



Introducción a la arquitectura de software



Contenidos

1. Objetivos de aprendizaje

2. Arquitectura de software

3. Tipos de patrones de diseño

4. Patrones arquitectónicos




Contenidos

5. Estilos arquitectónicos


6. Relación entre patrones de diseño, arquitectónicos y estilos arquitectónicos

7. Principios SOLID

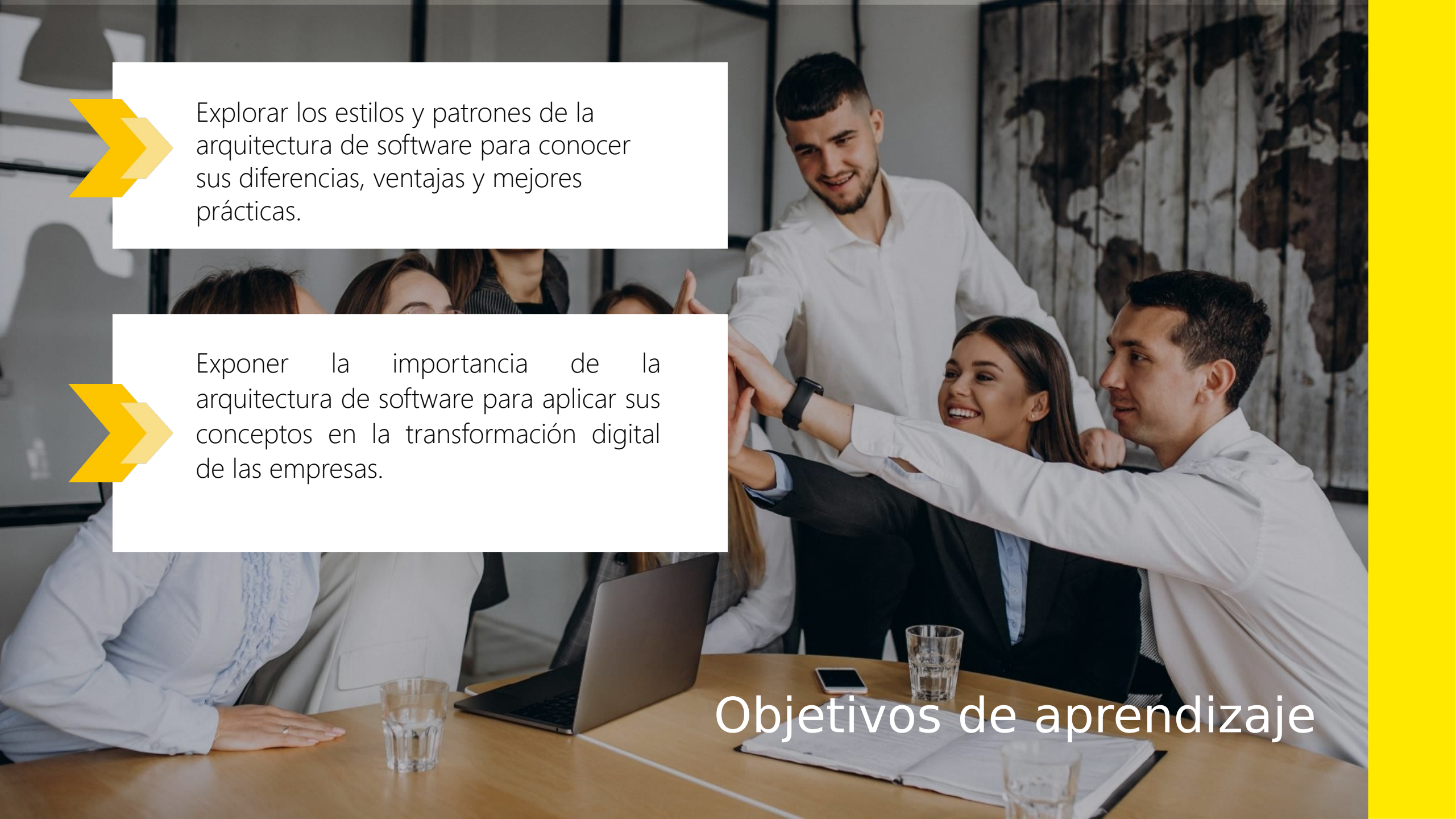
8. Otros conceptos



Explorar los estilos y patrones de la arquitectura de software para conocer sus diferencias, ventajas y mejores prácticas.



Exponer la importancia de la arquitectura de software para aplicar sus conceptos en la transformación digital de las empresas.



Objetivos de aprendizaje

Arquitectura de software

La arquitectura de software es la organización fundamental de un sistema enmarcada en sus componentes, las relaciones entre ellos, y el ambiente, y los principios que orientan su diseño y evolución (ISO/IEC/IEEE 42010: Defining «architecture», s. f.)

Tipos de patrones de diseño

Un patrón de diseño es básicamente una forma reutilizable de resolver un problema común en el desarrollo de software (Alcolea, 2021)



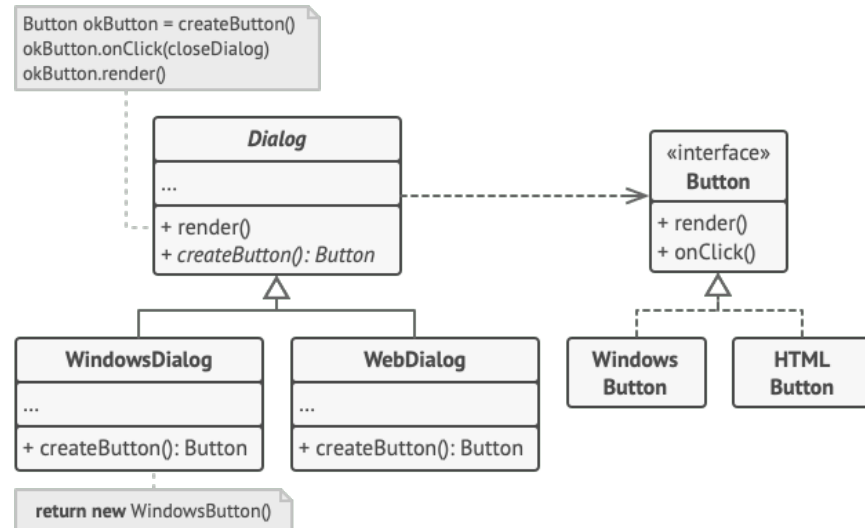
(¿Qué son los patrones de diseño de software?, 2022)

- **Patrones creacionales**
- **Patrones estructurales**
- **Patrones de comportamiento**

Patrones creacionales:

Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la flexibilidad y la reutilización del código existente de una manera adecuada a la situación. Esto le da al programa más flexibilidad para decidir qué objetos deben crearse para un caso de uso dado (¿Qué son los patrones de diseño de software?, 2022).

Patrones creacionales



(Factory Method, s. f.)

- Abstract Factory

- Builder Patterns

- Factory Method

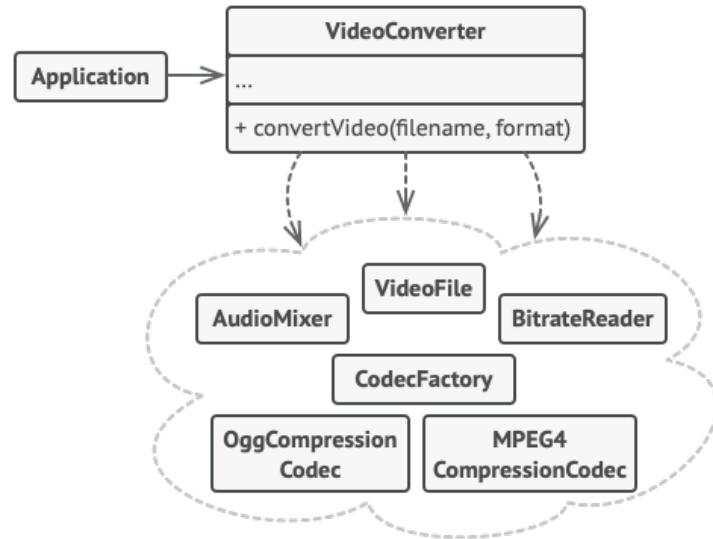
- Prototype

- Singleton

Patrones estructurales:

Facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y las estructuras de objetos. El concepto de herencia se utiliza para componer interfaces y definir formas de componer objetos para obtener nuevas funcionalidades (¿Qué son los patrones de diseño de software?, 2022).

Patrones estructurales



(Facade, s. f.)

- Adapter

- Bridge

- Composite

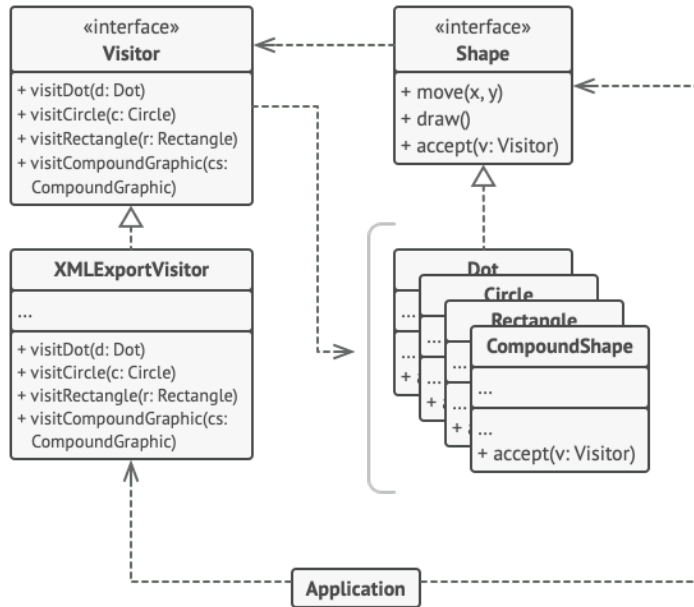
- Decorator

- Facade

Patrones de comportamiento:

El patrón de comportamiento se ocupa de la comunicación entre objetos de clase. Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones (¿Qué son los patrones de diseño de software?, 2022).

Patrones de comportamiento



(Visitor, s. f.)

- Chain of responsibility

- Command

- Interpreter

- Iterator

- Visitor

Patrones arquitectónicos

¿Cómo los relacionamos con patrones de diseño? El patrón de diseño describe la relación entre un conjunto de clases y objetos para resolver un problema de diseño común en un contexto específico.

Por lo tanto, Un patrón arquitectónico es una solución reutilizable de uso general para resolver un problema arquitectónico común en un contexto específico (¿Qué son los patrones arquitectónicos y los estilos arquitectónicos? - programador clic, s. f.).






Estilos arquitectónicos

Un estilo arquitectónico es un conjunto de restricciones arquitectónicas colaborativas que restringen los roles y funciones de los elementos arquitectónicos, y entre elementos que pueden existir en cualquier arquitectura que siga ese estilo.

El patrón de arquitectura son las preguntas de un contexto específico, el estilo de arquitectura es la solución de ese contexto específico.

(¿Qué son los patrones arquitectónicos y los estilos arquitectónicos? - programador clic, s. f.).

Estilos de arquitectura:

-  Monolítica
-  Cliente-Servidor
-  Orientado a servicios
-  Por capas
-  P2P

Patrones de arquitectura:

-  SOA
-  Microservicios
-  N-Capas, 3-capas
-  MVC
-  Microkernel
-  REST

Patrones de diseño:

-  Factory method
-  Singleton
-  Prototype
-  Facade
-  Observer
-  Strategy
-  Memento

Relación entre patrones de diseño, arquitectónicos y estilos arquitectónicos

Para entender la relación vamos a mirar ejemplos de cada uno:

Principios SOLID

Los objetivos de los principios SOLID son:

- Crear un software eficaz: que cumpla con su cometido y que sea robusto y estable.
 - Escribir un código limpio y flexible ante los cambios: que se pueda modificar fácilmente según necesidad, que sea reutilizable y mantenible.
 - Permitir escalabilidad: que acepte ser ampliado con nuevas funcionalidades de manera ágil.
- (Maluenda, 2022)

Principio de Responsabilidad Única (Single Responsibility Principle):

Reúne las cosas que cambian por las mismas razones. Separa aquellas que cambian por razones diferentes
(Maluenda, 2022).

Principio de abierto / cerrado (Open / Closed Principle):

Deberías ser capaz de extender el comportamiento de una clase, sin modificarla (Maluenda, 2022).

Principio de Sustitución de Liskov (Liskov Sustitution Principle):

si en alguna parte de nuestro código estamos usando una clase, y esta clase es extendida, tenemos que poder utilizar cualquiera de las clases hijas y que el programa siga siendo válido (Leiva, 2016).

Principio de Segregación de la Interfaz (Interface Segregation Principle):

Haz interfaces que sean específicas para un tipo de cliente (Maluenda, 2022).

Principio de inversión de dependencias (Dependency inversion Principle):

Gracias al principio de inversión de dependencias, podemos hacer que el código que es el núcleo de nuestra aplicación no dependa de los detalles de implementación, como pueden ser el framework que utilices, la base de datos, cómo te conectes a tu servidor...

Todos estos aspectos se especificarán mediante interfaces, y el núcleo no tendrá que conocer cuál es la implementación real para funcionar (Leiva, 2016).

Encapsulamiento

es un mecanismo que consiste en organizar datos y métodos de una estructura, conciliando el modo en que el objeto se implementa, es decir, evitando el acceso a datos por cualquier otro medio distinto a los especificados. Por lo tanto, la encapsulación garantiza la integridad de los datos que contiene un objeto (EcuRed, s. f.).

Acoplamiento

El acoplamiento es el grado en el que los módulos del software dependen unos de otros. Si para cambiar un módulo en un programa tenemos que cambiar otro módulo, entonces hay acoplamiento entre ellos (Garzas, 2014).

Cohesión

Decimos que algo tiene una alta cohesión, si tiene unos límites claros y todo lo que hace está contenido en un único sitio:

- Un balón tendría una alta cohesión: tiene una forma definida, límites claros (es redondeado), y todo lo que implica el concepto balón está en un único sitio.
- Internet sería un ejemplo de algo con baja cohesión: es algo muy amplio y sus límites no están definidos.

Don't Repeat Yourself (DRY)





DRY nos recuerda que cada comportamiento repetible en el código debe estar aislado (por ejemplo, separado en una función) para su reutilización (Peña, 2021).

Separation of Concerns (SoC)

Este principio afirma que el software se debe separar en función de los tipos de trabajo que realiza (Microsoft, 2022).

Ley de Demeter

Dice que un método de un objeto solo puede llamar a métodos de:






-  El propio objeto.
-  Un argumento del método.
-  Cualquier objeto creado dentro del método.
-  Cualquier propiedad/campo directo del propio objeto.

Una de las principales cosas que la Ley de Demeter quiere evitar que hagas es esto:
`objetoA.metodoDeA().metodoDeB().metodoDeC();`

(Garzas, 2014)

Keep it Simple, Stupid (KISS)

La idea principal de este es que cuando se tiene un sistema, normalmente funciona mucho mejor si se diseña y desarrolla de manera simple.

-  Mantén los métodos y las clases pequeños
-  Usar nombres claros para las variables
-  No reutilizar variables locales
-  Divide el problema en partes pequeñas
-  No abusar de los comentarios



(Zapata, 2020)

Inversion of Control (IoC)

Es un estilo de programación en el cual un framework o librería controla el flujo de un programa. Por ejemplo: cuando creamos una aplicación ASP.NET nosotros conocemos cuál es el ciclo de vida de una página, pero no lo controlamos ya que es ASP.NET el que lo hace, o cuando creamos aplicaciones de escritorio o móviles, los formularios también tienen su ciclo de vida que controla el framework (Valverde, 2014).

Atributos de calidad

Los atributos de calidad, se clasifican en las siguientes dos categorías:

-  Observables vía Ejecución: Son los atributos que se determinan del comportamiento del sistema, en tiempo de ejecución.
-  No Observables vía Ejecución: Son los atributos que se establecen durante el desarrollo del sistema.

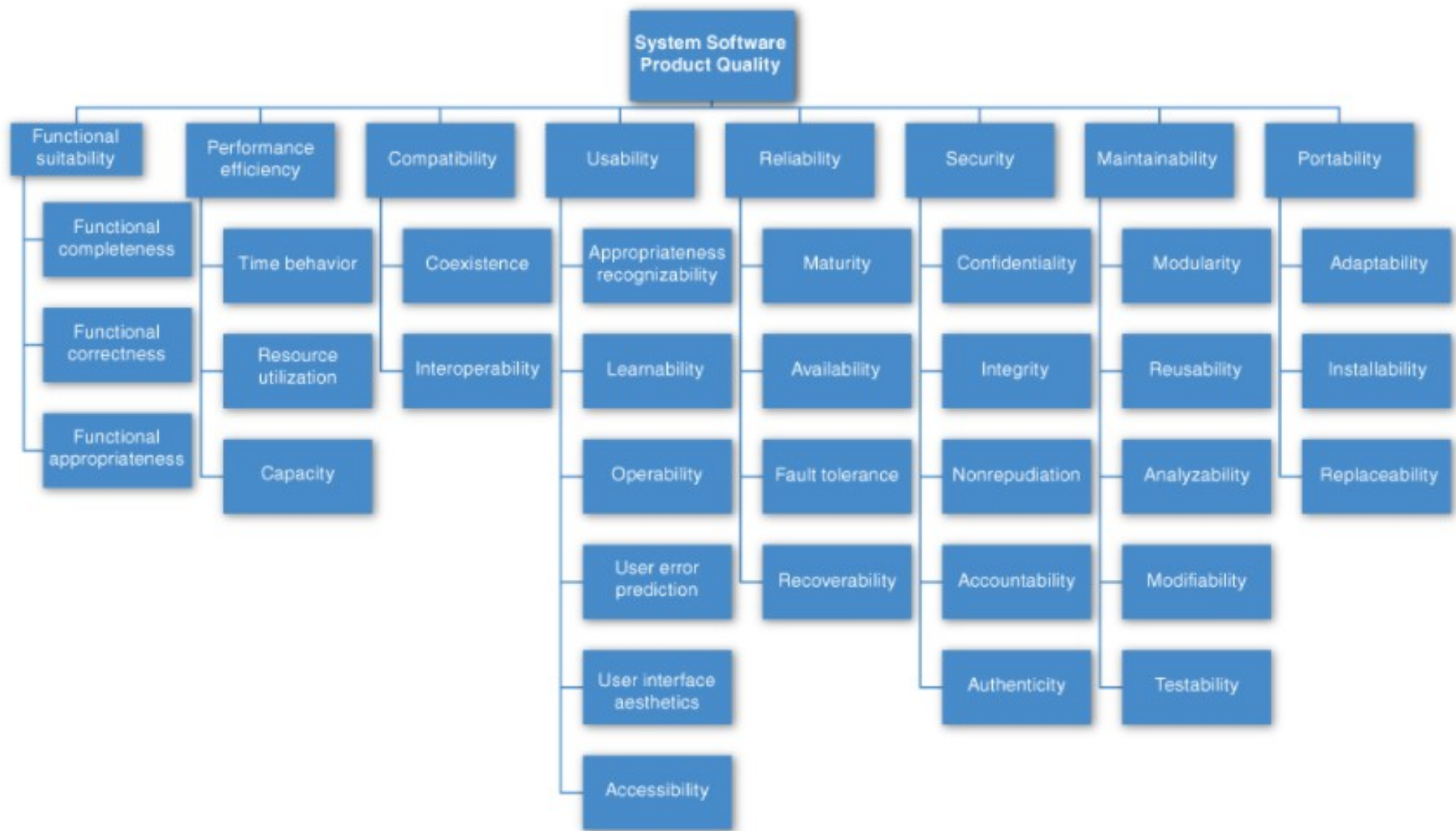


Figure 12.1. The ISO/IEC FCD 25010 product quality standard



Life
Thinking

Muchas gracias