UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

ESTRUCTURA DE COMPONENTES

OBJETIVO

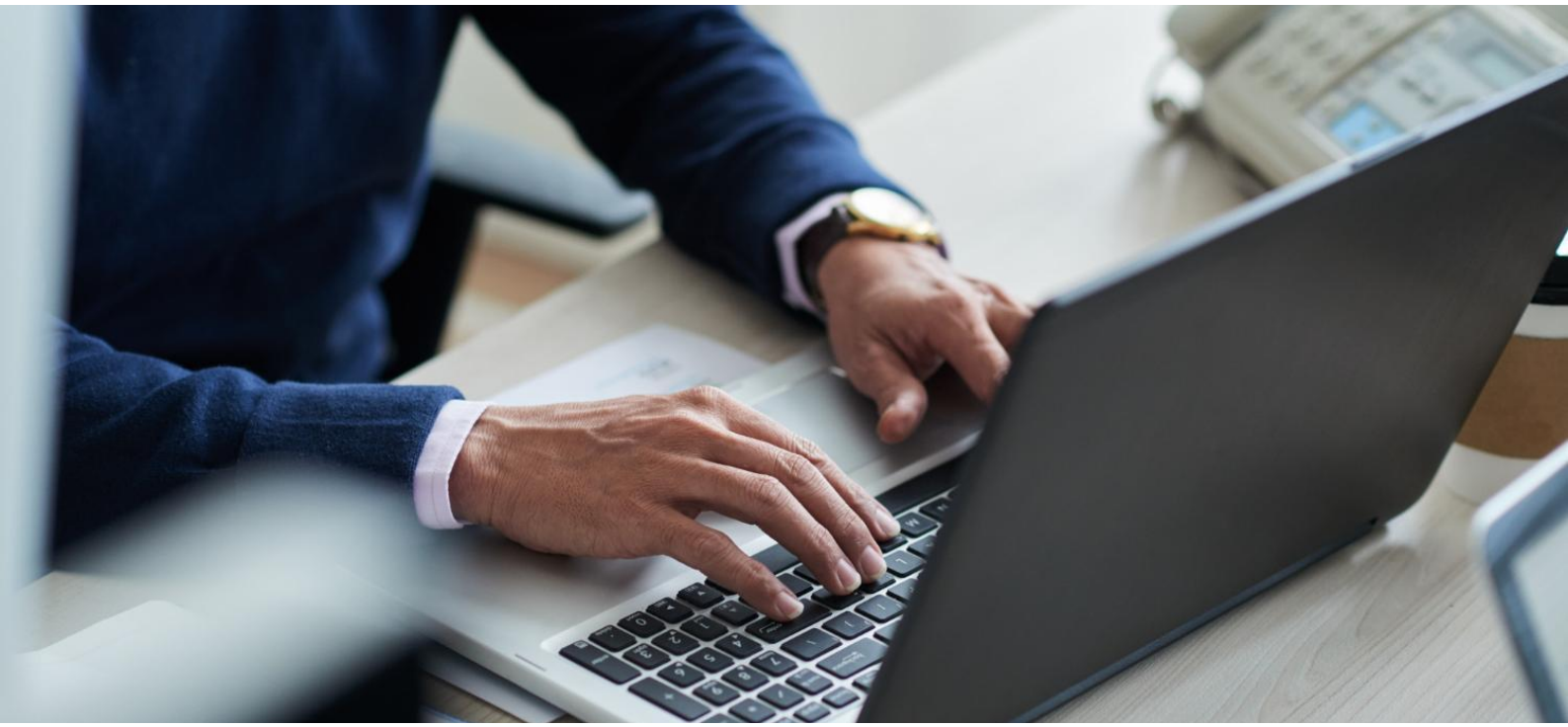
Identificar la estructura de los componentes del *software* como elemento clave de la modularidad del diseño o del cumplimiento de las cualidades arquitectónicas.

TABLA DE CONTENIDO

UCAB

Un *software* actual puede tener cientos de miles de líneas de código. Esto lo hace **inmanejable**, muy difícil para hacerle un cambio. Esta situación se complica cada vez más, porque cada vez los *softwares* están más interconectados.

Entonces es una buena práctica agrupar el código, lo que nos lleva al concepto de **componente**. Esta organización del código está fuertemente marcada por las tecnologías (ambientes de desarrollo) que se utilicen para su construcción.



A su vez, se hace necesario "organizar" estos componentes. Aquí lo llamaremos "estructurar". Estas estructuras pueden ser de diferentes maneras: algunas de ellas son muy exitosas en ciertas situaciones; estas estructuras son lo que conoceremos más adelante como **estilos arquitectónicos**. Por ello, es clave cómo vamos a organizar (estructurar) nuestros componentes en la actividad de diseño.

Por lo pronto, vamos a dejar en claro el concepto de componente y el de estructura de componentes.

01 Componente

Los componentes consisten en clases o funciones (dependiendo de la plataforma de implementación), cuyo diseño cae bajo la responsabilidad de los líderes tecnológicos o desarrolladores. Ofrecen un mecanismo específico del lenguaje para **agrupar** el código; a menudo se les anida para crear estratificación.

El componente más simple es un envoltorio de código: es un nivel más alto de **modularidad** que las clases (o funciones, en lenguajes no orientados a objetos). Este contenedor simple a menudo se denomina "biblioteca", ya que tiende a ejecutarse en la misma dirección de memoria que el código de llamada y se comunica a través de mecanismos de llamada de función del lenguaje. Las bibliotecas suelen ser dependientes en tiempo de compilación.

Estos componentes, a su vez, se organizan de diferentes maneras. ¡Veamos cómo!

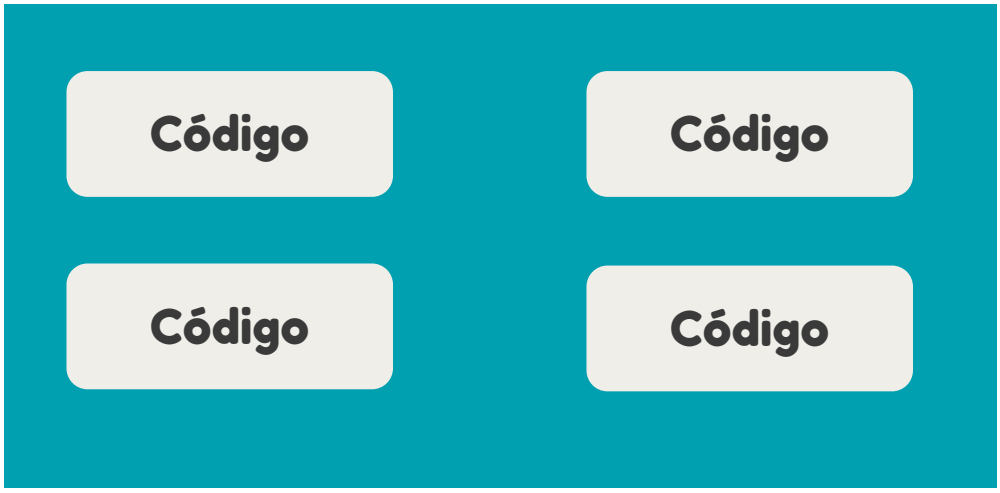
02 Estructuración de componentes

Aquí entra en juego la experticia del arquitecto de software: este es el rol, dentro del proceso de desarrollo, que tiene la responsabilidad de proponer y **custodiar** la estructura de componentes del *software* que se está desarrollando.

Claro está, él no trabaja solo, sino que colaborativamente analiza los pros y los contras de las posibles y conocidas estructuras, en función de lo que se desea construir.

Hay dos formas de organizar los componentes:

1. Agrupados en una sola unidad:



Fuente: Agrupados en una sola unidad. Adaptada de Richards y Ford (2020)

2 Envoltura simple o en capas

Fuente: Envoltura simple. Adaptada de Richards y Ford (2020)

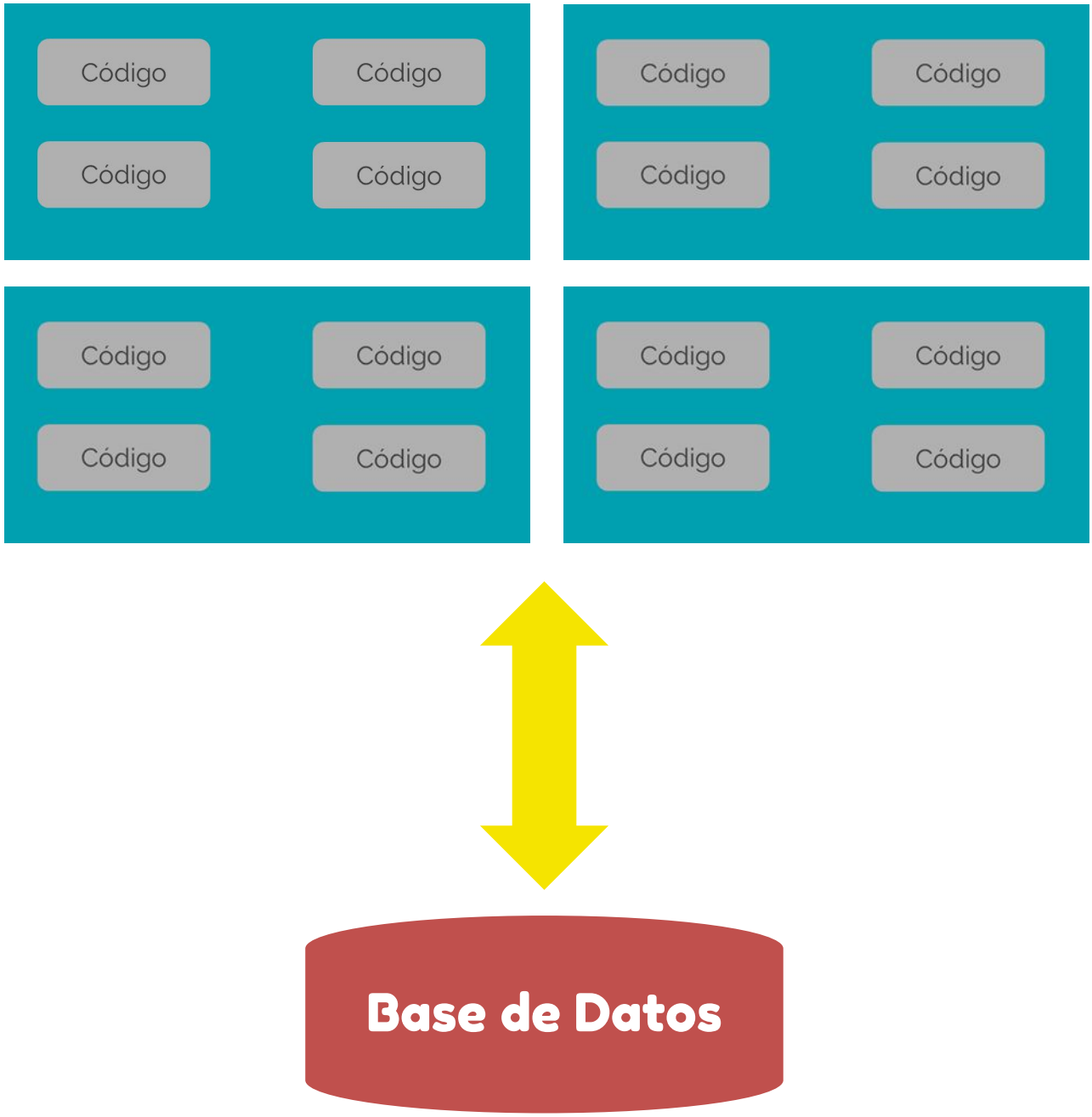
El arquitecto define, refina, gestiona y gobierna los componentes dentro de una arquitectura. Los **arquitectos de *software***, en colaboración con analistas de negocios, expertos en la materia, desarrolladores, ingenieros de control de calidad, arquitectos de operaciones y empresariales, crean el diseño inicial del software, incorporando las características de la arquitectura y los requisitos (básicamente los no funcionales) que exige el mismo.

Dado que los componentes representan un mecanismo de transporte de contenedores general, un arquitecto puede crear cualquier tipo de partición que desee. Existen varios estilos comunes, con diferentes conjuntos de trade-off (balance). Trade-off porque cada partición (estructuración) tiene **ventajas y desventajas**. La idea es seleccionar la que menos desventajas tiene.

Esta partición, veremos más adelante, se conoce como estilo arquitectónico.

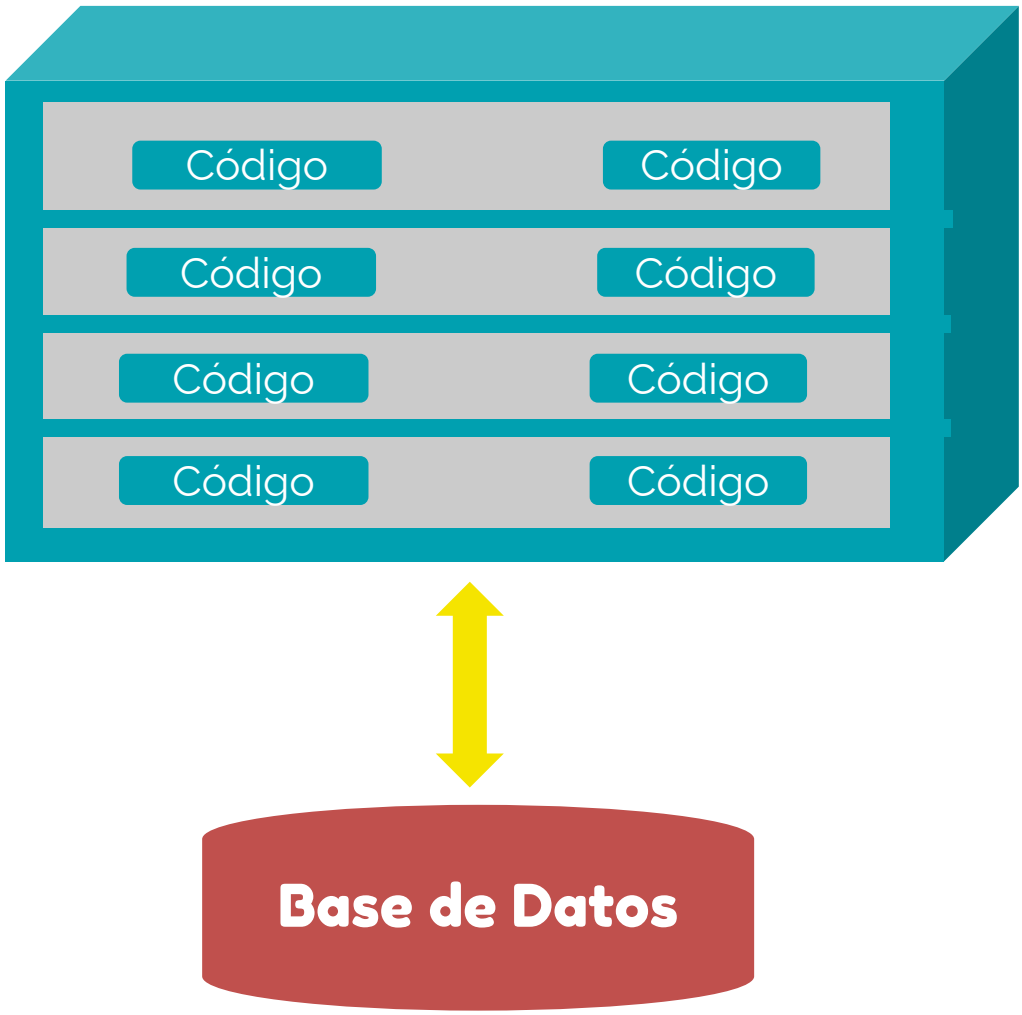
Sin embargo, el arquitecto debe comenzar por una partición (estructuración) básica, que luego se irá modificando en cada *sprint*. Esta partición básica puede ser de dos tipos:

- Monolítica:



Fuente: Monolítica. Adaptada de Richards y Ford (2020)

- Por capas:

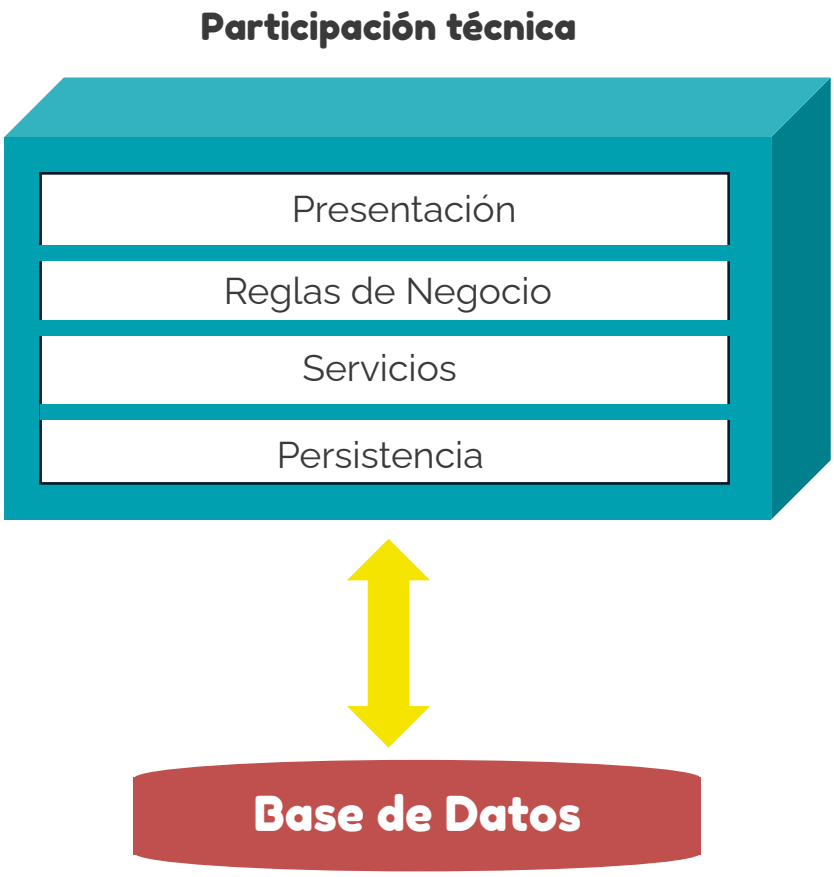


Fuente: Por capas. Adaptada de Richards y Ford (2020)

Las particiones monolíticas se hacen en función del dominio (figura 5):

Fuente: Partición por dominio. Adaptada de Richards y Ford (2020)

Y las de capa se hacen por el tipo de técnica que realiza cada capa (figura 6):

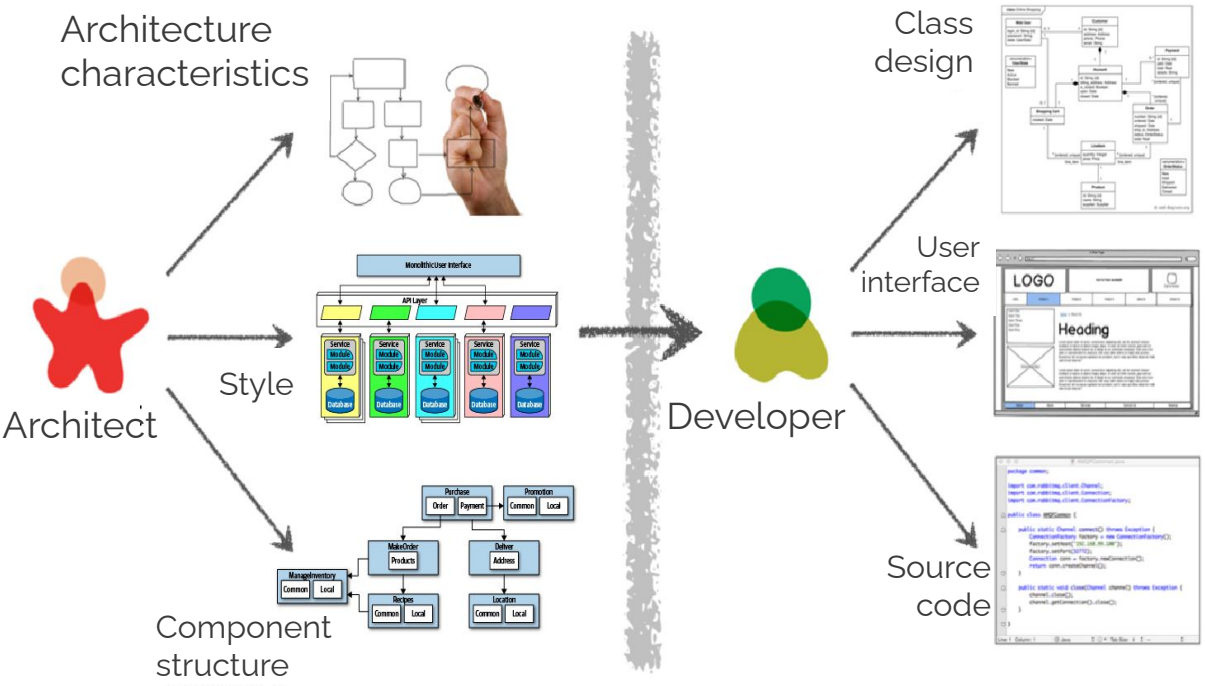


Fuente: Partición técnica. Adaptada de Richards y Ford (2020)

¿Qué hacen los desarrolladores?:

Toman componentes diseñados conjuntamente con el arquitecto y los **subdividen** en clases, funciones o subcomponentes. En general, el diseño de clases y funciones es responsabilidad compartida de arquitectos, líderes tecnológicos y desarrolladores, aunque la mayor parte se destina a los desarrolladores.

La siguiente figura muestra, en términos de diagramas, cuáles son las responsabilidades de cada actor:



Fuente: Responsabilidades. Extraída de Richards y Ford (2020)

El **arquitecto** se responsabiliza de:

- La elaboración de las cualidades de la arquitectura (estas se derivan de los requisitos no funcionales).
- Proponer los estilos arquitectónicos (pueden ser varios en una misma aplicación).
- Proponer la estructura de componentes.

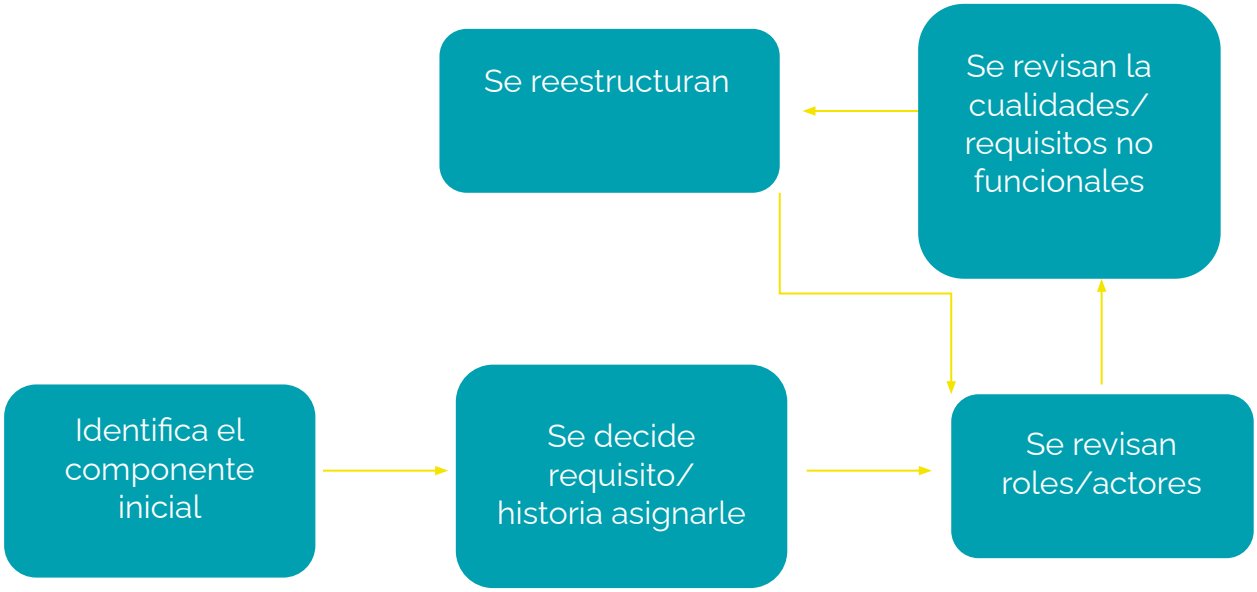
El(os) **desarrollador(es)**, por su parte, es(on) responsable(s) de**:**

- Definir el Diagrama de clases de diseño para cada *Sprint* (en cada Sprint se desarrollan uno o varios componentes).
- Definir las interfaces hombre-máquina (aquí puede existir la participación de un diseñador gráfico).
- Implementar el código correspondiente.

Para ello, los desarrolladores **toman** los componentes, diseñados conjuntamente con el arquitecto, y los subdividen en clases, funciones o subcomponentes.

¿Cómo lo hacen?

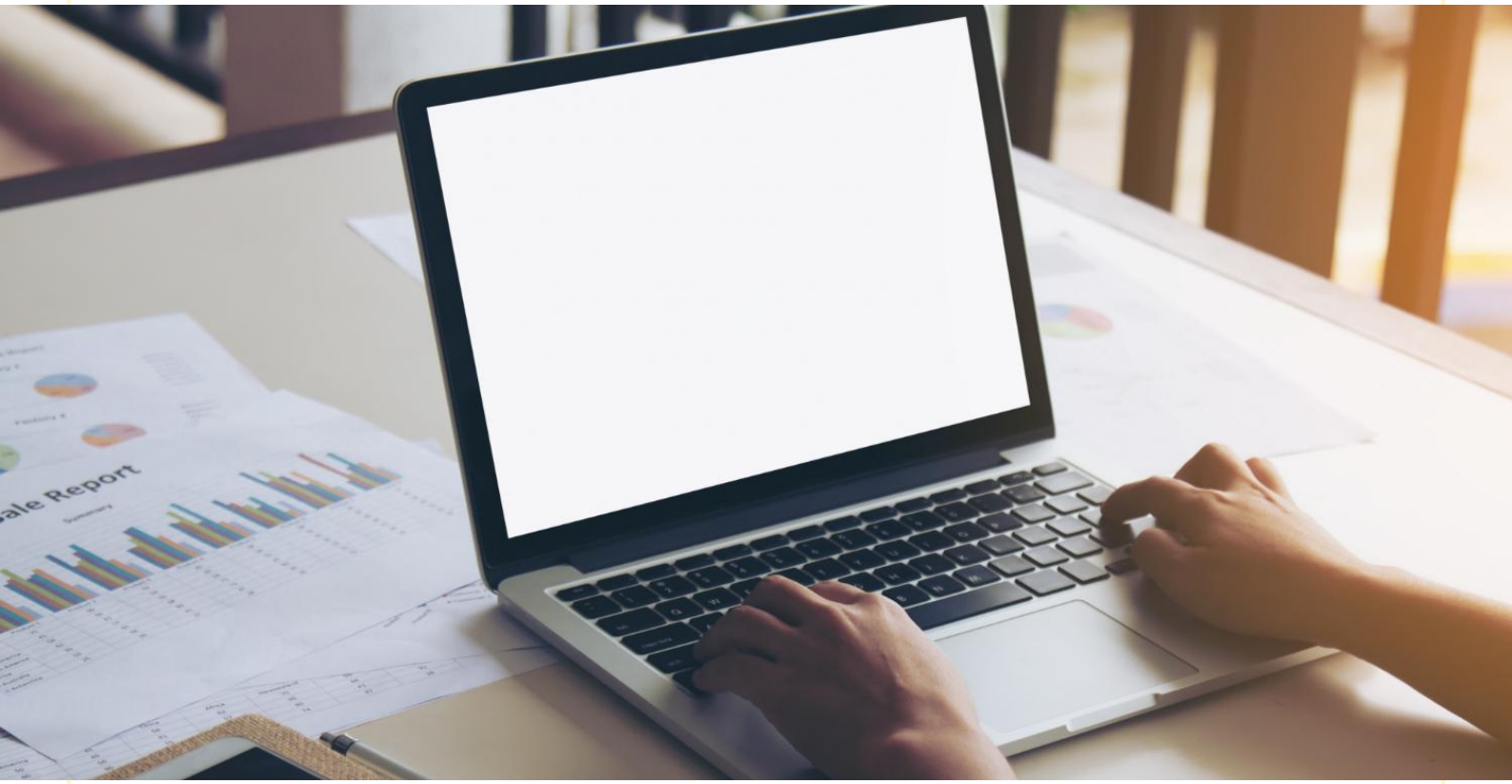
Siguen el proceso que se muestra en la figura, siempre de manera colaborativa:



Fuente: Proceso de subdivisión de componentes. Extraída de Richards y Ford (2020)

La **retroalimentación** es fundamental en el diseño de *software*. Los arquitectos deben iterar continuamente en el diseño de sus componentes con los desarrolladores. El diseño de *software* proporciona todo tipo de dificultades inesperadas: nadie puede anticipar todos los problemas desconocidos que suelen ocurrir durante los proyectos de *software*.

Por lo tanto, un enfoque **iterativo** para el diseño de componentes es clave. Es prácticamente imposible tener en cuenta los diferentes descubrimientos y casos extremos que surgirán. Además, a medida que la arquitectura y los desarrolladores profundizan en la construcción de la aplicación, obtienen un *trade-off* más matizado de dónde deberían estar el comportamiento y los roles.

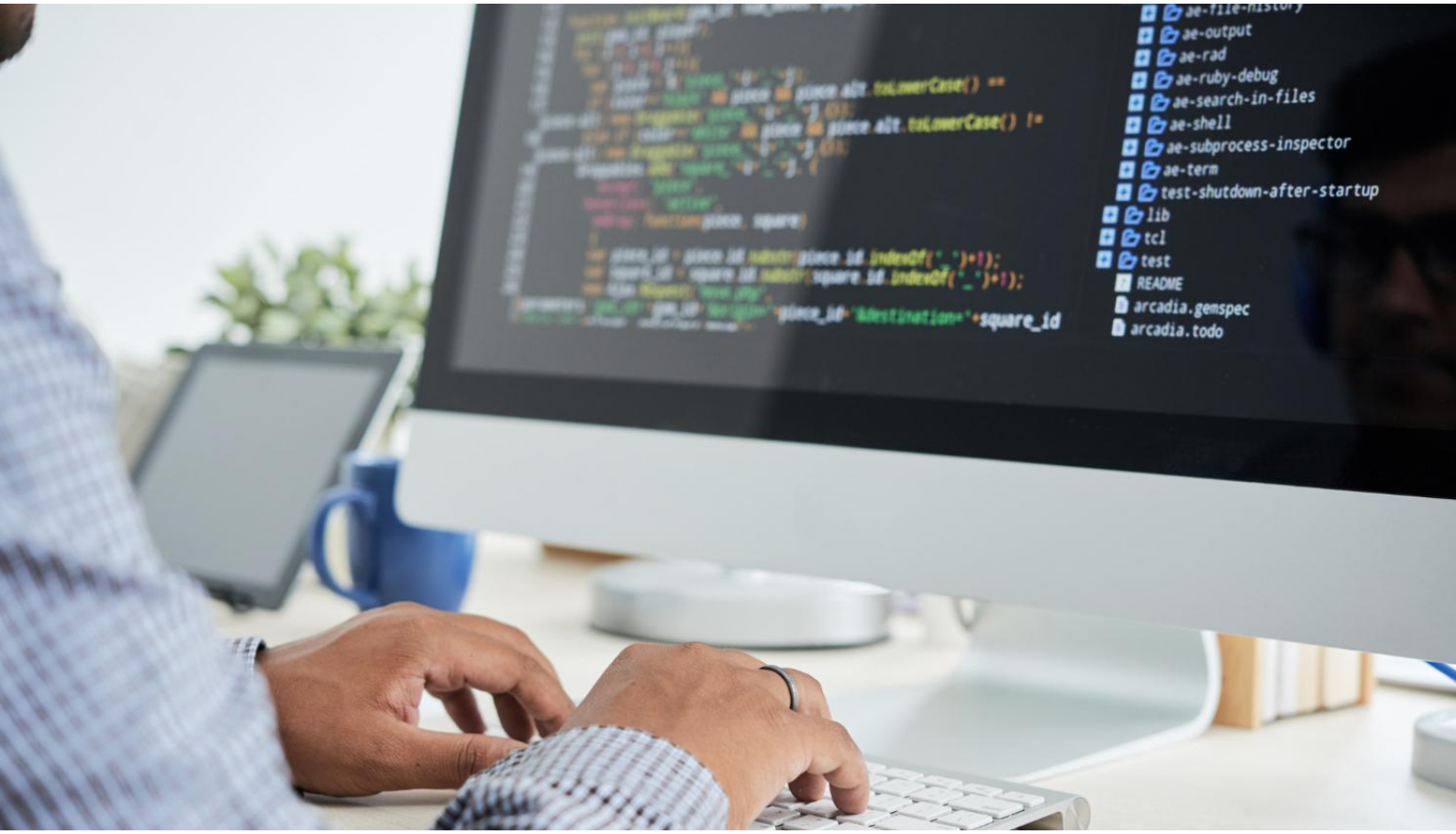


Encontrar la granularidad adecuada para los componentes es una de las tareas más difíciles de un arquitecto. Un diseño de componente demasiado detallado conduce a **demasiada** comunicación entre los componentes para lograr resultados.

Los componentes demasiado gruesos fomentan un alto acoplamiento interno, lo que conduce a dificultades en la capacidad de implementación y la capacidad de prueba, así como a **efectos secundarios** negativos, relacionados con la modularidad.

Nadie pone en duda que los ***softwares*** están en continua evolución: cada día aparecen "nuevas necesidades" que conllevan a la modificación (mejoras/cambios) del código que implementa el mismo. Además, hay una gran presión porque estos cambios sean realizados a la brevedad. Los mecanismos, entonces usados para responder a esto, son propios de la modularidad del diseño. El elemento clave para esta **modularidad** es el **componente**.

Ahora bien, un *software* con muchas líneas de código tendrá entonces muchos componentes. Estos deben, a su vez, ser organizados (estructurados) de forma tal que se satisfagan los requisitos funcionales lo mejor posible (ahora llamados "cualidades arquitectónicas"). Esta estructuración es responsabilidad del arquitecto, quien trabaja colaborativamente con el desarrollador. Hay dos formas iniciales de estructurar los componentes: monolítica y por capas. Son básicas y al inicio, cuando se tiene mucha incertidumbre sobre cómo quedará la estructuración al final, ellas son las más recomendadas para comenzar.



Richard, M. y Ford, N. (2020). Agrupados en una sola unidad [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

Richard, M. y Ford, N. (2020). Envoltura simple [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

Richard, M. y Ford, N. (2020). Monolítica [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

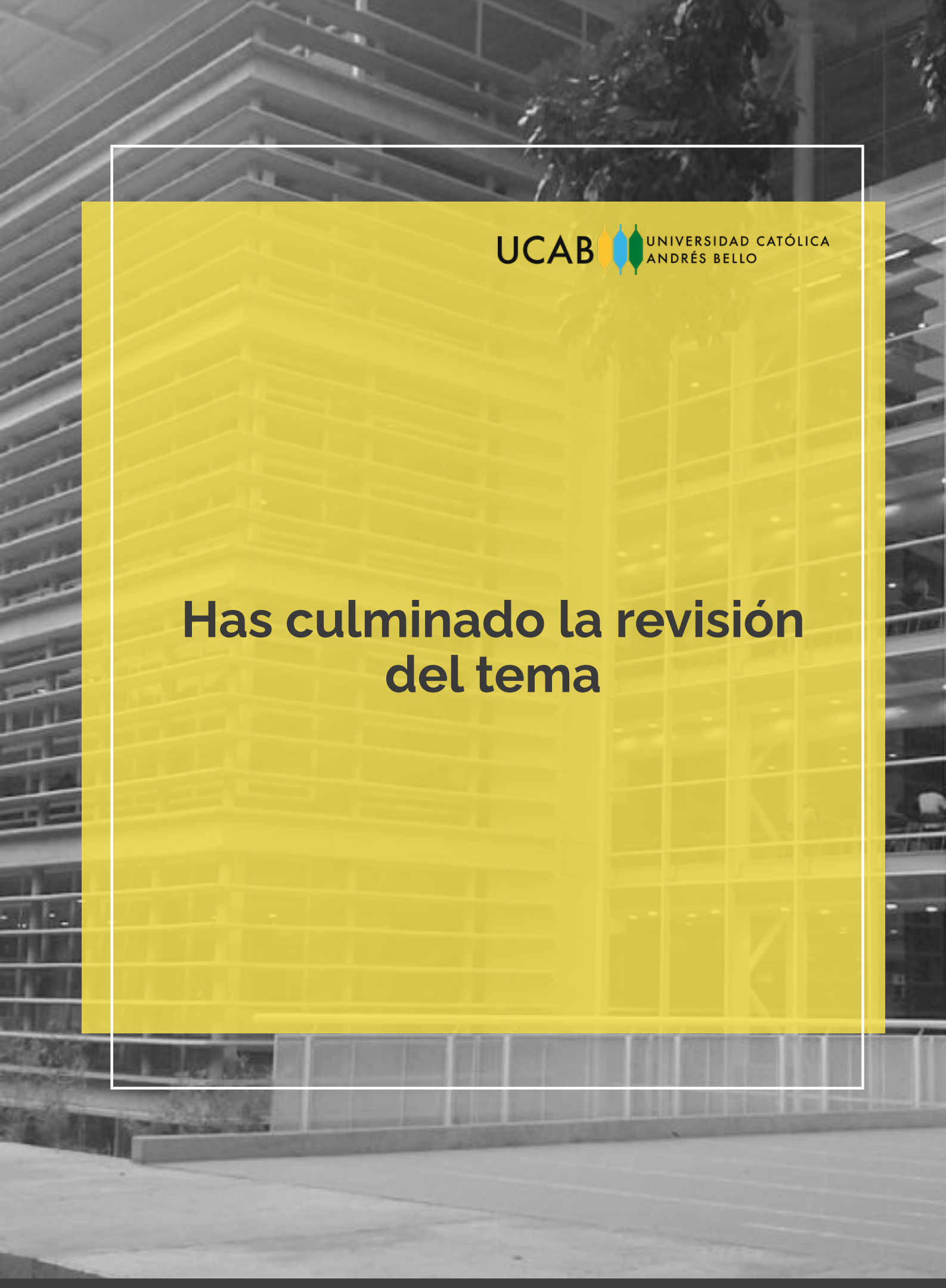
Richard, M. y Ford, N. (2020). Por capas [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

Richard, M. y Ford, N. (2020). Partición por dominio [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

Richard, M. y Ford, N. (2020). Partición técnica [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

Richard, M. y Ford, N. (2020). Responsabilidades [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.

Richard, M. y Ford, N. (2020). Proceso de subdivisión de componentes [Imagen]. Recuperado de *Fundamentals of Software Architecture*: *An Engineering Approach.* O'Reilly Inc.



UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Has culminado la revisión del tema