INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE

UN ENFOQUE PRÁCTICO

Oscar Javier Blancarte Iturralde

PRIMERA EDICIÓN

Oscar Blancarte Blog
software architect

Introducción a la arquitectura de software – Un enfoque práctico

❝La arquitectura de software no es el arte de hacer diagramas atractivos, sino el arte de diseñar soluciones que funcionen adecuadamente durante un periodo de tiempo razonable**❞**

— Oscar Blancarte (2020)

Datos del autor:

Ciudad de México.

e-mail: oscarblancarte3@gmail.com

Autor y edición:

Composición y redacción:

Oscar Javier Blancarte Iturralde

Edición:

Oscar Javier Blancarte Iturralde

Portada:

Arq. Jaime Alberto Blancarte Iturralde

Primera edición:

Se publicó por primera vez en Enero del 2020

Acerca del autor

Oscar Blancarte es originario de Sinaloa, México donde estudió la carrera de Ingeniería en Sistemas Computacionales y rápidamente se mudó a la Ciudad de México donde actualmente radica.

Oscar Blancarte es Arquitecto de software con más de 15 años de experiencia en el desarrollo y arquitectura de software. Certificado como Java Programmer (Sun microsystems), Análisis y Diseño Orientado a Objetos (IBM) y Oracle IT Architect (Oracle). A lo largo de su carrera ha trabajado para diversas empresas del sector de TI, entre las que destacan su participación en diseños de arquitectura de software y consultoría para clientes de los sectores de Retail, Telco y Health Care. Oscar Blancarte es además autor de su propio blog https://www.oscarblancarteblog.com desde el cual está activamente publicando temas interesantes sobre Arquitectura de software y temas relacionados con la Ingeniería de Software en general. Desde su blog ayuda a la comunidad a resolver dudas y es por este medio que se puede tener una interacción más directa con el autor.

Además, es un apasionado por el emprendimiento, lo que lo ha llevado a emprender en diversas ocasiones, como es el caso de Centripio, una plataforma de educación online, Snipping Code, una plataforma de productividad para desarrolladores donde pueden guardar pequeños fragmentos de código repetitivos y Reactive Programming, la plataforma en la cual publica sus libros e invita a otros desarrollar sus propias obras.

Otras obras del autor

Aplicaciones reactivas con React, NodeJS & MongoDB

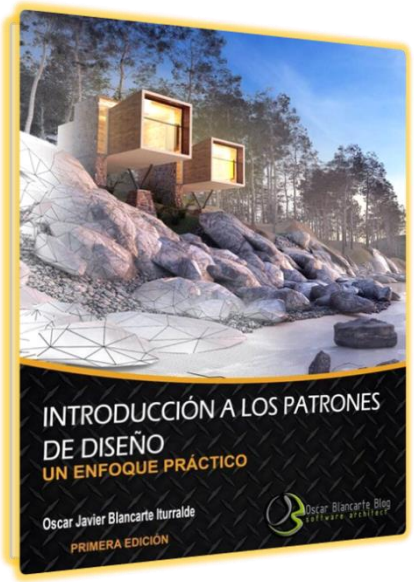
Es un libro enfocado a enseñar a construir aplicaciones reactivas utilizando React, que va desde crear una aplicación WEB, hasta construir tu propio API REST utilizando NodeJS + Express y persistiendo toda la información con MongoDB, la base de datos NoSQL más popular de la actualidad.

Con este libro no solo aprenderás a construir aplicaciones WEB, sino que aprenderás todo lo necesario desde desarrollo hasta producción.

Durante todo el libro vamos desarrollando un proyecto final llamado Mini Twitter, el cual es una réplica de la red social Twitter, en donde utilizamos absolutamente todos los conceptos que se enseñan durante la lectura del libro.

Ir al libro

Introducción a los patrones de diseño



Es el libro más completo para aprender patrones de diseño, pues nos enfocamos en enseñar los 25 patrones más utilizados y desde una filosofía del mundo real, es decir, que todos los patrones están acompañados de un problema que se te podría presentar en alguno de tus proyectos.

Todos los patrones se explican con un escenario del mundo real, en el cual se plantea la problemática, como se solucionaría sin patrones de diseño y cómo sería la mejor solución utilizando patrones, finalmente, implementaremos la solución y analizaremos los resultados tras ejecutar cada aplicación.

Ir al libro

Agradecimientos

Este libro tiene una especial dedicación a mi esposa Liliana y mi hijo Oscar, quienes son la principal motivación y fuerza para seguir adelante todos los días, por su cariño y comprensión, pero sobre todo por apoyarme y darme esperanzas para escribir este libro.

A mis padres, quien con esfuerzo lograron sacarnos adelante, darnos una educación y hacerme la persona que hoy soy.

A todos los lectores anónimos de mi blog y todas aquellas personas que, de buena fe, compraron y recomendaron mis libros y fueron en gran medida quienes me inspiraron para escribir este tercer libro.

Finalmente, quiero agradecerte a ti, por confiar en mí y ser tu copiloto en esta gran aventura para aprender el Arte de la arquitectura de software y muchas cosas más :)

Prefacio

A lo largo de mi experiencia, me ha tocado convivir con muchas personas que se hacen llamar así mismas "*Arquitectos de software*" (las comillas son a propósito), aludiendo a su experiencia en el desarrollo de software o simplemente por haber aguantado muchos años en una empresa, lo que los hace sentirse merecedores de ese título, ya que según ellos, son los únicos que conocen todo el sistema, sin embargo, esto puede ser muy peligroso, pues un arquitecto no es una persona que tiene mucho tiempo en la empresa o el líder de desarrollo o una persona con mucha experiencia en el desarrollo de software.

Un verdadero arquitecto de software no es solo una persona con fuertes conocimientos técnicos, sino que debe de tener muchos de los denominados "*Soft Skills*" o habilidades blandas, que van desde el liderazgo, la toma de decisiones, innovación y toma de riesgos. Un arquitecto es una persona capaz de ofrecer soluciones innovadoras que solucionen problemas complejos, aun así, no es solo eso, un Arquitecto debe de poder justificar y defender sus argumentos, llegando al grado de tener que confrontar a altos directivos y hacerles ver la importancia de sus decisiones.

Si bien, este es un libro introductorio, es, sin duda, el libro en donde he puesto la mayor parte de mis conocimientos, pues son conocimientos que me tomó muchos años adquirir, madurar y perfeccionar, pues la arquitectura de software no es un conocimiento que se adquiera con una simple lectura o curso, sino que se van adquiriendo con la experiencia, experiencia que yo quiero compartirles y que es sin lugar a duda, el libro que yo hubiera querido tener cuando inicié en la arquitectura de software.

No quisiera entrar en más detalle en este momento, ya que a medida que profundicemos en el libro, analizaremos el rol de un arquitecto de software y su relación con la arquitectura de software. Analizaremos la diferencia que existe entre patrones de diseño, patrones arquitectónicos, los tipos de arquitectura y cómo es que todo esto se combina para crear soluciones de software capaces de funcionar durante grandes periodos de tiempo sin verse afectadas drásticamente por nuevos requerimientos.

Cómo utilizar este libro

Este libro es en lo general fácil de leer y digerir, su objetivo es enseñar todos los conceptos de forma simple y asumiendo que el lector tiene poco o nada de conocimiento del tema, así, sin importar quien lo lea, todos podamos aprender.

Como parte de la dinámica de este libro, hemos agregado una serie de tipos de letras que hará más fácil distinguir entre los conceptos importantes, código, referencias a código y citas. También hemos agregado pequeñas secciones de tips, nuevos conceptos, advertencias y peligros, los cuales mostramos mediante una serie de íconos agradables para resaltar a la vista.

Texto normal:

Es el texto que utilizaremos durante todo el libro, el cual no enfatiza nada en particular.

Negritas:

El texto en negritas es utilizado para enfatizar un texto, de tal forma que buscamos atraer tu atención, ya que se trata de algo importante.

Cursiva:

Lo utilizamos para hacer referencia a fragmentos de código como una variable, método, objeto o instrucciones de líneas de comandos. También se usa para resaltar ciertas palabras técnicas.

Código

Para el código utilizamos un formato especial, el cual permite colorear ciertas palabras especiales, crear el efecto entre líneas y agregar el número de línea que ayude a referenciar el texto con el código.

```
1.  ReactDOM.render(
2.    <h1>Hello, world!</h1>,
3.    document.getElementById('root')
4.  );
```

El texto con fondo verde, lo utilizaremos para indicar líneas que se agregan al código existente.

~~Mientras que el rojo y tachado, es para indicar código que se elimina de un archivo existente.~~

Por otra parte, tenemos los íconos, que nos ayudan para resaltar algunas cosas:

Nuevo concepto: <concepto>
Cuando mencionamos un nuevo concepto o término que vale la pena resaltar.

Tip
Esta caja la utilizamos para dar un tip o sugerencia que nos puede ser de gran utilidad.

Importante
Esta caja se utiliza para mencionar algo muy importante.

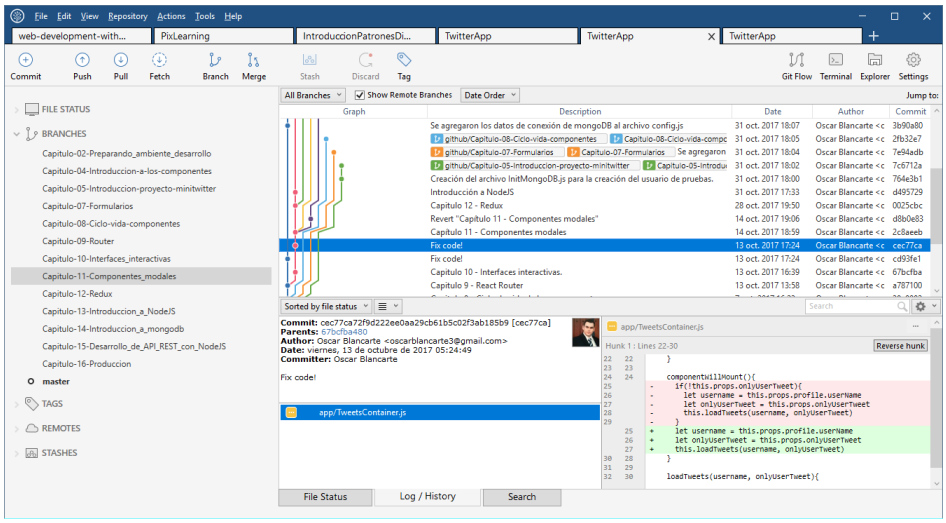
Error común
Esta caja se utiliza para mencionar errores muy comunes que pueden ser verdadero dolor de cabeza o para mencionar algo que puede prevenir futuros problemas.

Código fuente

Todo el código fuente de este libro está disponible en GitHub y lo puedes descargar en la siguiente URL:

https://github.com/oscarjb1/introduction-to-software-architecture

La segunda y más recomendable opción es, utilizar un cliente de Git, como lo es Source Tree y clonar el repositorio:



Requisitos previos

Si bien, he escrito este libro lo más simple y claro posible, cabe resaltar que este es un libro que requiere conocimientos sólidos en programación y el proceso de construcción de software en general.

Para comprender mejor todos los conceptos que explicaremos a lo largo de este libro también es necesario conocimiento sólido en patrones de diseño.

INTRODUCCIÓN

Desde los inicios de la ingeniería software, los científicos de la computación lucharon por tener formas más simples de realizar su trabajo, ya que las cuestiones más simples, como: imprimir un documento, guardar un archivo o compilar el código, eran tareas que podría tardar desde un día hasta una semana.

> Solo como anécdota, yo recuerdo que uno de mis maestros de programación me comentó que ellos para poder ejecutar un programa que solo sumara dos números, era necesario escribir una gran cantidad de código, grabarlo en tarjetas perforadas y llevarlo al departamento de informática y esperar desde 1 día hasta una semana para que dieran el resultado de la compilación, y si faltaba una coma o un punto, se tenía que repetir todo el proceso.

Hoy en día, existen herramientas que van indicando en tiempo real si hay errores de sintaxis en el código, pero no solo eso, además, son lo suficientemente inteligentes para autocompletar lo que se va escribiendo, incluso, nos ayudan a detectar posibles errores en tiempo de ejecución.

La realidad es que, a medida que la tecnología avanza, tenemos cada vez más herramientas a nuestra disposición, como: lenguajes de programación, IDE's, editores de código, frameworks, librerías, plataformas en la nube y una gran cantidad de herramientas que hacen la vida cada vez más simple, así que, los retos de hoy en día no son compilar el código, imprimir una hoja, guardar en una base de datos, tareas que antes eran muy difíciles. Lo curioso es que, hoy en día hay tantas alternativas para hacer cualquier cosa, y por increíble que parezca, **el reto de un programador hoy en día es, decidirse por qué tecnología irse**, eso es increíble, tenemos tantas opciones para hacer lo que sea, que el reto no es hacer las cosas, sino decidir con que tecnologías lo construiremos.

Ahora bien, yo quiero hacerte una pregunta, ¿crees que hacer un programa hoy en días es más fácil que hace años?

Seguramente a todos los que les haga esta pregunta concordarán con que hoy en día es más fácil, sin embargo, y por increíble que parezca, el hecho de que las tecnologías sean cada vez más simples, nos trae nuevas problemáticas y justo aquí donde quería llegar.

A medida que las tecnologías son más simples y más accesibles para todas las personas del mundo, las aplicaciones se enfrentan a retos que antes no existían, como la concurrencia, la seguridad, la alta disponibilidad, el performance, la usabilidad, la reusabilidad, testabilidad, funcionalidad, modificabilidad, portabilidad, integridad, escalabilidad, etc, etc. Todos estos son conceptos que prácticamente no existían en el pasado, porque las aplicaciones se desarrollaban para una audiencia muy reducida y con altos conocimientos técnicos, además, se ejecutaban en un Mainframe, lo que reducía drásticamente los problemas de conectividad o intermitencia, pues todo se ejecuta desde el mismo servidor.

Nuevo concepto: Mainframe
Los Mainframe son súper computadores capaces de realizar millos de instrucciones por segundo (MIPS). Son utilizadas para el procesamiento de grandes volúmenes de datos y pueden estar varios Mainframe conectados entre sí en la misma ubicación física (site).

Seguramente has escuchado alguna vez eso que dicen las mamás de hoy en día, "*¡Ay!, los niños de hoy son más inteligentes, desde bebés aprender a utilizar las tablets o las computadoras*". La realidad no es que los niños de ahora son más inteligentes, ya que el ser humano siempre ha sido inteligente, lo que pasa es que, cada vez se desarrollan aplicaciones con una usabilidad mejor y son altamente intuitivas, lo que ocasiona que cualquier persona pueda utilizarlas, incluso si conocimientos.

Pues bien, debido a todos estos cambios, más la llegada de nuevas arquitecturas como Cloud computing y SOA, Microservicios, REST, es que nace la necesidad del arquitecto de software, el cual tiene como principal responsabilidad asegurarse de que una aplicación cumpla con los **atributos de calidad**, entre otras responsabilidades más, como dictar los estándares de codificación, herramienta, plataformas y tecnologías, entre otras cosas que analizaremos más adelante.

Para los que no les quede claro que es un atributo de calidad, son por lo general los requerimientos no funcionales.

¿Alguien aquí ha escuchado que son los requerimientos funcionales y no funcionales?

Nuevo concepto: Requerimientos funcionales
los requerimientos funcionales son todos aquellos que nacen de la necesidad de los usuarios para cubrir un requerimiento de negocio y son solicitados explícitamente, como, por ejemplo, que el sistema me permite registrar clientes, crear facturas, administrar el inventario, etc.

Nuevo concepto: Requerimientos no funcionales
los requerimientos no funcionales son aquellos que no son solicitados explícitamente por los usuarios, pero van implícitos en un requerimiento, por ejemplo, que la aplicación sea rápida, segura y que pueda escalar si el número de usuarios aumenta, etc.

TIP: Los atributos de calidad son por lo general los requerimientos no funcionales
Si bien, analizaremos con detalles los atributos de calidad más adelante, imagina que los atributos de calidad son los requerimientos no funcionales, y que si bien, el cliente no te los solicitará, su ausencia podría implicar el fracaso de proyecto.

Entonces, un arquitecto deberá siempre cuidar que la aplicación cumpla con los atributos de calidad, y será su responsabilidad que una solución no solo cumpla los requerimientos solicitados explícitos (requerimientos funcionales), sino que, además, deberá cuidar los requerimientos no funcionales, incluso, si el cliente no los ha solicitado, pues su cumplimiento garantizará el funcionamiento de la aplicación por un largo periodo de tiempo.

Índice

Por dónde empezar

Capítulo 1

Para muchos, dominar la arquitectura de software es uno de los objetivos que han buscado durante algún tiempo, sin embargo, no siempre es claro el camino para dominarla, ya que la arquitectura de software es un concepto abstracto que cada persona lo interpreta de una forma diferente, dificultando con ello comprender y diseñar con éxito una arquitectura de software.

Para iniciar este libro y comprender mejor todos los conceptos que analizaremos será necesario que abramos la mente y nos borremos esa tonta idea de que un arquitecto es aquella persona que tiene más tiempo en la empresa, o la que conoce todo el sistema, o el que desarrollo la solución, todas esas cosas nos hacen caer en nuestros laureles y nos hacen pensar que ya somos arquitectos y que nadie más puede hacerlo mejor. Dicho lo anterior, aprendamos desde cero los conceptos básicos:

¿Qué es la arquitectura de software?

Dado que este es un libro de arquitectura de software, lo más normal es preguntarnos; entonces **¿Qué es la arquitectura de software?** Sin embargo, temo decirte que no existe un consenso para definir con exactitud que es la arquitectura de software, por el contrario, existe diversas publicaciones donde se dan definiciones diferentes, lo que hace complicado decir con exactitud que es la arquitectura de software, por ello, vamos a analizar algunas definiciones para tratar de buscar la definición que usaremos en este libro.

Cabe mencionar que las definiciones que analizaremos a continuación son solo alguna de las muchísimas definiciones que existen, te invito a que veas el artículo publicado por el Software Ingineering Institute titulado "*What Is Your Definition of Software Architecture*" el cual recopila una gran cantidad de definiciones realizadas por personas, publicaciones e instituciones. Te dejo el link a continuación:

https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513807

Veamos alguna de las definiciones más relevantes sobre que es la arquitectura de software:

> *"La arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema "*
>
> *— "An introduction to Software Architecture" de David Garlan y Mary Shaw*

> *"La Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. "*
>
> *— Software Engineering Institute (SEI)*

"El conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos, y las propiedades de ambos. "

— Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al, AddisonWesley, 2010

"La arquitectura de software de un programa o sistema informático es la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos. "

— Software Architecture in Practice (2nd edition), Bass, Clements, Kazman; AddisonWesley 2003

"La arquitectura se define como la organización fundamental de un sistema, encarnada en sus componentes, sus relaciones entre sí y con el entorno, y los principios que rigen su diseño y evolución "

— ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of SoftwareIntensive Systems

"Una arquitectura es el conjunto de decisiones importantes sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces mediante las cuales se compone el sistema "

— Rational Unified Process, 1999

"La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema "

— Wikipedia

"La arquitectura del software es "la estructura de los componentes de un programa/sistema, sus interrelaciones y los principios y directrices que rigen su diseño y evolución en el tiempo".

— Garlan & Perry, 1995

Como podemos observar, existen muchas definiciones sobre que es la arquitectura de software, lo que hace complicado desde un inicio dar una definición exacta y que deje conformes a todos, sin embargo, como arquitectos de software estamos forzados a poder dar una definición, lo que nos deja dos opciones, adoptar una de las existente o creamos nuestra propia definición basado en las anteriores.

Antes de elegir la que sería la definición que utilizaremos en este libro, quiero que analicemos las definiciones anteriores, podrás observar que, si bien todas son diferentes, todas coinciden en que la arquitectura se centra en la estructura del sistema, los componentes que lo conforman y la relación que existe entre ellos. Esto quiere decir que sin importar que definición utilicemos, debe de quedar claro que la arquitectura de software se centra en **la estructura del sistema, los componentes que lo conforman y la relación que existe entre ellos**.

Dicho lo anterior, a mí en lo particular me agrada la definición de Wikipedia, pues es sumamente minimalista, clara y concisa. Sin embargo, yo le agrego algo más para dejarla de la siguiente manera:

Nuevo concepto: Arquitectura de software
La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, el cual consiste en un conjunto de patrones y abstracciones que proporcionan un marco claro para la implementación del sistema.

Cabe mencionar que esta es la definición que para mí más se acerca a lo que es la arquitectura de software y que estaré utilizado a lo largo de este libro para

definirla, sin embargo, siéntete libre de adoptar una existente o hacer tu propia definición.

¿Qué son los patrones de diseño?

Es común que cuando hablamos de arquitectura de software salgan términos como patrones, sin embargo, existen dos tipos de patrones, los patrones de diseño y los patrones arquitectónicos, los cuales no son lo mismo y no deberían ser confundidos por ninguna razón.

En principio, un patrón de diseño es la solución a un problema de diseño, el cual debe haber comprobado su efectividad resolviendo problemas similares en el pasado, también tiene que ser reutilizable, por lo que se deben poder usar para resolver problemas parecidos en contextos diferentes.

Nuevo concepto: Patrones de diseño
es la solución a un problema de diseño, el cual debe haber comprobado su efectividad resolviendo problemas similares en el pasado, también tiene que ser reutilizable, por lo que se deben poder usar para resolver problemas parecidos en contextos diferentes.

Los patrones de diseño tienen su origen en la Arquitectura (construcción), cuando en 1979 el Arquitecto Christopher Alexander publicó el libro *Timeless Way of Building*, en el cual hablaba de una serie de patrones para la construcción de edificios, comparando la arquitectura moderna con la antigua y cómo la gente había perdido la conexión con lo que se considera calidad.

Él utilizaba las siguientes palabras: "Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez."

Más tarde, Christopher Alexander y sus colegas publicaron el volumen *A Pattern Language* en donde intentaron formalizar y plasmar de una forma práctica las generaciones de conocimientos arquitectónicos. En la obra se refieren a los patrones arquitectónicos de la siguiente manera:

Los patrones no son principios abstractos que requieran su redescubrimiento para obtener una aplicación satisfactoria, ni son específicos a una situación particular o cultural; son algo intermedio. Un patrón define una posible solución correcta para un problema de diseño dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones. (1977).

Entre las cosas que se describen en el libro se encuentra la forma de diseñar la red de transporte público, carreteras, en qué lugar deben ir las perillas de las puertas, etc.

Hasta ese momento los patrones conocidos tenían un enfoque arquitectónico y hablan de cómo construir estructuras, pero fue hasta 1987 cuando Ward Cunningham y Kent Beck, motivados por el pobre entrenamiento que recibían los nuevos programadores en programación orientación a objetos, se dieron cuenta de la gran semejanza que existían entre una buena arquitectura propuesta por Christopher Alexander y la buena arquitectura de la programación orientada a objetos. De tal manera que utilizaron gran parte del trabajo de Christopher para diseñar cinco patrones de interacción hombre-máquina y lo publicaron en el artículo OOPSLA-87 bajo el título *Using Pattern Languages for OO Programs*.

Sin embargo, fue hasta principios de la década de 1990 cuando los patrones de diseño tuvieron su gran debut en el mundo de la informática a partir de la publicación del libro *Design Patterns*, escrito por el grupo Gang of Four (GoF) compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, en el que se recogían 23 patrones de diseño comunes que ya se utilizaban sin ser reconocidos como patrones de diseño.

Tipos de patrones de diseño

Los patrones de diseño se dividen en tres tipos, las cuales agrupan los patrones según el tipo de problema que buscan resolver, los tipos son:

Patrones Creacionales: Son patrones de diseño relacionados con la creación o construcción de objetos. Estos patrones intentan controlar la forma en que los objetos son creados, implementando mecanismos que eviten la creación directa de objetos.

Patrones Estructurales: Son patrones que tiene que ver con la forma en que las clases se relacionan con otras clases. Estos patrones ayudan a dar un mayor orden a nuestras clases ayudando a crear componentes más flexibles y extensibles.

Patrones de Comportamiento: Son patrones que están relacionados con procedimientos y con la asignación de responsabilidad a los objetos. Los patrones de comportamiento engloban también patrones de comunicación entre ellos.

¿Cómo diferenciar un patrón de diseño?

Una de las grandes problemáticas a la hora de identificar los patrones de diseño, es que se suelen confundir los patrones arquitectónicos que más adelante analizaremos.

Como regla general, los patrones de diseño tienen un impacto relativo con respecto a un componente, esto quiere decir que tiene un impacto menor sobre todo el componente. Dicho de otra forma, si quisiéramos quitar o remplazar el patrón de diseño, solo afectaría a las clases que están directamente relacionadas con él, y un impacto imperceptible para el resto de componentes que conforman la arquitectura.

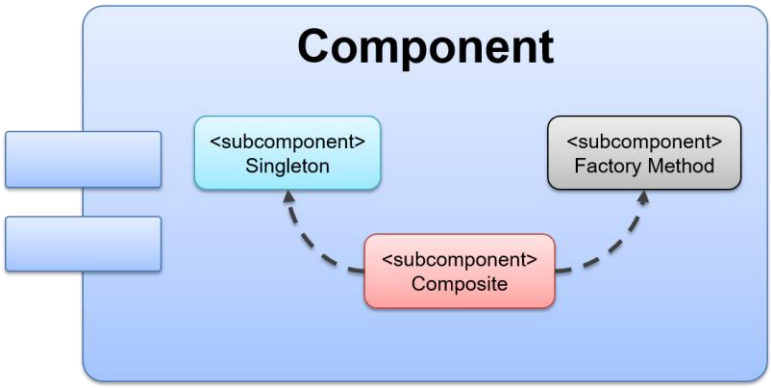


Fig 1: Muestra un componente con 3 patrones de diseño implementados.

Como podemos ver en la imagen anterior, un componente puede implementar varios patrones de diseño, sin embargo, estos quedan ocultos dentro del componente, lo que hace totalmente transparente para el resto de módulos los patrones implementados, las clases utilizadas y las relaciones que tiene entre sí.

Como regla general, los patrones de diseño se centran en cómo las clases y los objetos se crean, relacionan, estructuran y se comportan en tiempo de ejecución, pero siempre, centrado en las clases y objetos, nunca en componentes.

Otra de las formas que tenemos para identificar un patrón de diseño es analizar el impacto que tendría sobre el componente que caso de que el patrón de diseño fallara, en tal caso, un patrón de diseño provocaría fallas en algunas operaciones del componente, pero el componte podría seguir operando parcialmente.

Cuando un arquitecto de software analiza un componente, siempre deberá crear abstracciones para facilitar su análisis, de tal forma, que eliminará todos aquellos aspectos que no son relevantes para la arquitectura y creará una representación lo más simple posible.

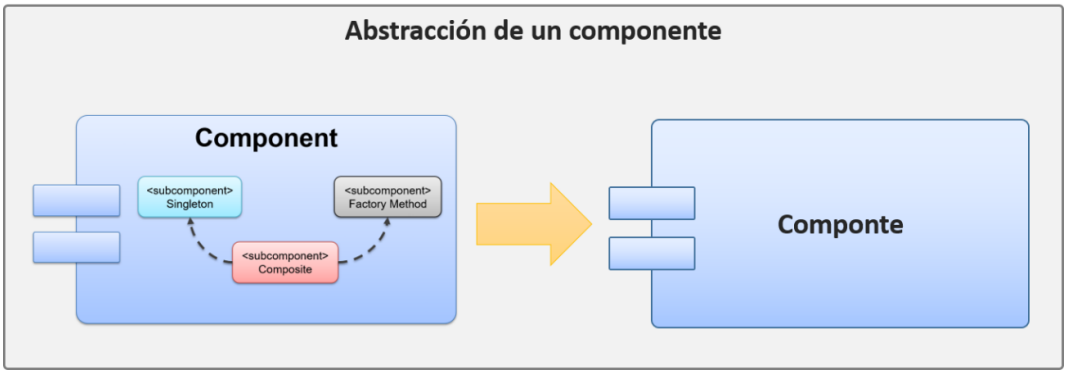


Fig 2: Abstracción de un componente

Nuevo concepto: Abstracción
Es el proceso por medio del cual aislamos a un elemento de su contexto o resto de elementos que lo acompañan con el propósito de crear una representación más simple de él, el cual se centra en lo que hace y no en como lo hace.

Como podemos ver en la imagen anterior, un arquitecto debe de centrarse en los detalles que son realmente relevantes para la arquitectura y descartar todos aquellos que no aporten al análisis o evaluación de la arquitectura.

Sé que en este punto estarás pensando, pero un arquitecto también debe de diseñar la forma en que cada componente individual debe de trabajar, y eso es correcto, sin embargo, existe otra etapa en la que nos podremos preocupar por detalles de implementación, por hora, recordemos que solo nos interesan los componentes y la forma en que estos se relacionan entre sí.

¿Dónde puedo aprender patrones de diseño?

Como comentamos hace un momento, este libro no se centra en la enseñanza de los patrones de diseño, sino al contrario, es un pre-requisito para poder comprender mejor este libro.

Si quieres regresar un paso atrás y aprender de lleno los patrones de diseño, te invito a que veas mi otro libro "**Introducción a los patrones de diseño – Un enfoque práctico**" en el cual explico de lleno los principales patrones de diseño con ejemplos del mundo real, los cuales podrás descargar y ejecutar directamente en tu equipo.

El libro lo podrás ver en: https://reactiveprogramming.io/books/design-patterns/es

¿Qué son los patrones arquitectónicos?

A diferencia de los patrones de diseño, los patrones arquitectónicos tienen un gran impacto sobre el componente, lo que quiere decir que cualquier cambio que se realice una vez construido el componente podría tener un impacto mayor.

> *"Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares a los patrones de diseño de software, pero tienen un alcance más amplio. Los patrones arquitectónicos abordan diversos problemas en la ingeniería de software, como las limitaciones de rendimiento del hardware del equipo, la alta disponibilidad y la minimización de un riesgo empresarial. "*
>
> *— Wikipedia*

> *"Expresa una organización estructural fundamental o esquema para sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para organizar las relaciones entre ellos. "*
>
> *— TOGAF*

> *"Los patrones de una arquitectura expresan una organización estructural fundamental o esquema para sistemas complejos. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades únicas e incluye las reglas y pautas que permiten la toma de decisiones para organizar las relaciones entre ellos. El patrón de arquitectura para un sistema de software ilustra la estructura de nivel macro para toda la solución de software. Un patrón arquitectónico es un conjunto de principios y un patrón de grano grueso que proporciona un marco abstracto para una familia de sistemas. Un patrón arquitectónico mejora la partición y promueve la reutilización del diseño al proporcionar soluciones a problemas recurrentes con frecuencia. Precisamente hablando, un patrón arquitectónico comprende un conjunto de principios que dan forma a una aplicación."*
>
> *— Achitectural Patterns - Pethuru Raj, Anupama Raman, Harihara Subramanian*

"Los patrones de arquitectura ayudan a definir las características básicas y el comportamiento de una aplicación."

— Software Architecture Patterns - Mark Richards

"Los patrones arquitectónicos son un método para organizar bloques de funcionalidad para satisfacer una necesidad. Los patrones se pueden utilizar a nivel de software, sistema o empresa. Los patrones bien expresados le dicen cómo usarlos, y cuándo. Los patrones se pueden caracterizar según el tipo de solución a la que se dirigen. "

— MITRE

Nuevamente podemos observar que existen diversas definiciones para lo que es un patrón arquitectónico, incluso, hay quienes utilizan el termino patrón para referirse a los patrones de diseño y patrones arquitectónicos, sin embargo, queda claro que existe una diferencia fundamental, los patrones de diseño se centran en las clases y los objetos, es decir, como se crean, estructuran y cómo se comportan en tiempo de ejecución, por otra parte, los patrones arquitectónicos tiene un alcance más amplio, pues se centra en como los componentes se comportan, su relación con los demás y la comunicación que existe entre ellos, pero también abordar ciertas restricciones tecnológicas, hardware, performance, seguridad, etc.

Los patrones de diseño son diferentes a los patrones arquitectónicos.
Tanto los patrones de diseño cómo los patrones arquitectónicos pueden resultar similares ante un arquitecto inexperto, pero por ninguna razón debemos confundirlos, ya son cosas diferentes.

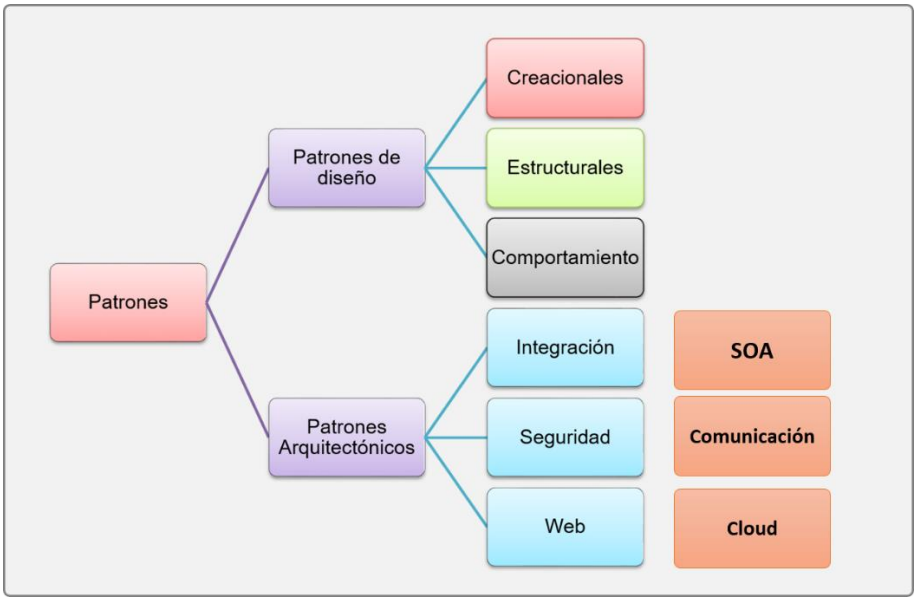


Fig 3: Tipos de patrones

La siguiente imagen nos ayudará a entender un poco mejor los distintos patrones de diseño y patrones arquitectónicos que existen para ayudarnos a diferenciarlos mejor. Cabe mencionar que existen muchos más tipos de patrones arquitectónicos que los que hay en la imagen.

Más adelante tendremos un capítulo dedicado exclusivamente para analizar los principales patrones arquitectónicos.

¿Cómo diferenciar un patrón arquitectónico?

Los patrones arquitectónicos son fáciles de reconocer debido a que tiene un impacto global sobre la aplicación, e incluso, el patrón rige la forma de trabajar o comunicarse con otros componentes, es por ello que a cualquier cambio que se realice sobre ellos tendrá un impacto directo sobre el componente, e incluso, podría tener afectaciones con los componentes relacionados.

Un ejemplo típico de un patrón arquitectónico es el denominado "Arquitectura en 3 capas", el cual consiste en separar la aplicación 3 capas diferentes, las cuales corresponden a la capa de presentación, capa de negocio y capa de datos:

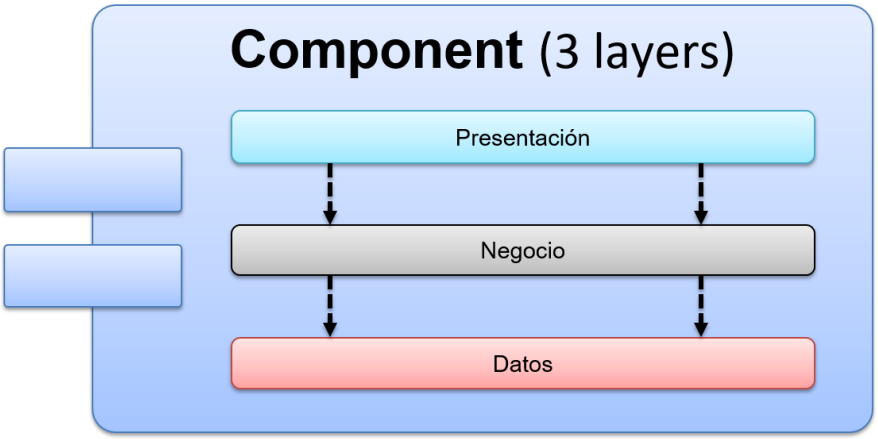


Fig 4: Arquitectura en 3 capas

En la imagen anterior podemos apreciar un componte que implementa la arquitectura de 3 capas, dicho patrón arquitectónico tiene la finalidad de separar la aplicación por capas, con la finalidad de que cada capa realiza una tarea específica. La capa de presentación tiene la tarea de generar las vistas, la capa de negocio tiene la responsabilidad de implementar la lógica de negocio, cómo implementar las operaciones, realizar cálculos, validaciones, etc, por último, la capa de datos tiene la responsabilidad de interactuar con la base de datos, como guardar, actualizar o recuperar información de la base de datos.

Cuando el usuario entra a la aplicación, lo hará a través de la capa de presentación, pero a medida que interactúe con la aplicación, este requerirá ir a la capa de negocio para consumir los datos, finalmente la capa de datos es la que realizará las consultas a la base de datos.

Dicho lo anterior, si alguna de las 3 capas falla, tendremos una falla total de la aplicación, ya que, si la capa de presentación falla, entonces el usuario no podrá ver nada, si la capa de negocios falla, entonces la aplicación no podrá guardar o solicitar información a la base de datos y finalmente, si la capa de datos falla, entonces no podremos recuperar ni actualizar información de la base de datos. En

cualquiera de los casos, en usuario quedará inhabilitado para usar la aplicación, tendiendo con ello una falla total de la aplicación.

Por otra parte, si decidimos cambiar el patrón arquitectónico una vez que la aplicación ha sido construida, tendremos un impacto mayor, pues tendremos que modificar todas las vistas para ya no usar la capa de negocios, la capa de negocio tendrá que cambiar para ya no acceder a la capa de datos, y la capa de datos quizás tenga que cambiar para adaptarse al nuevo patrón arquitectónico, sea como sea, la aplicación tendrá un fuerte impacto.

Además del impacto que tendrá el componente, hay patrones que su modificación podría impactar a otros componentes, incluso, compontes externos que están fuera de nuestro dominio, lo que complicaría aún más las cosas.

¿Qué son los estilos arquitectónicos?

El último término que deberemos aprender por ahora son los estilos arquitectónicos, los cuales también son diferentes a los patrones arquitectónicos.

Para comprender que es un estilo arquitectónico, es necesario regresarnos un poco a la arquitectura tradicional (construcción), para ellos, un estilo arquitectónico es un **método específico de construcción, caracterizado por las características que lo hacen notable y se distingue por las características que hacen que un edificio u otra estructura sea notable o históricamente identificable.**

En el software aplica exactamente igual, pues un estilo arquitectónico determina las características que debe tener un componente que utilice ese estilo, lo cual hace que sea fácilmente reconocible. De la misma forma que podemos determinar a qué periodo de la historia pertenece una construcción al observar sus características físicas, materiales o método de construcción, en el software podemos determinar que estilo de arquitectura sigue un componente al observar sus características.

Entonces, ¿Qué son los estilos arquitectónicos?, veamos algunas definiciones:

"Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural; Un vocabulario de componentes y conectores, con restricciones sobre cómo se pueden combinar. "

— M. Shaw and D. Garlan, Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996.

"Un estilo de arquitectura es una asignación de elementos y relaciones entre ellos, junto con un conjunto de reglas y restricciones sobre la forma de usarlos"

—Clements et al., 2003

"Un estilo arquitectónico es una colección con nombre de decisiones de diseño arquitectónico que son aplicables en un contexto de desarrollo dado, restringen decisiones de diseño arquitectónico que son específicas de un sistema particular dentro de ese contexto, y obtienen cualidades beneficiosas en cada uno sistema resultante. "

—R. N. Taylor, N. Medvidović and E. M. Dashofy, Software architecture: Foundations, Theory and Practice. Wiley, 2009.

"La arquitectura del software se ajusta a algún estilo. Por lo tanto, como cada sistema de software tiene una arquitectura, cada sistema de software tiene un estilo; y los estilos deben haber existido desde que se desarrolló el primer sistema de software. "

—Giesecke et al., 2006; Garlan et al., 2009.

Nuevamente podemos observar que no existe una única definición para describir lo que es un estilo arquitectónico, lo que hace difícil tener una definición de referencia. En mi caso, me gusta decir que un estilo arquitectónico es:

Nuevo concepto: Estilo arquitectónico
Un estilo arquitectónico establece un marco de referencia a partir del cual es posible construir aplicaciones que comparten un conjunto de atributos y características mediante el cual es posible identificarlos y clasificarlos.

Como siempre, siéntete libre de adoptar la definición que creas más acertada.

Los estilos arquitectónicos son diferentes a los patrones arquitectónicos.
A pesar de que puedan parecer similares por su nombre, los patrones arquitectónicos y los estilos arquitectónicos son diferentes y por ningún motivo deben de confundirse.

La relación entre patrones de diseño, arquitectónicos y estilos arquitectónicos

Para una gran mayoría de los arquitectos, incluso experimentados, les es complicado diferenciar con exactitud que es un patrón de diseño, un patrón arquitectónico y un estilo arquitectónico, debido principalmente a que, como ya vimos, no existe definiciones concretas de cada uno, además, existe una línea muy delgada que separa a estos tres conceptos.

Para comprender mejor estos conceptos será necesario de apoyarnos de la siguiente imagen:

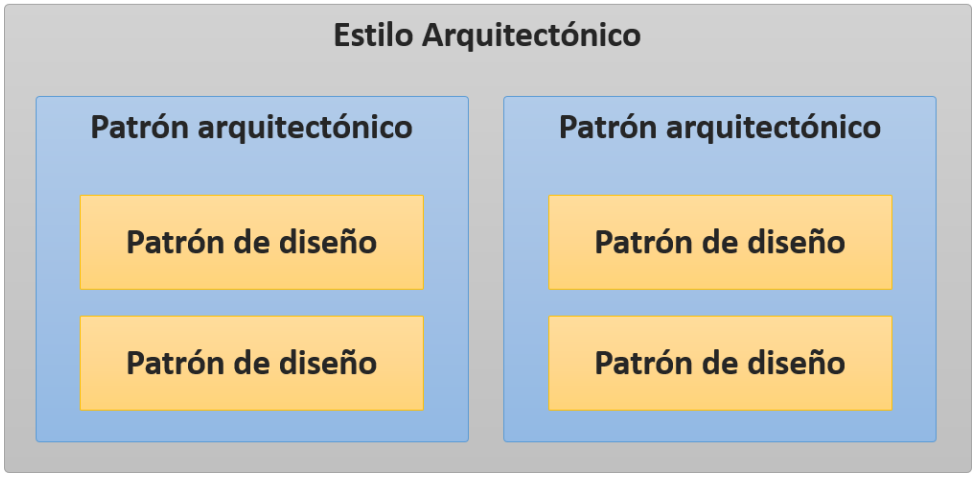


Fig 5: La relación entre patrones de diseño, patrones arquitectónicos y estilos arquitectónicos

Como podemos ver en la imagen, los estilos arquitectónicos son los de más alto nivel, y sirve como base para implementar muchos de los patrones arquitectónicos que conocemos, de la misma forma, un patrón arquitectónico puede ser implementado utilizando uno o más patrones de diseño. De esta forma, tenemos que los patrones de diseño son el tipo de patrón más específico y que centra en resolver como las clases se crean, estructura, relacionan y se comportan en tiempo de ejecución, por otra parte, los patrones arquitectónicos se enfocan en los componentes y como se relacionan entre sí, finalmente, los estilos arquitectónicos son marcos de referencia mediante los cuales es posible basarse para crear aplicaciones que compartan ciertas características.

Aun con esta explicación, siempre existe una especial confusión entre los patrones arquitectónicos y los estilos arquitectónicos, es por ello que debemos de recordar que un patrón arquitectónico existe para resolver un problema recurrente, mientras que los estilos arquitectónicos no existen para resolver un problema concreto, si no que más bien sirve para nombrar un diseño arquitectónico recurrente.

Estilo arquitectónico
Los estilos arquitectónicos no existen para resolver un problema de diseño, en su lugar, sirven para nombrar un diseño arquitectónico recurrente.

A pesar de que estos 3 conceptos tengan un propósito diferente (aunque relacionado), es importante que cualquier arquitecto de software entienda la diferencia, pues confundirlos puede ser un error grave, es por ello que en este libro nos centraremos en aprender de lleno los principales estilos y patrones arquitectónicos, dejando de lado los patrones de diseño, pues asumimos que a estas alturas ya los dominas.

Si quieres profundizar en los patones de diseño, te puedo recomendar que revises mi libro de "**Introducción a los patrones de diseño – un enfoque práctico**", en el cual explicamos los 25 principales patrones de diseño utilizando ejemplos del mundo real.