciones (*guías de estilo*).
- Procesos (*actividades a seguir*).

Estos **modelos** están orientados a resolver los retos de la ingeniería del Software, muy relacionados con los atributos del buen Software indicados en la sección 1.3.

- Heterogeneidad de plataformas.
- Entrega más rápida.
- Confianza.
- Gastos en el Hardware/Software.
- Adaptabilidad a nuevas tecnologías.
- Usabilidad.
- Mantenimiento.

## 1.5. Ciclo de Vida

Qué hacer →Cómo hacerlo →Hacerlo →Probarlo →Usarlo →Mantenerlo

Para la realización del Software tendremos que tener en cuenta:

- Escala
- Productividad
- Calidad (ISO)
  - Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad, Portabilidad.
- Consistencia
- Tasa de cambio

Este ciclo de vida se organiza en fases. A cada fase se obtendrá un resultado que se utilizará en las siguientes fases. El *ciclo de vida* del Software se enfoca en manejar la complejidad y el cambio a lo largo de un largo proceso.

Existen distintos *ciclos de vida*, nombrados como modelos (sección 1.3):

- Informal
- Convencional
- Incremental
- Evolutivo
- Prototipado (puede ser incluido en los anteriores modelos)

Estos modelos tienen un equilibrio entre los siguientes factores:

- Velocidad de desarrollo
- Calidad
- Visibilidad
- Sobrecarga de gestión
- Exposición al riesgo
- Relaciones públicas

Bloque 2

Ingengería de Requisitos



## Bloque 3

### Diseño Estructurado de Alto Nivel

# Bloque 4

## Objetos

### 4.1. Orientación a objetos

#### 4.1.1. Enfoque estructurado

Se denomina enfoque estructurado a la forma de pensar el software en términos de funciones de transformación de datos (se disocia entre funciones y datos, y las tareas se interpretan como una transformación de los últimos).

#### Ejemplo: Pintar un círculo

El enfoque estructurado resuelve el problema de pintar un círculo de la siguiente forma:

- Usa una definición de círculo que esté acorde con los recursos de software (en este caso la expresión algebraica).

$$R^2 \leq (x - x_0)^2 + (y - y_0)^2 \quad (4.1)$$

donde el radio  $R$  y las coordenadas del centro son las constantes que especifican un círculo concreto.

- Disocia la definición de círculo en dos partes y las reinterpreta:
  - Considera que  $R$  y el centro son datos para pintar el círculo y añade el color.
  - Convierte la expresión declarativa en una función operativa que transforma el conjunto de datos precedentes en  $(x, y, \text{color})$  de todos los píxeles para pintar el círculo en la pantalla.

- Como resultado final se obtiene un sistema capaz de pintar un círculo en términos de un proceso de transformación de datos.

El sistema software se expresa como una función  $F(x)$  que transforma el conjunto de datos  $(R, x_0, y_0)$  en otro conjunto de datos, en este caso de píxeles.

(insertar figura 1.2)

A este tipo de esquema se le denomina *diagrama de flujo de datos*. **El diagrama de flujo de datos es un esquema asíncrono** (no expresa secuencias); las flechas sólo indican los flujos de datos, no el orden de ejecución.

El principal problema del enfoque estructurado es latente en el momento en el que queremos añadir más elementos e interactuar con ellos, por ejemplo, pintar varios círculos y actuar sobre los mismos de forma selectiva, digamos borrar el segundo que se pintó. Podríamos hacer un bucle para crear  $n$  círculos, pero si queremos guardarlos tendríamos que añadir tantas variables como círculos, con el objetivo de retener cada conjunto de constantes. Este sistema es una duplicación del sistema para solo un caso.

La disgregación de los conceptos en datos y funciones tiene sus pros y sus contras, por ejemplo, este enfoque permite trabajar directamente con la idea de base de datos o archivo, lo cual puede ser beneficioso. Sin embargo, esta disociación implica **disminuir nuestro nivel de abstracción**.

#### 4.1.2. Enfoque orientado a objetos

El enfoque orientado a objetos es la forma particular de pensar el software en términos de elementos que colaboran entre sí para realizar tareas. Este enfoque nos da un nivel de abstracción superior al estructurado, asociando cada elemento del problema a un elemento software. Cada elemento software tiene las propiedades íntegras de cada elemento del discurso (lo que *hace a una cosa ser una cosa*).

##### Ejemplo: Pintar un círculo

En el ejemplo anterior, definimos un objeto **círculo** que cumple las propiedades de un círculo según la definición que hemos adaptado para nuestro sistema (en este caso, el objeto contiene un centro y un radio) y tiene los mecanismos para pintarse y crearse como elemento.

El enfoque de objetos piensa:

- En variables software capaces de recordar las constantes de un círculo, capaces de pintar un círculo y capaces de crearse a sí mismas como variables.
- En el sistema software en términos de la interacción de estas variables, dadas sus respectivas capacidades para ejecutar operaciones, es decir, cómo rela-

cionar todas las variables para conseguir que se realice la tarea de pintar círculos.

(introducir figura 1.8)

Este esquema muestra el sistema software, donde se aprecian las relaciones entre las variables software que ejecutan la tarea de pintar un círculo. Como vemos, el sistema software, aun aplicando el mismo algoritmo, tiene una organización diferente, y por tanto sus propiedades también varían.

## Objetos

Esto que hemos ido llamando *variables software* se conocen en este enfoque como **objetos**, y amplían la idea de la variable software tratada en el enfoque estructurado, ya que tienen capacidad de expresar cualquier cosa, incluso operaciones. Otra definición complementaria de objeto es la siguiente: *Un objeto es un elemento software cualitativamente distinto capaz de expresar un concepto más amplio, más ambiguo: cosa*

Los objetos interactúan entre ellos mediante **mensajes**, solicitudes a objetos para que ejecuten operaciones.

# Bloque 5

## Arquitectura

Como bien se comentó en el bloque 2

# Índice alfabético

buen Software, 4

costes, 3

Requisitos, 7

retos Software, 5