UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

REPRESENTACIÓN DE LA ARQUITECTURA

OBJETIVOS

Identificar los diferentes tipos de diagrama para representar el diseño de un *software*, a partir de dos tendencias: la clásica con UML o la ágil, que utiliza Model4C, para garantizar una correcta comunicación entre el equipo de desarrollo.

Reconocer la importancia de la consistencia representacional como expresión de la relación entre las partes de una arquitectura de *software*, utilizando diagramas, a fin de mantener la efectividad en momentos de cambios en un *software*.

TABLA DE CONTENIDO

UCAB

INTRODUCCIÓN

¡La evolución de un *software* hoy día es constante! Cambia con frecuencia, mucha frecuencia, pues las demandas de nuevas funcionalidades o características de calidad, tales como seguridad y confiabilidad, siempre van en aumento. Esto trae como consecuencia que el diseño también vaya variando. El diseño es **dinámico**.

Un arquitecto de *software* debe ir **registrando** las decisiones arquitectónicas que tome en función de estas demandas, y debe también "representar" su propuesta de solución ante las nuevas demandas. Al describir visualmente la arquitectura, el creador (arquitecto) a menudo debe mostrar diferentes vistas de la arquitectura. Por ejemplo, es probable que el arquitecto muestre una descripción general de toda la topología de la arquitectura y luego profundice en las partes individuales para precisar mejor los detalles del diseño. Sin embargo, si el arquitecto muestra una parte sin indicar dónde se encuentra dentro de la arquitectura general, confunde a los espectadores (el equipo de desarrollo).

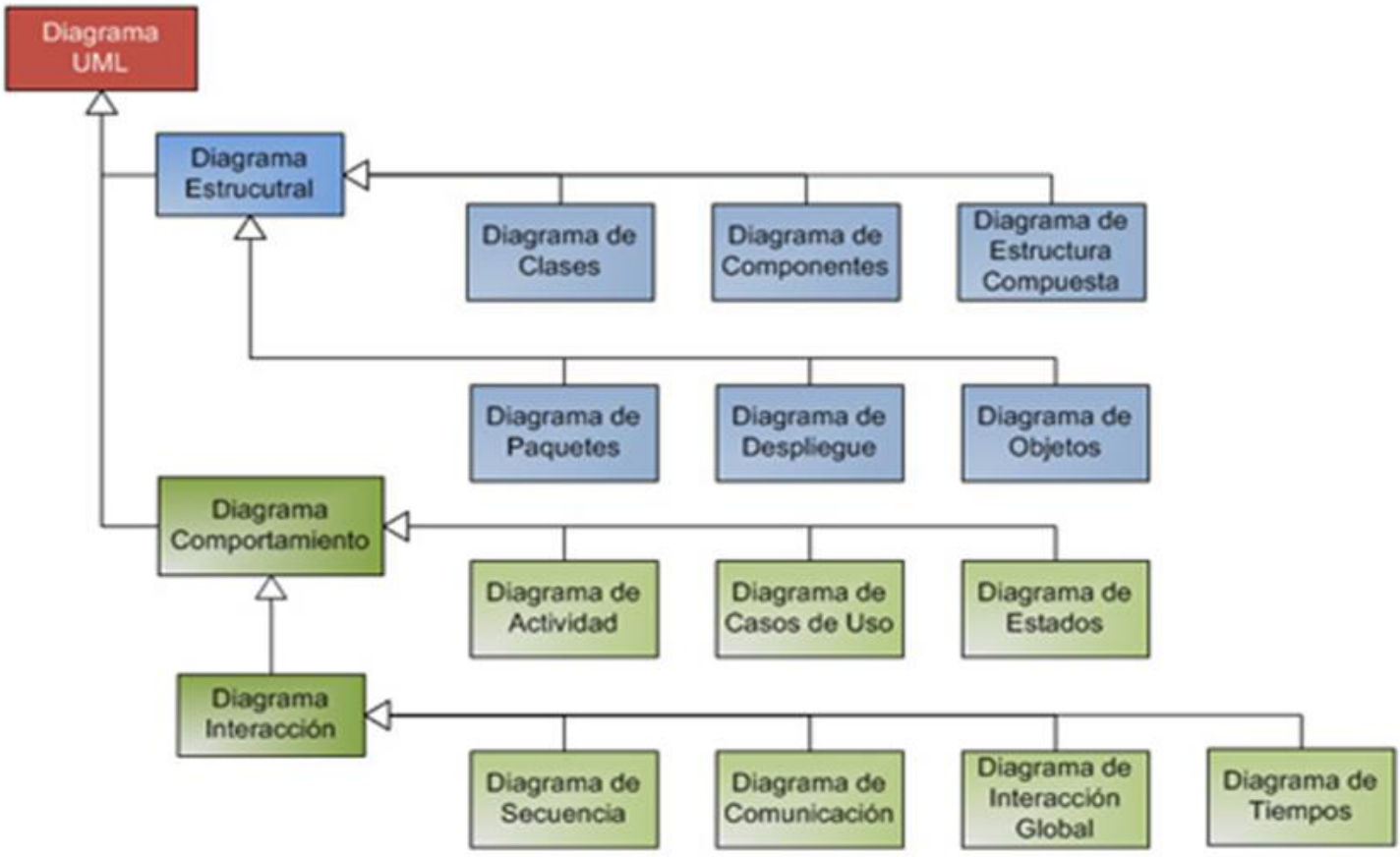
La consistencia representacional es la práctica de mostrar siempre la relación entre las partes de una arquitectura, utilizando **diagramas**. Un diagrama es un dibujo que tiene una sintaxis, es decir, no es un "dibujo libre". Esta sintaxis puede ser explícita, si se usa un lenguaje particular tal como UML (por sus siglas en inglés Lenguaje Unificado de Modelado), o con una leyenda, propuesta por su creador. Hay dos tendencias para hacer esta representación: la clásica con UML o la ágil que utiliza el Modelo 4C.

01 UML

Con la llegada del paradigma de orientación a objetos, los desarrolladores se centraron más en el diseño. Por ello comienza la preocupación de cómo lograr diseños de calidad que sean **confiables**, extensibles, escalables, etc., ya que si se toman decisiones de diseño acertadas se propicia la calidad del *software*.

Por otro lado, hacer cambios en la etapa de diseño no es costoso. El primer problema a resolver es: ¿cómo representó el diseño?

En ese momento (1995) surgieron cualquier cantidad de propuestas sobre posibles diagramas para representar el **diseño** de un *software*. Se impone UML; sin embargo, UML propone 13 diagramas (ver la siguiente figura):

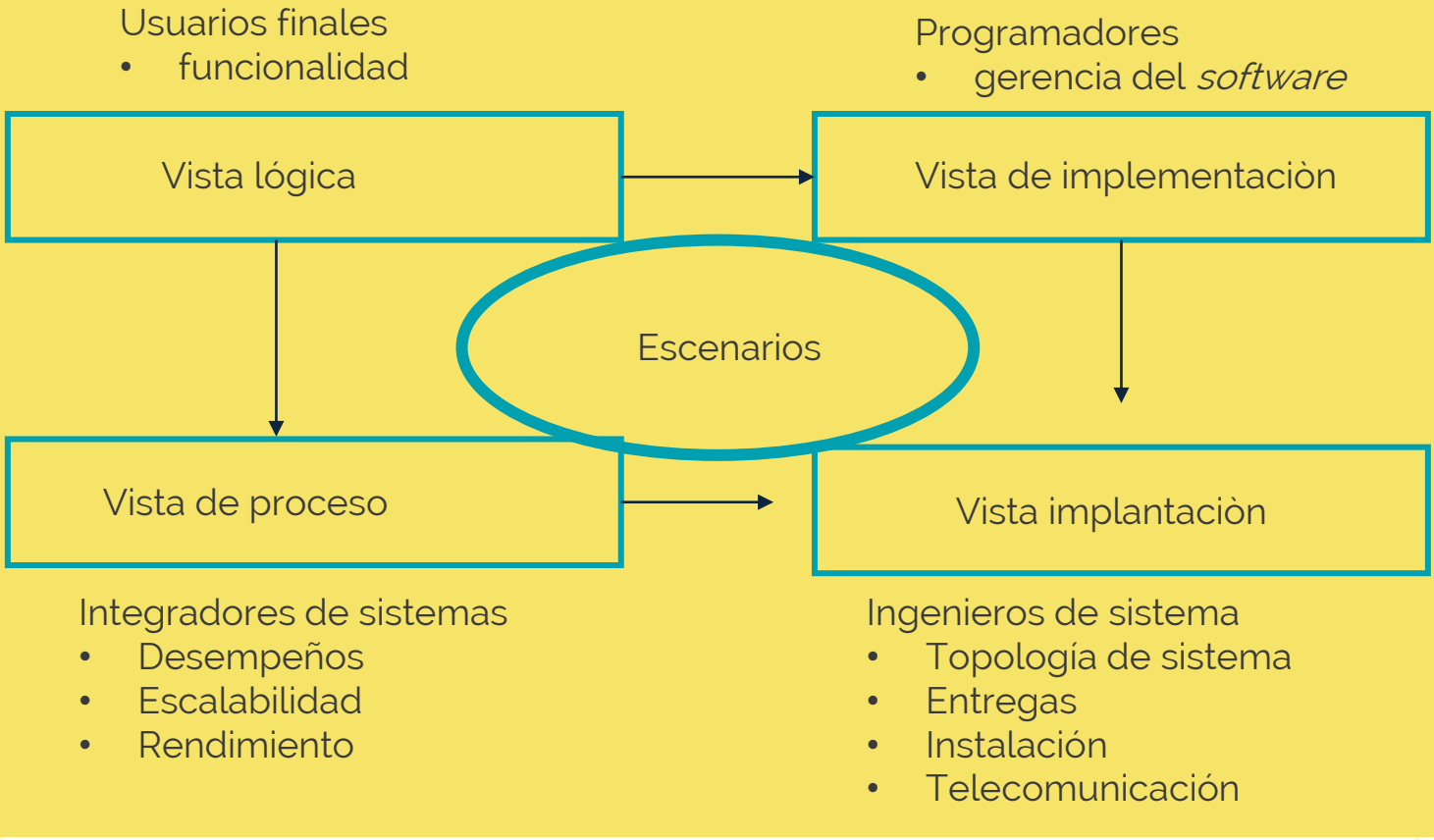


Fuente: Elaboración propia

¿Debemos usar los 13 diagramas para representar el diseño? No. Pero ¿cuántos y cuáles?

Kruchten (1995), experto en **arquitectura**, propone su modelo de 4+1 vistas.

Para él, el desarrollo de los sistemas debe estar centrado en su arquitectura y propone que esta sea especificada con cinco vistas ((ver la siguiente figura):



Fuente: 4+1 vistas. Traducida de Kruchten (1995)

Revisemos cada una de ellas:

- Vista lógica: describe el modelo objeto del diseño. Se usa de UML el diagrama de clases de diseño para su representación.

- La vista de implementación: describe la organización estática del software en el ambiente de desarrollo. Se usa de UML el Diagrama de componentes para su representación.

- La vista de proceso: describe los aspectos de diseño relacionados con la **concurrencia** y la sincronización. Se usa de UML los Diagramas de secuencia para su representación.

- La vista implantación: describe el mapa del SW dentro del HW y refleja los aspectos de distribución. Se usa de UML el Diagrama de despliegue para su representación.

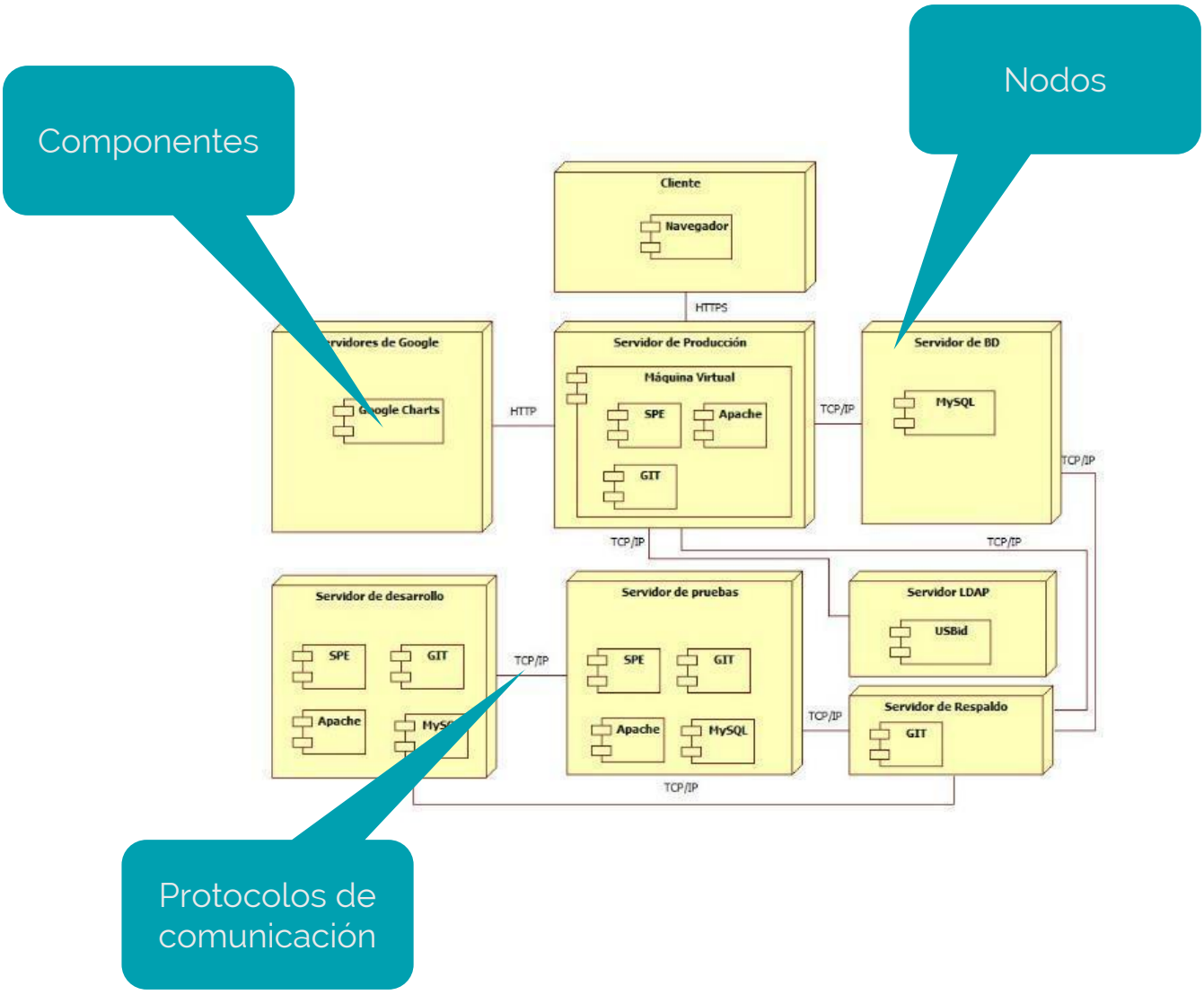
- Los escenarios: representan la funcionalidad deseada. Se usa de UML los Diagramas de casos de uso o escenarios, constituyendo así la quinta vista.

Esta propuesta de representación de la arquitectura potenció a **UML** (*Unified Modeling Language*). Provee una sintaxis visual que se utiliza para construir modelos y no está enlazado a ninguna metodología. Precisemos un poco sobre los diagramas de UML que nos quedan pendientes por describir:

Diagramas despliegue: muestran la **configuración** de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos; esto es, la implantación de los componentes sobre los dispositivos físicos.

Se relaciona con el diagrama de componentes, ya que un nodo incluye uno o más componentes (desarrollados o no por la empresa). Responde a la pregunta: ¿qué software funciona en cuál *hardware*?

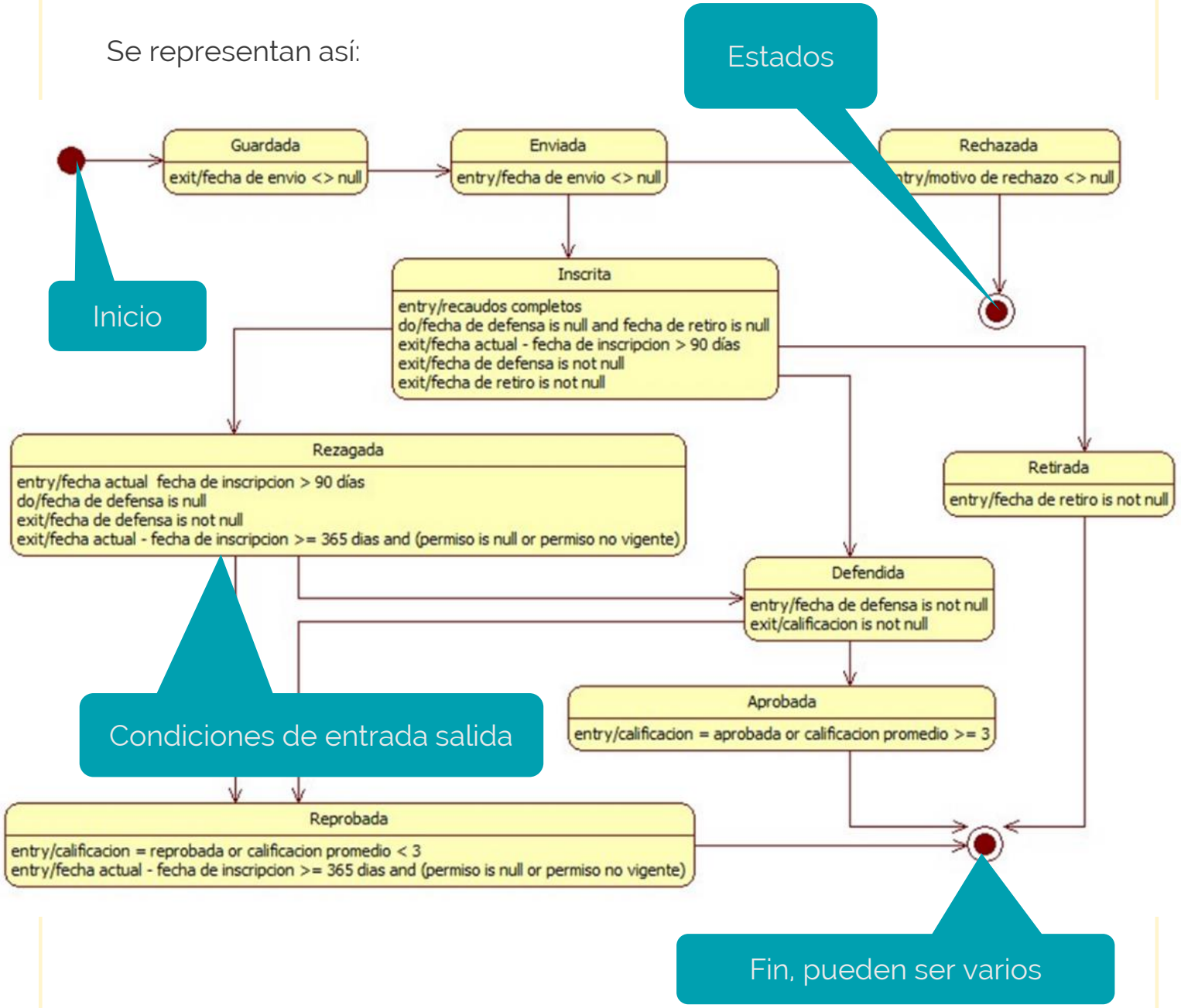
Los diagramas de despliegue se representan así:



Fuente: Elaboración propia

- **Diagramas de estados:** si bien Kruchten no los propone para representar la arquitectura, son muy útiles para modelar el **ciclo de vida** de un objeto. Muestran una máquina de estados que consta de estados, transiciones, eventos y actividades. Cubren la vista dinámica de un sistema y resaltan el comportamiento dirigido por eventos de un clasificador. Típicamente se usan durante la fase de análisis y diseño, cuando se desea entender el funcionamiento de las clases con mucho detalle.

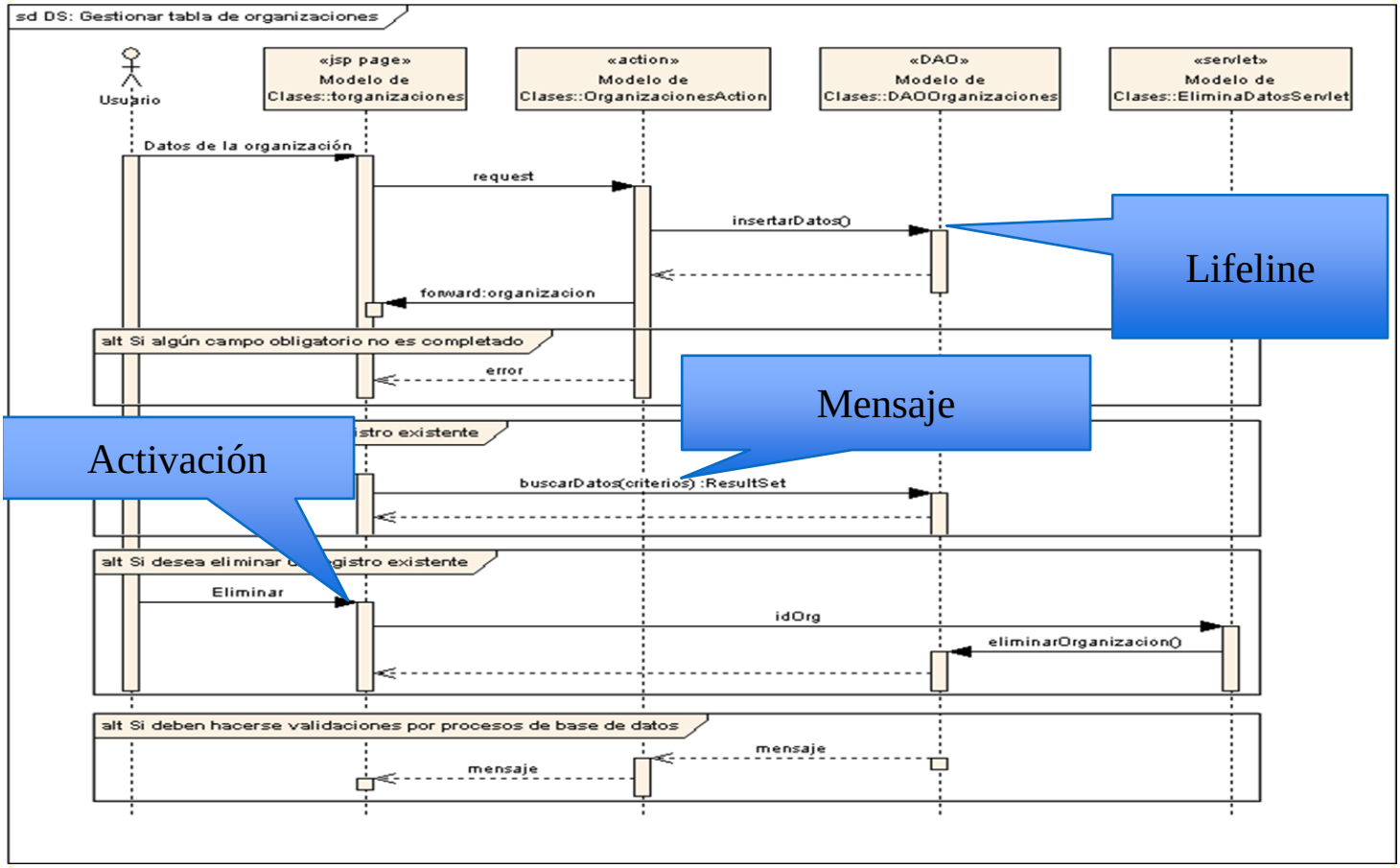
Se representan así:



Fuente: Elaboración propia

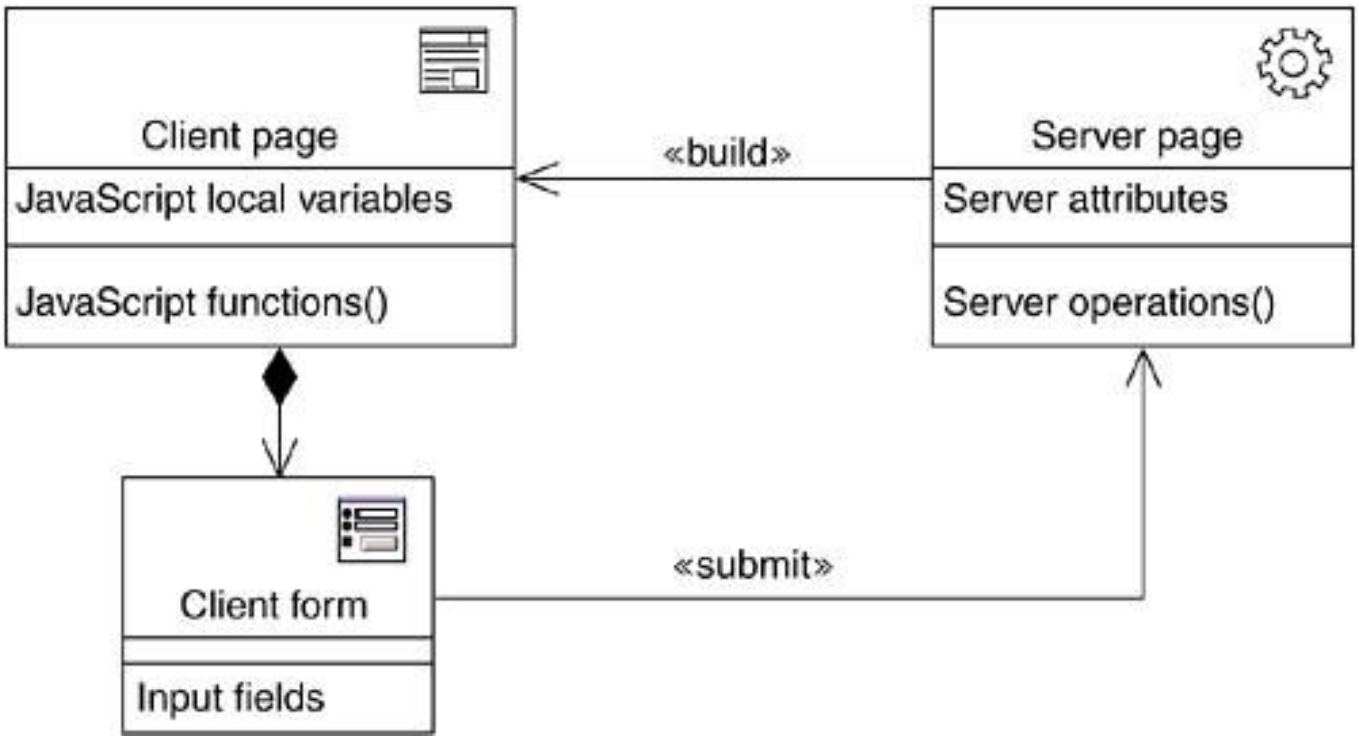
- **Diagramas de secuencia:** son un tipo de diagrama de interacción. Las interacciones son unidades simples de comportamiento entre clasificadores. Un **clasificador** es un elemento en el modelo que describe características estructurales y de comportamiento. Las clases, actores, componentes, subsistemas, casos de uso, etcétera, son clasificadores. Un mensaje es una forma de comunicación durante una interacción.

Se representan así:



Fuente: Elaboración propia

- **Diagramas WAE**: con la llegada del mundo web se hizo una propuesta de Diagramas de secuencia estereotipados, donde los clasificadores, en lugar de clases, son páginas. Un estereotipo es un mecanismo de extensión que define un nuevo tipo de elemento dentro de un nuevo tipo de modelos; tienen una **semántica** adicional. Este mecanismo lo propone UML para aquellos sistemas/situaciones que no pueden ser modelados completamente con un diagrama de clases y/o diagramas de secuencia. Los diagramas WAE: *Web Application Extension*, definen un conjunto de estereotipos, etiquetas y restricciones que nos permiten modelar aplicaciones web. Fue propuesto por Conallen (2000).



Fuente: Web Application Extension for UML. Extraída de Conallen (2002)

Los sistemas web se basan en el **estilo arquitectónico** cliente-servidor. Es decir, estilo por capas, con solo dos capas. Repasemos un poco la metáfora del restaurant, a fin de que hagamos un buen diseño, basado en este estilo arquitectónico.

Metáfora del restaurant

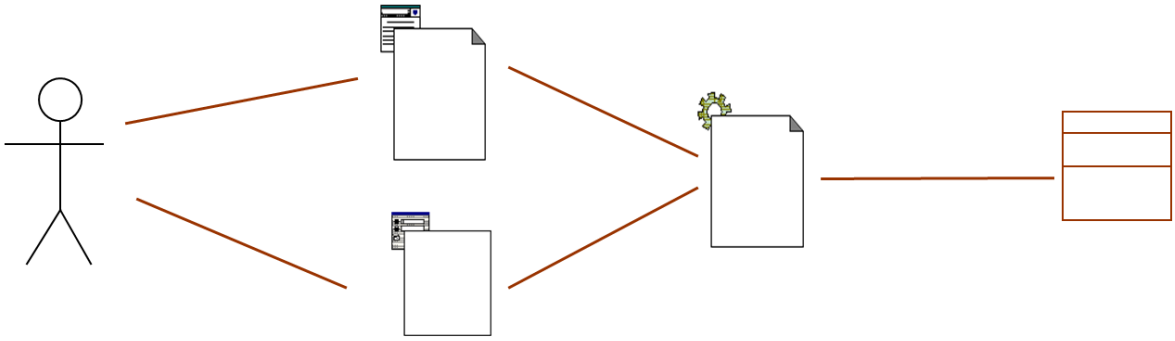
Fuente: Elaboración propia

En un restaurante los clientes son atendidos por el mesonero. Este le indica al chef los platos a preparar (los servicios solicitados). El chef manipula los alimentos para elaborar los platos solicitados y luego le entrega al mesonero los platos y este se los sirve a los clientes.

Aquí hay un conjunto de relaciones no permitidas:

- Los clientes no "hablan" con el chef.
- Los mesoneros no manipulan alimentos.
- Los clientes no manipulan los alimentos.

Si llevamos esta metáfora a UML con WAE, tendríamos:



Fuente: Elaboración propia

- Los clientes (usuarios) son atendidos por las páginas cliente.
- Estas solicitan servicios a las páginas servidor (chef).
- Y estos son los que actúan sobre las bases de datos.

Nótese que una página cliente no accede a las bases de datos y un usuario no accede a una página servidor.

02 Modelo 4C

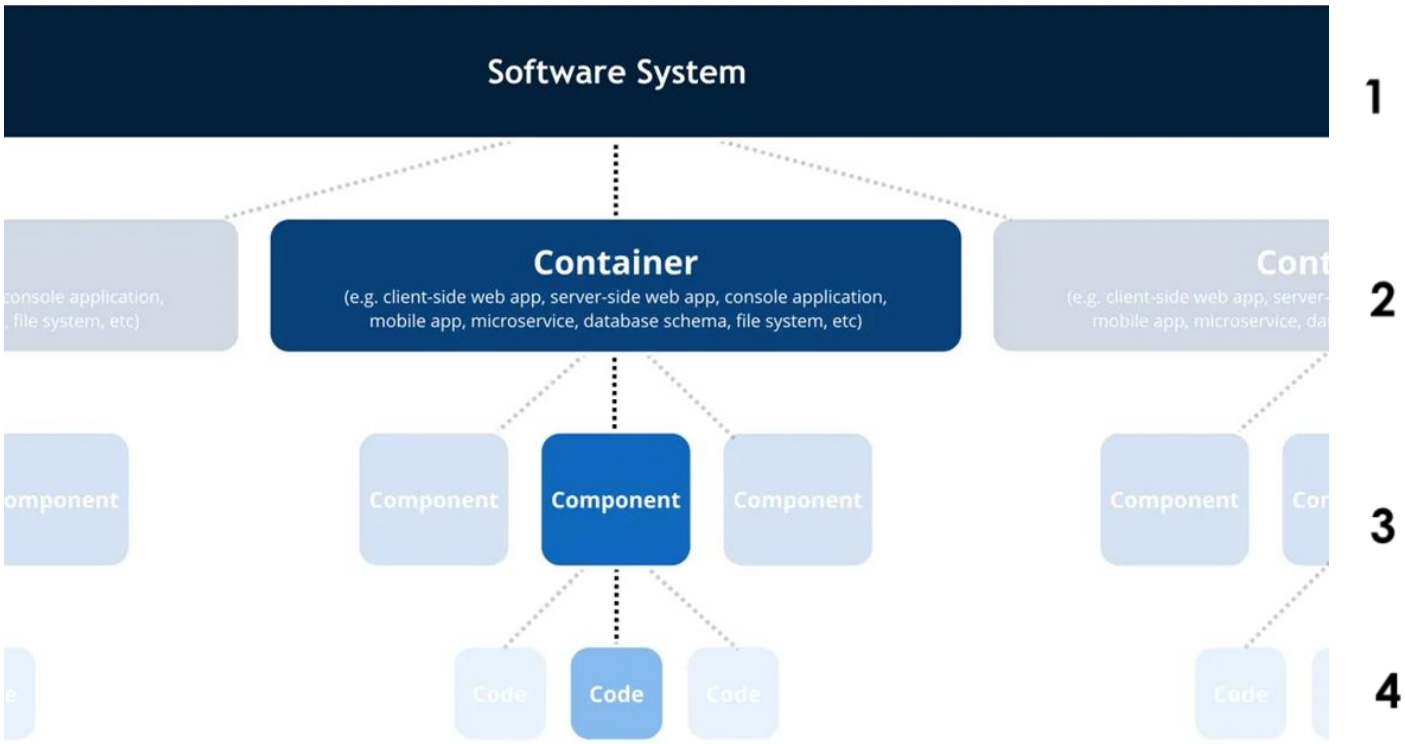
> El modelo 4C consiste en un conjunto **jerárquico** de diagramas de arquitectura de *software* para: contexto, contenedores, componentes y código. La jerarquía de los diagramas 4C proporciona diferentes niveles de abstracción; cada uno es relevante para una audiencia diferente. Propuesto por Simón Brown (2014), parte de algunos de los diagramas de UML, pero se hace conocido a partir del 2011.

El modelo 4C considera las **estructuras estáticas** de un sistema de *software* en términos de contenedores (aplicaciones, almacenes de datos, microservicios, etc.), componentes y clases. También considera a las personas que utilizan los sistemas de *software* que construimos.

No siempre tenemos la misma audiencia: usuario, ejecutivos, desarrolladores, QA, etc.

Hay que evitar la ambigüedad en los diagramas, incluyendo una cantidad suficiente de texto, así como también una clave o leyenda para la notación utilizada.

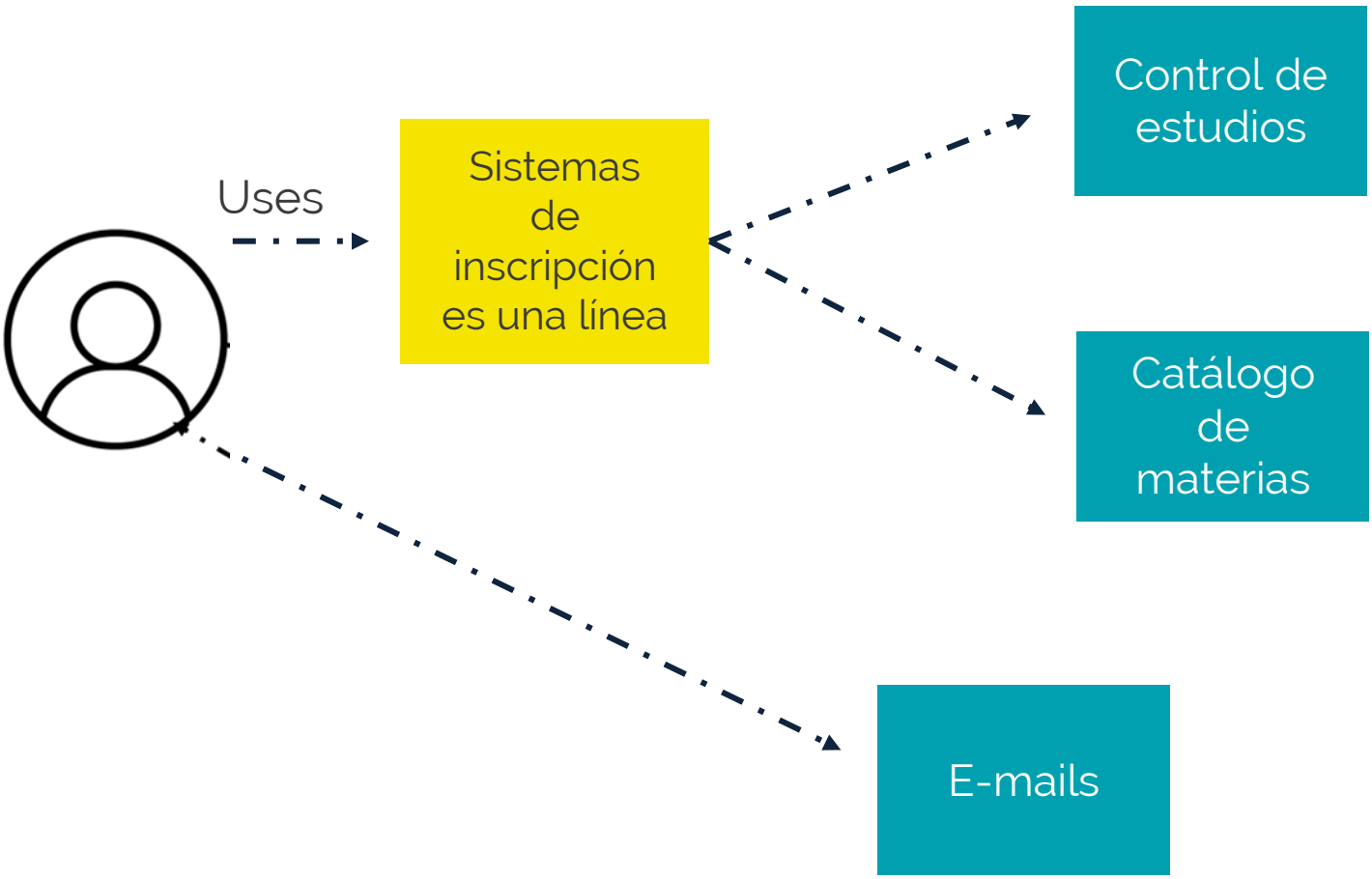
Sus niveles son:



Fuente: Niveles del Modelo 4C. Adaptada de Brown (2014)

1 Contexto del sistema:

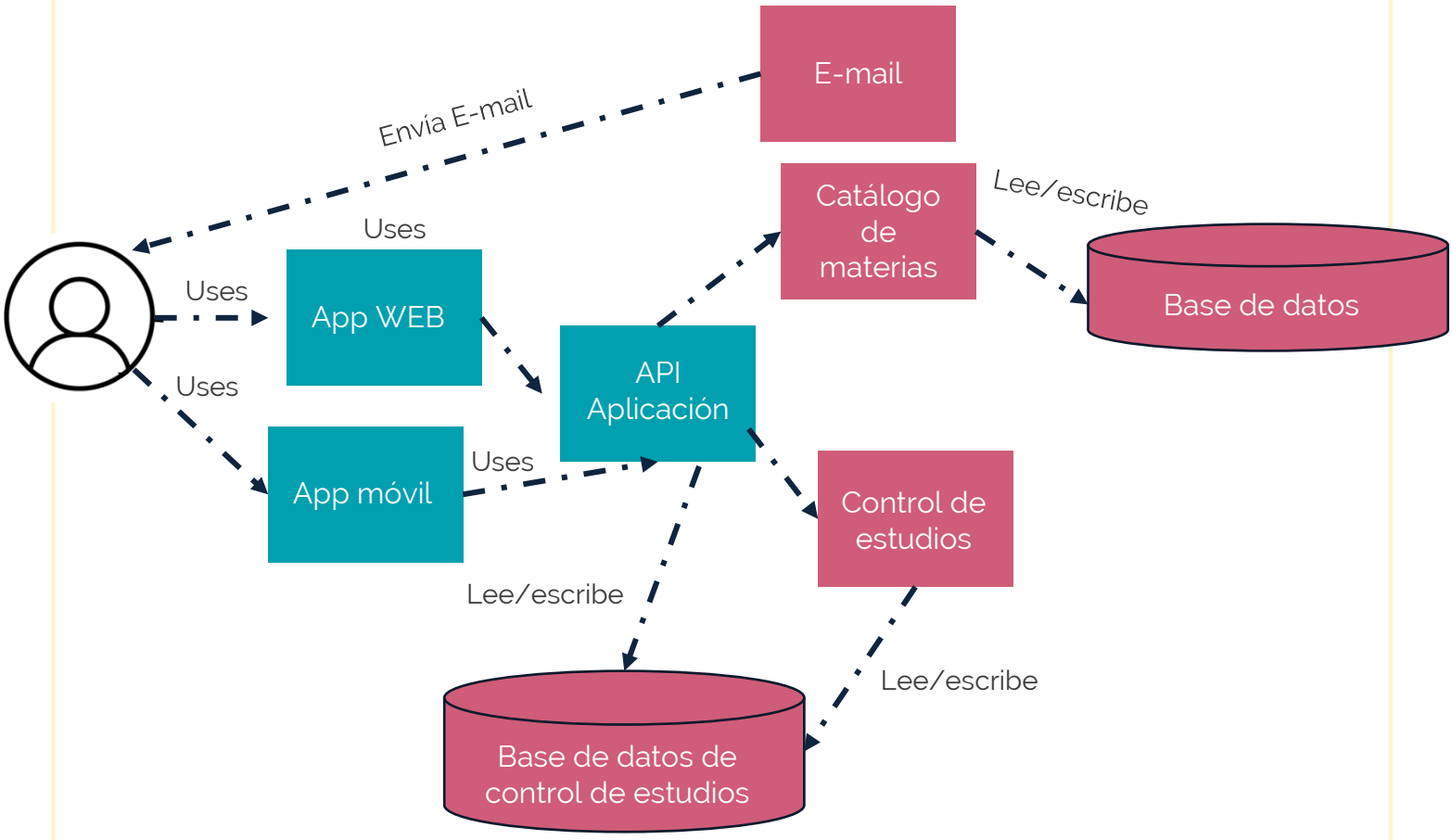
- En el **nivel 1** un diagrama de contexto del sistema muestra el sistema de *software* que se está construyendo, cómo encaja en el mundo en términos de las personas que lo utilizan y los otros sistemas de *software* con los que interactúa.
- Aquí el detalle no es importante.
- Contiene:
  - Usuario
  - Sistemas, todos conectados.



Fuente: Elaboración propia

2 Contenedores:

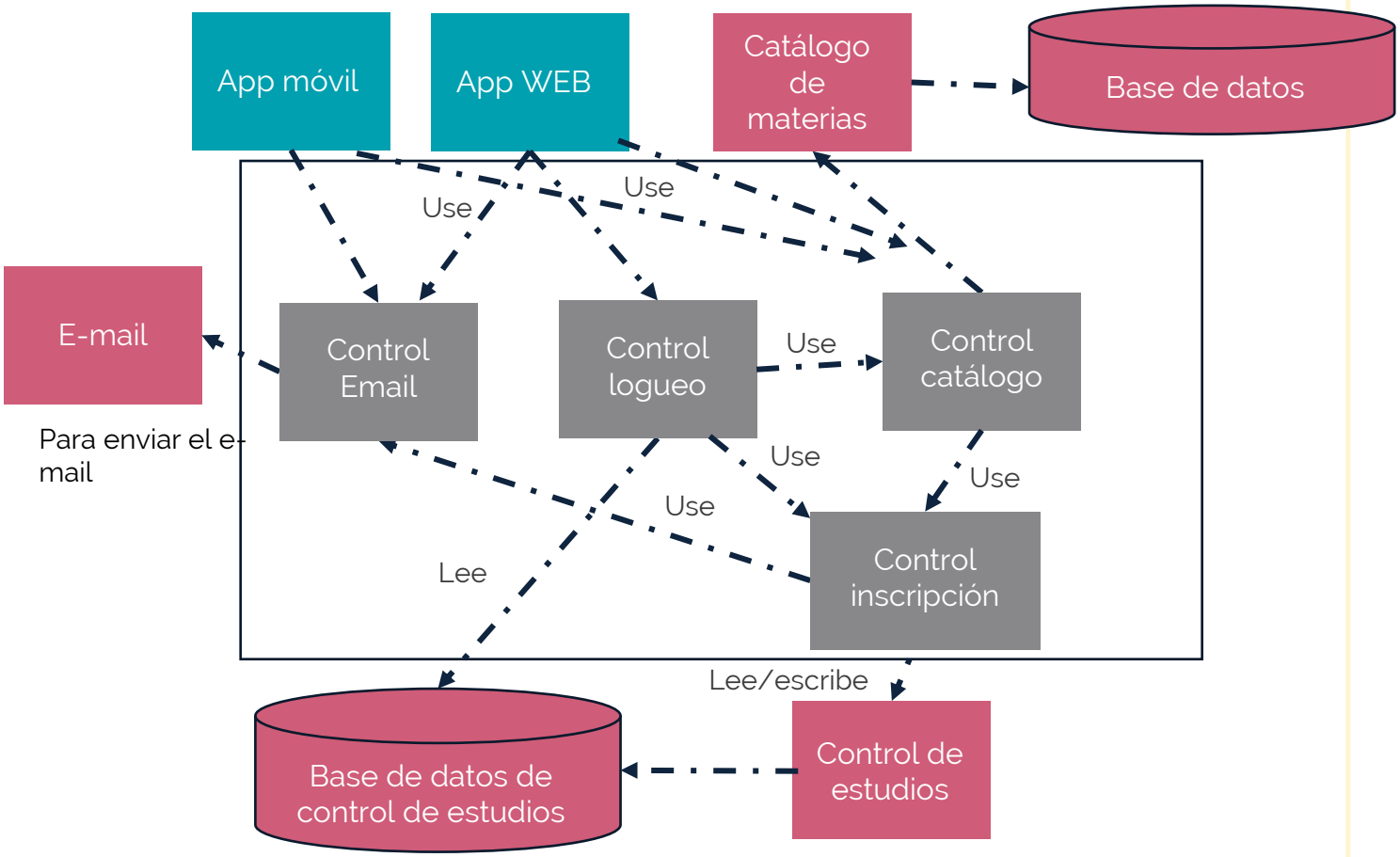
- El **nivel 2** muestra una estructura de alto nivel.
- Amplía el sistema de *software* y muestra los contenedores (aplicaciones, almacenamiento de datos, microservicios, etc.) que componen este sistema.
- Las decisiones tecnológicas son también una parte fundamental de este diagrama.
- Contiene:
  - Contenedores.
  - Relaciones entre ellos.
  - Usuarios.



Fuente: Elaboración propia

3 Componentes:

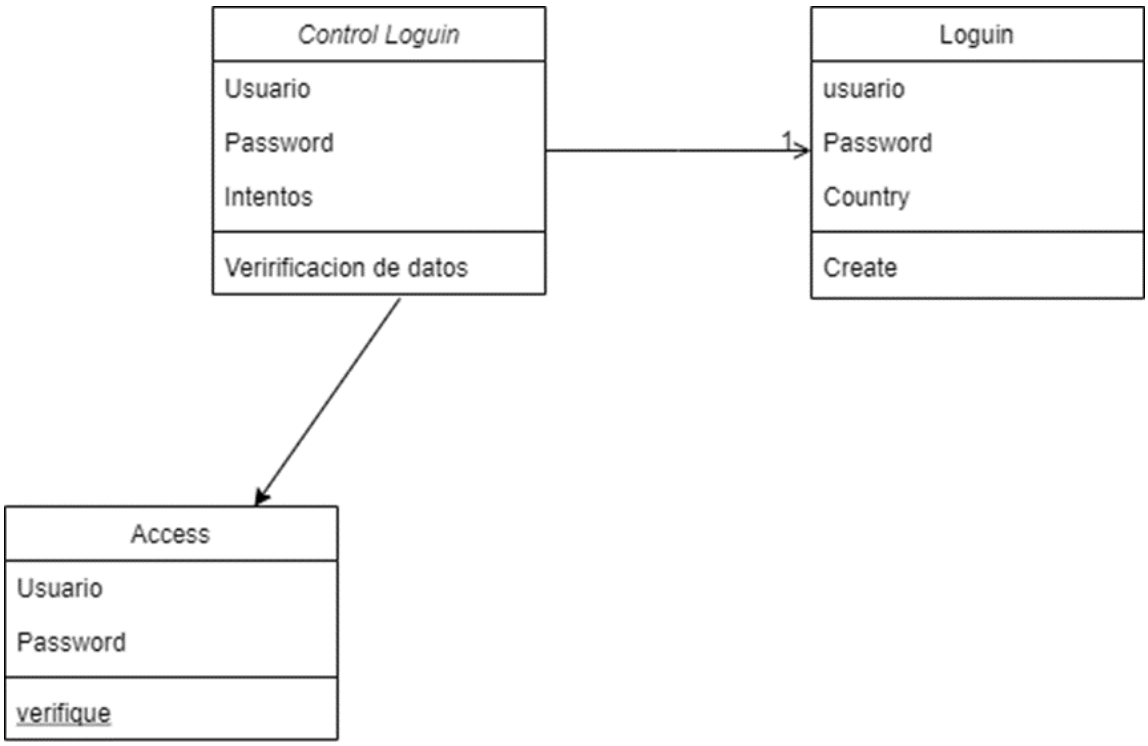
- El **nivel 3** expande un contenedor individual para mostrar los componentes que contiene.
- Estos componentes deben asignarse a abstracciones reales (por ejemplo, una agrupación de códigos), en función de su código.
- Por cada componente:
  - Su nombre
  - Responsabilidades
  - Detalles de implementación.
- Contiene:
  - Componentes
  - Contenedores necesarios, usuarios y relaciones (solo los necesarios)



Fuente: Elaboración propia

4 Clases:

- El **nivel 4**, por último, es un diagrama de clases UML que muestra los elementos de código (interfaces y clases).
- Solo muestra métodos y atributos relevantes.
- Contiene:
  - Clases
  - Métodos
  - Atributos
  - Paquetes.

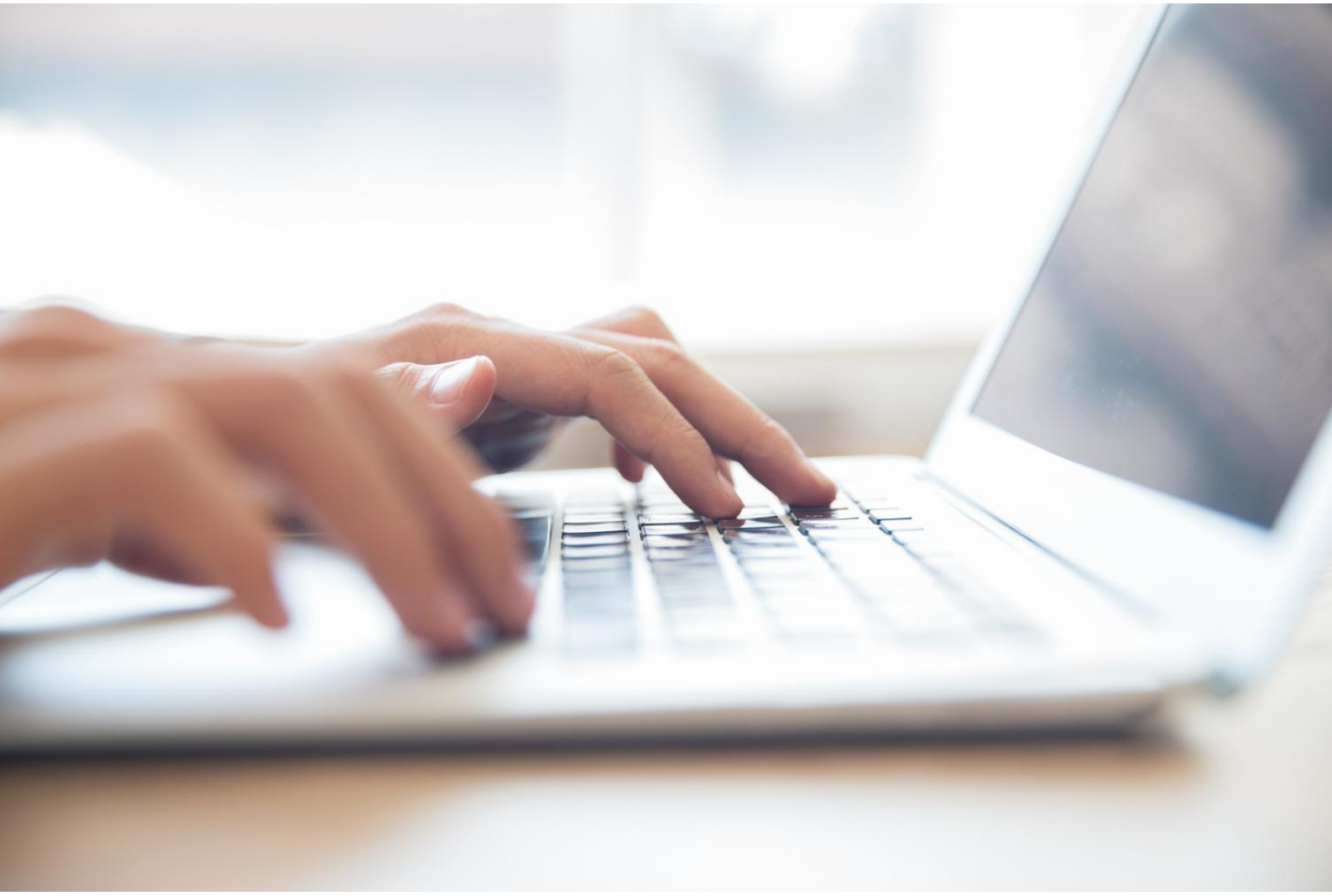


Fuente: Elaboración propia

Para efectos del proyecto, utilizaremos la notación Modelo 4C, dado que estamos en un entorno ágil.

Estos cuatro diagramas, luego de elaborarlos para el último *Sprint* del proyecto, se colocarán en el DAS.

Una de las metas del equipo de desarrollo es **disminuir** el esfuerzo cada vez que nos pidan hacer un cambio en un *software*. Una de las estrategias más poderosas para lograr esto es contar con la documentación adecuada y actualizada del diseño del software. De esta forma, iremos "directo al grano", ubicamos rápidamente cuál es el cambio que debemos hacer y dónde. Por ello, la representación de la arquitectura es clave. Con este tema tenemos claro cómo representarla, ya sea usando las cuatro vistas + 1 de Kruchten (en UML) o el Modelo 4C de Brown. Además que con una representación no ambigua garantizamos una mejor comunicación entre el equipo de desarrollo.



REFERENCIAS

Brown, S. (2014). *Software Architecture for developers*. https://softwarearchitecturefordevelopers.com/

Conallen, J. (2000). *Building Web Applications with UML*. Addison Wesley.

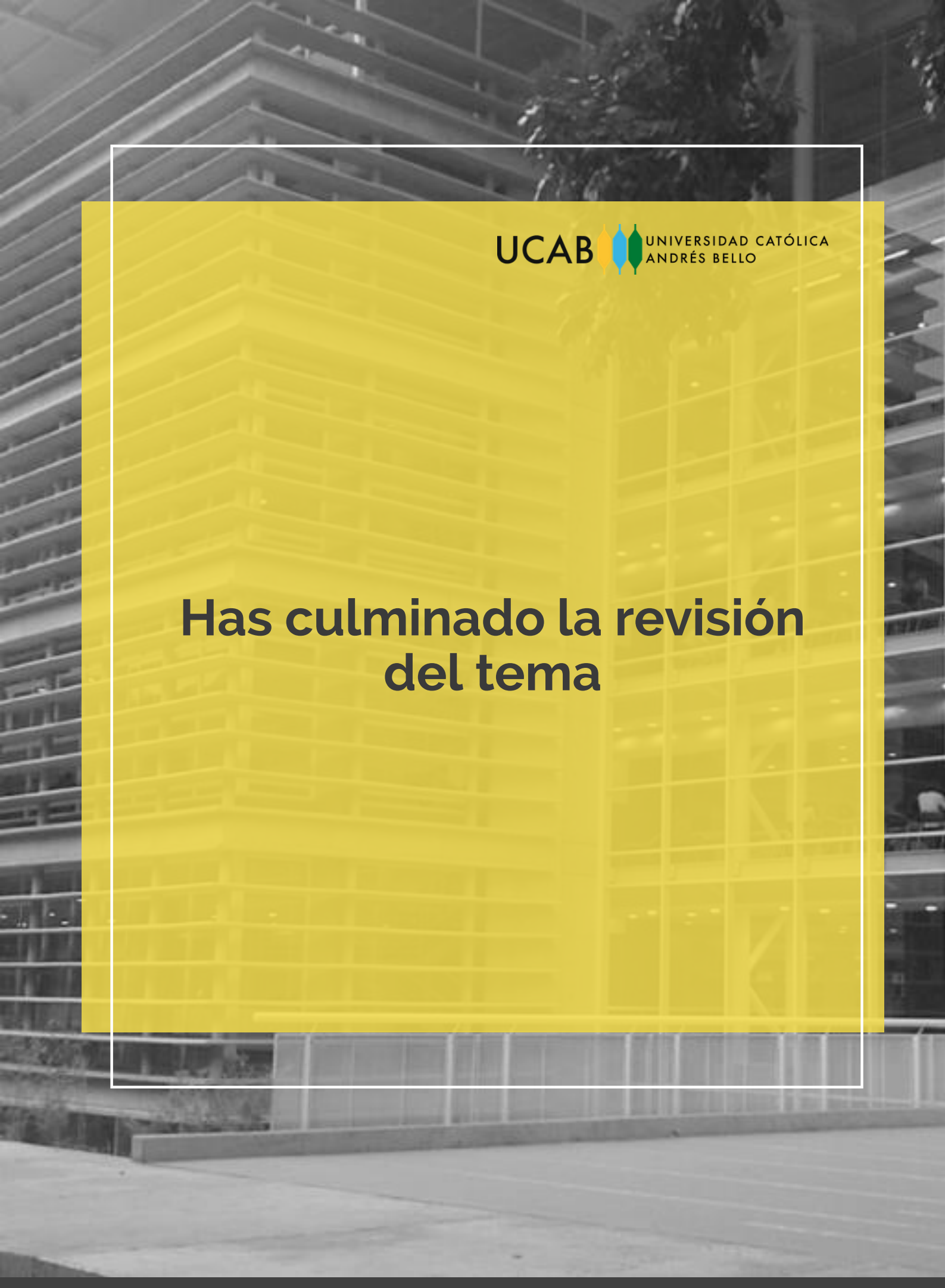
Kruchtten, P. (1995). Architectural Blueprints—The "4+1" View Model of Software. *Architecture IEEE Software*, *12*(6), pp. 42-50.

Referencias de las imágenes:

Brown, S. (2014). Niveles del Modelo 4C [Imagen]. Recuperado de *Software Architecture for developers* https://softwarearchitecturefordevelopers.com/

Conallen, J. (2000). Web Application Extension for UML [Imagen]. Recuperado de *Building Web Applications with UML*. Addison Wesley.

Kruchtten, P. (1995). 4+1 vistas [Imagen]. Recuperado de Architectural Blueprints—The "4+1" View Model of Software. *Architecture IEEE Software*, *12*(6), pp. 42-50



UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Has culminado la revisión del tema