



Vicerrectorado de Modalidad Abierta y a Distancia

## Modelado de Sistemas

Guía didáctica





Facultad Ingenierías y Arquitectura

## Modelado de Sistemas

### Guía didáctica

Carrera	PAO Nivel
Tecnologías de la Información	IV

#### Autor:

Roddy Andrés Correa Tenesaca



D S O F \_ 2 0 4 4

# Universidad Técnica Particular de Loja

## Modelado de Sistemas

### Guía didáctica

Roddy Andrés Correa Tenesaca

### Diagramación y diseño digital

Ediloja Cía. Ltda.

Marcelino Champagnat s/n y París

edilojacialtda@ediloja.com.ec

[www.ediloja.com.ec](http://www.ediloja.com.ec)

ISBN digital 978-9942-47-177-2

Año de edición: octubre 2024

Edición: primera edición

Loja-Ecuador



### Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0)

Usted acepta y acuerda estar obligado por los términos y condiciones de esta Licencia, por lo que, si existe el incumplimiento de algunas de estas condiciones, no se autoriza el uso de ningún contenido.

Los contenidos de este trabajo están sujetos a una licencia internacional Creative Commons **Reconocimiento-NoComercial-CompartirIgual 4.0** (CC BY-NC-SA 4.0). Usted es libre de **Compartir** – copiar y redistribuir el material en cualquier medio o formato. **Adaptar** – remezclar, transformar y construir a partir del material citando la fuente, bajo los siguientes términos:

**Reconocimiento**- debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier

forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciatante. *No Comercial-no puede hacer uso del material con propósitos comerciales. Compartir igual-Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.* No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia.  
<https://creativecommons.org/licenses/by-nc-sa/4.0/>



# Índice

<b>1. Datos de información .....</b>	<b>10</b>
1.1 Presentación de la asignatura .....	10
1.2 Competencias genéricas de la UTPL.....	10
1.3 Competencias específicas de la carrera .....	10
1.4 Problemática que aborda la asignatura .....	10
<b>2. Metodología de aprendizaje .....</b>	<b>12</b>
<b>3. Orientaciones didácticas por resultados de aprendizaje .....</b>	<b>14</b>
<b>Primer bimestre .....</b>	<b>14</b>
<b>Resultado de aprendizaje 1: .....</b>	<b>14</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>14</b>
<b>Semana 1 .....</b>	<b>14</b>
Unidad 1. Introducción al modelado de sistemas .....	15
1.1. Definición de modelado de sistemas .....	15
1.2. Importancia del modelado de sistemas .....	17
1.3. El Modelado de sistemas y la programación orientada a objetos .....	19
1.4. Aplicaciones prácticas del modelado en diversos ámbitos.....	20
Actividades de aprendizaje recomendadas .....	22
Autoevaluación 1.....	22
<b>Resultado de aprendizaje 2: .....</b>	<b>25</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>25</b>
<b>Semana 2 .....</b>	<b>25</b>
Unidad 2. Modelo 4 + 1 vistas de Krutchen.....	25
2.1. Introducción al modelo 4 + 1 de Krutchen .....	26
2.2. Vista de escenario.....	27
2.3. Vista lógica .....	28
2.4. Vista de proceso .....	28

2.5. Vista de desarrollo .....	29
2.6. Vista física .....	30
Actividades de aprendizaje recomendadas .....	31
Autoevaluación 2.....	32
<b>Resultado de aprendizaje 3: .....</b>	<b>34</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>34</b>
<b>Semana 3 .....</b>	<b>34</b>
Unidad 3. Diagramas UML básicos.....	34
3.1. ¿Qué es UML?.....	35
3.2. Herramientas de modelado UML.....	35
3.3. Vistas UML .....	36
3.4. Tipos de vistas UML.....	36
3.5. Vista de contexto .....	38
Actividades de aprendizaje recomendadas .....	41
<b>Resultado de aprendizaje 3: .....</b>	<b>43</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>43</b>
<b>Semana 4 .....</b>	<b>43</b>
Unidad 3. Diagramas UML básicos.....	43
3.6. Vista de escenarios o casos de uso .....	43
Actividades de aprendizaje recomendadas .....	48
<b>Resultado de aprendizaje 3: .....</b>	<b>49</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>49</b>
<b>Semana 5 .....</b>	<b>49</b>
Unidad 3. Diagramas UML básicos.....	49
3.6. Vista de escenarios o casos de uso .....	49
Actividades de aprendizaje recomendadas .....	55
<b>Resultado de aprendizaje 3: .....</b>	<b>56</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>56</b>
<b>Semana 6 .....</b>	<b>56</b>



Unidad 3. Diagramas UML básicos.....	56
3.7. Vista dinámica o de procesos.....	56
Actividades de aprendizaje recomendadas .....	61
<b>Resultado de aprendizaje 3: .....</b>	<b>62</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>62</b>
<b>Semana 7 .....</b>	<b>62</b>
Unidad 3. Diagramas UML básicos.....	62
3.7. Vista dinámica o de procesos.....	62
Actividades de aprendizaje recomendadas .....	67
Autoevaluación 3.....	68
<b>Resultados de aprendizaje 1 a 3: .....</b>	<b>71</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>71</b>
<b>Semana 8 .....</b>	<b>71</b>
Actividades finales del bimestre .....	71
<b>Segundo bimestre .....</b>	<b>74</b>
<b>Resultado de aprendizaje 4: .....</b>	<b>74</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>74</b>
<b>Semana 9 .....</b>	<b>74</b>
Unidad 4. Diagramas UML avanzados.....	74
4.1. Vista lógica .....	75
Actividades de aprendizaje recomendadas .....	85
<b>Resultado de aprendizaje 4: .....</b>	<b>86</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>86</b>
<b>Semana 10 .....</b>	<b>86</b>
Unidad 4. Diagramas UML avanzados.....	86
4.2. Vista de desarrollo .....	86
Actividades de aprendizaje recomendadas .....	95
<b>Resultado de aprendizaje 4: .....</b>	<b>96</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>96</b>



<b>Semana 11 .....</b>	<b>96</b>
Unidad 4. Diagramas UML avanzados.....	96
4.2. Vista de desarrollo .....	96
Actividades de aprendizaje recomendadas .....	100
<b>Resultado de aprendizaje 4: .....</b>	<b>101</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>101</b>
<b>Semana 12 .....</b>	<b>101</b>
Unidad 4. Diagramas UML avanzados.....	101
4.3. Vista física .....	101
Actividades de aprendizaje recomendadas .....	106
Autoevaluación 4.....	107
<b>Resultado de aprendizaje 4: .....</b>	<b>110</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>110</b>
<b>Semana 13 .....</b>	<b>110</b>
Unidad 5. Modelado avanzado.....	110
5.1. Modelado ágil (C4).....	110
Actividades de aprendizaje recomendadas .....	120
<b>Resultado de aprendizaje 4: .....</b>	<b>121</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>121</b>
<b>Semana 14 .....</b>	<b>121</b>
Unidad 5. Modelado avanzado.....	121
5.1. Modelado ágil (C4).....	122
Actividades de aprendizaje recomendadas .....	125
<b>Resultado de aprendizaje 5: .....</b>	<b>127</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>127</b>
<b>Semana 15 .....</b>	<b>127</b>
Unidad 5. Modelado avanzado.....	127
5.2. Modelado de componentes cloud .....	128



5.3. Importancia de modelar para diseñar flujos de automatización en prácticas DevOps .....	130
Actividades de aprendizaje recomendadas .....	133
Autoevaluación 5.....	134
<b>Resultados de aprendizaje 4 y 5:.....</b>	<b>137</b>
<b>Contenidos, recursos y actividades de aprendizaje recomendadas .....</b>	<b>137</b>
<b>Semana 16 .....</b>	<b>137</b>
Actividades finales del bimestre .....	137
<b>4. Solucionario .....</b>	<b>140</b>
<b>5. Glosario .....</b>	<b>145</b>
<b>6. Referencias Bibliográficas .....</b>	<b>148</b>
<b>7. Anexos .....</b>	<b>149</b>





## 1. Datos de información

### 1.1 Presentación de la asignatura



### 1.2 Competencias genéricas de la UTPL

Organización y planificación del tiempo.

### 1.3 Competencias específicas de la carrera

- Diseñar aplicaciones de software que mediante técnicas avanzadas de modelado permiten solucionar los requerimientos del cliente, utilizando estándares de la industria.
- Modelar procesos de negocio utilizando técnicas y marcos de referencia para identificar problemas, oportunidades de mejora y proponer alternativas que permitan dar soporte a la estrategia del negocio.

### 1.4 Problemática que aborda la asignatura

- Capacidad de análisis, entendimiento y comprensión de un problema dentro de un contexto determinado.

- Necesidad de documentar una solución *software* haciendo uso de marcos de referencia, *estándares* como ISO y IEEE, nomenclatura, elementos y estereotipos UML.
- Recomendar soluciones de *software* (de escritorio, móviles o web), haciendo uso de diagramas, vistas, puntos de vista para exponerlos a los clientes o involucrados.
- Sugerir soluciones arquitectónicas de *software* candidatas ante un problema o necesidad.





## 2. Metodología de aprendizaje

Estimados estudiantes:

¡Bienvenidos a la asignatura de Modelado de sistemas! En este curso, adoptaremos una Metodología centrada en la resolución de problemas, con el propósito fundamental de idear soluciones de diseño eficaces para el desarrollo de *software* de alta calidad. Nuestro enfoque principal será el empleo del modelado como una herramienta esencial para representar soluciones de manera documentada, estandarizada y eficiente. El objetivo de esta asignatura es equipar a los estudiantes con las habilidades necesarias para abordar problemas reales y diseñar soluciones que puedan implementarse utilizando una variedad de tecnologías y enfoques.

### Enfoque práctico:

- La metodología se centra en la resolución de problemas prácticos relacionados con el desarrollo de *software*.
- Se espera que los estudiantes apliquen sus conocimientos previos y experiencias para diseñar soluciones efectivas utilizando el modelado de sistemas.
- A lo largo del curso, trabajaremos con un caso práctico de referencia para aplicar los conceptos aprendidos.
- Los estudiantes utilizarán este caso como un marco de referencia para entender cómo aplicar el modelado en situaciones del mundo real.

### Desarrollo de habilidades analíticas:

- Se fomentará la aplicación del modelado como una herramienta analítica para comprender, evaluar y diseñar soluciones.
- Los estudiantes practicarán la identificación de requisitos y la conceptualización de modelos que aborden problemas específicos.



## **Exploración de conceptos avanzados:**

- Si bien se proporciona una base sólida, se alienta a los estudiantes a investigar y aplicar conceptos avanzados.
- La guía brinda acceso a bibliografía y recursos en línea para enriquecer la comprensión y aplicación de los temas.

## **Bibliografía y recursos:**

- Se proporciona una lista de lecturas recomendadas y recursos en línea para ampliar el conocimiento.
- Estos recursos serán valiosos para abordar el caso práctico y comprender cómo se aplican los conceptos en el mundo real.

Lo invito a empaparse en un enfoque práctico y desafiante del Modelado de Sistemas, donde la aplicación directa de los conocimientos adquiridos le guiará hacia la resolución efectiva de problemas y la conceptualización de soluciones de diseño.

¡Bienvenidos a una experiencia educativa orientada a la acción!



### 3. Orientaciones didácticas por resultados de aprendizaje



#### Primer bimestre

##### Resultado de aprendizaje 1:

Crea modelos que apoyen el proceso de análisis de sistemas.

El modelado de sistemas facilita la comprensión y la comunicación de los aspectos más importantes en un proyecto, lo que resulta relevante para un análisis y especificación detallada. Para alcanzar el resultado de aprendizaje “Crea modelos que apoyen al proceso de análisis de sistemas”, empezaremos introduciendo conceptos fundamentales sobre el modelado que le permitirán sentar las bases para plantear modelos que le permitirán capturar los requisitos, procesos y componentes del sistema.

#### Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.



#### Semana 1

En esta primera unidad, la guía lo introducirá en el mundo del modelado de sistemas, una práctica esencial en el desarrollo de software que nos permitirá visualizar y comprender la complejidad de los sistemas de manera efectiva. A lo largo de las próximas semanas, exploraremos conceptos cruciales que nos ayudarán a representar estructuras, comportamientos y



relaciones en un formato accesible. Comenzaremos definiendo el modelado de sistemas, descubriremos la importancia de representar sistemas de manera visual y cómo esta práctica se traduce en una herramienta invaluable para desarrolladores, diseñadores y otros interesados (*stakeholders*) del proyecto. A lo largo de este proceso educativo, no solo adquirirá conocimientos teóricos, sino que también aplicará estos conceptos a través de ejercicios y proyectos prácticos que enriquecerán su experiencia de aprendizaje.

## **Unidad 1. Introducción al modelado de sistemas**

---

### **1.1. Definición de modelado de sistemas**

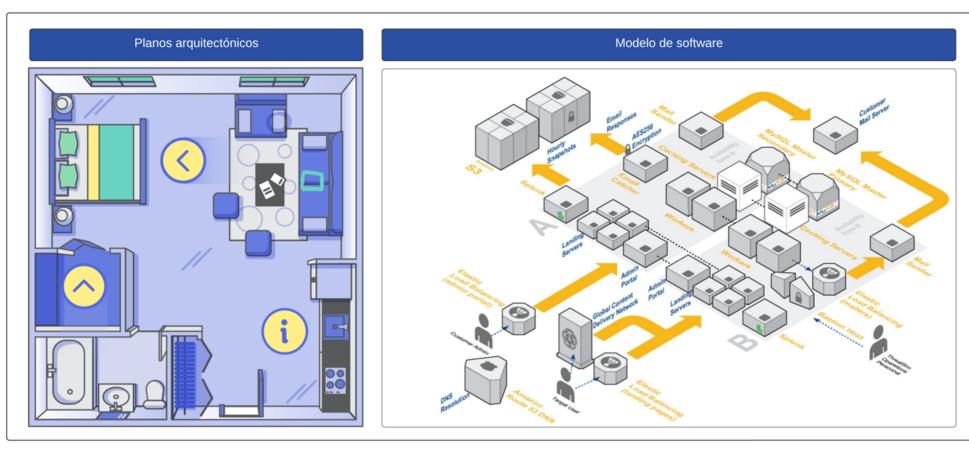
En el contexto del desarrollo de *software*, el modelado de sistemas emerge como una disciplina esencial, encargada de crear representaciones abstractas y visuales para sistemas complejos. Según James Rumbaugh (1991), coautor del influyente libro “Object-oriented modeling and design”, el modelado implica *“la construcción de abstracciones que describen la estructura, el comportamiento y otras características de un sistema o proceso”*. Este enfoque resalta la importancia de elaborar representaciones que no solo reflejen la realidad, sino que simplifiquen y se enfoquen en los aspectos cruciales del sistema.

Comparado con otros campos de la ingeniería, el modelado de sistemas es similar al uso de planos en arquitectura o esquemas en ingeniería mecánica. El uso de modelos para describir cómo se ven y se comportan las construcciones complejas es familiar para la mayoría: al igual que un arquitecto recurre a los planos para visualizar cómo debería ser una construcción, el ingeniero de *software* piensa en términos de modelos de objetos para describir la estructura y el comportamiento de sistemas de *software*. La figura 1 representa la analogía entre los planos arquitectónicos de una casa y un modelo de *software*.



**Figura 1**

*Comparación entre Planos Arquitectónicos de una Casa y el Modelo de Software: Analogía de la representación de Estructuras.*



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

La analogía con la ingeniería tradicional se extiende al concepto de abstracción. Al igual que un plano arquitectónico no detalla cada tornillo o pieza, un modelo de sistemas no describe cada línea de código. Sin embargo, ambas proporcionan una guía fundamental para la construcción, permitiendo a los ingenieros de software centrarse en los aspectos clave, simplificando la comprensión y facilitando la comunicación en equipos multidisciplinarios.

En resumen, el modelado de sistemas es el arte y la ciencia de crear representaciones que facilitan la comprensión y el diseño efectivo de sistemas de software. Al adoptar este enfoque, los desarrolladores no solo construyen software, sino que también construyen un lenguaje común que facilita la colaboración y el éxito en proyectos de envergadura.

## 1.2. Importancia del modelado de sistemas

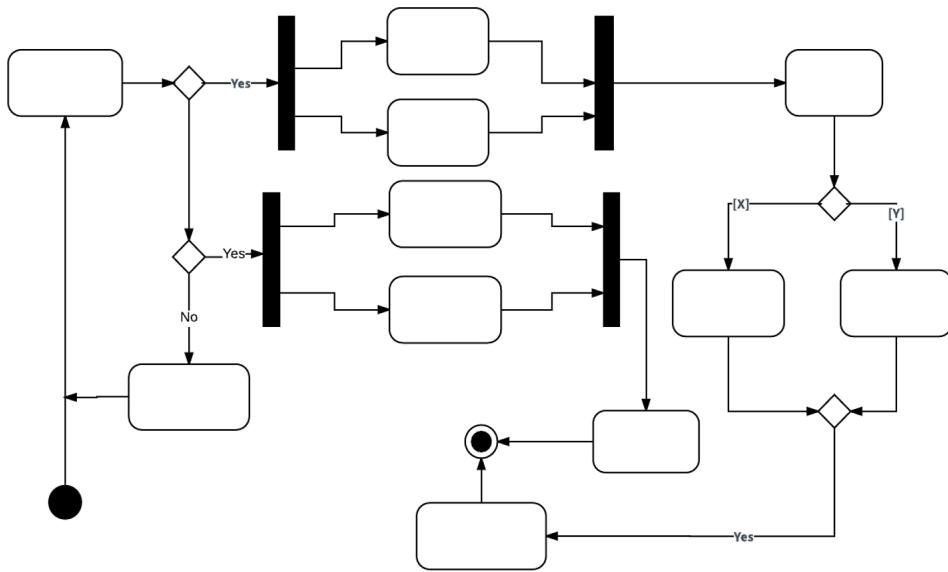
La importancia de modelar sistemas en el desarrollo de software se basa en la capacidad de abordar la complejidad inherente a los sistemas y proporcionar representaciones claras y estructuradas. Diversos autores y expertos en ingeniería de software han subrayado la relevancia de esta práctica crucial a la hora de abordar diversos casos de desarrollo de software.

Grady Booch (1990), colaborador en el desarrollo del método de modelado UML, destaca que el modelado "no es solo una herramienta que utilizamos para dar forma a nuestras visiones construidas; es una disciplina para representar nuestro entendimiento". Esta perspectiva resalta que el modelado no solo sirve como medio de documentación, sino como un proceso intrínseco para comprender, comunicar y estructurar la arquitectura de sistemas complejos. La figura 2 representa un flujo genérico de un diagrama de actividades.



**Figura 2**

Representación de diagrama UML



Nota. Tomado de *Tutorial de diagrama de actividades UML* [Ilustración], Lucidchart, 2024, [lucidchart](#), CC BY 2024.

La importancia de modelar sistemas se manifiesta también en la capacidad de anticipar problemas y efectuar ajustes antes de la implementación. Booch, menciona que “*los modelos son herramientas predictivas*”, permitiendo a los desarrolladores identificar posibles conflictos, definir requisitos más claramente y mejorar la calidad del software resultante.

Ivar Jacobson (2000), agrega otra dimensión a la importancia del modelado al describirlo como “*un vehículo para que las personas piensen*”. El modelado no solo es una herramienta para representar ideas, si no un medio para explorar y refinar conceptos, fomentando la creatividad y la innovación en el proceso de diseño de software.

Modelar sistemas no es simplemente una práctica técnica; es una disciplina esencial para la comprensión profunda, la comunicación efectiva y la mejora de la calidad en el desarrollo de software. Al adoptar esta perspectiva, los ingenieros de software no solo construyen sistemas, sino que también construyen conocimiento compartido y colaborativo que impulsa el éxito en proyectos complejos.

### 1.3. El Modelado de sistemas y la programación orientada a objetos

El modelado de sistemas y la Programación Orientada a Objetos (POO) están intrínsecamente entrelazados, compartiendo una relación complementaria que potencia la eficacia del desarrollo de software.

Grady Booch (2000), menciona que “*la esencia de la Programación Orientada a Objetos es la abstracción*”. La POO se centra en la creación de objetos, como elementos autónomos, que encapsulan datos y comportamientos. Estos objetos, a su vez, se deben modelar y representar visualmente en diagramas, permitiendo a los desarrolladores comprender e implementar sistemas de software. Le invito a revisar el [anexo 1](#) donde encontrará una Representación UML del modelado estructural de objetos.

El modelado de sistemas mediante UML (*Unified Modeling Language*) proporciona un conjunto de herramientas y notaciones que complementan los principios fundamentales de la POO. Los diagramas de clases, por ejemplo, permiten representar la estructura de un sistema, mostrando cómo las clases interactúan y se relacionan entre sí. La relación se extiende también a los diagramas de secuencia, que ilustran la interacción entre objetos a lo largo del tiempo, reflejando la ejecución de métodos y la comunicación entre componentes del sistema.

Ivar Jacobson destaca que “*el modelado y la Programación Orientada a Objetos son herramientas gemelas*”, subrayando cómo el acto de modelar proporciona una base sólida para la implementación práctica en la POO. Esta relación entre modelado y POO refuerza la importancia de una comprensión profunda de los principios de la POO al realizar modelado de sistemas. Al



visualizar y conceptualizar sistemas de manera Orientada a Objetos, los desarrolladores están mejor equipados para traducir esos modelos en implementaciones efectivas en código.

#### **1.4. Aplicaciones prácticas del modelado en diversos ámbitos**

El modelado de sistemas es una forma de proporcionar una representación estructurada y visual de los sistemas que sirve independientemente del contexto para el cual se esté usando. En la tabla 1, se describen varios ejemplos de sistemas en los cuales el modelado desempeña un papel fundamental, seguido de una explicación detallada sobre la importancia del modelado en cada uno de estos ámbitos. Cabe destacar que estos son algunos casos, las prácticas de modelado de sistemas son aplicables a todos los casos y contextos.



**Tabla 1**

*Casos de ejemplo del papel que desempeña el modelado de sistemas en diversos casos de implementación*

Ejemplo	Detalle
<b>Sistema de reservas de vuelos y hoteles</b>	El modelado de este sistema proporciona una vista de los casos de reserva y la interacción con los usuarios, lo que otorga una visión clara y general que facilita la implementación y evaluación de los procesos.
<b>Sistema de gestión de inventarios</b>	El modelado en este escenario permite optimizar los niveles de stock y mejorar la eficiencia operativa. Mediante el modelado, es posible representar y visualizar, por ejemplo, los flujos de inventario para identificar áreas de mejora y tomar decisiones informadas para una gestión más efectiva de los recursos.
<b>Sistema de control de tráfico</b>	El modelado permite planificar y gestionar los flujos de tráfico, lo que permite identificar cuellos de botella para optimizar la coordinación de señales de tráfico para una circulación más eficiente.
<b>Sistema de gestión de proyectos de construcción</b>	El modelado de sistemas de gestión de proyectos ayuda a planificar, programar y asignar recursos eficientemente. Mediante el modelado de sistemas es posible representar a través de diagramas de secuencia los pasos, los involucrados y el flujo óptimo para automatizar la ejecución de proyectos y garantizar su finalización dentro del tiempo y el presupuesto previstos.
<b>Sistema de gestión de bibliotecas</b>	Estos sistemas se benefician del modelado por ejemplo para organizar y catalogar libros. Al modelar la estructura de la biblioteca y los procesos de préstamo, es posible mejorar disponer de una vista general de los diversos procesos para automatizar la gestión de los recursos bibliográficos.



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas*, por Correa, R., 2024, Ediloja Cía. Ltda.

Claro está que estos son solo algunos ejemplos. La importancia del modelado radica en cualquier ámbito de aplicación, ya que ofrece importantes ventajas a los equipos de desarrollo. A continuación, se destacan algunas de estas capacidades en la siguiente infografía:

#### Capacidades del modelado en equipos de desarrollo



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el capítulo 1 del libro "[Software engineering with UML](#)" de Bhuvan Unhelkar. Identifique en qué fase del ciclo de vida del desarrollo de software se aplica el proceso de modelado de sistemas y reflexione sobre su importancia.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.
2. Revise el siguiente video sobre [Introduction to software development lifecycle | What is software development?](#), este le proporcionará una visión general del ciclo de vida del desarrollo de software, explicando las fases clave como planificación, análisis, diseño, desarrollo, pruebas, implementación y mantenimiento. Es fundamental comprender cómo cada etapa contribuye al desarrollo de un software eficiente y de calidad.
3. Le invito a realizar la siguiente la autoevaluación para comprobar sus conocimientos. En este cuestionario, usted podrá evaluar su nivel de comprensión sobre los temas revisados en esta primera unidad. El objetivo es que pueda medir sus conocimientos sobre la importancia del modelado de sistemas e identificar posibles puntos de mejora que le sirvan para mejorar sus conocimientos.



### [Autoevaluación 1](#)

Elija la opción correcta a las siguientes interrogantes:

1. ¿Qué es el **modelado de sistemas**?
  - a. La construcción de sistemas utilizando diagramas de flujo.



- b. La creación de representaciones visuales de sistemas complejos.
  - c. La implementación de sistemas utilizando un lenguaje específico.
  - d. La documentación de requerimientos no funcionales.
- 2. ¿Cuál es una de las principales ventajas del modelado de sistemas?**
- a. Elimina la necesidad de escribir código.
  - b. Simplifica la comprensión y facilita la comunicación entre los stakeholders.
  - c. Reduce el tiempo de desarrollo a la mitad.
  - d. Sustituye la necesidad de pruebas de software.
- 3. ¿Qué similitud tiene el modelado de sistemas con otras disciplinas de ingeniería?**
- a. No tiene ninguna similitud.
  - b. Se utiliza únicamente para documentar requerimientos.
  - c. Es similar al uso de planos en arquitectura y esquemas en ingeniería mecánica.
  - d. Se enfoca en la implementación directa del código.
- 4. ¿Por qué es importante la abstracción en el modelado de sistemas?**
- a. Permite detallar cada línea de código.
  - b. Facilita la visualización de los aspectos clave del sistema sin detallar todo.
  - c. No es importante; los detalles son más cruciales.
  - d. Solo se usa para simplificar diagramas.
- 5. ¿Cómo se complementa el modelado de sistemas y la programación orientada a objetos?**
- a. El modelado de sistemas proporciona una forma en cómo se puede representar y diseñar los objetos.
  - b. Elimina la necesidad de diseñar objetos.
  - c. Reemplaza la programación orientada a objetos.
  - d. Se usa solo para documentar la arquitectura.
- 6. ¿Qué representa un diagrama de clases en UML?**
- a. La interacción entre usuarios y el sistema.



- b. La estructura de un sistema mostrando cómo las clases interactúan y se relacionan.
  - c. La secuencia de actividades en un proyecto.
  - d. La infraestructura física de un sistema.
7. **¿Cuál es un beneficio clave de modelar sistemas antes de la implementación?**
- a. Asegura que no habrá errores en el código.
  - b. Reduce el costo de los recursos de *hardware*.
  - c. Elimina la necesidad de pruebas de *software*.
  - d. Ayuda a identificar problemas y realizar ajustes antes de la implementación.
8. **¿Cómo contribuye el modelado de sistemas a la gestión de la complejidad en el desarrollo de software?**
- a. Detallando cada aspecto del sistema de manera exhaustiva.
  - b. Eliminando la necesidad de equipos multidisciplinarios.
  - c. Simplificando la implementación sin abstracción.
  - d. Dividiendo el sistema en partes manejables y enfocándose en aspectos particulares.
9. **¿Qué se entiende por “modelos como herramientas predictivas” en el contexto del desarrollo de software?**
- a. Modelos que solo se utilizan para predecir costos.
  - b. Modelos que ayudan a identificar posibles conflictos y mejorar la calidad del software.
  - c. Modelos que se usan únicamente para la documentación final.
  - d. Modelos que predicen la eficiencia de los desarrolladores.
10. **¿Qué herramienta es comúnmente utilizada para el modelado de sistemas orientados a objetos?**
- a. JavaScript
  - b. Unified Modeling Language (UML)
  - c. SQL
  - d. HTML

[Ir al solucionario](#)





## Resultado de aprendizaje 2:

Identifica las partes esenciales de un sistema utilizando los diferentes tipos de modelado.

Para alcanzar el resultado planteado, usted aprenderá sobre el modelo 4+1 vistas de Kruchten que es un enfoque fundamental para el modelado de sistemas de software. Este modelo se compone de cinco vistas: lógica, de desarrollo, de procesos, física y de casos de uso. Cada una de estas vistas aporta una perspectiva única que, en conjunto, le brindará una visión completa del sistema. Al familiarizarte con estas vistas, aprenderá a identificar las partes esenciales de un sistema, lo que le proporcionará una base sólida para el diseño y la conceptualización de soluciones arquitectónicas en el futuro.

### Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.



#### Semana 2

#### Unidad 2. Modelo 4 + 1 vistas de Krutchen

En esta unidad, revisaremos el modelo 4+1 vistas de Philippe Kruchten, una metodología que ofrece una perspectiva única y estructurada para analizar sistemas desde diferentes ángulos. Durante esta semana, exploraremos en detalle los cinco puntos de vista del modelo. Desde la perspectiva lógica hasta la de desarrollo, examinaremos cómo este modelo proporciona una herramienta poderosa para comprender la arquitectura de sistemas desde diversas dimensiones. Destacaremos la necesidad de considerar múltiples

perspectivas y la interacción entre ellas en escenarios específicos, asegurando una comprensión sólida tanto teórica como práctica del Modelo 4+1. Utilizando recursos como el artículo original de Kruchten (1995), y obras especializadas, los participantes identificarán la utilidad de los diagramas en su representación y exposición, garantizando así un aprendizaje significativo en una semana dedicada a este estudio.

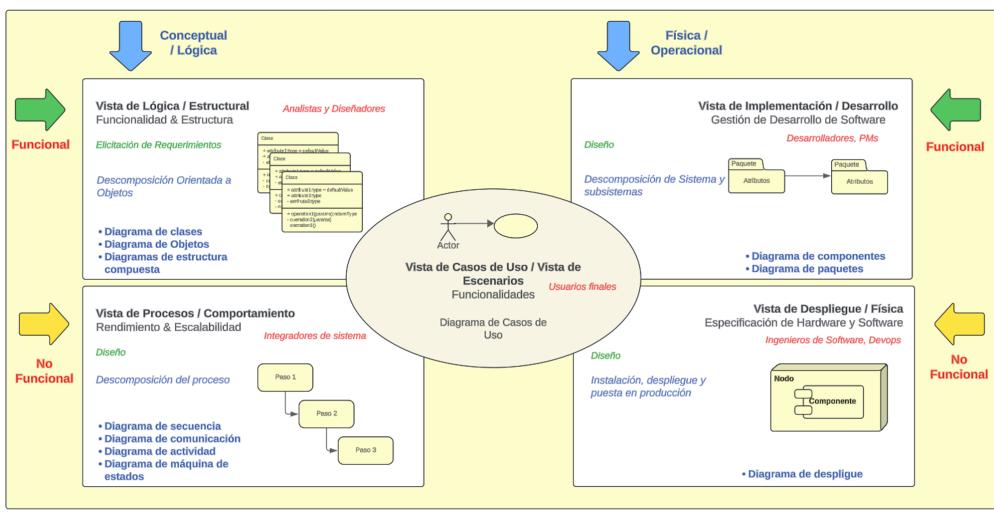
## 2.1. Introducción al modelo 4 + 1 de Krutchen

El modelo 4+1 de Philippe Kruchten es una forma de representar diferentes puntos de vista en forma de una estructura que toma como base los escenarios o casos de uso para especificar diferentes niveles de abstracción. Este modelo hace uso de 5 puntos de vista distintos, cada uno se enfoca en aspectos específicos del sistema, que en conjunto proporcionan una comprensión integral y general del diseño arquitectónico. Los cinco puntos de vista son la vista lógica, la vista de desarrollo, la vista de proceso, la vista física y la vista escenario. En la siguiente figura se representan los tipos de diagramas y artefactos que deben ser utilizados en la representación de cada una de las vistas del modelo 4+1 de Krutchen.



**Figura 3**

Modelo 4+1 de Kruchten



Nota. Adaptado de 4+1 View Model of Software Architecture [Ilustración], por bashcode, 2016, [Slideshare](#), CC BY 4.0.

La necesidad de considerar múltiples perspectivas radica en comprender que un único punto de vista no permite capturar la estructura total de un sistema. Cada vista del modelo 4+1 tiene un objetivo claro y un nivel de representación que permite a los arquitectos y desarrolladores comprender diferentes aspectos sin perder la necesidad global. Esta metodología, respaldada por la experiencia de Kruchten, proporciona un marco robusto que guía a la creación de arquitecturas de software sólidas y comprensibles desde múltiples dimensiones.

## 2.2. Vista de escenario

La vista de escenario en el modelo 4+1 de Kruchten es importante para establecer el contexto, alcance y límites del sistema. Su uso tiene como objetivo crear una representación que permita fácilmente a los involucrados comprender las diferentes interacciones necesarias del sistema para cumplir con sus objetivos de negocio. Antes de iniciar el diseño de cualquier sistema,

es crucial partir desde la etapa de análisis, esto implica identificar las necesidades de los usuarios, comprender los procesos de negocio y determinar cómo se va a abordar el problema.

La vista de escenario es la representación de alto nivel, que considera los requisitos funcionales más relevantes y debe ser representada mediante el diagrama de casos de uso. Esta representación permite comprender las interacciones entre el sistema y su entorno, así como para capturar los requisitos funcionales desde el punto de vista del usuario, identificando las interacciones entre los actores y el sistema (Cockbum, 2000).

### 2.3. Vista lógica

La vista lógica del modelo 4+1 de Kruchten se enfoca en la estructura interna del sistema y en cómo se organiza y gestiona la lógica de negocio. Permite crear una representación detallada de las clases, objetos y relaciones que componen el sistema, permitiendo así describir su estructura y funcionalidad. El desglose en clases y objetos ayuda a identificar mecanismos comunes y componentes de diseño en diferentes secciones del sistema.

El diagrama de clases es uno de los diagramas más utilizados para representar la vista lógica. Este diagrama muestra las clases en un sistema junto con sus atributos y métodos, proporcionando una descripción clara de la estructura de datos del sistema y las interrelaciones entre las diferentes entidades de clase. El diagrama de clases es muy importante para entender cómo se estructuran y comportan los objetos del sistema (Booch, 2005).

### 2.4. Vista de proceso

La vista de procesos se centra en los aspectos dinámicos del sistema, describiendo cómo se ejecutan los procesos y cómo se comunican entre sí. Esta vista proporciona una representación detallada de los procesos y flujos de trabajo que se automatizan en la implementación del sistema.



Esta vista proporciona una guía detallada para representar tanto los procesos de negocio como los de desarrollo de software. En este contexto, es crucial destacar que los procesos, flujos y secuencias que se desarrollan a nivel lógico deben ser claramente especificados. Esto permite establecer formas de representar el proceso a través de las comunicaciones entre objetos, facilitando la comprensión y optimización de los procesos, lo que contribuye significativamente a la eficiencia operativa y al éxito del proyecto.

Los diagramas UML utilizados para representar la vista de procesos son:

- **Diagrama de actividad:** se enfoca en representar los procesos y las decisiones que se toman en tiempo de ejecución. Esto ayuda a tener claro el paso a paso de las tareas y también cómo fluye la información entre ellas.
- **Diagrama de secuencia:** describe la interacción entre diferentes objetos en el sistema en tiempo de ejecución. Es útil para mostrar casos de interacción entre los componentes del sistema, mostrando el orden de ejecución, llamadas y tipo de los mensajes enviados entre sí.

## 2.5. Vista de desarrollo

Esta vista se enfoca en la representación y estructura del código fuente y otros artefactos que pueden ser módulos o submódulos del sistema, los cuales pueden ser organizados mediante el uso de estilos y patrones arquitectónicos. Estos estilos y patrones ofrecen soluciones probadas y reconocidas para resolver problemas comunes en el diseño de software, contribuyendo así a mejorar la eficiencia, mantenibilidad y escalabilidad del sistema.

En esta vista, se emplean dos tipos de diagramas principales: el diagrama de componentes y el diagrama de paquetes.

- **Diagrama de componentes:** se utiliza para describir los componentes, módulos y submódulos del sistema y sus conexiones en cuanto a implementación.



- **Diagrama de paquetes:** se utiliza para describir la organización de los artefactos de software en paquetes, módulos o submódulos.

La vista de desarrollo desempeña un papel crucial en garantizar la calidad y mantenibilidad del software. Ofrece una orientación detallada para organizar y gestionar los artefactos de desarrollo, lo que facilita la colaboración entre los miembros del equipo y promueve la evolución continua del sistema a lo largo del tiempo (Martín, 2009).

## 2.6. Vista física

Esta vista se enfoca en la configuración física del sistema y en cómo se organizan los componentes del sistema en el contexto de implementación. Toma como referencia principalmente los requisitos no funcionales del sistema, como disponibilidad, confiabilidad, rendimiento, escalabilidad, entre otros. Además, define una forma de representación de la infraestructura de hardware y software necesaria para ejecutar el sistema en un entorno de ejecución. Los componentes del sistema, como aplicaciones, servidores y bases de datos, son elementos importantes que se pueden representar en la vista de despliegue.

La vista de física también aborda los entornos de implementación del sistema, como desarrollo, pruebas y producción. Describe cómo se configuran y gestionan estos entornos para admitir el desarrollo, la prueba y la ejecución del software en cada etapa del ciclo de vida del proyecto (Humble, 2010).

El tipo de diagrama comúnmente utilizado para representar esta vista es el diagrama de despliegue.





## Actividades de aprendizaje recomendadas



Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el artículo denominado [The 4+1 View Model of architecture.](#)

Identifique y describa cada una de las cinco vistas del modelo 4 + 1 y explique cómo cada vista contribuye a la comprensión global del sistema. Este ejercicio le ayudará a entender cómo el modelado arquitectónico puede clarificar la estructura del sistema y facilitar su diseño y comunicación.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.

2. Revise el video [What's software architecture : the 4+1 architectural views](#), reflexione sobre cómo cada vista ayuda a entender y comunicar diferentes aspectos de la arquitectura del software. Este video le proporcionará una comprensión mucho más visual de cómo cada vista contribuye a una representación completa del sistema, ayudándole a aplicar estos conceptos en el diseño y análisis de sistemas de manera más efectiva.
3. En la unidad 2 ha revisado el modelo 4+1 vistas de Kruchten, modelo que proporciona una perspectiva integral de la arquitectura de software desde diferentes perspectivas. Este conocimiento le servirá para analizar y diseñar sistemas complejos de manera más efectiva. Ahora, para consolidar y aplicar lo aprendido, le invito a realizar la siguiente autoevaluación mediante el desarrollo de esta, podrá poner en práctica sus conocimientos y medir su conocimiento de los conceptos clave estudiados en esta semana.



## Autoevaluación 2

Elija la opción correcta a las siguientes interrogantes:

1. **¿Cuál es el objetivo principal del Modelo 4+1 Vistas de Kruchten?**

- a. Simplificar la programación de sistemas.
- b. Proporcionar una perspectiva integral de la arquitectura de software desde diferentes ángulos.
- c. Reemplazar los diagramas UML.
- d. Eliminar la necesidad de documentación en proyectos de software.

2. **¿Qué vista en el Modelo 4+1 se enfoca en las interacciones entre el sistema y su entorno?**

- a. Vista lógica.
- b. Vista de desarrollo.
- c. Vista de proceso.
- d. Vista de escenario.

3. **¿Qué tipo de diagrama se utiliza comúnmente para representar la vista lógica?**

- a. Diagrama de actividad.
- b. Diagrama de clases.
- c. Diagrama de despliegue.
- d. Diagrama de secuencia.

4. **¿Qué vista del Modelo 4+1 se centra en los aspectos dinámicos del sistema?**

- a. Vista física.
- b. Vista lógica.
- c. Vista de proceso.
- d. Vista de desarrollo.

5. **¿Qué tipo de diagrama se usa para mostrar el flujo de trabajo en la Vista de proceso?**

- a. Diagrama de clases.
- b. Diagrama de paquetes.
- c. Diagrama de actividad.



d. Diagrama de despliegue.

**6. En la vista de desarrollo, ¿qué diagrama representa la estructura de agrupación de los artefactos de software?**

- a. Diagrama de clases.
- b. Diagrama de actividad.
- c. Diagrama de componentes.
- d. Diagrama de paquetes.

**7. ¿Qué vista del Modelo 4+1 se enfoca en la configuración física del sistema?**

- a. Vista física.
- b. Vista lógica.
- c. Vista de escenario.
- d. Vista de desarrollo.

**8. ¿Qué diagrama se utiliza para representar la topología de red en la Vista física?**

- a. Diagrama de clases.
- b. Diagrama de paquetes.
- c. Diagrama de despliegue.
- d. Diagrama de secuencia.

**9. ¿Cuál es una de las ventajas principales de utilizar múltiples vistas en el Modelo 4+1?**

- a. Eliminar la necesidad de pruebas de software.
- b. Capturar diferentes aspectos del sistema sin perder coherencia global.
- c. Reducir el tiempo de desarrollo a la mitad.
- d. Simplificar el código fuente.

**10. ¿Qué vista del Modelo 4+1 se utiliza para entender los requisitos funcionales desde la perspectiva del usuario?**

- a. Vista de escenario.
- b. Vista física.
- c. Vista lógica.
- d. Vista de desarrollo.

[Ir al solucionario](#)





### Resultado de aprendizaje 3:

Utiliza UML como lenguaje de construcción de modelos para modelar componentes de un sistema de información.

UML, o Lenguaje de Modelado Unificado, es una herramienta importante para la construcción de modelos de sistemas de software debido a su capacidad para proporcionar una representación visual clara y estandarizada de los diferentes componentes y sus interacciones. Al utilizar UML, podrá descomponer un sistema en sus partes, visualizar cómo estos componentes se relacionan entre sí y cómo interactúan para cumplir con los requisitos del sistema. Para lograr el resultado de aprendizaje, primero se familiarizará con los diferentes tipos de diagramas UML básicos, como los diagramas de contexto, casos de uso y secuencia. Luego, aplicará estos diagramas para modelar componentes específicos de un sistema, mostrando sus estructuras internas y sus conexiones. Esto le permitirá diseñar, analizar y comunicar la arquitectura de sistemas de manera efectiva y precisa.

### Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.



### Semana 3

#### Unidad 3. Diagramas UML básicos

En esta unidad, revisaremos los primeros diagramas UML básicos. Durante las próximas semanas, aprenderemos una variedad de tipos de diagramas UML, cada uno diseñado para representar diferentes puntos de vista del sistema. Empezaremos con diagramas de estructura y complementaremos

el conocimiento de diagramas de comportamiento, además podrá identificar herramientas que le permitan crear diagramas UML de una forma sencilla y práctica haciendo uso correcto de la notación. Al finalizar esta unidad, estará capacitado para utilizar UML como una forma de crear modelos, para representar y analizar los componentes de un sistema.

### 3.1. ¿Qué es UML?

El Lenguaje de Modelado Unificado (UML), es un estándar de la industria para la visualización, especificación, construcción y documentación de sistemas *software*. Es una notación utilizada por los ingenieros de *software* para capturar y comunicar la estructura y el comportamiento de un sistema de *software* de manera clara y concisa.

UML, es un Lenguaje de Modelado Unificado que puede ser utilizado para representar el sistema en diferentes niveles y puede ser usado en múltiples etapas del ciclo de vida de desarrollo. Su objetivo es establecer un lenguaje común en el que todos los involucrados en un proyecto tengan la facilidad de comprender, comunicar y especificar los diferentes contextos del producto de *software*. UML especifica diferentes vistas y por cada vista múltiples diagramas que permiten representar un sistema, desde el contexto, su estructura hasta su comportamiento dinámico.

### 3.2. Herramientas de modelado UML

Para crear modelos UML, es necesario apoyarse en herramientas que permitan crear este tipo de especificaciones. En ese sentido, le invito a revisar la siguiente infografía donde se destacan algunas herramientas CASE que se usan en empresa y que le pueden servir en sus actividades de modelado.

[Herramientas para el modelado UML](#)



### 3.3. Vistas UML

Las vistas UML son representaciones que muestran aspectos específicos de un sistema desde diferentes ángulos. Cada vista UML se centra en un conjunto particular de elementos y relaciones dentro del sistema, lo que permite a los diseñadores y desarrolladores enfocarse en áreas específicas de interés.

Las vistas UML son importantes en el ciclo de desarrollo de software por diversas razones, entre las que se destaca:

- **Comprendión holística:** permite representar diferentes aspectos, como la estructura, el comportamiento y la interacción de los componentes.
- **Comunicación efectiva:** facilitan la comunicación entre los miembros del equipo y las partes del proyecto.
- **Análisis detallado:** permite exponer cada aspecto de construcción del sistema, permite identificar posibles problemas y tomar decisiones durante el ciclo de vida de desarrollo.
- **Documentación integral:** es una forma sencilla de documentar el diseño, construir el modelo y crear la arquitectura del sistema.

### 3.4. Tipos de vistas UML

Existen varios tipos de vistas UML, cada una enfocada en aspectos específicos del sistema. A continuación, en la tabla 2, se describen las áreas, la clasificación de vistas y tipos de diagramas UML que permiten crear representaciones de cada una de las perspectivas del modelado de sistemas:



**Tabla 2***Vistas y diagramas UML*

Área	Vista	Diagrama
Estructural	Estática	Diagrama de clases.
		Estructura interna.
	Diseño	Diagrama de colaboración.
		Diagrama de componentes.
Dinámica	Casos de uso	Diagrama de casos de uso.
	Máquinas de estado	Diagrama de máquina de estados.
	Actividad	Diagrama de actividad.
	Interacción	Diagrama de secuencia. Diagrama de comunicación.
Física	Despliegue	Diagrama de despliegue.
Gestión de modelo	Gestión del proyecto	Diagrama de paquetes.
	Perfil	

Nota. Adaptado de *The unified modeling language reference manual*, por Jacobson, L., Booch, J., 2021.

De acuerdo con lo expuesto en la tabla, las vistas que se estudiarán en esta guía tienen los siguientes objetivos:

- **Vista de contexto:** representar las interacciones entre el sistema y las entidades externas, proporcionando una visión general de alto nivel del sistema y su entorno.
- **Vista de estructural:** se centra en la estructura estática del sistema, mostrando los componentes y sus relaciones.
- **Vista de dinámica:** se enfoca en el comportamiento dinámico del sistema y la interacción entre sus componentes.



- **Vista de casos de uso:** representa los diferentes escenarios del sistema y la interacción de los actores.
- **Vista de despliegue:** muestra la distribución física de los componentes del sistema en el entorno de implementación.

En las siguientes secciones, se amplía a detalle cada una de estas vistas UML.

### 3.5. Vista de contexto

La vista de contexto es una forma de representar las interacciones entre un sistema y su entorno. Es la vista de más alto nivel que permite de un único vistazo entender cómo el sistema interactúa con entidades externas, como otros sistemas, usuarios o dispositivos. Describe el alcance, los límites del sistema y sus interfaces con las entidades externas. Este diagrama es una forma sencilla y uno de los primeros diagramas que se elabora para definir el alcance del sistema y a comprender cómo se integra en su contexto.

La vista de contexto es importante por varias razones:

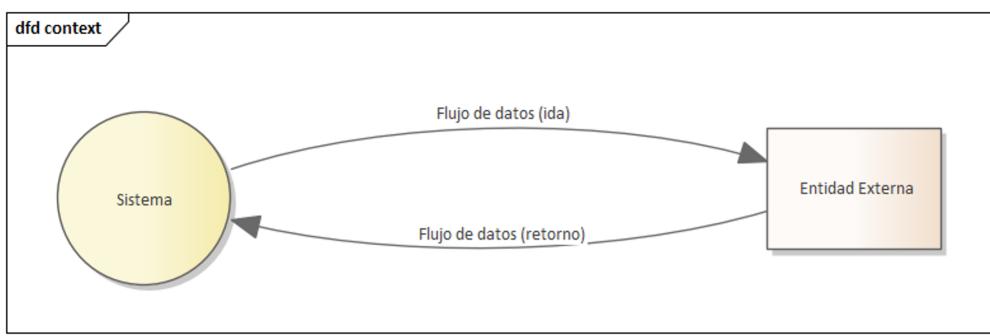
- **Definición de alcance:** permite establecer claramente los límites del sistema, identificando qué está dentro y fuera del alcance de este.
- **Claridad en las interacciones:** proporciona una comprensión de las interacciones entre el sistema y sus entidades externas.
- **Identificación de interfaces:** permite identificar las interfaces del sistema con otros sistemas o usuarios.
- **Comunicación efectiva:** facilita la comunicación entre las partes interesadas.

### 3.5.1. Diagrama de contexto

El diagrama de contexto ofrece una vista general de cómo el sistema interactúa con su entorno. En este diagrama, el sistema es el único elemento central, mientras que las entidades externas se representan como cajas externas que interactúan alrededor con el sistema. Los componentes de diseño de un diagrama de contexto son los que se describen en la figura 4:

**Figura 4**

*Componentes de un diagrama de contexto.*



Nota. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

#### El sistema:

- Representado como un único proceso o entidad central.
- Es el sujeto del diagrama y el foco principal.

#### Las entidades externas:

- Entidades externas que interactúan con el sistema.
- Pueden ser usuarios, sistemas externos, organizaciones, etc.

#### Los flujos de datos:

- Representan las entradas y salidas de datos entre el sistema y los actores.

- Dibujados como flechas que indican la dirección del flujo de información.

### 3.5.2. Modelando un diagrama de contexto

Para modelar un diagrama de contexto es importante partir por efectuar un análisis profundo del problema para identificar las necesidades de alto nivel de la solución. Con esta base usted podrá elaborar un diagrama de contexto de manera clara y sencilla:

#### 1. Identificar el sistema:

- Definir claramente el sistema que se está modelando.
- Dibujar el sistema como un círculo o un rectángulo en el centro del diagrama.

#### 2. Identificar los actores externos:

- Identificar todas las entidades externas que interactúan con el sistema.
- Dibujar cada actor alrededor del sistema.

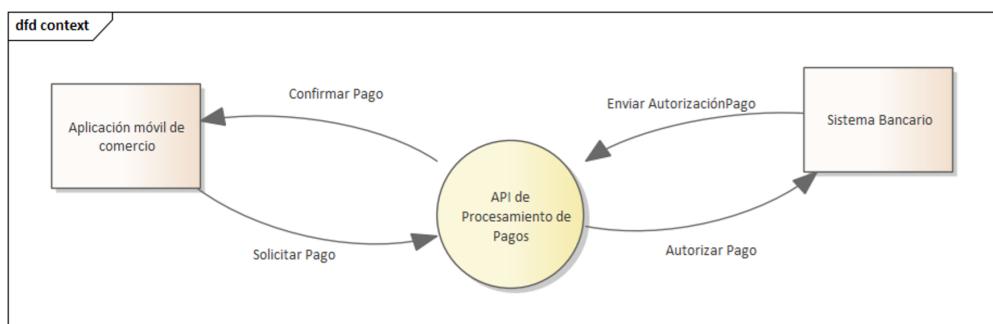
#### 3. Definir los flujos de datos:

- Identificar los datos que se intercambian entre el sistema y los actores.
- Dibujar flechas para representar cada flujo de datos, indicando la dirección del flujo.

Consideremos un sistema que proporciona una API para el procesamiento de pagos con tarjeta de crédito. El diagrama de contexto para este sistema podría ser el que se representa en la figura 5:

## Figura 5

Ejemplo de diagrama de contexto.



Nota. Esta figura representa la solución a nivel de diagrama de contexto del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía.

En este contexto, el diagrama de la figura es clave para comprender la interacción entre los componentes del sistema.



En una API, esta interactúa directamente con un sistema, sin la necesidad de un usuario. No obstante, en el caso de soluciones completas como ERP o CRM, los usuarios finales se relacionan directamente con la aplicación y utilizan sus características para manejar procesos de negocio.



### Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Observe el siguiente video [Context diagrams overview](#) y refuerce los conceptos aprendidos.
2. Realice un breve ejemplo considerando el contexto del ejemplo anteriormente descrito en el apartado 3.5.2. Modifique el diagrama y

agregue al contexto una plataforma e-Commerce como un cliente del API.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.

3. Revise el blog [\*What is system context diagram?\*](#) Explore los ejemplos de diagramas de contexto presentados y analice cómo se utilizan para mostrar las interacciones entre un sistema y su entorno.





## Resultado de aprendizaje 3:

Utiliza UML como lenguaje de construcción de modelos para modelar componentes de un sistema de información.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 4

##### Unidad 3. Diagramas UML básicos

En esta semana, abordaremos la Vista de casos de uso. Esta perspectiva proporciona una visión de alto nivel que nos permite identificar y capturar requisitos funcionales del sistema. Nos centraremos específicamente en el diagrama de casos de uso, explorando su estructura y relaciones entre los diferentes elementos, la nomenclatura adecuada y la especificación necesaria para comprender cómo se utilizarán y comportarán los sistemas en diversos escenarios.

#### 3.6. Vista de escenarios o casos de uso

La vista de casos de uso permite comprender los requisitos funcionales de un sistema. Esta vista se enfoca en identificar y capturar los diferentes casos de uso del sistema, para mostrar cómo los actores externos (usuarios y sistemas) interactúan con el sistema.

La vista de casos de uso es importante por:

- **Identifica los requisitos funcionales:** permite identificar y clarificar las necesidades funcionales del sistema desde el punto de vista de los actores.

- **Claridad en las interacciones:** permite representar la interacción entre el actor y el sistema.
- **Validación del diseño:** permite validar el diseño del sistema, identificando todos los casos de uso relevantes.
- **Comunicación efectiva:** facilita la comunicación entre los desarrolladores y los demás interesados, proporcionando una representación visual de alto nivel de las interacciones del sistema y las funcionalidades.



### 3.6.1. Diagrama de casos de uso

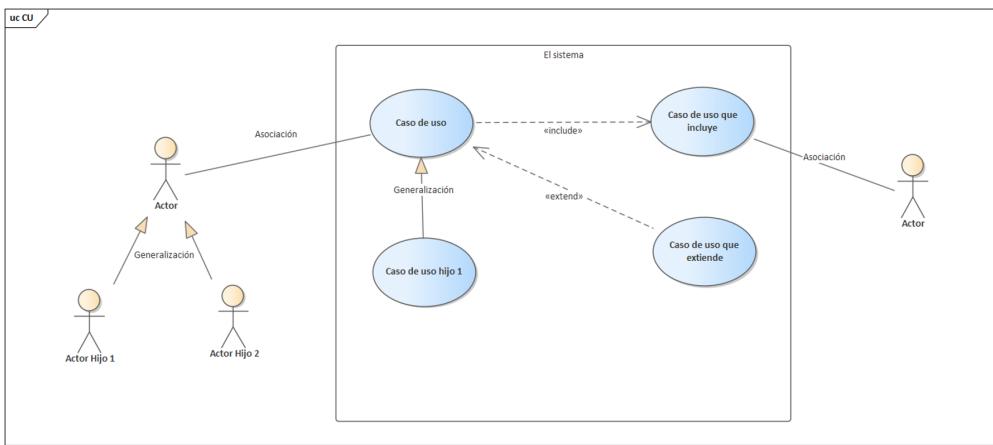
Un caso de uso es una técnica de modelado utilizada para definir y organizar los requisitos funcionales de un sistema. Representa las interacciones entre los actores externos y el sistema para lograr un objetivo particular. Los diagramas de casos de uso son esenciales para comprender cómo se utiliza un sistema y para identificar los requisitos del sistema desde la perspectiva del usuario.

Un caso de uso describe una secuencia de pasos que un actor realiza en el sistema para generar un resultado observable. Como se observa en la figura 6, los componentes de un diagrama de casos de uso permiten ilustrar la interacción entre actores y el sistema para cumplir con ciertos objetivos.

Los actores son entidades externas que pueden ser usuarios, otros sistemas o dispositivos que interactúan con el sistema. El sistema se representa generalmente como un rectángulo que agrupa todos los casos de uso. Los casos de uso son elipses/óvalos dentro del sistema que describen funcionalidades específicas expuestas por el sistema a los actores. Las asociaciones son líneas que conectan los actores con los casos de uso. Cada asociación expresa un significado diferente, por ejemplo, para el caso de las relaciones de <<include>> y <<extend>>, estas permiten indicar que un caso de uso incluye o extiende la funcionalidad de otro caso de uso, también las relaciones de generalización pueden demostrar herencia y especialización tanto entre actores como entre casos de uso.

**Figura 6**

Componentes de un diagrama de casos de uso



Nota. Esta figura representa los componentes de un diagrama de casos de uso. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

### 3.6.2. Modelando un diagrama de casos de uso

Para ilustrar cómo crear un diagrama de casos de uso, vamos a desarrollar un ejemplo práctico para un sistema de API de procesamiento de pagos de tarjeta de crédito. Aquí, identificaremos los actores, los casos de uso, y las relaciones entre ellos, aplicando algunos de los *tips* mencionados.

#### Actores

- **Aplicación móvil de comercio:** una aplicación utilizada por los clientes para realizar compras y enviar solicitudes de pago.
- **Sistema bancario:** el sistema bancario que procesa las transacciones de pago.

Para completar este ejercicio en su cuaderno de apuntes o en un documento Word, siga las siguientes instrucciones:



- Enumera todos los posibles usuarios y sistemas externos que podrían interactuar con su sistema. Los actores pueden ser usuarios, sistemas externos, dispositivos, etc.
- Agrupa actores similares para simplificar el diagrama.

## Casos de uso

- **Procesar pago:** la aplicación móvil de comercio envía una solicitud de pago.
- **Autorizar pago:** el sistema bancario autoriza la transacción.
- **Verificar transacción:** el administrador del sistema verifica el estado de las transacciones.

Una forma correcta para el nombramiento correcto de casos de uso es haciendo uso del **verbo + sustantivo**. Esto ayuda a describir claramente la acción y el objetivo del caso de uso. Por ejemplo:

- **Procesar pago:** representa la acción de procesar un pago.
- **Verificar transacción:** representa la acción de comprobar una transacción.

## Relaciones

- Procesar Pago incluye (<<include>>) Autorizar Pago.
- Verificar Transacción extiende (<<extend>>) Procesar Pago.

Use relaciones de inclusión y extensión correctamente.

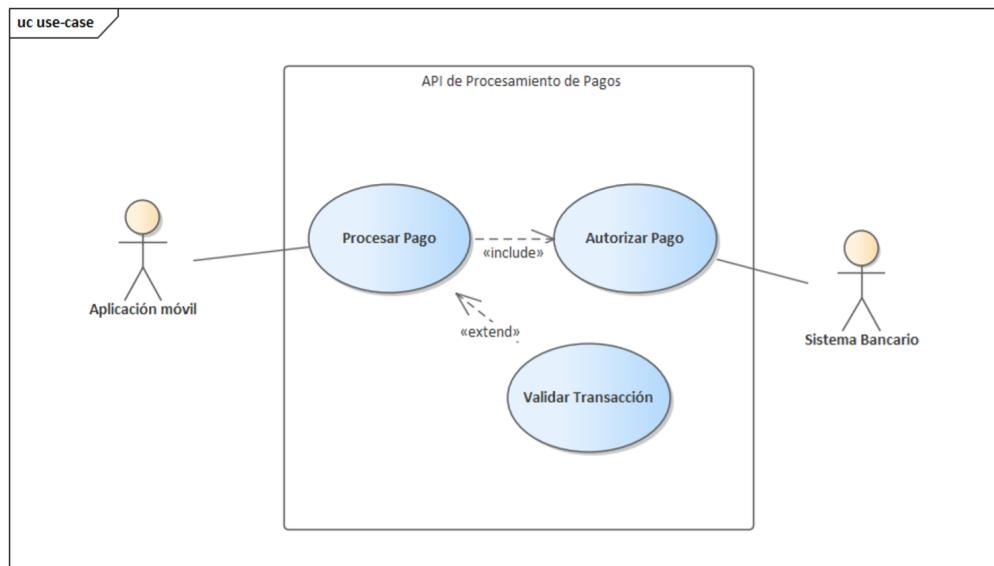
- **<<include>>**: utilízalo cuando un caso de uso siempre incluya la funcionalidad de otro caso de uso.
  - **Ejemplo:** un caso de uso “Procesar Pedido” puede incluir “Verificar Disponibilidad de Stock”.

- <<extend>>: utilízalo cuando un caso de uso opcionalmente extienda la funcionalidad de otro caso de uso bajo ciertas condiciones.
  - **Ejemplo:** un caso de uso “Registrar Usuario” puede ser extendido por “Enviar Email de Confirmación” bajo ciertas condiciones.

En la figura 7, se describe el diagrama resultante del proceso de elaboración del diagrama y representación a través de elementos UML.

### Figura 7

*Representación de un diagrama de casos de uso en base al ejemplo práctico desarrollado en la guía didáctica.*



Nota. Esta figura representa la solución a nivel de diagrama de casos de uso del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise la sección de casos de uso en el capítulo 2 del libro “Software engineering with UML” de Bhuvan Unhelkar. Refuerce los conceptos e identifique los componentes clave de un diagrama de casos de uso y analice los ejemplos proporcionados en el libro en mención.
2. Imagine que la API de procesamiento de pagos de tarjeta de crédito ha sido actualizado para incluir una nueva funcionalidad: la capacidad de “Reembolsar Pago”. Analice y plantee de qué forma debería representar el nuevo requerimiento en el diagrama del caso práctico.

**Nota:** por favor, complete las actividades en su cuaderno de apuntes o en un documento Word.





## Resultado de aprendizaje 3:

Utiliza UML como lenguaje de construcción de modelos para modelar componentes de un sistema de información.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 5

##### Unidad 3. Diagramas UML básicos

En esta semana, revisaremos cómo realizar una especificación detallada de casos de uso. Eso es importante porque permite crear una narrativa en función de la secuencia de interacción entre el actor y el sistema. A continuación, haciendo uso de la norma IEEE 830-1998, comprenderemos cómo crear especificaciones textuales y aplicaremos estos conceptos a través del caso práctico que venimos desarrollando.

#### 3.6. Vista de escenarios o casos de uso

##### 3.6.3. Especificación textual de casos de uso

La especificación textual de casos de uso es una actividad importante para identificar y describir de manera detallada cómo interactúan los actores con el sistema para lograr objetivos específicos. Esta técnica sirve como base para el diseño, desarrollo y pruebas del sistema y hace uso de las prácticas y recomendaciones de la norma IEEE 830-1998.

La norma IEEE 830-1998 proporciona directrices detalladas para la Especificación de Requisitos de Software, incluyendo la especificación de casos de uso. Algunos puntos clave de esta norma que pueden ser útiles para la especificación textual de casos de uso incluyen:

- **Objetivos del usuario:** asegurarse de que cada caso de uso esté orientado a cumplir con los objetivos del usuario final.
- **Requisitos funcionales:** identificar y documentar los requisitos funcionales específicos que deben ser cumplidos por el sistema.
- **Requisitos no funcionales:** incluir requisitos de rendimiento, seguridad, usabilidad, entre otros, que puedan afectar la implementación del caso de uso.
- **Formato y estructura:** facilita la comprensión y revisión de los escenarios del sistema por parte de los *stakeholders*.

Un caso de uso debe ser especificado textualmente, con el objetivo de eliminar la ambigüedad de cada caso de uso, y aportar con una narrativa que sea fácil de comprender para garantizar la comunicación entre los *stakeholders* del proyecto. También permite detallar precondiciones, flujos principales, flujos alternativos, y postcondiciones, facilitando la identificación temprana de posibles problemas y requerimientos que no hayan sido inicialmente considerados. Esto asegura que el sistema a implementar cumpla efectivamente con las expectativas del cliente y los usuarios finales.

En la tabla 3, se describe cada uno de los ítems que según la norma se deben considerar en esta especificación de casos de uso:

**Tabla 3**  
*Formato de especificación de casos de uso*

Identificador	Número o nombre único que identifica al caso de uso	
<b>Nombre</b>	Nombre descriptivo que refleja la funcionalidad del caso de uso.	
<b>Descripción</b>	Descripción breve del propósito y la funcionalidad del caso de uso.	
<b>Actores Involucrados</b>	Lista de actores que participan en la ejecución del caso de uso. Actor 1 Actor 2	
<b>Precondición</b>	Precondición del caso de uso	
<b>Flujo Principal</b>	Paso	Acción
	1	
	2	
	...	
<b>Flujo Alternativo</b>	Paso	Acción
	1	
	2	
<b>Post condiciones</b>	Condiciones que deben ser verdaderas al finalizar la ejecución del caso de uso.	
<b>Requisitos Especiales</b>	Requisitos adicionales, como restricciones de rendimiento o seguridad, que afectan al caso de uso.	
<b>Extensiones</b>	Puntos de extensión donde se pueden agregar funcionalidades adicionales a través de otros casos de uso (usos extendidos).	
<b>Frecuencia de Uso</b>	Número estimado de veces que se espera que se ejecute el caso de uso en un período de tiempo específico.	
<b>Formato de Entrada y Salida</b>	Descripción de los formatos esperados de entrada y salida del sistema para este caso de uso, si corresponde.	
<b>Referencias Cruzadas</b>	Referencias a otros documentos relacionados que puedan proveer información relevante para el caso de uso.	
<b>Autor</b>	Nombre de la persona o equipo responsable de la especificación del caso de uso.	



Identificador	Número o nombre único que identifica al caso de uso
Fecha de Creación	Fecha en la que se creó la especificación del caso de uso.
Estado	Estado actual del caso de uso (por ejemplo, propuesto, en desarrollo, aprobado, implementado).

Nota. Esta tabla contiene cada uno de los ítems a considerar en una especificación textual de un diagrama de casos de uso. Tomado de Guía Didáctica de Modelado de Sistemas, por Correa, R., 2024, Ediloja Cía. Ltda.

### 3.6.4. Desarrollando la especificación textual del diagrama de caso de uso

Para ilustrar cómo se estructura un diagrama de casos de uso, vamos a continuar desarrollando el ejemplo práctico aplicado a un sistema de API de procesamiento de pagos de tarjeta de crédito.

La especificación textual para el caso de uso de la figura 7, sería el que se presenta en la siguiente tabla:

**Tabla 4***Ejemplo de especificación textual de casos de uso*

<b>Identificador</b>	<b>CU-ProcesarPago-001</b>								
<b>Nombre</b>	Procesar pago, autorizar pago, verificar transacción.								
<b>Descripción</b>	Sistema de API para el procesamiento de pagos de tarjeta de crédito que permite a los usuarios realizar transacciones seguras y eficientes.								
<b>Actores Involucrados</b>	<ul style="list-style-type: none"> <li>• Aplicación móvil de comercio.</li> <li>• Sistema bancario.</li> </ul>								
<b>Precondición</b>	<ul style="list-style-type: none"> <li>• El sistema debe estar disponible y operativo.</li> <li>• Los usuarios deben tener fondos suficientes para realizar los pagos.</li> </ul>								
<b>Flujo Principal</b>	<table border="1"> <thead> <tr> <th>Paso</th> <th>Acción</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>La Aplicación Móvil de Comercio envía una solicitud de pago al Sistema Bancario.</td> </tr> <tr> <td>2</td> <td>El Sistema Bancario verifica y autoriza la transacción.</td> </tr> <tr> <td>3</td> <td>El Administrador del Sistema verifica el estado de la transacción.</td> </tr> </tbody> </table>	Paso	Acción	1	La Aplicación Móvil de Comercio envía una solicitud de pago al Sistema Bancario.	2	El Sistema Bancario verifica y autoriza la transacción.	3	El Administrador del Sistema verifica el estado de la transacción.
Paso	Acción								
1	La Aplicación Móvil de Comercio envía una solicitud de pago al Sistema Bancario.								
2	El Sistema Bancario verifica y autoriza la transacción.								
3	El Administrador del Sistema verifica el estado de la transacción.								
<table border="1"> <thead> <tr> <th>Paso</th> <th>Acción</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Si no hay fondos suficientes, el pago es rechazado.</td> </tr> <tr> <td>2</td> <td>Si la transacción no es válida, se debe mostrar un mensaje informativo al usuario.</td> </tr> <tr> <td>3</td> <td>Si la transacción no se encuentra, se debe mostrar un mensaje de error en pantalla.</td> </tr> </tbody> </table>	Paso	Acción	1	Si no hay fondos suficientes, el pago es rechazado.	2	Si la transacción no es válida, se debe mostrar un mensaje informativo al usuario.	3	Si la transacción no se encuentra, se debe mostrar un mensaje de error en pantalla.	
Paso	Acción								
1	Si no hay fondos suficientes, el pago es rechazado.								
2	Si la transacción no es válida, se debe mostrar un mensaje informativo al usuario.								
3	Si la transacción no se encuentra, se debe mostrar un mensaje de error en pantalla.								
<b>Post condiciones</b>	<ul style="list-style-type: none"> <li>• El pago se procesa correctamente y se registra en el sistema.</li> <li>• Se notifica el estado del pago a la Aplicación Móvil de Comercio.</li> </ul>								

Identificador	CU-ProcesarPago-001
<b>Requisitos Especiales</b>	<ul style="list-style-type: none"> <li>El sistema debe mantener altos estándares de seguridad para proteger la información financiera de los usuarios.</li> <li>Debe ser capaz de manejar múltiples transacciones simultáneamente con alta disponibilidad.</li> </ul>
<b>Extensiones</b>	<ul style="list-style-type: none"> <li>UC-002 (Autorizar Pago) se extiende opcionalmente si se requiere una verificación adicional de seguridad.</li> <li>UC-003 (Verificar Transacción) se extiende para incluir más detalles sobre el estado de la transacción.</li> </ul>
<b>Frecuencia de Uso</b>	Alta
<b>Formato de Entrada y Salida</b>	<ul style="list-style-type: none"> <li>Datos de transacción desde la Aplicación Móvil de Comercio hacia el Sistema Bancario.</li> <li>Respuesta de autorización o error desde el Sistema Bancario hacia la Aplicación Móvil de Comercio y el Administrador del Sistema.</li> </ul>
<b>Referencias Cruzadas</b>	Diagrama de Arquitectura del Sistema, Documentación de Seguridad, Especificaciones Técnicas de Integración.
<b>Autor</b>	Equipo de Desarrollo
<b>Fecha de Creación</b>	DD-MM-YYYY
<b>Estado</b>	En Desarrollo

Nota. Esta tabla representa la solución a nivel de especificación textual de un diagrama de casos de uso del caso práctica utilizado en la guía didáctica.  
 Tomado de Guía Didáctica de Modelado de Sistemas, por Correa, R., 2024, Ediloja Cía. Ltda.





## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise la sección de vista de casos de uso en el libro “El lenguaje unificado de modelado: manual de referencia, segunda edición”. Reforzar los conceptos aprendidos en clase mediante la lectura de esta sección. Identifique y anote las definiciones clave, ejemplos y mejores prácticas presentadas en el libro para mejorar su comprensión de los diagramas de casos de uso.
2. Suponga que la API de procesamiento de pagos de tarjeta de crédito ha sido actualizado para incluir una nueva funcionalidad: la capacidad de “Reembolsar Pago”. Analice de qué forma debería representar la especificación textual de la nueva capacidad en el diagrama del caso práctico.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.



## **Resultado de aprendizaje 3:**

Utiliza UML como lenguaje de construcción de modelos para modelar componentes de un sistema de información.

### **Contenidos, recursos y actividades de aprendizaje recomendadas**



#### **Semana 6**

##### **Unidad 3. Diagramas UML básicos**

En esta semana, estudiaremos La vista dinámica de los sistemas, nos enfocaremos en cómo se comportan y evolucionan en el tiempo. Revisaremos los diagramas que permiten representar esta vista y también aprenderá la forma de identificar la lógica y secuencia de las acciones e interacciones que suceden dentro del sistema.

#### **3.7. Vista dinámica o de procesos**

Esta vista permite capturar el comportamiento dinámico de un sistema en tiempo de ejecución. Además, se utiliza para representar los procesos de negocio, algoritmos y flujos de trabajo en un alto nivel, facilitando una forma comprensible de diagramar las acciones y mecanismos que se ejecutan en el sistema.

Para la representación de esta vista deberá utilizar los diagramas de actividades y de secuencias. El diagrama de actividades sirve para representar el flujo de control de las actividades del sistema, representando las decisiones, bifurcaciones y actividades secuenciales. Mientras que, el diagrama de secuencias se utiliza para representar cómo los objetos interactúan entre sí en un periodo de tiempo determinado, mostrando la secuencia de mensajes intercambiados entre ellos.



### 3.7.1 Diagrama de actividades

El diagrama de actividades es una herramienta robusta en el modelado de sistemas que se utiliza para representar el flujo de actividades dentro de un proceso o procedimiento. En este tipo de diagrama, se muestran las actividades como nodos en un gráfico dirigido, y las transiciones entre estas actividades se representan mediante flechas que indican el flujo de control. Básicamente, se podría resumir al diagrama de actividades como una representación gráfica del flujo de actividades dentro de un proceso o procedimiento.

Este tipo de diagrama permite crear una representación visual clara y comprensible del flujo de actividades dentro de un proceso o procedimiento, identificar decisiones y condiciones en los procesos. Además, permite definir los caminos que la lógica establece para alcanzar los objetivos.

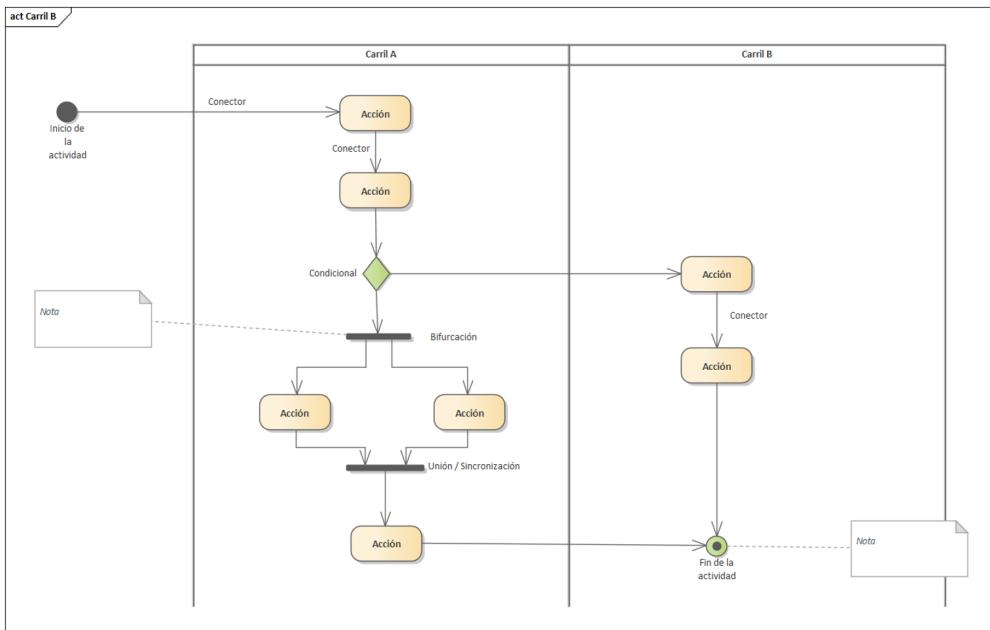
Un diagrama de actividades se utiliza principalmente para:

- **Modelado de procesos de negocio:** permite modelar y analizar los procesos de negocio, identificando las actividades involucradas y las relaciones entre ellas.
- **Diseño de algoritmos:** permite diseñar algoritmos y procedimientos, representando las diferentes etapas y condiciones que se toman durante el tiempo de ejecución.
- **Visualización de procesos de software:** permite visualizar y comprender los procesos internos de un sistema, desde tareas simples hasta flujos de trabajo de alta complejidad.

En un diagrama de actividades, se utilizan varios elementos para representar las actividades, las decisiones, el flujo de control y otros aspectos del proceso o procedimiento que se está modelando. A continuación, en la figura 8, se describe los elementos principales que se encuentran en un diagrama de actividades:

**Figura 8**

Componentes UML de un Diagrama de Componentes.



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

- El **inicio de la actividad** se representa con un círculo pequeño que marca el punto de inicio del flujo.
- Las **acciones** son rectángulos con bordes redondeados que describen tareas o pasos específicos en el proceso.
- Los **carriles** son secciones verticales u horizontales que dividen el diagrama para indicar diferentes actores o roles responsables.
- El **flujo de decisión** se representa con un rombo, donde se evalúa una condición y se dirige el flujo hacia diferentes caminos según los resultados.
- Las **uniones** (sincronización) son barras horizontales o verticales que combinan múltiples flujos en uno solo.
- Las **bifurcaciones** son similares a las uniones, pero funcionan al revés, dividiendo un flujo en varios pasos concurrentes.

- El **fin de la actividad** se indica con un círculo sólido rodeado por un anillo, señalando el fin del proceso.
- Las **notas** son rectángulos con una esquina doblada, utilizados para agregar comentarios o aclaraciones adicionales.
- Los **conectores** son flechas que muestran la dirección del flujo de control de una acción o decisión.

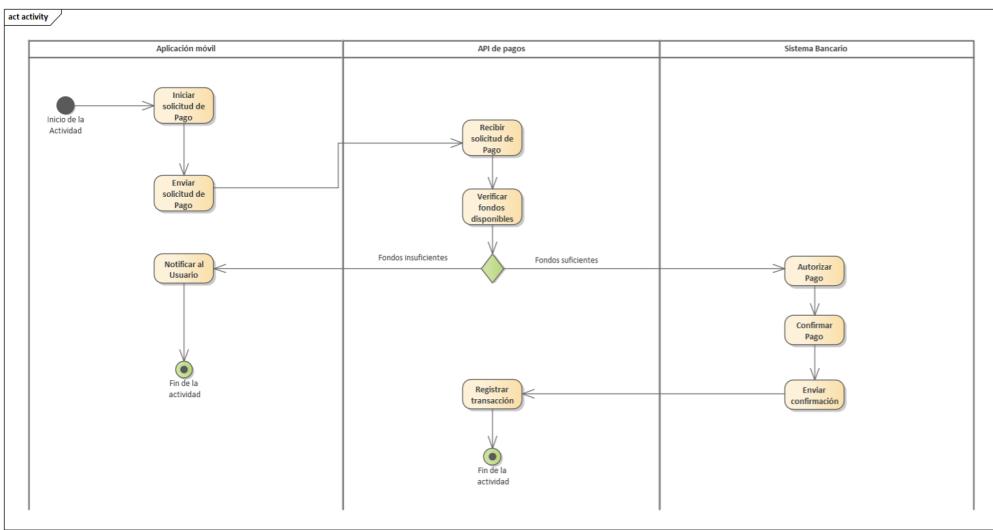


### 3.7.2 Modelando el diagrama de actividades

Para ilustrar cómo se modela un proceso utilizando un diagrama de actividades, continuaremos con el caso práctico de la guía. Este diagrama detallará el flujo de trabajo desde el inicio de una transacción de pago hasta su finalización, incluyendo acciones como la verificación de fondos, la autorización del pago, y el manejo de reembolsos. Este ejercicio representado en la figura 9, nos permitirá visualizar cómo se gestionan las decisiones y las actividades concurrentes dentro del sistema.

**Figura 9**

Ejemplo del diagrama de actividades



Nota. Esta figura representa la solución a nivel de diagrama de componentes del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

Este diagrama de actividades se organiza en tres carriles para representar las interacciones entre la Aplicación Móvil de Comercio, la API, y el Sistema Bancario. El Inicio de la Actividad representa el inicio del proceso, seguido por la acción Inicio de Pago en la Aplicación Móvil de Comercio. La aplicación envía una Solicitud de Pago a la API, que recibe la solicitud y procede a Verificar Fondos Disponibles en el Sistema Bancario.

En el flujo de decisión, si los Fondos son Insuficientes, la API notifica a la Aplicación Móvil, que a su vez Notifica al Usuario y finaliza la actividad. Si los Fondos son Suficientes, la API Autoriza el Pago y el Sistema Bancario Confirma el Pago. Luego, el Sistema Bancario Envía una Confirmación a la API, que registra la transacción.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el diagrama de actividades y modifíquelo para incluir la funcionalidad de **reembolso** en el proceso. Considere todos los posibles escenarios y caminos que puede tomar el proceso. Reflexione sobre ¿Cómo esta modificación afecta el flujo general del proceso?, y documente los cambios realizados.
2. Revise la sección de vista de actividades en el libro “*Software engineering with UML*” de Bhuvan Unhelkar para reforzar los conceptos aprendidos en clase. Identifique y refuerce las definiciones, ejemplos y buenas prácticas presentadas en esta sección. Aplique estos conocimientos para perfeccionar el diagrama de actividades desarrollado en el ejercicio práctico.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.





## Resultado de aprendizaje 3:

Utiliza UML como lenguaje de construcción de modelos para modelar componentes de un sistema de información.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 7

##### Unidad 3. Diagramas UML básicos

En esta semana, revisaremos el diagrama de secuencias para comprender y visualizar la interacción entre objetos en un sistema. Este diagrama le proporcionará una representación clara y detallada de los diferentes elementos del sistema que se comunican entre sí en tiempo de ejecución. Además, profundizaremos estos conceptos mediante el uso de ejemplos prácticos y actividades, con esto, desarrollará las habilidades necesarias para crear y analizar diagramas de secuencias que capturen de manera precisa el comportamiento dinámico de su sistema.

#### 3.7. Vista dinámica o de procesos

##### 3.7.3. Diagrama de secuencias

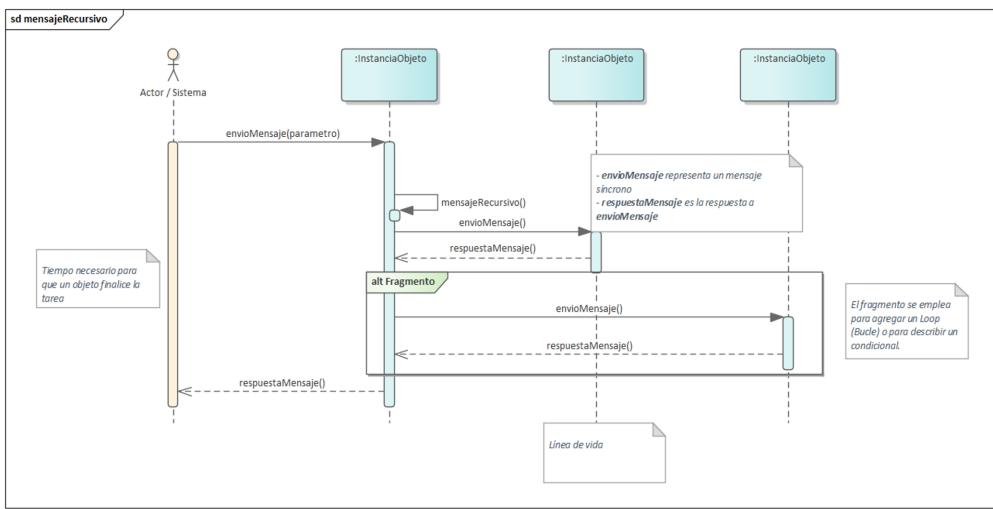
El diagrama de secuencias es un tipo de diagrama de interacción en UML (Lenguaje Unificado de Modelado) que se utiliza para representar la interacción entre objetos en un sistema a lo largo del tiempo. Se centra en la secuencia de mensajes que se intercambian entre los objetos para realizar una función específica o para cumplir un escenario de interacción determinado.

Su objetivo es generar un punto de vista que permita describir la interacción entre objetos. En este sentido, se podría resumir que este diagrama se usa principalmente para:

- **Modelado de comportamiento:** facilitan la representación del comportamiento dinámico de un sistema, mostrando cómo interactúan los objetos entre sí.
- **Diseño y desarrollo de software:** permiten comprender y especificar cómo se comportarán los sistemas en tiempo de ejecución.
- **Documentación y comunicación:** facilitan la documentación y comunicación del diseño y la funcionalidad de un sistema a todas las partes involucradas.

El diagrama de secuencia consta de varios elementos o artefactos que representan diferentes aspectos de la interacción entre objetos en un sistema. Estos elementos ayudan a proporcionar una representación detallada y precisa de cómo los objetos se comunican entre sí a lo largo del tiempo. A continuación, en la figura 10 se describen los principales elementos del diagrama de secuencia:

**Figura 10**  
**Componentes de un Diagrama de Secuencia**



Nota. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

En este tipo de diagramas, los componentes básicos incluyen al actor y al sistema.

- El **actor** puede ser un usuario, un dispositivo o un sistema externo, mientras que el sistema es la entidad principal que se está modelando.
- La **instancia** representa el uso de un objeto de una clase que participa en la interacción.
- Los **mensajes** se muestran con flechas dirigidas desde el quién inicia la acción hasta el receptor. Un mensaje síncrono espera siempre una respuesta inmediata.
- La **línea de vida** es una línea vertical que nace desde cada objeto y permite representar el tiempo de ejecución.
- Los **fragmentos** se utilizan para agrupar partes del diagrama y definir comportamientos como alternativas o bucles.
- La **respuesta del mensaje** síncrono se representa con una línea discontinua que retorna la respuesta generada de la acción.

En un contexto avanzado, los diagramas de secuencia también incluyen **mensajes asíncronos** y otros elementos que permiten representar interacciones más complejas. Los **mensajes asíncronos** son aquellos que no requieren una respuesta inmediata y se representan con una línea de flecha con una punta abierta. Estos mensajes son esenciales para modelar sistemas donde las operaciones pueden continuar independientemente del resultado inmediato. Otros componentes avanzados pueden incluir **fragmentos combinados** como condicionales, bucles y regiones críticas, que permiten especificar comportamientos complejos y concurrentes. Estos componentes son cruciales para representar la lógica detallada y las condiciones bajo las cuales los objetos interactúan.

Los diagramas de secuencia son una buena forma de representar también la comunicación entre microservicios o componentes de software en implementaciones modernas y de alta transaccionalidad. En un contexto real, los diagramas de secuencia pueden representar cómo los microservicios se comunican, enviando mensajes y esperando respuestas síncronas y asíncronas, dependiendo del caso de uso implementado.



Cuando no se tienen identificadas previamente las clases, una buena práctica es empezar por utilizar las **tarjetas CRC (Clase, Responsabilidad, Colaborador)**, como una forma de representar las clases candidatas.

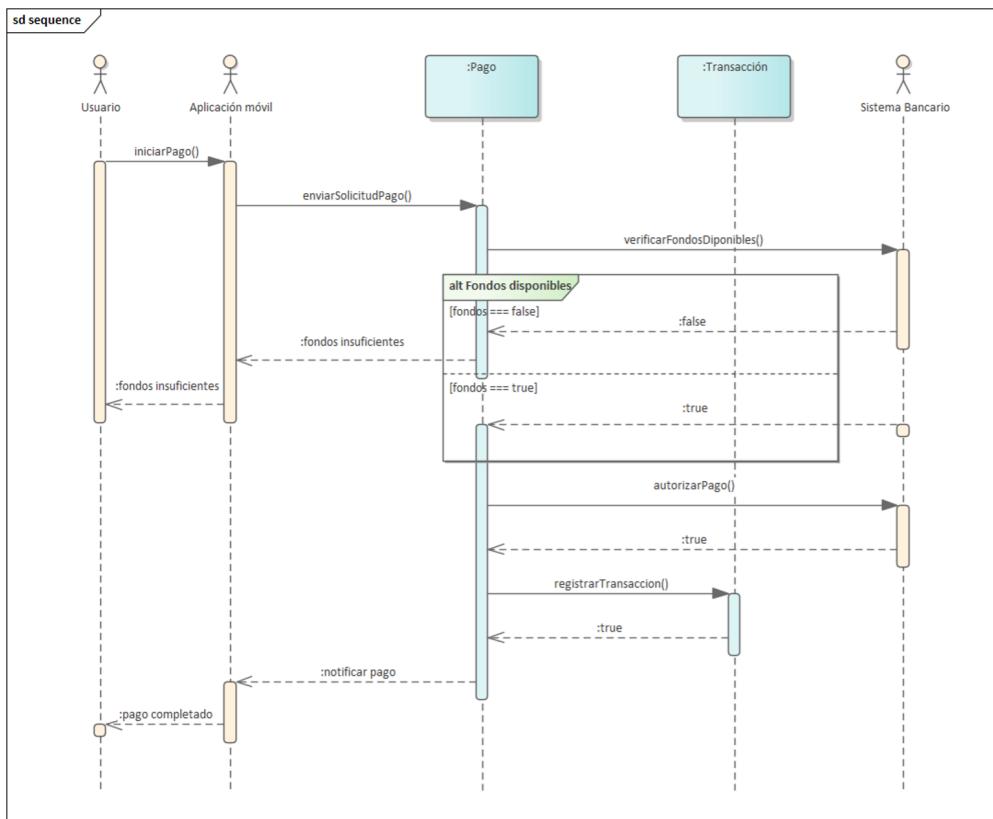
### 3.7.4. Modelado un diagrama de secuencias

Para continuar con nuestro caso práctico del sistema de API de procesamiento de pagos de tarjeta de crédito, se desarrolla un diagrama de secuencia que representa el flujo de interacción entre la aplicación móvil de comercio, la API y el sistema bancario. La figura 11 muestra cómo los actores y sistemas intercambian mensajes durante el proceso de pago, desde la solicitud inicial hasta la confirmación y registro de la transacción.



**Figura 11**

Ejemplo diagrama de secuencias



Nota.

Esta figura representa la solución a nivel de diagrama de secuencias del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

El diagrama de secuencia comienza con el actor (usuario) que inicia el pago a través de la Aplicación Móvil. La aplicación envía una Solicitud de Pago a la API, que a su vez es procesada en la instancia de: Pago, verifica los fondos disponibles enviando un mensaje al Sistema Bancario. El Sistema Bancario responde con la disponibilidad de fondos de manera síncrona. Si los fondos

son suficientes, la API procede a Autorizar el Pago con el Sistema Bancario, que luego confirma la Transacción también de manera síncrona. La API envía una Confirmación de Pago a la Aplicación Móvil, que notifica al usuario.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el diagrama de secuencia proporcionado en el caso práctico. Agregue una condición adicional para manejar la situación en la que el usuario cancela la transacción después de que se ha iniciado el proceso de pago. Incluya los mensajes y las instancias de clases necesarias para manejar esta nueva condición. Esta actividad le ayudará a comprender cómo los diagramas de secuencia pueden adaptarse a situaciones imprevistas y cómo manejar escenarios alternativos.
2. Revise el capítulo sobre diagramas de secuencia en el libro “El lenguaje unificado de modelado: manual de referencia, segunda edición”. Enfóquese en la sección que describe las diferentes interacciones y componentes utilizados en los diagramas de secuencia y refuerce su comprensión sobre el tema abordado en esta semana. Este ejercicio te permitirá profundizar en el uso de diagramas de secuencia para representar cómo los objetos interactúan a lo largo del tiempo.

**Nota:** por favor, complete las actividades en su cuaderno de apuntes o en un documento Word.

3. Le invito a realizar la siguiente autoevaluación para comprobar su comprensión de los conceptos estudiados en la unidad 3. En esta autoevaluación, prepárese para aplicar y evaluar su nivel de comprensión de los temas ya repasados.





## Autoevaluación 3

Elija la opción correcta a las siguientes interrogantes:

**1. ¿Qué es UML y por qué es importante en el desarrollo de software?**

- a. UML es un lenguaje de programación funcional.
- b. UML es un estándar de la industria para la visualización, especificación, construcción y documentación de sistemas software intensivos en procesos.
- c. UML es una metodología para gestionar proyectos de software.
- d. UML es un *framework* de desarrollo de software para automatizar la generación de clases.

**2. ¿Cuál es el propósito principal de la vista de contexto en UML?**

- a. Mostrar la estructura interna del sistema.
- b. Representar el paso a paso de los procesos internos del sistema.
- c. Visualizar la distribución física de los componentes del sistema en el entorno de implementación.
- d. Proporcionar una representación visual de las interacciones entre el sistema y las entidades externas, definiendo el alcance y los límites del sistema.

**3. ¿Cuál es el propósito principal de la vista de casos de uso en UML?**

- a. Identificar y documentar los requisitos funcionales del sistema desde el punto de vista del usuario, mostrando cómo interactúan los actores externos con el sistema para lograr objetivos específicos.
- b. Identificar la estructura estática del sistema y sus componentes.
- c. Visualizar la distribución física de los componentes del sistema en el entorno de implementación.
- d. Representar el comportamiento dinámico del sistema mediante la secuencia de actividades y eventos.



**4. ¿Qué elementos son representados por el diagrama de casos de uso en UML?**

- a. Actores como entidades externas, casos de uso como acciones realizadas por estos actores dentro del sistema, y relaciones que muestran las interacciones entre actores y casos de uso.
- b. Componentes internos del sistema y sus interacciones.
- c. Interacciones dinámicas entre objetos durante la ejecución del sistema.
- d. La estructura física y despliegue de los componentes del sistema.

**5. ¿Cuál es uno de los objetivos de la especificación textual de casos de uso según la norma IEEE 830-1998?**

- a. Mejorar la comunicación entre los miembros del equipo de desarrollo.
- b. Proporcionar una descripción de las interacciones entre actores y casos de uso.
- c. Establecer las relaciones de generalización y especialización entre casos de uso en el diagrama de casos de uso.
- d. Eliminar la ambigüedad al definir claramente los objetivos, acciones y resultados esperados de cada caso de uso.

**6. ¿Qué elementos son incluidos típicamente en una especificación textual detallada de casos de uso?**

- a. Objetivos del usuario, diagramas de secuencia y flujos de control del sistema.
- b. Precondiciones, flujos principales, flujos alternativos y postcondiciones.
- c. Requisitos no funcionales y diagramas de estructura interna del sistema.
- d. Actores externos y sus interacciones con el sistema en casos de uso.

**7. ¿Cuál es el objetivo del diagrama de actividades en UML?**

- a. Mostrar cómo los objetos interactúan entre sí en tiempo de ejecución.



- b. Diseñar procesos de negocio, identificando actividades y relaciones entre ellas.
  - c. Visualizar la estructura estática de un sistema de software y sus componentes.
  - d. Capturar el comportamiento dinámico de un sistema a través de diagramas de secuencia y colaboración.
8. ¿Qué elemento de un diagrama de actividades UML se utiliza para evaluar condiciones y dirigir el flujo de control hacia diferentes caminos?
- a. Uniones.
  - b. acciones.
  - c. Inicio de la actividad.
  - d. Notas.
9. ¿Cuál es el principal propósito del diagrama de secuencias en UML?
- a. Representar la estructura estática de un sistema de software.
  - b. Modelar el comportamiento dinámico de un sistema, mostrando cómo interactúan los objetos a lo largo del tiempo para cumplir funciones específicas.
  - c. Documentar los recursos físicos y lógicos de un sistema distribuido.
  - d. Analizar la arquitectura y los patrones de diseño de un sistema orientado a microservicios.
10. ¿Qué elemento de un diagrama de secuencias UML representa una instancia específica de una clase que participa en la interacción?
- a. Línea de vida.
  - b. Mensaje síncrono.
  - c. Fragmento combinado.
  - d. Actor.

[Ir al solucionario](#)



## Resultados de aprendizaje 1 a 3:

- Crea modelos que apoyen el proceso de análisis de sistemas.
- Identifica las partes esenciales de un sistema utilizando los diferentes tipos de modelado.
- Utiliza UML como lenguaje de construcción de modelos para modelar componentes de un sistema de información.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 8

#### Actividades finales del bimestre

Esta semana está dedicada al repaso y su preparación para la evaluación bimestral. En este primer bimestre, hemos explorado una variedad de temas fundamentales en el modelado de sistemas haciendo uso de UML como notación. Iniciamos desde la importancia del modelado y los beneficios del uso de UML, abordamos el modelo 4+1 de Kruchten, los diagramas de contexto, casos de uso, actividades y secuencias, con lo cual hemos construido una base sólida de conocimientos y habilidades. Ahora es el momento de consolidar todo lo aprendido, resolver cualquier duda y prepararse de manera efectiva para la evaluación bimestral.

¡Muchos éxitos en esta importante actividad académica!



## Estrategia de repaso y preparación



### A. Revisión de contenidos

Revise y consolide el conocimiento adquirido en las primeras siete semanas:

- **Semana 1:** repase la importancia del modelado en el desarrollo de software y los beneficios de usar UML.
- **Semana 2:** revise el modelo 4+1 de Kruchten y sus diferentes vistas.
- **Semana 3:** repase los conceptos y componentes de los diagramas de contexto.
- **Semana 4:** revise los diagramas de casos de uso, su especificación y cómo desarrollar ejemplos prácticos.
- **Semana 5:** profundice en la especificación textual de casos de uso y los detalles de los pasos involucrados.
- **Semana 6:** revise los diagramas de actividades, sus componentes y su aplicación en el caso práctico.
- **Semana 7:** repase los diagramas de secuencia y su uso en la vista dinámica del sistema.

### B. Actividades de repaso

Realice las actividades prácticas para reforzar los conceptos:

- **Elabore resúmenes de los contenidos:** realice un resumen de cada semana, destacando los puntos clave y conceptos importantes.
- **Trabaje en ejercicios prácticos:** vuelve a efectuar los ejercicios prácticos desarrollados en cada semana, intente mejorarlo incluyendo nuevas funcionalidades o condiciones. Este es un punto significativo para abordar en su evaluación bimestral.
- **Participe de foros de discusión:** comparta dudas y apóyese en sus compañeros y tutor a través de los foros de discusión.

### C. Cuestionario del primer bimestre

Realice el cuestionario del primer bimestre en Canvas para complementar sus actividades calificadas:

- **Cuestionario en Canvas:** realice el cuestionario del primer bimestre dentro del tiempo asignado para obtener su calificación.
- **Autoevaluación:** revise sus respuestas y corrija los errores. Identifique las áreas donde necesita reforzar su



### D. Recursos adicionales

Utilice recursos adicionales para aclarar dudas y ampliar el conocimiento:

- **Lecturas recomendadas:** revise los capítulos y actividades de aprendizaje recomendadas.
- **Videos y tutoriales:** busque videos y tutoriales en línea que refuerzen los conceptos que le resulten más complejos.
- **Consultas al profesor:** aproveche las tutorías para plantear preguntas a su profesor sobre cualquier duda o tema que no haya comprendido completamente.

### E. Planificación de estudio

Organice su tiempo de estudio de manera que le permita abarcar todos los contenidos del bimestre:

- **Calendario de estudio:** cree un calendario de estudio para distribuir su tiempo de manera equitativa entre las diferentes semanas y temas.
- **Tiempo de descanso:** asegúrese de incluir tiempo para descansos y actividades recreativas para evitar el agotamiento.



A medida que concluimos este primer bimestre dedicado al modelado de sistemas con UML, quiero felicitarlo por el esfuerzo y la dedicación que ha mostrado en cada semana de estudio. Le animo a aplicar todo lo aprendido en el cuestionario del primer bimestre en **Canvas** y a prepararse de manera sólida para la evaluación bimestral. Utilice estos conocimientos como base para enfrentar con éxito los retos del siguiente bimestre.



## Segundo bimestre

### Resultado de aprendizaje 4:

Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.

Para alcanzar el resultado planteado, usted deberá diseñar representaciones gráficas para un sistema ya que es fundamental para facilitar la comunicación y validación del diseño. Al utilizar UML avanzado, como diagramas de clases, interfaces y paquetes, podrá diseñar la estructura y las relaciones entre los componentes del sistema. Esto permite a los equipos discutir y explorar el diseño con claridad, identificando posibles mejoras o problemas antes de la implementación. Además, el modelado ágil con C4 ofrece una metodología flexible para representar el sistema en diferentes niveles de detalle, permitiendo facilitar las necesidades de comunicación y validación en cada etapa del desarrollo.

### Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.



### Semana 9

#### Unidad 4. Diagramas UML avanzados

Bienvenidos al segundo bimestre del curso de Modelado de sistemas. En esta etapa, continuaremos estudiando los diagramas UML avanzados, comenzando con uno de los más valiosos y comúnmente utilizados: el



diagrama de clases. Este tipo de diagrama es esencial para representar la estructura estática de un sistema, mostrando las clases, sus atributos, métodos y las relaciones entre ellas.

El diagrama de clases es una forma de representar y facilitar la transición de los requisitos a un diseño detallado y estructurado. Durante esta semana, estudiaremos los componentes básicos y avanzados de los diagramas de clases, cómo se construyen y su importancia en el desarrollo de *software*.

A través de ejemplos prácticos y actividades, aplicaremos estos conceptos al caso práctico del API de procesamiento de pagos de tarjeta de crédito, permitiéndonos ver cómo se materializan los conceptos teóricos en un contexto real. Prepare sus herramientas de modelado y su curiosidad, ya que nos adentraremos en los detalles que forman la columna vertebral de cualquier sistema de *software* bien diseñado.

#### 4.1. Vista lógica

La vista lógica es una de las cinco vistas definidas en el modelo 4+1 de Kruchten, el cual es un marco utilizado para describir la arquitectura de sistemas *software*. Esta vista se centra en el modelado de la funcionalidad del sistema desde la perspectiva de los desarrolladores, mostrando cómo se organizan los elementos del sistema para cumplir con los requisitos funcionales. A través de la vista lógica, los arquitectos y desarrolladores pueden obtener una visión detallada de la estructura interna del sistema, lo cual es crucial para asegurar que los Requisitos del Usuario se traducen correctamente en componentes funcionales y coherentes.

La vista lógica es fundamental porque:

- **Claridad en la estructura del sistema:** proporciona una representación clara y detallada de la estructura del sistema. Esto facilita la comprensión a los desarrolladores, permitiendo un mejor entendimiento de la organización del sistema y detectar posibles inconsistencias o áreas de mejora.



- **Facilita el diseño y desarrollo:** ayuda a los desarrolladores a planificar y coordinar la implementación de las clases y las relaciones.
- **Mejora la mantenibilidad:** una representación lógica bien estructurada facilita el mantenimiento y la evolución del sistema. Esto es especialmente importante en sistemas grandes y complejos, donde la mantenibilidad es clave para garantizar la adaptabilidad a futuros requisitos.
- **Soporta la reusabilidad:** una correcta estructura promueve la reusabilidad de clases y componentes en diferentes partes del sistema o en proyectos futuros.

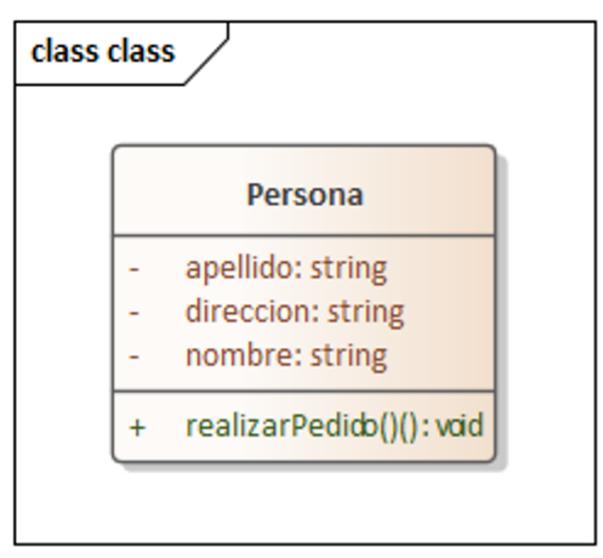


#### 4.1.1. Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura del sistema mostrando sus clases, atributos, operaciones y las relaciones entre los objetos. Los componentes básicos de un diagrama de clases incluyen:

**Clases:** se representa como un rectángulo como se describe en la figura 12, dividido en tres partes: el nombre de la clase, los atributos y los métodos.

**Figura 12**  
*Representación de una Clase en UML.*



Nota. Esta figura representa la composición de una Clase UML. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

**Atributos:** los atributos son las propiedades de una clase y se enumeran en la segunda parte de la clase. Definen las características del objeto que la clase representa. Por ejemplo: nombre: *String*, apellido: *String*, dirección: *String*.

**Métodos:** los métodos son las operaciones o funciones que pueden realizar los objetos de la clase. Se enumeran en la tercera parte de la clase. Por ejemplo: realizarPedido(): *void*.

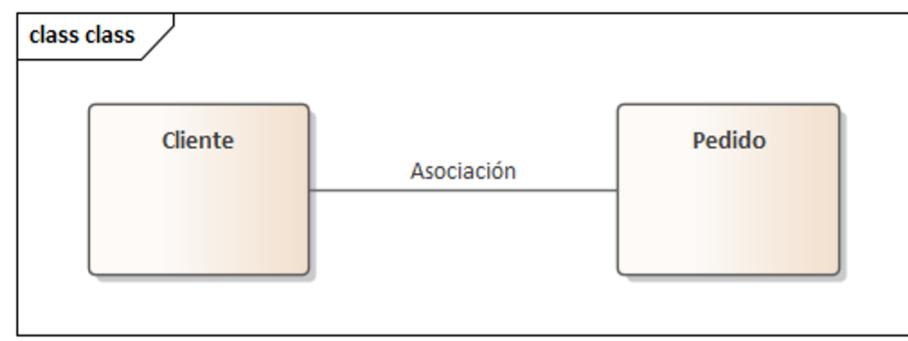
**Relaciones:** muestran cómo las clases interactúan y se relacionan entre sí.

Los tipos principales de relaciones son:

- **Asociación:** se utiliza cuando un objeto de una clase necesita interactuar con un objeto de otra clase. La siguiente figura muestra un ejemplo:

**Figura 13**

*Relación de Asociación.*



Nota. Esta figura representa una relación de asociación entre clases.

Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

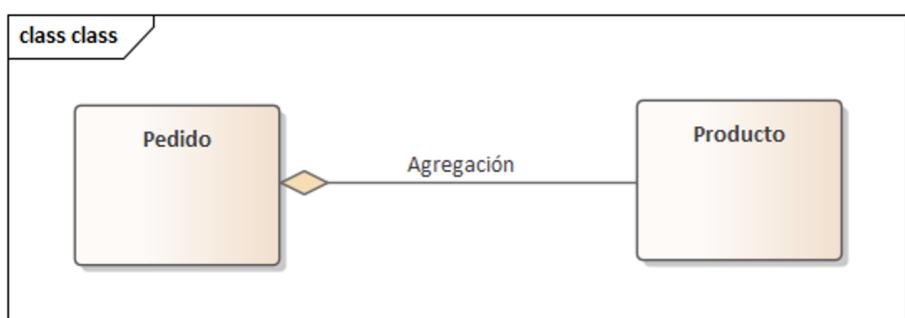
*Uso:* utilice asociación cuando una clase necesita utilizar los servicios o interactuar con otra clase de manera regular.

- **Agregación:** la agregación es un tipo de asociación que indica una relación “todo/parte” entre el objeto contenedor (todo) y los objetos contenidos (partes). La parte puede existir independientemente del todo. Observe el ejemplo:



**Figura 14**

*Relación de Agregación.*



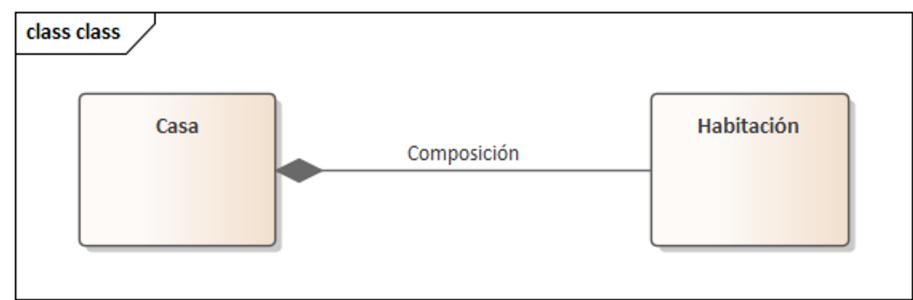
Nota. Esta figura representa una relación de agregación entre clases.  
Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

*Uso:* utilice agregación cuando una clase forma parte de otra clase, pero puede existir independientemente de la clase contenedora.

- **Composición:** la composición es una forma fuerte de agregación donde las partes no pueden existir sin el todo. Si el contenedor es destruido, también lo son las partes.

**Figura 15**

*Relación de Composición.*



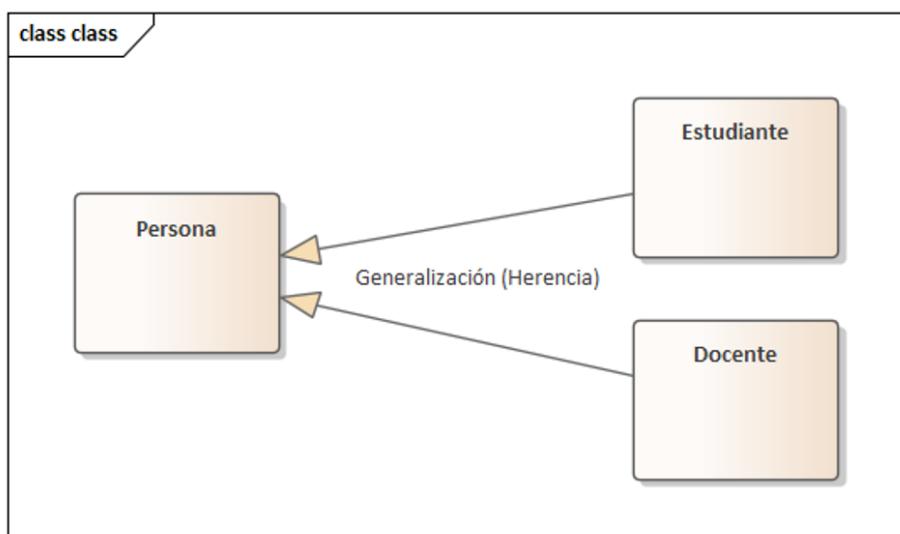
Nota. Esta figura representa una relación de composición entre clases.  
Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

*Uso:* utilice composición cuando una clase forma una parte integral de otra clase y no puede existir sin la clase contenedora.

- **Generalización (herencia):** la herencia muestra una relación jerárquica entre una clase general (superclase) y una clase más específica (subclase). La subclase hereda los atributos y métodos de la superclase.

**Figura 16**

*Relación de Generalización.*



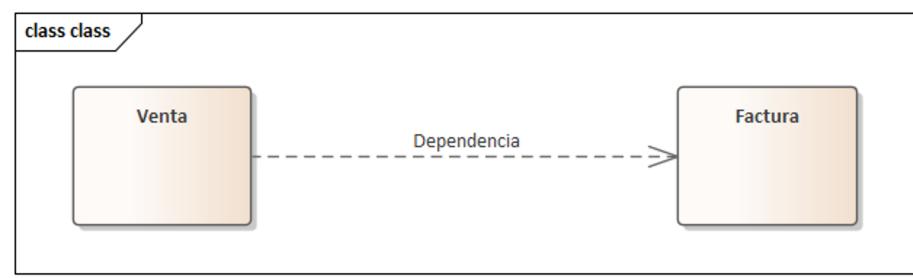
Nota. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

*Uso:* utilice herencia para modelar una jerarquía donde una clase comparte características comunes con otra clase más general.

- **Dependencia:** la dependencia indica que una clase utiliza temporalmente otra clase. Es una relación débil comparada con la asociación.

**Figura 17**

*Relación de dependencia.*



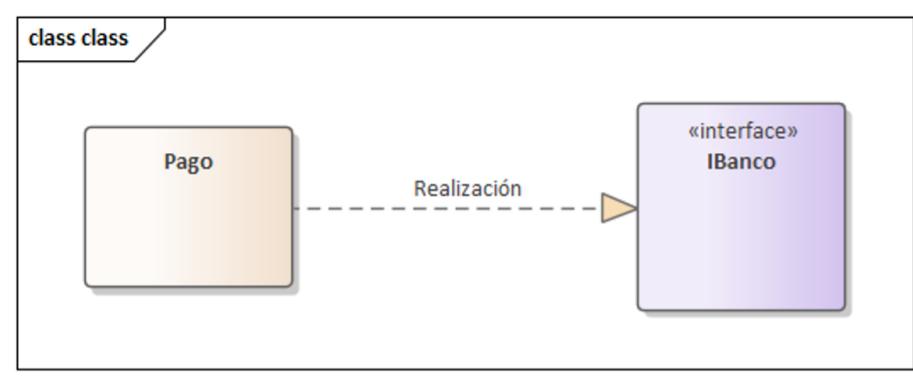
Nota. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

*Uso:* utilice dependencia cuando una clase necesita usar otra clase para realizar una tarea específica y temporal.

- **Realización:** la realización se utiliza para mostrar que una clase implementa una interfaz. Una interfaz define un contrato que una clase debe cumplir.

**Figura 18**

*Relación de Realización.*



Nota. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

Uso: utilice realización cuando una clase necesite implementar una interfaz definida por otra clase.



Las interfaces en UML son útiles para definir comportamientos comunes que pueden ser compartidos por múltiples clases, promoviendo la reutilización del código y la flexibilidad en el diseño.

**Multiplicidad:** indica el número de instancias de una clase que pueden estar asociadas con una instancia de otra clase. Es crucial para definir las relaciones entre objetos y entender cómo interactúan dentro del sistema.

Se representa utilizando números o intervalos dentro de un estereotipo (por ejemplo, 0..1, 1..\*, 1..1, \*, etc.), donde:

- **0**: indica que no hay instancias asociadas.
- **1**: indica exactamente una instancia asociada.
- **(asterisco)**: indica cualquier número de instancias (cero o más).

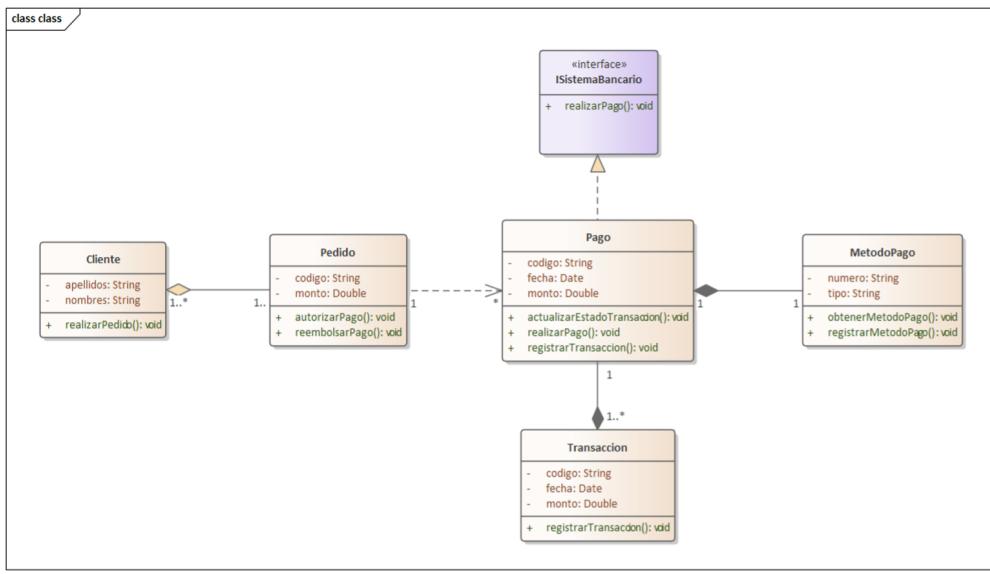
**Ejemplo:** un cliente puede tener varios pedidos, expresado como 1..\* para indicar que un cliente puede tener uno o varios pedidos.

#### 4.1.2. Modelando el diagrama de clases

Para desarrollar el diagrama de clases del sistema de API de procesamiento de pagos de tarjeta de crédito, primero identificaremos las clases principales involucradas, sus atributos y métodos, y luego definiremos las relaciones entre ellas. Este mecanismo nos permitirá crear una representación clara de la estructura del sistema como la que se describe en la figura 19.

**Figura 19**

Ejemplo de Diagrama de Clases.



Nota. Esta figura representa la solución a nivel de diagrama de clases del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

El diagrama de clases modela las entidades principales y sus interacciones en el sistema de procesamiento de pagos de tarjeta de crédito las cuales se describen a continuación:

- **Cliente** representa los usuarios que interactúan con el sistema a través de la aplicación móvil de comercio.
- **Pedido** es creado por un cliente y puede involucrar transacciones de pago.
- **Pago** gestiona las transacciones individuales, registrando detalles como el estado y el identificador.
- **MetodoPago** define los métodos disponibles para realizar pagos.
- **ISistemaBancario** proporciona la conexión entre el sistema y el banco para procesar pagos.

- **Transacción** registra los detalles de cada transacción de pago realizada, como el monto y la fecha.

Al nombrar clases, utilice sustantivos que describan claramente el propósito o la entidad que representa la clase. Esto facilita la comprensión y el mantenimiento del sistema. Por ejemplo:

- Cliente en lugar de Persona para especificar claramente su rol en el sistema.
- Pedido en lugar de TransaccionVenta para hacer explícito que se trata de un pedido realizado por un cliente.
- ProcesadorPagos en lugar de Interfaz para dejar claro que es la clase específica para el procesamiento de pagos.



Utilizar nombres claros y específicos ayuda a evitar confusiones y asegura que todos los miembros del equipo de desarrollo tengan una comprensión común del sistema.

Las relaciones entre estas clases reflejan cómo interactúan en el sistema:

- Un Cliente puede tener varios Pedidos.
- Cada Pedido puede estar asociado con múltiples Pago.
- Cada Pago debe ser realizada a través de un método de pago (MetodoPago) específico.
- La interfaz bancaria (ISistemaPago) facilita la comunicación entre el sistema y el banco.

### Especificación de multiplicidad:

- Un cliente puede efectuar varios pedidos, pero un pedido pertenece a un único cliente (1..\*).
- Un pedido puede tener múltiples pagos, pero un pago pertenece a un único pedido (1..\*).
- Cada transacción de pago se realiza utilizando un método de pago (MetodoPago) específico (1..1).



- Cada pago utiliza una interfaz bancaria para comunicarse con el sistema bancario (1..1).
- Cada transacción registra los detalles del pago (1..1).



La multiplicidad en los diagramas de clases especifica la cantidad de instancias que pueden estar asociadas entre dos clases. Es importante utilizarla para clarificar las relaciones y entender mejor cómo interactúan los objetos en el sistema.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el libro “El lenguaje unificado de modelado” de Ivar Jacobson, capítulo sobre Diagramas de clases. Explore cómo se aborda la modelización de clases y las relaciones en el contexto del UML. Busque ejemplos prácticos que refuerzen los conceptos aprendidos y reflexione sobre cómo estos conceptos pueden aplicarse al diagrama de clases que ha desarrollado hasta ahora.
2. Modifique el diagrama de clases del sistema de API para procesamiento de pagos de tarjeta de crédito desarrollado anteriormente. Agregue una nueva funcionalidad como “Gestión de Cuentas de Usuario”. Asegúrese de identificar las nuevas clases necesarias, definir las relaciones apropiadas y actualizar la multiplicidad según sea necesario.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.





## Resultado de aprendizaje 4:

Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 10

##### Unidad 4. Diagramas UML avanzados

En la semana 10, revisaremos dos conceptos fundamentales en la Programación Orientada a Objetos y en el Diseño de software: en el estudio de interfaces y paquetes UML.

Las interfaces son contratos que definen un conjunto de métodos que una clase debe implementar, promoviendo la reutilización y flexibilidad del código al establecer un comportamiento común compartido por múltiples clases.

Mientras que los paquetes son mecanismos de agrupamiento que organizan elementos relacionados del modelo en estructuras jerárquicas, ayudando a gestionar la complejidad de sistemas grandes mediante una vista modular y estructurada. Esto nos permitirá crear la vista de desarrollo, que será la que nos permitirá tener una visión general de cómo se estructurará el modelo y los componentes de nuestro sistema.

#### 4.2. Vista de desarrollo

Esta vista se enfoca en representar cómo los componentes del sistema software se organizan, distribuyen y se relacionan entre sí en el entorno de desarrollo y despliegue. En esta vista el diagrama de componentes a través de la representación de aspectos físicos, módulos, bibliotecas y otros

artefactos de software, y los **paquetes** que permiten agrupar y organizar los componentes relacionados lógicamente dentro del sistema, juegan un papel importante para definir cómo se va a organizar el sistema.

También las **interfaces** son elementos esenciales que definen los puntos de interacción entre los componentes del sistema, ya que a través de estas se especifican qué servicios se proporcionan o requieren entre los componentes.

La vista de desarrollo es fundamental por:

- **Claridad en la implementación física:** proporciona una representación de cómo se implementa físicamente el sistema, permitiendo a los desarrolladores a comprender la estructura y distribución de los componentes del software.
- **Soporte para el despliegue:** facilita la planificación y organiza el despliegue del sistema, permitiendo a los equipos de desarrollo prepararse para etapas post-desarrollo.
- **Optimización y mantenimiento:** permite identificar oportunidades para optimizar la estructura del sistema y garantizar la mantenibilidad al detallar las dependencias y comunicaciones entre los componentes.

#### 4.2.1. Interfaces

Las interfaces son puntos de conexión entre diferentes componentes del sistema, tanto internos como externos. Estas interfaces definen los servicios que un componente puede proporcionar o requerir, facilitando la comunicación y la integración entre distintos módulos del software.

Funcionan como contratos que especifican los métodos, operaciones o servicios que un componente puede ofrecer o requerir. Establecen un nivel de abstracción que permite a los desarrolladores separar la implementación interna de un componente de su interacción con otros componentes, promoviendo la modularidad y la reutilización del código.

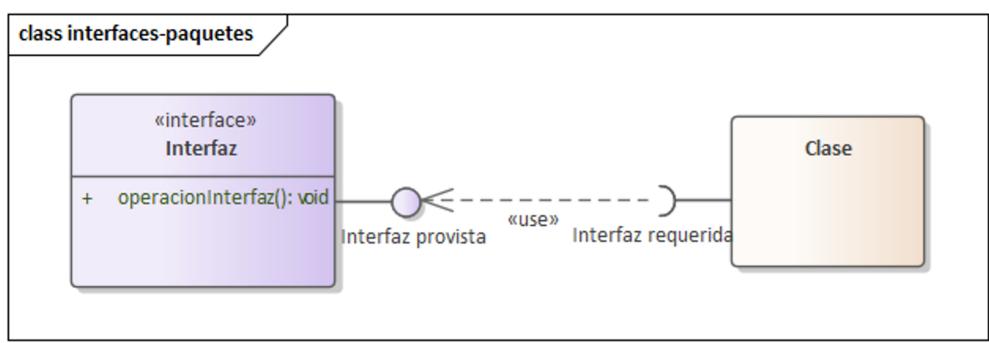


Las interfaces son fundamentales en el diseño y desarrollo de software por varias razones clave:

- **Separación de responsabilidades**: permiten separar la definición de un servicio o funcionalidad de su implementación concreta, lo que facilita la independencia entre los componentes que interactúan.
- **Interoperabilidad**: facilitan la comunicación entre diferentes partes del sistema, incluso si están desarrolladas por equipos diferentes o en tecnologías distintas.
- **Flexibilidad y mantenibilidad**: al definir claramente los puntos de interacción entre componentes, las interfaces facilitan los cambios en la implementación sin afectar a otros módulos que dependen de esos servicios.

## Componentes UML de interfaces

**Figura 20**  
*Componentes UML de una Interfaz*



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

Las interfaces se representan con un rectángulo con el nombre de la interfaz en la parte superior y los métodos o servicios que define en la parte inferior. Las interfaces pueden ser interfaces provistas o interfaces.



- **Interfaz provista:** representa una interfaz que un componente implementa y proporciona para que otros componentes la utilicen. Se identifica con un círculo.
- **Interfaz requerida:** representa una interfaz que un componente necesita para poder utilizar los servicios proporcionados por otro componente. Se identifica con un medio círculo.



#### 4.2.2. Paquetes

Los paquetes son mecanismos utilizados para organizar y estructurar los componentes de un sistema. Proporcionan un nivel adicional de abstracción sobre los componentes individuales, agrupándolos lógicamente según su funcionalidad. Sirven para agrupar elementos relacionados, como clases, interfaces y otros paquetes, dentro de una unidad que pertenece a un dominio particular. Estos elementos pueden ser organizados jerárquicamente para reflejar la estructura modular del sistema y facilitar la comprensión y mantenimiento del código.

La siguiente figura representa un ejemplo de un diagrama de paquetes.

**Figura 21**  
*Paquetes UML*



Nota. Tomado de *¿Qué es un diagrama de paquetes?* [Ilustración], por Lucidchart, 2024, [Lucidchart](#), CC BY 4.0.

Los paquetes juegan un papel crucial en el desarrollo de software por varias razones:

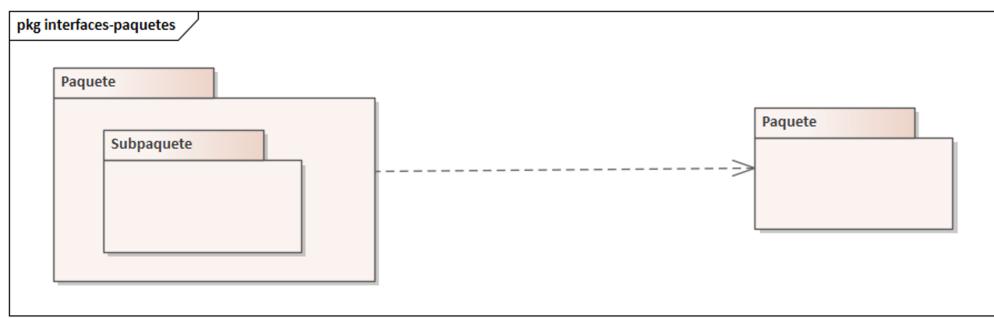
- **Organización modular:** permiten dividir el sistema en unidades más pequeñas y manejables, facilitando la asignación de responsabilidades y la colaboración entre equipos de desarrollo.
- **Reutilización:** promueven la reutilización del código al encapsular funcionalidades relacionadas que pueden ser compartidas por diferentes partes del sistema.
- **Gestión de la complejidad:** ayudan a gestionar la complejidad del software al proporcionar una estructura clara y modular que facilita la navegación y comprensión del sistema.

Un ejemplo común de uso de paquetes es en el diseño de arquitecturas basadas en capas, donde cada capa del sistema se representa típicamente como un paquete que contiene componentes relacionados (por ejemplo, capa de presentación, capa de lógica de negocio, capa de acceso a datos).

## Componentes UML de paquetes

A continuación, la figura representa los componentes UML de un diagrama de paquetes.

**Figura 22**  
*Componentes UML de paquetes*



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas* [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda

En los diagramas de paquetes, estos se representan como carpetas con el nombre del paquete en la parte superior. Pueden contener elementos como clases, interfaces, subpaquetes u otros paquetes. Además de su estructura interna, también permiten representar las relaciones entre paquetes, que son esenciales para mostrar la dependencia y la organización modular del sistema.

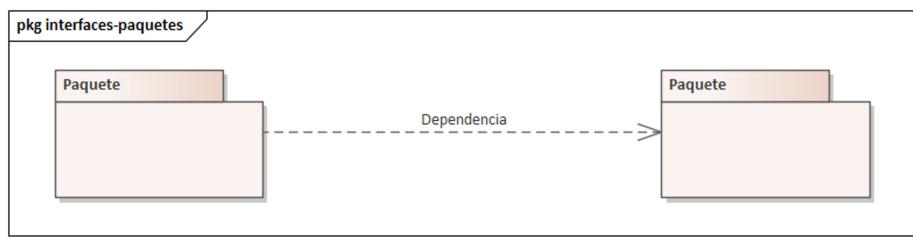
## Relaciones entre paquetes

Las relaciones entre paquetes se pueden representar utilizando diferentes tipos de conectores para expresar cómo interactúan entre sí. A continuación, se describen las relaciones principales y cómo se representan:

- **Dependency (Dependencia):** indica que un paquete depende de otro para su implementación o comportamiento, pero sin requerir la importación explícita de los elementos del paquete dependiente. Observe el ejemplo en la siguiente figura:

**Figura 23**

*Paquetes relación dependencia*



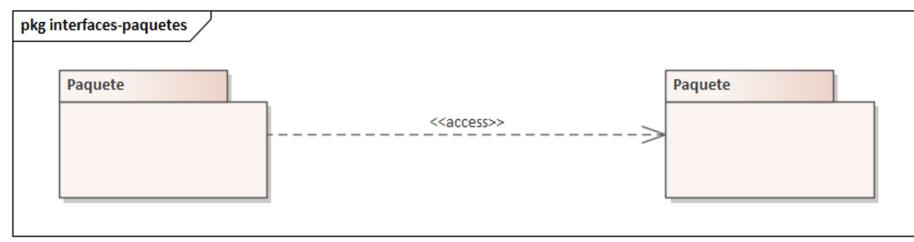
Nota. Esta figura representa la relación de dependencia entre paquetes. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

- **Access (Acceso):** indica que un paquete accede al contenido de otro paquete, pero sin dependencia directa de implementación.



**Figura 24**

Paquetes relación << access >>

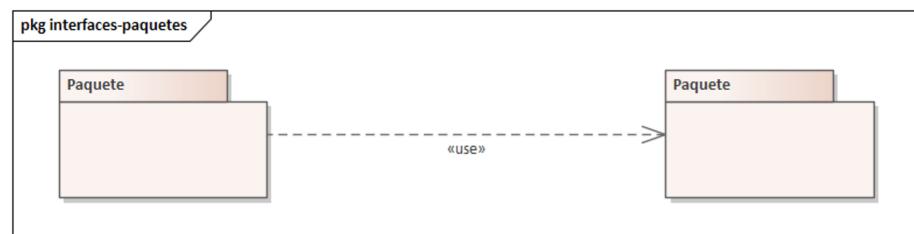


Nota. Tomado de *Guía Didáctica de Modelado de Sistemas* [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

- **Use (Uso):** representa que un paquete utiliza elementos del otro paquete en su implementación. Se muestra con una línea sólida desde el paquete que usa hacia el paquete que es usado.

**Figura 25**

Paquetes relación << use >>



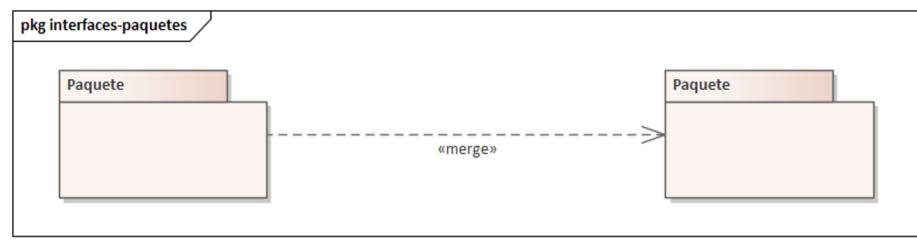
Nota. Tomado de *Guía Didáctica de Modelado de Sistemas* [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

- **Merge (Fusión):** indica que dos o más paquetes se fusionan en un solo paquete. Se representa con una línea sólida con un rombo hueco en el extremo que apunta hacia el paquete resultante de la fusión.



**Figura 26**

Paquetes relación << merge >>

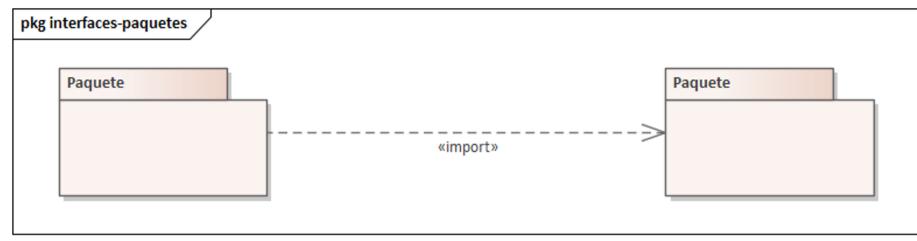


Nota. Tomado de *Guía Didáctica de Modelado de Sistemas* [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

- **Import (Importación):** indica que un paquete importa los elementos de otro paquete para su uso dentro del mismo. Se representa con una línea punteada desde el paquete que importa hacia el paquete que es importado.

**Figura 27**

Paquetes relación << import >>



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas* [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

Estas relaciones ayudan a especificar cómo se estructuran y relacionan los paquetes dentro del sistema, facilitando la comprensión de la organización modular y las dependencias entre los componentes del software.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el capítulo correspondiente a la Vista de desarrollo y el diagrama de componentes en el libro “El lenguaje unificado de modelado: manual de referencia” de Ivar Jacobson. Preste especial atención a la sección sobre Interfaces y paquetes, y cómo se representan y utilizan en los diagramas de componentes.
2. Reflexione sobre cómo los conceptos presentados se aplican en el caso práctico que hemos desarrollado hasta ahora. Realice un ejemplo.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.





## Resultado de aprendizaje 4:

Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 11

##### Unidad 4. Diagramas UML avanzados

En esta semana, estudiaremos el diagrama de componentes, un diagrama importante en el modelado de sistemas software. Este diagrama forma parte de la vista de desarrollo del modelo 4+1 de Kruchten, que permite estructurar los componentes físicos del sistema y sus interacciones. Este tipo de diagramas permite a los desarrolladores comprender cómo se organizan y comunican entre sí los componentes en el entorno de implementación. A lo largo de esta semana, exploraremos los conceptos clave del diagrama de componentes y su importancia en el proceso de desarrollo de software.

#### 4.2. Vista de desarrollo

##### 4.2.3. Diagrama de componentes

El diagrama de componentes permite representar la estructura y las relaciones entre los componentes físicos del sistema. Este tipo de diagrama se enfoca en mostrar cómo los diversos módulos, bibliotecas, archivos ejecutables y otros elementos de software se organizan y se relacionan para formar una aplicación o sistema completo. Cada componente se representa mediante un rectángulo con su nombre y puede incluir detalles adicionales como interfaces, dependencias y relaciones con otros componentes.

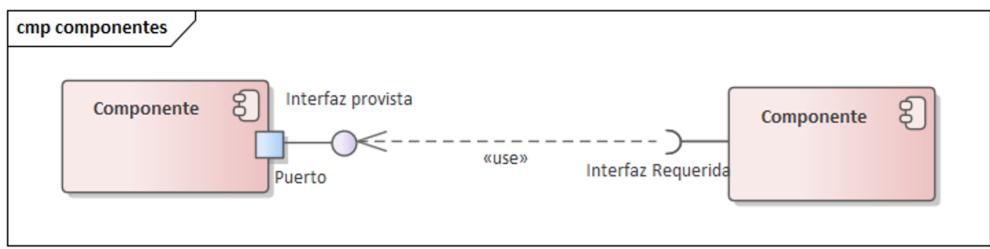
La importancia del diagrama de componentes radica en su capacidad para visualizar claramente la arquitectura física y lógica del sistema, facilitando la comprensión del diseño y la implementación. Esto ayuda a los desarrolladores a planificar la distribución de los componentes en diferentes plataformas o entornos, asegurando la escalabilidad, el rendimiento y la eficiencia del sistema *software*. Además, es útil para identificar puntos de integración y posibles cuellos de botella en la implementación.

Dentro del contexto del diagrama de componentes, varios elementos juegan roles importantes en la representación y la comunicación entre los componentes del sistema.

- Los **componentes** son representados como rectángulos con su nombre, encapsulando funcionalidades específicas y ofreciendo interfaces claras para la interacción con otros componentes.
- Los **puertos** actúan como puntos de conexión dentro de los componentes, permitiendo la comunicación entrante y saliente.
- Las **interfaces provistas** definen los servicios o funcionalidades que el componente ofrece a otros componentes.
- Las **interfaces requeridas** especifican las capacidades que el componente necesita para operar correctamente. Estas relaciones y comunicaciones entre componentes facilitan una representación clara y detallada de la arquitectura física y lógica del sistema *software*, ayudando en la planificación del despliegue, la gestión de la complejidad y la optimización del diseño.

**Figura 28**

Componentes UML del Diagrama de Componentes



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

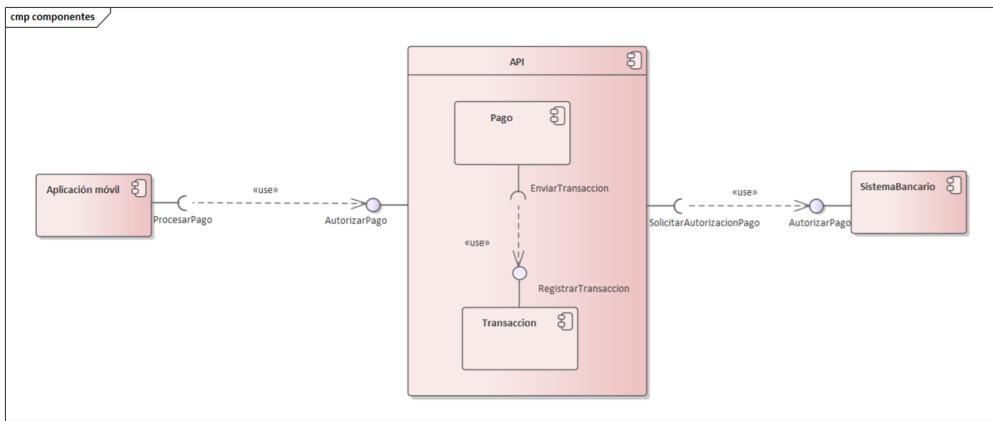
#### 4.2.4. Modelando un diagrama de componentes

En esta sección práctica, aplicaremos los conceptos aprendidos sobre el diagrama de componentes en el contexto del sistema de API para procesamiento de pagos de tarjeta de crédito. El diagrama de componentes nos permitirá visualizar cómo los diferentes elementos del sistema interactúan y se organizan físicamente para ofrecer sus funcionalidades.

El diagrama de componentes para nuestro sistema de API de pagos de tarjeta de crédito, que puede observarse en la figura que aparece más adelante, muestra los componentes clave y sus interacciones. El componente principal es la API de pagos, que interactúa con otros componentes como el sistema bancario y métodos de pago. Cada componente está representado con su nombre y los puertos que utilizan para comunicarse. Las interfaces provistas y requeridas entre los componentes especifican cómo se conectan y qué servicios ofrecen o necesitan para funcionar correctamente.

**Figura 29**

Ejemplo de Diagrama de Componentes



Nota. Esta figura representa la solución a nivel de diagrama de componentes del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

En este ejercicio la aplicación móvil interactúa directamente con el componente API a través de la interfaz provista AutorizarPago. Este diseño permite que la aplicación móvil envíe solicitudes de autorización de pagos de manera eficiente y segura.

Internamente en la API, se ha estructurado dos subcomponentes principales: Pago y transacción. El subcomponente pago utiliza la interfaz provista transacción para llevar a cabo el registro detallado de cada pago procesado, asegurando una gestión precisa y robusta de las transacciones financieras.

Finalmente, para autorizar los pagos, el componente API se comunica con el sistema bancario a través de la interfaz provista específica para este propósito.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el libro “El lenguaje unificado de modelado: manual de referencia” de Grady Booch, James Rumbaugh, e Ivar Jacobson. Enfóquese en la sección que describe los Diagramas de componentes para reforzar los conceptos aprendidos esta semana.
2. Agregue un componente adicional al diagrama de componentes del caso práctico para implementar un “Servicio de Notificaciones” que se encargue de enviar confirmaciones de pago al cliente. Asegúrese de incluir las interfaces necesarias y describir cómo interactúan con los otros componentes del sistema.

**Nota:** por favor, complete la actividad en su cuaderno de apuntes o en un documento Word.



## **Resultado de aprendizaje 4:**

Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.

### **Contenidos, recursos y actividades de aprendizaje recomendadas**



#### **Semana 12**

#### **Unidad 4. Diagramas UML avanzados**

En esta semana, nos adentraremos en la vista física de la arquitectura del sistema, explorando cómo los componentes del *software* se asignan a la infraestructura de *hardware*. Esta vista es esencial para entender cómo y dónde se ejecuta el sistema, asegurando que todos los componentes funcionan correctamente en su entorno de operación. También, aprenderemos a crear y utilizar diagramas de despliegue, una herramienta fundamental para visualizar la distribución del *software* en el *hardware* y para planificar el despliegue del sistema en entornos reales. Al finalizar esta semana, será capaz de modelar la infraestructura física de su sistema y comprender la importancia del despliegue adecuado para el rendimiento y la escalabilidad del *software*.

#### **4.3. Vista física**

La vista física es utilizada para describir la arquitectura de sistemas de *software*. Esta vista se centra en la infraestructura de *hardware* y en cómo los componentes del *software* se despliegan en nodos físicos, virtuales o en *cloud*. A diferencia de las otras vistas, que se enfocan principalmente en la



funcionalidad y en el comportamiento del sistema, esta vista se preocupa por cómo los elementos del sistema se organizan y se ejecutan en el *hardware* disponible.

La vista física es fundamental por:

- **Planificación del despliegue:** permite a los arquitectos y a los ingenieros de sistemas visualizar cómo se asignan los componentes del *software* a los servidores, a las bases de datos, y a otros recursos de *hardware* y como se configuran en su entorno de operación.
- **Optimización del rendimiento:** la vista física permite optimizar la distribución de las cargas de trabajo entre los recursos computacionales disponibles.
- **Escalabilidad y flexibilidad:** esta vista ayuda a los desarrolladores y a los administradores de sistemas a decidir cuándo y cómo agregar más recursos para manejar el aumento de la demanda o para mejorar la redundancia y la disponibilidad del sistema.
- **Seguridad y confiabilidad:** permite identificar y mitigar los riesgos asociados con la infraestructura física, como la seguridad de los datos, la protección contra fallos de *hardware*, y la redundancia para la recuperación ante desastres.
- **Gestión de la infraestructura:** la vista física facilita la gestión y el mantenimiento de la infraestructura del sistema. Proporciona una guía clara para la configuración, la monitorización, y la actualización de los componentes del *hardware*, así como para la resolución de problemas relacionados con la infraestructura.

La vista física se representa a través de diagramas de despliegue. Estos diagramas muestran la configuración física del sistema, ilustrando los nodos (servidores, dispositivos, etc.), los artefactos (componentes del *software*), y las conexiones entre ellos. Los diagramas de despliegue permiten visualizar cómo los elementos del *software* interactúan con el *hardware* y cómo se comunican entre sí a través de la red.



Los Requerimientos No Funcionales (NFR, por sus siglas en inglés) son importantes en la vista física porque determinan las propiedades y las características del sistema que no están directamente relacionadas con la funcionalidad, pero que son esenciales para su operación eficiente y efectiva.

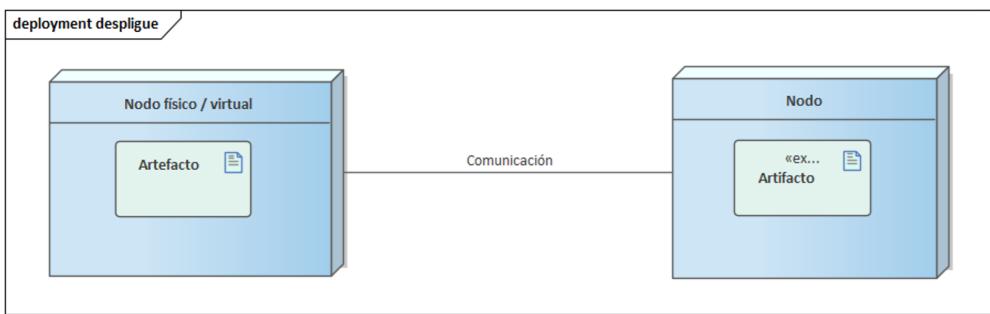
#### 4.3.1. Diagrama de despliegue

El diagrama de despliegue permite representar la disposición del *hardware* en el que se ejecutan los componentes del *software* y las conexiones entre estos componentes. Al proporcionar una representación gráfica de la infraestructura de *hardware* y *software*, el diagrama de despliegue permite visualizar cómo los artefactos de *software* se distribuyen a través de los nodos físicos, cómo interactúan y se comunican entre sí en el entorno de ejecución.

La importancia del diagrama de despliegue radica en su capacidad para proporcionar una visión clara de cómo el sistema se distribuye físicamente, facilitando la comprensión de la arquitectura de *hardware* y la implementación del *software*. Este diagrama es fundamental para la planificación y gestión del despliegue, ya que ayuda a distribuir los componentes del *software* en los diferentes nodos de *hardware*, asegurando que cada artefacto esté correctamente ubicado y configurado para su funcionamiento óptimo. Además, permite optimizar el rendimiento del sistema al identificar posibles cuellos de botella y mejorar la distribución de la carga. En términos de seguridad y confiabilidad, el diagrama de despliegue permite planificar e implementar medidas de seguridad como *firewalls* y sistemas de respaldo, garantizando la protección de los datos y la continuidad del servicio en caso de fallos. También facilita la planificación de la escalabilidad y el mantenimiento del sistema a largo plazo, permitiendo agregar o actualizar nodos y artefactos de manera eficiente.

**Figura 30**

*Componentes UML del Diagrama de Despliegue*



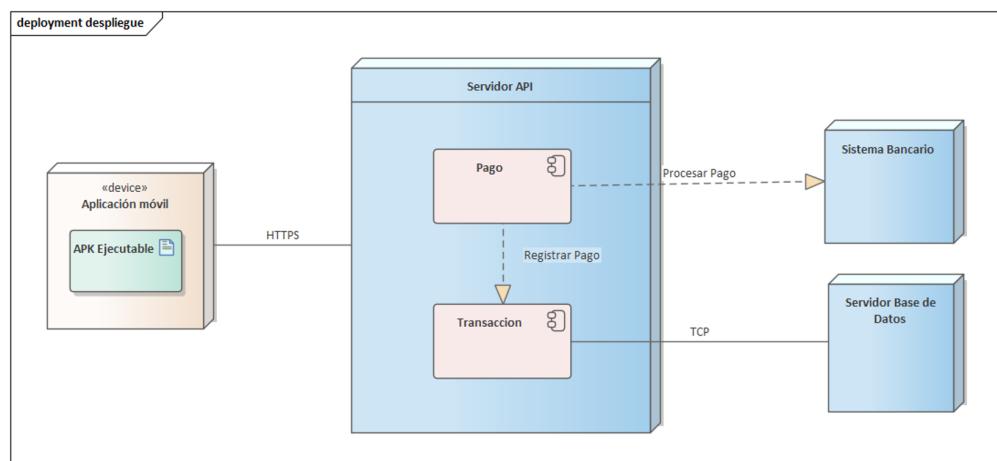
Nota. Tomado de *Guía Didáctica de Modelado de Sistemas* [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

En el diagrama de despliegue mostrado en la figura 30, se pueden identificar varios componentes clave que representan la infraestructura y la distribución del software del sistema. Los nodos representan dispositivos físicos o virtuales, como servidores, estaciones de trabajo, dispositivos móviles y routers, que forman parte de la infraestructura del sistema. Dentro de estos nodos, se encuentran los artefactos, que son unidades de implementación de software como archivos ejecutables, librerías, bases de datos y otros componentes. Las interfaces, que también se visualizan en el diagrama, representan los puntos de interacción entre los nodos y los artefactos, definiendo los servicios que los artefactos proveen o requieren. Las conexiones en el diagrama muestran la comunicación entre nodos y artefactos, incluyendo redes físicas o virtuales y enlaces de comunicación. Los componentes de software, que son módulos de software residiendo en artefactos desplegados en los nodos, abarcan aplicaciones, servicios web y otros módulos funcionales que forman parte del sistema.

#### 4.3.2. Modelando un diagrama de despliegue

En esta sección práctica, aplicaremos los conceptos aprendidos sobre diagramas de despliegue al caso de estudio que hemos venido desarrollando: un sistema de API para el procesamiento de pagos con tarjeta de crédito. Este ejercicio nos permitirá visualizar la disposición física de los componentes del sistema y cómo interactúan entre sí en el entorno de ejecución.

**Figura 31**  
*Ejemplo de Diagrama de Despliegue*



Nota. Esta figura representa la solución a nivel de diagrama de despliegue del caso práctico utilizado en la guía didáctica. Tomado de Guía Didáctica de Modelado de Sistemas [Ilustración], por Correa, R., 2024, Ediloja Cía. Ltda.

En el diagrama de despliegue mostrado en la figura 31, se puede observar la disposición física de los componentes del sistema de API para el procesamiento de pagos con tarjeta de crédito. En el nodo "Device" se representa la aplicación móvil, que contiene un ejecutable APK. Esta aplicación se conecta de manera segura (HTTPS) con el servidor API, que es el corazón del sistema. El servidor API a su vez contiene dos componentes



principales: "Pago" y "Transacción". El componente "Pago" realiza una implementación directa del componente "Transacción" para registrar los pagos de manera efectiva.

Externamente, el componente "Pago" también realiza una implementación directa con el nodo del "Sistema Bancario" para procesar los pagos, asegurando así la comunicación fluida y segura entre el sistema interno y los servicios bancarios externos.

Por otro lado, el componente "Transacción" establece una conexión TCP con el servidor de base de datos para almacenar las transacciones registradas, asegurando la integridad y disponibilidad de los datos.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el capítulo relevante del libro "El lenguaje unificado de modelado" de Jacobson para consolidar los conceptos de UML, que ilustren la aplicación de estos diagramas en el modelado de sistemas. Identifique cómo se aplican los conceptos y prácticas del diagrama de despliegue en la modelización de sistemas. Busquen ejemplos adicionales que amplíen su comprensión sobre la distribución física de componentes y artefactos en sistemas complejos.
2. Analice el diagrama de despliegue del API de pagos proporcionado y propongan la creación de un nuevo nodo dentro del contexto del API. Detallén qué función o servicio específico cumplirá este nuevo nodo y cómo se integrará con los nodos y componentes existentes.

**Nota:** por favor, complete las actividades en su cuaderno de apuntes o en un documento Word.

3. Le invito a realizar la siguiente autoevaluación donde pondrá a prueba los conocimientos adquiridos en la unidad 4 sobre Diagramas UML avanzados. A lo largo de esta unidad, hemos explorado diversos aspectos del modelado avanzado, incluyendo la vista lógica con diagramas de clases detallados, la vista de desarrollo que abarca interfaces, paquetes y componentes, así como la vista física mediante diagramas de despliegue. Estos conceptos no solo son fundamentales para comprender la estructura y el comportamiento detallado de los sistemas *software*, sino que también son cruciales para diseñar arquitecturas robustas y escalables.



#### Autoevaluación 4

Elija la opción correcta a las siguientes interrogantes:

- 1. ¿Por qué es fundamental la vista lógica en el modelado de sistemas *software* según el modelo 4+1 de Kruchten?**
  - a. Para representar la distribución física del sistema.
  - b. Para asegurar que los requisitos del usuario se traduzcan correctamente en componentes funcionales.
  - c. Para modelar los casos de uso del sistema.
  - d. Para describir las interacciones entre los actores del sistema.
- 2. ¿Cuáles son los componentes básicos de un diagrama de clases en UML?**
  - a. Objetos, relaciones, y mensajes.
  - b. Atributos, métodos, y fragmentos.
  - c. Clases, atributos, y métodos.
  - d. Paquetes, interfaces, y dependencias.
- 3. ¿Cuál es la diferencia principal entre la relación de agregación y la relación de composición en un diagrama de clases de UML?**
  - a. La agregación implica una relación “todo/parte” donde la parte puede existir independientemente, mientras que la



- composición indica una relación fuerte donde las partes no pueden existir sin el todo.
- b. La agregación es una relación temporal, mientras que la composición es una relación permanente.
  - c. La composición implica una relación “todo/parte” donde la parte puede existir independientemente, mientras que la agregación indica una relación fuerte donde las partes no pueden existir sin el todo.
  - d. La composición y la agregación son términos intercambiables en UML, sin diferencias significativas.
- 4. ¿Cuál es el propósito principal de las interfaces en el diseño de software?**
- a. Definir la estructura interna de una clase.
  - b. Separar la implementación de un servicio de su definición.
  - c. Facilitar la comunicación entre usuarios y sistemas.
  - d. Garantizar la seguridad de los datos en un sistema.
- 5. ¿Cuál es uno de los principales beneficios de utilizar paquetes en el desarrollo de software?**
- a. Incrementar la complejidad del sistema.
  - b. Reducir la reutilización del código.
  - c. Organizar componentes relacionados según su funcionalidad.
  - d. Aumentar la dependencia entre los módulos del sistema.
- 6. ¿Cuál es el papel de las interfaces provistas en un diagrama de componentes UML?**
- a. Definir las dependencias entre componentes.
  - b. Especificar los servicios y funcionalidades que un componente ofrece a otros.
  - c. Mostrar las relaciones jerárquicas entre clases y componentes.
  - d. Gestionar la interacción con los actores finales.
- 7. ¿Cuál es la función principal de los puertos en un diagrama de componentes UML?**
- a. Representar la estructura jerárquica de los módulos.
  - b. Facilitar la comunicación entre diferentes componentes.

c. Definir las relaciones de herencia entre interfaces.

d. Organizar los archivos ejecutables dentro del sistema.

**8. ¿Cuál es el propósito principal del diagrama de despliegue?**

a. Diseñar la estructura de clases y objetos en el código.

b. Visualizar la disposición física del *hardware* y *software*.

c. Definir las relaciones de herencia entre componentes.

d. Detallar los casos de uso y escenarios de interacción.

**9. ¿Qué representan los nodos y los artefactos en un diagrama de despliegue UML?**

a. Nodos: métodos y operaciones. Artefactos: dependencias entre componentes.

b. Nodos: dispositivos físicos o virtuales. Artefactos: unidades de implementación de *software*.

c. Nodos: interfaces de usuario. Artefactos: servicios web expuestos.

d. Nodos: capas de presentación. Artefactos: acciones de negocio.

**10. ¿Cuál es el papel de las interfaces en un diagrama de despliegue UML?**

a. Definir las relaciones entre clases y objetos.

b. Especificar los métodos de prueba de *software*.

c. Representar los puntos de interacción entre componentes.

d. Organizar los artefactos en capas de arquitectura.

[Ir al solucionario](#)





## Resultado de aprendizaje 4:

Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 13

##### Unidad 5. Modelado avanzado

En la semana 13, damos inicio a una nueva unidad donde estudiaremos una forma de hacer el Modelado de sistemas avanzado. Para ello exploraremos el Modelado Ágil con C4, desarrollado por Simón Brown, que es una técnica moderna y ágil para la representación de la arquitectura de software. Este enfoque utiliza un conjunto de diagramas jerárquicos para representar diferentes niveles de abstracción del sistema, desde un diagrama de contexto general hasta los detalles de los componentes individuales.

Al finalizar la semana, usted estará capacitado para aplicar el modelo C4 en sus proyectos, facilitando un mejor diseño y comunicación de la arquitectura de software.

##### 5.1. Modelado ágil (C4)

En el entorno moderno de desarrollo de software, la agilidad y la capacidad de adaptarse rápidamente a los cambios son cruciales. Los métodos ágiles, como Scrum y Kanban, han transformado la forma en que los equipos desarrollan software, poniendo un fuerte énfasis en la iteración rápida, la colaboración y la entrega continua de valor. En este contexto, el modelado ágil se convierte en una forma importante de crear un modelo de software más rápido y que garantice la alineación con los objetivos del proyecto.

El método C4, desarrollado por Simón Brown, es una técnica de modelado ágil que utiliza una serie de diagramas jerárquicos para representar diferentes niveles de abstracción y puntos de vista en el modelado de sistemas. C4 significa:

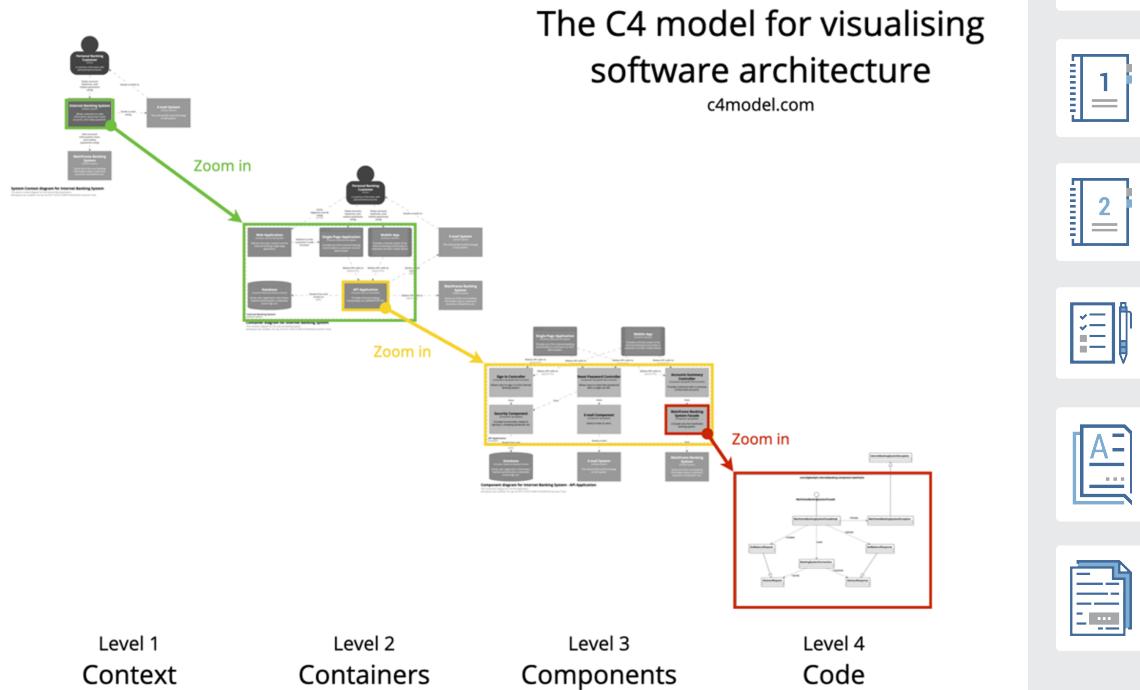
- **Contexto (Context):** representa una visión general de muy alto nivel del sistema y su entorno, incluye los usuarios y otros sistemas con los que interactúa.
- **Contenedores (Containers):** describe los contenedores de alto nivel que componen el sistema, como aplicaciones web, bases de datos, microservicios, funciones o cualquier otro componente de sistemas.
- **Componentes (Components):** permite ampliar los contenedores en componentes individuales, mostrando cómo se agrupan y colaboran para cumplir con las necesidades del sistema.
- **Código (Code):** es el nivel más detallado, describe el diseño interno de los componentes, a menudo utilizando diagramas de clases UML para mostrar las relaciones y la estructura del código.

La siguiente figura es una representación de alto nivel del uso de modelado C4 y sus diferentes niveles de especificación.



**Figura 32**

Representación del modelo C4



Nota. Tomado de *The C4 model for visualising software architecture* [Ilustración], por C4 model, 2024, [C4model](#), CC BY 4.0.

La notación del modelo C4 para diseñar modelos ágiles se basa en una representación clara y concisa de los elementos arquitectónicos en cada uno de sus niveles. En el nivel de Contexto (C1), se utiliza un diagrama simple que muestra el sistema como un rectángulo central con los actores externos representados como personas o sistemas alrededor de él. Las interacciones entre el sistema y los actores se indican con líneas etiquetadas, describiendo el tipo de interacción o relación.

En el nivel de Contenedores (C2), los contenedores se representan como rectángulos dentro del sistema, cada uno etiquetado con su nombre y responsabilidad. Las interacciones entre los contenedores y con actores

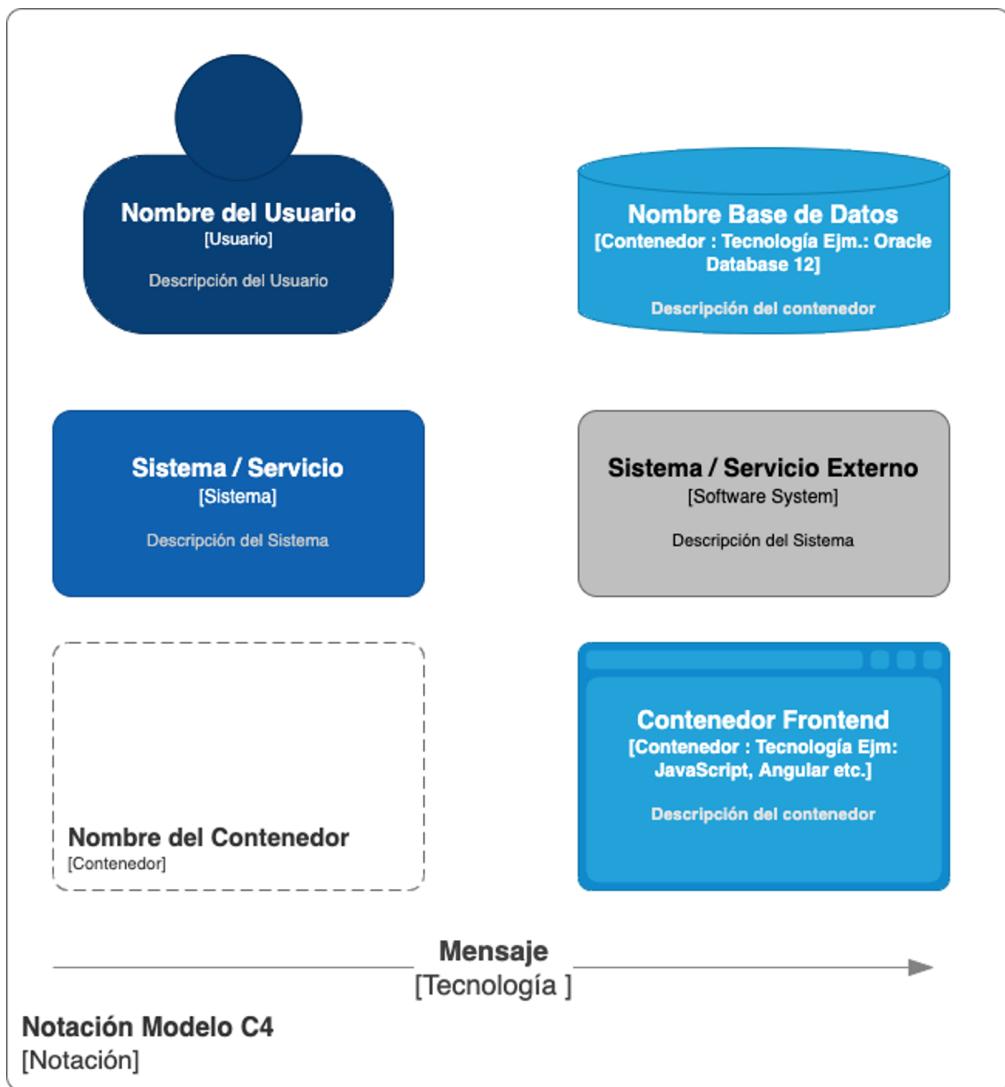
externos se indican con líneas etiquetadas, detallando las comunicaciones o dependencias. El nivel de Componentes (C3) desglosa cada contenedor en componentes más pequeños, representados también como rectángulos con sus nombres y responsabilidades, y las relaciones entre ellos se indican de manera similar. Finalmente, en el nivel de Código (C4), se utilizan diagramas UML clásicos como diagramas de clases y secuencias para mostrar la implementación detallada de los componentes, con clases, interfaces, métodos y sus interacciones representadas según los estándares UML. Esta notación estructurada permite una comprensión clara y detallada de la arquitectura del sistema en cada nivel de abstracción.

La figura a continuación resume los principales componentes de la notación C4.



**Figura 33**

Notación del Modelo C4



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

### 5.1.1. Nivel C1: contexto

Este es el nivel más alto en el modelo C4 y proporciona una visión general del sistema y su entorno, muy similar al diagrama de contexto UML. Este diagrama muestra cómo el sistema interactúa con usuarios externos (personas) y otros sistemas externos (sistemas). Es una representación simplificada que ayuda a todos los *stakeholders* a entender el alcance del sistema y sus interacciones principales sin entrar en conceptos técnicos.

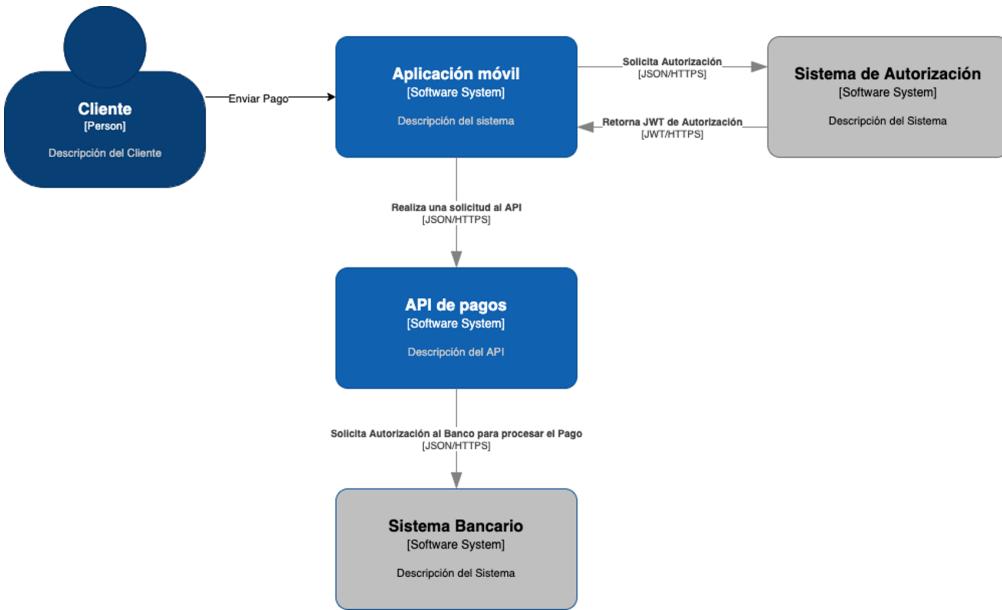
El diagrama de contexto es importante porque:

- **Define el alcance:** ayuda a delimitar claramente qué es parte del sistema y qué no, lo cual es esencial para establecer expectativas y responsabilidades.
- **Facilita la comunicación:** es una herramienta visual que todos, desde desarrolladores hasta *stakeholders* no técnicos, pueden entender, promoviendo una mejor comunicación.
- **Identifica interacciones clave:** muestra cómo el sistema interactúa con otros sistemas y usuarios, ayudando a identificar puntos críticos de integración y dependencia.

Continuando con el caso práctico desarrollado en esta guía sobre la API de pagos, la representación visual del C1, tendría los siguientes detalles:

**Figura 34**

Ejemplo del nivel C1 del Modelado C4



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

Este nivel representa cada uno de los elementos en el nivel más alto de representación:

- **Clients:** Los clientes interactúan con el sistema a través de la **Aplicación Móvil**. Utilizan esta aplicación para iniciar pagos.
- **Aplicación Móvil:** Es el componente que permite a los clientes enviar solicitudes de pago al **API de Pagos**.
- **Sistema de Autenticación:** Antes de que un cliente pueda realizar un pago, la aplicación móvil interactúa con el **Sistema de Autenticación** para verificar la identidad y nivel de acceso del usuario.

- **API de Pagos:** Este es el sistema central que recibe las solicitudes de pago de la aplicación móvil. Una vez recibe una solicitud de pago, realiza las siguientes acciones:
  - **Autorización del Pago:** Interactúa con el **Sistema Bancario** para autorizar la transacción.
  - **Procesamiento del Pago:** Una vez autorizado, el API de Pagos procesa el pago y registra la transacción.
- **Sistema Bancario:** Es el sistema externo que autoriza y procesa los pagos. El API de Pagos se comunica con el sistema bancario para verificar y completar las transacciones.



### 5.1.2. Nivel C2: contenedor

El diagrama de contenedores, o C2, es el segundo nivel en el modelo C4, permite descomponer a cada uno de los componentes de nivel C1, descomponerlo en contenedores. Un contenedor es una aplicación o servicio ejecutable, como una aplicación web, una base de datos, o un servicio de *backend*. Este diagrama muestra en cuanto a detalle cómo se compone cada uno de los contenedores y los componentes que facilitan la operación del sistema.

El diagrama de contenedores es importante porque:

- **Descompone el sistema:** proporciona más detalle de la arquitectura del sistema al mostrar los diferentes contenedores y cómo se relacionan entre sí.
- **Define interfaces y protocolos:** ayuda a identificar las interfaces y protocolos utilizados para la comunicación entre los contenedores.
- **Facilita la implementación y el despliegue:** proporciona una guía clara para los equipos de desarrollo y operaciones sobre cómo implementar y desplegar los diferentes contenedores del sistema.

Continuando con el caso práctico del API de pagos, descompondremos el sistema en sus contenedores principales y mostraremos sus interacciones. Para la API de pagos, identificamos los siguientes contenedores:

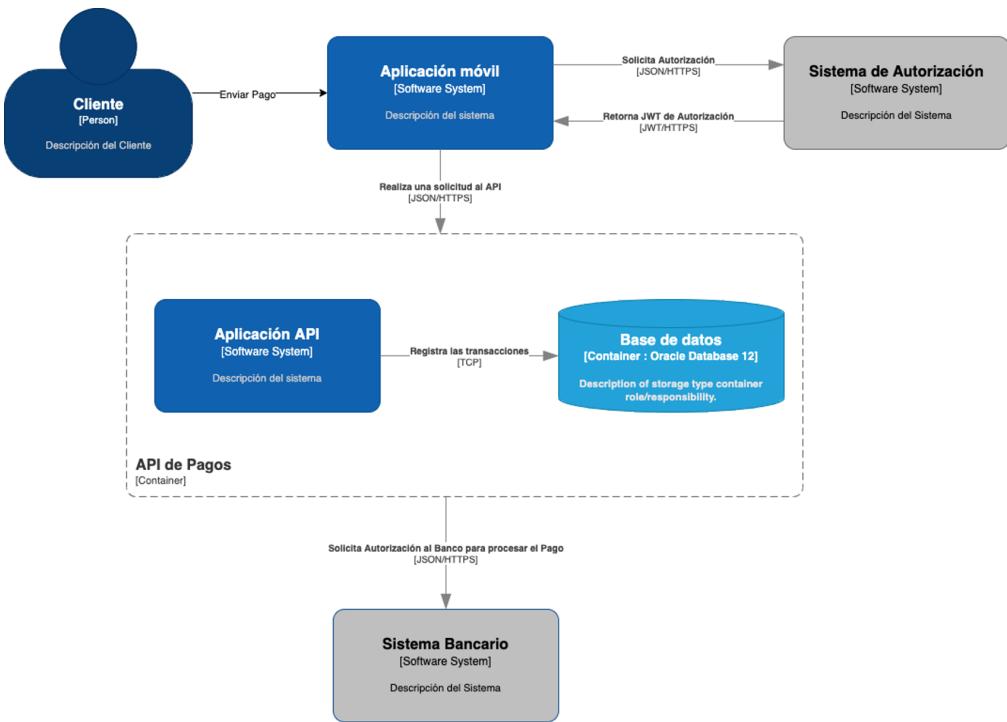
- **Aplicación móvil:** cliente móvil que permite a los usuarios iniciar pagos.
- **API de pagos:** *backend* que procesa las solicitudes de pago.
- **Servicio de autenticación:** servicio externo que autentica a los usuarios.
- **Sistema bancario:** sistema externo que procesa y autoriza los pagos.

Para este punto es fundamental que nos enfoquemos en los contenedores que queremos detallar, en este caso, la API de pagos. Este será el componente en el que nos enfocaremos para descomponer.

La figura a continuación representa la solución con respecto a contenedor del caso práctico utilizado en la guía didáctica.

**Figura 35**

Ejemplo del nivel C2 del Modelado C4



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

- **Aplicación Móvil:** Los clientes utilizan la aplicación móvil para iniciar pagos. Esta aplicación envía solicitudes HTTPS al contenedor del **API de Pagos**.
- **API de Pagos:** Este contenedor procesa las solicitudes de pago recibidas de la aplicación móvil. Realiza las siguientes acciones:
  - **Sistema Bancario:** Una vez autenticado el usuario, el API de Pagos se comunica con el **Sistema Bancario** para autorizar y procesar el pago.
  - **Base de Datos:** Registra la transacción en la **Base de Datos**.

- **Servicio de Autenticación:** Este contenedor es responsable de autenticar y autorizar a los actores antes de que puedan realizar un pago.
- **Sistema Bancario:** Este contenedor externo procesa y autoriza los pagos. El API de Pagos envía solicitudes a este sistema para completar las transacciones.
- **Base de Datos:** Este contenedor almacena todas las transacciones realizadas. El API de Pagos se comunica con la base de datos para registrar cada transacción y asegurar su persistencia.



### Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Visite la [web oficial del C4 model](#) y profundice en los conceptos clave del Modelado Ágil. Asegúrese de revisar las explicaciones detalladas y ejemplos de los diagramas de contexto, contenedores, componentes y código.
2. Tome el ejemplo del API de pagos y amplíe los diagramas C1 y C2. En el diagrama C1, agregue un nuevo sistema externo que interactúe con la API de pagos. En el diagrama C2, introduzca un nuevo contenedor que represente un servicio de notificaciones que envía confirmaciones de pago a los usuarios.

**Nota:** por favor, complete las actividades en su cuaderno de apuntes o en un documento Word.



## Resultado de aprendizaje 4:

Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 14

#### Unidad 5. Modelado avanzado

En esta semana, avanzaremos explorando los niveles C3 (Componente) y C4 (Código). Estos niveles son esenciales para detallar la estructura interna del sistema, permitiendo a los desarrolladores y arquitectos comprender cómo se implementan y organizan los diferentes elementos del sistema. A lo largo de esta semana, aprenderá a crear diagramas que desglosen el sistema en componentes más pequeños y específicos, y finalmente, llegará al nivel de código que muestra la implementación concreta haciendo uso de UML que hemos venido estudiando a lo largo del ciclo académico. Para complementar, continuaremos con nuestro caso práctico del API de pagos, construyendo y analizando los diagramas C3 y C4, lo que le permitirá ver la aplicación práctica de estos conceptos en un contexto real.

## 5.1. Modelado ágil (C4)

### 5.1.3. Nivel C3: componente

Este nivel permite desglosar aún más el sistema representado en los niveles C1 y C2, proporcionando una visión detallada de los componentes individuales que forman parte de cada contenedor. Este nivel es fundamental para los desarrolladores y arquitectos, ya que muestra cómo se implementan los diferentes elementos y artefactos del sistema.

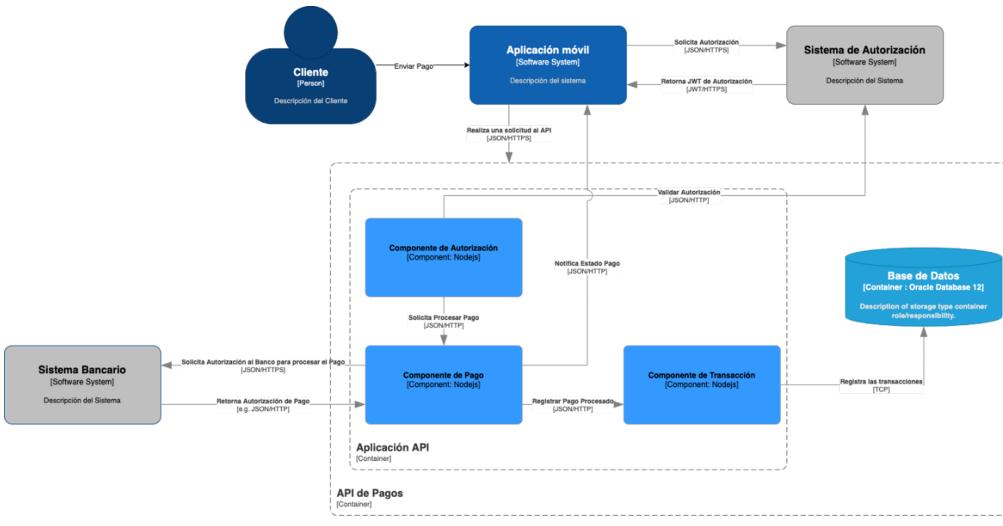
El nivel C3 es crucial por varias razones:

- **Detallada estructura interna:** proporciona una visión granular de la arquitectura interna de los contenedores, permitiendo identificar claramente los componentes y sus responsabilidades.
- **Facilita el desarrollo y la mantenibilidad:** permite descomponer el sistema en componentes manejables y más pequeños, de tal manera que los desarrolladores pueden centrarse en partes específicas del sistema.
- **Promueve el uso de patrones de diseño:** este nivel de detalle facilita la identificación y uso de patrones de diseño, permitiendo que el sistema sea robusto, escalable y fácil de mantener.
- **Mejora la comunicación:** establece un lenguaje común para que los desarrolladores y arquitectos acuerden la implementación y las interacciones entre los componentes del sistema.

Para desarrollar el diagrama de componentes C3 del caso que se viene analizando, se va a representar en la figura 36, la descomposición del contenedor de aplicación API.

**Figura 36**

Ejemplo del nivel C3 del Modelado C4



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

En el diagrama de componentes (C3) del API de pagos, descomponemos el contenedor de API de pagos:

- **Aplicación Móvil:** Este componente interactúa con el API de Pagos a través de una interfaz HTTPS. No se desglosa más, ya que se considera un contenedor en este contexto.
- **API de Pagos:**
  - **Componente de Autorización de Pago:** Responsable de gestionar las solicitudes de autorización de pago. Realiza validaciones iniciales y comunica con el Sistema Bancario para obtener la autorización.
  - **Componente de Procesamiento de Pago:** Este componente procesa el pago, actualizando los registros pertinentes y comunicándose con el Componente de Transacción para registrar la operación.

- **Componente de Transacción:** Registra las transacciones, este componente se comunica con la base de datos para almacenar la información de cada transacción.
- **Sistema Bancario:** Interactúa con el Componente de Autorización de Pago para validar y autorizar las transacciones.
- **Base de Datos:** El Componente de Transacción se conecta a esta base de datos para almacenar los detalles de las transacciones.

#### 5.1.4. Nivel C4: código

El diagrama de código es el nivel más detallado en el modelo C4. En este nivel, se desglosan los componentes del sistema hasta el nivel de clases y métodos individuales, con el objetivo de brindar un detalle más completo de la implementación del sistema. Este nivel es especialmente útil para los desarrolladores que desarrollan en el código fuente del sistema.

El nivel C4 es crucial por varias razones:

- **Detalle de implementación completo:** proporciona una visión detallada de cómo se implementan los componentes.
- **Facilita el desarrollo y la depuración:** permite identificar rápidamente problemas y optimizar la implementación.
- **Documentación precisa:** facilita una documentación detallada de la implementación, lo cual es útil para el mantenimiento y la evolución del software.
- **Promueve buenas prácticas de codificación:** ayuda a asegurar que se sigan buenas prácticas de desarrollo y uso, patrones de diseño en cuanto a código.

En el nivel C4, los diagramas UML (*Unified Modeling Language*) se utilizan para representar la estructura y el comportamiento del código de manera detallada. Los diagramas UML más comunes utilizados en este nivel son:

- **Diagrama de clases:** muestra las clases del sistema, sus atributos, métodos y las relaciones entre ellas.

- **Diagrama de secuencia:** representa la interacción entre objetos a lo largo del tiempo, mostrando el flujo de mensajes y las llamadas a métodos.
- **Diagrama de actividad:** detalla el flujo de trabajo dentro de un sistema, mostrando las actividades y las transiciones entre ellas.
- **Diagrama de estado:** describe los estados por los que pasa un objeto a lo largo de su vida y las transiciones entre esos estados.



Para referir su implementación, recuerde que estos diagramas se estudiaron en las unidades 3 y 4. Finalmente, este nivel permite a los desarrolladores y arquitectos asegurar que el sistema esté bien estructurado, sea escalable, mantenable y esté alineado con los requisitos y expectativas del negocio.



## Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise la documentación oficial del [modelo C4](#) para reforzar su conocimiento sobre los cuatro niveles de modelado. Preste especial atención a las diferencias entre el Modelado C4 y los enfoques tradicionales de UML. Prepare una tabla comparativa sobre las características principales del modelo C4 y UML en el contexto de desarrollo ágil.
2. Utilizando el caso práctico del API de pagos desarrollado en las secciones anteriores, extienda el modelo C1 (diagrama de Contexto) y C2 (diagrama de Contenedores) agregando un nuevo nodo que represente un sistema externo adicional que interactúe con la API de pagos. Puede ser un sistema de notificaciones o un sistema de reporte financiero. Describa las interacciones y actualice los diagramas para reflejar esta nueva integración.



**Nota:** por favor, complete las actividades en su cuaderno de apuntes o en un documento Word.





## Resultado de aprendizaje 5:

Selecciona los componentes que debe incluir en un proceso para solucionar un problema específico.

En el contexto de la construcción de sistemas modernos, es esencial entender cómo seleccionar los componentes adecuados que conforman un proceso, especialmente al trabajar con tecnologías de Cloud y DevOps. Estos enfoques permiten diseñar soluciones eficientes, escalables y automatizadas que responden a necesidades específicas del negocio. Para alcanzar este resultado de aprendizaje, analizará diferentes tipos de componentes en la nube y prácticas de DevOps, lo que le permitirá identificar cuáles son más adecuados para resolver problemas específicos. Al comprender las características, ventajas y limitaciones de cada componente, podrás tomar decisiones informadas que optimicen el proceso y satisfagan los objetivos del sistema.

### Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.



#### Semana 15

#### Unidad 5. Modelado avanzado

En esta semana, nos adentraremos en dos aspectos fundamentales para la arquitectura moderna de sistemas: el modelado de componentes *Cloud* y *DevOps*.

Durante este periodo, exploraremos cómo el entorno de computación en la nube ha transformado la manera en que diseñamos, implementamos y gestionamos sistemas complejos. Aprenderemos la importancia de identificar los componentes clave en arquitecturas basadas en la nube y cómo modelarlos de una forma sencilla con el objetivo de facilitar la implementación de las fases principales de la práctica *DevOps*.

## 5.2. Modelado de componentes cloud

La computación en la nube (*Cloud*) ha revolucionado la forma en que diseñamos, implementamos y gestionamos sistemas de *software*. El modelado de componentes en entornos *cloud* se ha convertido en una práctica crucial para asegurar la escalabilidad, la disponibilidad y la eficiencia operativa de las aplicaciones.

La computación en la nube permite a las organizaciones acceder a recursos informáticos, como servidores, almacenamiento, bases de datos, redes y *software*, y a diferentes modelos de uso a través de *Internet*, en lugar de tener estos recursos localmente en sus propios *DataCenters*. Esto proporciona flexibilidad, escalabilidad y economía de escala significativas, permitiendo a las empresas adaptarse rápidamente a las demandas cambiantes y reducir costos operativos.

Sin embargo, al ir a la nube y disponer de múltiples servicios, es importante crear modelos que nos permitan identificar rápidamente como están implementadas nuestras aplicaciones y los servicios de los cuales hacemos uso. Por tal motivo, y a diferencia de un contexto tradicional de desarrollo de *software*, los modelos de sistemas enfocados en el despliegue son mucho más relevantes en estos contextos modernos.

### 5.2.1. Principales componentes en arquitecturas cloud

- **Servicios cloud:** existe una variedad de servicios y modelos, como Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS)

y Software como Servicio (SaaS). Cada modelo proporciona diferentes capacidades y niveles de responsabilidad.

- **Microservicios:** es muy común que los sistemas que hacen uso de microservicios se diseñen e implementen utilizando microservicios, que son pequeños servicios independientes y altamente cohesivos. Con el objetivo de crear implementaciones altamente dinámicas.
- **Almacenamiento en la nube:** las soluciones de almacenamiento distribuido y escalable son esenciales en arquitecturas *cloud*. Esto incluye bases de datos distribuidas, sistemas de archivos distribuidos y almacenamiento de objetos, que permiten gestionar grandes volúmenes de datos de manera eficiente y segura.
- **Redes Definidas por Software (SDN):** permiten la gestión centralizada y programática de la red, lo que facilita la conectividad y la seguridad en entornos *cloud*. Esto es crucial para garantizar la comunicación fluida entre los componentes del sistema distribuido.

### Modelado de componentes *cloud* con UML

Para modelar implementaciones que hacen uso de componente *cloud* de manera efectiva, podemos utilizar diagramas UML específicos como diagramas de componentes, diagramas de despliegue y diagramas de secuencia, ya que estos permiten hacer una representación al nivel de servicios y componentes.

También es muy común en la industria utilizar Modelado C4 o en algunos casos crear representaciones sencillas y prácticas a través del uso de identificadores de los diferentes componentes *cloud*. Es importante recordar que, aunque no existe una receta o estándar para especificar soluciones que hacen uso de servicios *cloud*, en el mercado existen herramientas altamente especializadas y que proveen una curva de aprendizaje sencilla, por tanto, es significativo partir de la base y apoyarse en herramientas para crear este tipo de representaciones que hacen uso de técnicas modernas de construcción de software.

En el modelado a este nivel, se debe valorar que el diagrama o modelo debe proveer:

- **Visibilidad y comprensión:** permite una visión clara y comprensión de la arquitectura del sistema en entornos distribuidos y escalables.
- **Optimización y eficiencia:** ofrece una visión general que facilita la optimización de recursos y procesos, mejorando la eficiencia operativa y reduciendo costos.
- **Escalabilidad y flexibilidad:** ayuda a identificar casos en los que los sistemas deben escalar horizontalmente según sea necesario, adaptándose a cambios en la carga de trabajo y requisitos del negocio.

Algunas herramientas en las que usted puede apoyar su proceso de modelado de componentes *cloud* y que actualmente se usan en la industria son:

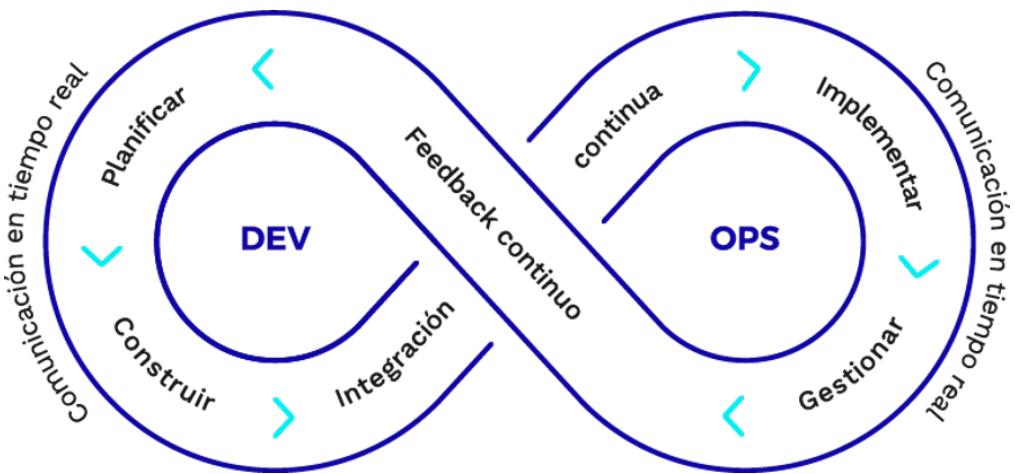
- **Lucidchart:** permite crear diagramas de arquitectura *cloud*, incluyendo diagramas de componentes y despliegue.
- **Visual Paradigm:** permite modelar arquitecturas haciendo uso de componentes de nube y soporta UML.
- **io:** herramienta gratuita y de código abierto que soporta la creación de diagramas UML, de arquitectura *cloud* y tiene la capacidad de extender sus características haciendo uso de Widgets o módulos externos.

### 5.3. Importancia de modelar para diseñar flujos de automatización en prácticas DevOps

DevOps es una práctica que integra el desarrollo de software (*Dev*) con las operaciones del sistema (*Ops*), promoviendo la colaboración continua entre equipos para acelerar la entrega de software de manera confiable y eficiente. En el contexto del modelado de sistemas, podemos utilizar diferentes tipos de flujos para representar el *End-to-End* de este proceso:

**Figura 37**

Flujo del proceso de DevOps.



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

Esta figura muestra el flujo continuo de integración (CI) y entrega (CD) de código desde la fase de desarrollo hasta la implementación y operación en producción. Este flujo permite cubrir todo el ciclo de desarrollo de software y puede incluir etapas como compilación, pruebas automatizadas, despliegue y monitoreo (Fowler, 2008).

Ofrece un enfoque más amplio y que une dos mundos, el desarrollo y las operaciones, por tanto, realizar un modelo para preparar el ciclo de desarrollo es importante ya que se cubren muchas aristas que requieren un abordaje más completo, documentado y detallado de que se hará la implementación. Además, el modelado de estos flujos ofrece una forma adecuada de hacer selección de las herramientas adecuadas para cada fase del proceso de Integración Continua y Entrega Continua (CI/CD). Aquí se destaca cómo el modelado inicial puede influir en la elección de herramientas:

- **Análisis de requisitos y capacidades:** mediante el modelado, se puede identificar los requisitos específicos del sistema, el stack tecnológico y

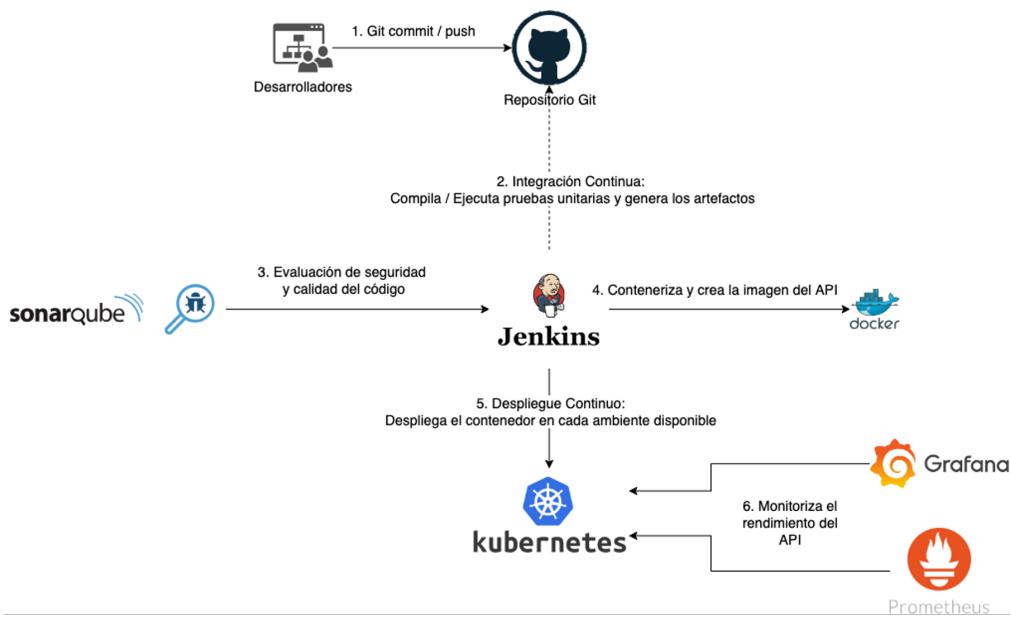
las necesidades de integración, de pruebas requeridas, y los entornos de implementación y operación del ciclo.

- **Selección de herramientas:** basándose en el modelado inicial, los equipos pueden seleccionar herramientas que se alineen con los componentes y servicios específicos de la arquitectura *cloud*, así como con los requisitos de integración y despliegue. Por ejemplo:
  - **Para la Integración Continua (CI),** herramientas como Jenkins, GitLab CI, o Azure DevOps se pueden seleccionar según la necesidad de integración y compilación automatizada de código.
  - **Para la Entrega Continua (CD),** herramientas como Docker, Kubernetes, o herramientas específicas de despliegue automático pueden ser utilizadas en función de la arquitectura de implementación.
  - **Compatibilidad con la arquitectura cloud:** las herramientas seleccionadas deben ser compatibles con la infraestructura y los servicios *cloud* utilizados, garantizando que el despliegue y la operación del software sean eficientes y seguros en el entorno *cloud*.

Dando continuidad al caso que hemos venido desarrollando, API de pagos por tarjeta de crédito para un sistema de comercio electrónico. Esta API debe ser seguro, escalable y eficiente, permitiendo a los usuarios realizar transacciones de forma rápida y segura. Cabe destacar que el modelo de la figura 38, es una representación que no sigue un modelo o estándar, pero que sigue un flujo y hace uso de identificadores que ofrecen una solución general de cómo se podría automatizar el flujo de desarrollo (Kurose, 2017).

**Figura 38**

Ejemplo de flujo automatizado DevOps



Nota. Tomado de *Guía Didáctica de Modelado de Sistemas [Ilustración]*, por Correa, R., 2024, Ediloja Cía. Ltda.

Le invito a revisar la siguiente infografía, donde podrá identificar cómo se realizará el paso a paso de la automatización de este modelo:

#### Proceso de automatización del modelo – Caso práctico



#### Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Vaya al video “[DevOps en 5 minutos | ¿qué es DevOps? | explicación DevOps](#)” para obtener una introducción rápida y clara sobre los principios y beneficios de *DevOps*, reforzando así sus conocimientos

en esta metodología clave para el desarrollo ágil y la entrega continua de software.

2. Revise el siguiente artículo [¿Qué es DevOps?](#), y profundice en cada uno de los conceptos de *DevOps* desde el punto de vista de AWS.
3. Le invito a realizar la siguiente autoevaluación, donde usted podrá evaluar los conocimientos adquiridos en la unidad 5. A lo largo de esta unidad, hemos estudiado diversas técnicas y metodologías que son fundamentales para entender y diseñar arquitecturas modernas, robustas y escalables. Desde el modelado ágil con C4 hasta el diseño de componentes en la nube y las prácticas integradas de *DevOps*, hemos abordado la importancia del modelado y cómo el uso de herramientas y enfoques pueden transformar la manera en que desarrollamos y desplegamos software.



## Autoevaluación 5

Elija la opción correcta a las siguientes interrogantes:

- 1. ¿Qué significa el acrónimo C4 en el contexto del modelado ágil de arquitecturas de software?**
  - a. Contexto, contenedores, componentes, código.
  - b. Componentes, contenedores, código, contexto.
  - c. Contenedores, componentes, código, contexto.
  - d. Código, contenedores, contexto, componentes.
- 2. ¿Cuál es el objetivo principal del nivel C1 (contexto) en el modelo C4?**
  - a. Definir los contenedores de alto nivel del sistema.
  - b. Mostrar la estructura interna de los componentes del sistema.
  - c. Ofrecer una visión general de alto nivel del sistema y su entorno.
  - d. Describir el diseño interno de los componentes del sistema.
- 3. ¿Qué representa el nivel C2 (contenedores) en el modelo C4?**
  - a. Detalles de implementación de los componentes del sistema.
  - b. Una visión general del sistema y su entorno.



- c. Interfaces y conexiones utilizadas para la comunicación entre contenedores.

- d. Descomposición del sistema en componentes ejecutables.

**4. ¿Por qué es importante el modelado ágil en el desarrollo de software?**

- a. Porque garantiza la documentación exhaustiva de todos los detalles técnicos.
- b. Porque permite adaptarse rápidamente a los cambios y mantener la alineación con los objetivos del proyecto.
- c. Porque reemplaza completamente a los métodos tradicionales de desarrollo de software.
- d. Porque reduce la necesidad de colaboración entre equipos multidisciplinarios.

**5. ¿Qué papel juegan los diagramas jerárquicos en el método C4?**

- a. Representan detalles de implementación de los componentes del sistema.
- b. Descomponen el sistema en diferentes niveles de abstracción.
- c. Proporcionan una perspectiva general del sistema y su entorno.
- d. Definen las interfaces y conexiones utilizadas para la comunicación entre contenedores.

**6. ¿Cuál es la principal ventaja de utilizar el modelo C4 en el desarrollo de software?**

- a. Facilita la documentación exhaustiva de cada línea de código.
- b. Mejora la escalabilidad del sistema sin necesidad de planificación adicional.
- c. Ofrece una comunicación clara y concisa sobre la arquitectura del sistema.
- d. Elimina la necesidad de adaptarse a cambios en los requisitos del proyecto.

**7. ¿Cuál es el objetivo principal del nivel C3 (componente) en el modelo C4?**

- a. Detallar la estructura interna de los contenedores del sistema.
- b. Describir las interacciones entre los diferentes contenedores.



c. Describe una vista detallada de los componentes individuales dentro de cada contenedor.

d. Muestra la implementación concreta de los componentes del sistema.

**8. ¿Qué papel desempeña el nivel C4 (código) en el modelo C4?**

a. Proporciona una vista general del sistema y su entorno.

b. Detalla las comunicaciones entre los componentes del sistema.

c. Describe los diferentes componentes que forman parte del sistema.

d. Desglosa los componentes hasta el nivel de clases y métodos individuales.

**9. ¿Cuál es la principal ventaja de utilizar el nivel C3 (componente) en el modelado ágil C4?**

a. Mejora la comunicación entre los stakeholders del proyecto.

b. Proporciona una documentación exhaustiva del sistema.

c. Facilita la identificación de interfaces y conexiones del sistema.

d. Permite descomponer el sistema en partes pequeñas para mejorar la mantenibilidad y actualización del software.

**10. ¿Por qué es útil el nivel C4 (código) en el contexto del desarrollo de software?**

a. Porque define claramente las interacciones entre los diferentes componentes del sistema.

b. Porque proporciona una vista general del sistema y su entorno.

c. Porque facilita una comunicación clara y concisa sobre la arquitectura del sistema.

d. Porque desglosa los componentes hasta el nivel de clases y métodos, permitiendo una comprensión completa de la implementación del sistema.

[Ir al solucionario](#)





## Resultados de aprendizaje 4 y 5:

- Diseña representaciones para facilitar la comunicación, discusión, exploración y validación de la especificación y diseño de un sistema.
- Selecciona los componentes que debe incluir en un proceso para solucionar un problema específico.

### Contenidos, recursos y actividades de aprendizaje recomendadas



#### Semana 16

#### Actividades finales del bimestre

Esta semana está dedicada al repaso y la preparación para la evaluación bimestral. A lo largo del segundo bimestre, hemos explorado una variedad de temas fundamentales en el modelado de sistemas utilizando UML, desde la sección de los Diagramas UML avanzados hasta la unidad 5. Modelo avanzado (Modelado Ágil) y sus diferentes secciones. Hasta este punto, hemos construido una base sólida de conocimientos y habilidades. Ahora es el momento de consolidar todo lo aprendido, resolver cualquier duda y prepararse de manera efectiva para la evaluación bimestral. A continuación, se presentan las indicaciones y actividades recomendadas para lograr este objetivo.

#### Estrategia de repaso y preparación

##### A. Revisión de contenidos

Revise y consolide el conocimiento adquirido en el segundo bimestre:

- **Semana 9:** revise el uso de diagramas de clase para representar la vista lógica.
- **Semana 10:** revise los conceptos de interfaces y paquetes, y la importancia de estos desde el punto de vista de desarrollo.

- **Semana 11:** repase el diagrama de componentes y refuerce su conocimiento sobre la importancia de estos para organizar su proyecto en módulos y submódulos.
- **Semana 12:** revise cómo usar y representar la vista física a través del diagrama de despliegue.
- **Semana 13:** profundice en la especificación de los niveles C1 y C2 del Modelo C4.
- **Semana 14:** revise como obtener provecho de ellos niveles C3 y C4 del Modelo C4.
- **Semana 15:** comprenda la importancia de modelar los flujos DevOps.

## B. Actividades de repaso

**Lleve a cabo** las actividades prácticas para reforzar los conceptos:

- **Elabore resúmenes de los contenidos:** realice un resumen de cada semana, destacando los puntos clave y conceptos importantes.
- **Trabaje en ejercicios prácticos:** vuelva a efectuar los ejercicios prácticos desarrollados en cada semana, intente mejorarlo, incluyendo nuevas funcionalidades o condiciones. Este es un punto significativo para abordar en su evaluación bimestral.
- **Participe de foros de discusión:** participe en foros de discusión con sus compañeros para compartir dudas, resolver preguntas y discutir conceptos importantes.

## C. Cuestionario del segundo bimestre

Desarrolle el cuestionario del segundo bimestre disponible en Canvas para evaluar sus conocimientos:

- **Cuestionario en Canvas:** realice el cuestionario del segundo bimestre en la plataforma Canvas. Asegúrese de completarlo dentro del tiempo asignado para simular las condiciones del examen real.
- **Autoevaluación:** revise sus respuestas y corrija los errores. Identifique las áreas donde necesita reforzar su



## D. Recursos adicionales

Utilice recursos adicionales para aclarar dudas y ampliar el conocimiento:

- **Lecturas recomendadas:** revise los capítulos y secciones recomendadas en los libros de referencia utilizados durante el curso.
- **Videos y tutoriales:** busque videos y tutoriales en línea que expliquen los conceptos que te resulten más difíciles.
- **Consultas al profesor:** aproveche las tutorías para plantear preguntas a su profesor sobre cualquier duda o tema que no haya comprendido completamente.

## E. Planificación de estudio

Organice su tiempo de estudio de manera efectiva:

- **Calendario de estudio:** cree un calendario de estudio para distribuir su tiempo de manera equitativa entre las diferentes semanas y temas.
- **Tiempo de descanso:** asegúrese de incluir tiempo para descansos y actividades recreativas para evitar el agotamiento.



A medida que concluimos este segundo bimestre y ciclo académico dedicado al modelado avanzado de sistemas con UML y Modelado Ágil, quiero felicitarlo por el esfuerzo y la dedicación que ha mostrado en cada semana de estudio. Le animo a aplicar todo lo aprendido en el cuestionario del segundo bimestre en Canvas y a prepararse de manera sólida para la evaluación. Utilice estos conocimientos como base para enfrentar con éxito los retos del siguiente ciclo académico.





## 4. Solucionario

### Autoevaluación 1

Pregunta	Respuesta	Retroalimentación
1	b	El modelado de sistemas permite abstraer y visualizar aspectos importantes del sistema.
2	b	Los modelos proporcionan una forma visual clara de entender el sistema y comunicar ideas.
3	c	Al igual que en otras disciplinas, el modelado de sistemas proporciona una guía fundamental para la implementación.
4	b	La abstracción permite centrarse en los aspectos más importantes, con el objetivo de simplificar la complejidad.
5	a	El modelado, especialmente con UML, complementa los principios de la POO, facilitando la visualización y diseño de objetos de una forma más clara.
6	b	Los diagramas de clases muestran las relaciones entre las clases, la estructura interna y su relación entre objetos.
7	d	El modelado permite prever problemas y corregirlos antes de que se conviertan en costosos errores durante la implementación.
8	d	La abstracción y división del sistema en componentes manejables facilita su comprensión y gestión.
9	b	Los modelos permiten prever problemas y conflictos, mejorando la planificación y calidad del software final.
10	b	UML es una herramienta estándar ampliamente utilizada para el modelado de sistemas orientados a objetos, proporcionando diversas notaciones y diagramas para representar la estructura y comportamiento de los sistemas.

[Ir a la autoevaluación](#)



## Autoevaluación 2

Pregunta	Respuesta	Retroalimentación
1	b	El Modelo 4+1 Vistas de Kruchten ofrece una visión integral y estructurada de la arquitectura de sistemas desde cinco perspectivas distintas.
2	d	La vista de escenario, se enfoca en las interacciones entre el sistema y su entorno.
3	b	El diagrama de clases se usa para representar la estructura interna del sistema, mostrando clases, atributos y relaciones.
4	c	La Vista de proceso describe cómo se ejecutan y comunican los procesos dentro del sistema.
5	c	El diagrama de actividad se utiliza para representar el flujo de trabajo y los pasos secuenciales en un proceso.
6	d	El diagrama de paquetes describe cómo se agrupan los artefactos de software en módulos o paquetes en la vista de desarrollo.
7	a	La Vista física se enfoca en la configuración física del sistema, incluyendo la infraestructura de hardware y software necesaria para su implementación.
8	c	El diagrama de despliegue se utiliza para representar la configuración física y la topología de red en la vista física.
9	b	Utilizar múltiples vistas permite capturar diversos aspectos del sistema, proporcionando una comprensión integral sin perder coherencia global.
10	a	La vista de escenario se utiliza para comprender los requisitos funcionales desde la perspectiva del usuario mediante diagramas de casos de uso.

[Ir a la autoevaluación](#)



### Autoevaluación 3

Pregunta	Respuesta	Retroalimentación
1	b	UML es un estándar de la industria que proporciona una notación en múltiples diagramas para representar diferentes puntos de vista de un sistema.
2	d	La vista de contexto se utiliza para definir los límites y el alcance del sistema.
3	a	La Vista de casos de uso se enfoca en identificar y documentar los escenarios funcionales del sistema desde el punto de vista del usuario.
4	a	El diagrama de casos de uso representa actores como entidades externas, casos de uso como acciones realizadas por estos actores dentro del sistema.
5	d	La especificación de casos de uso, según la norma IEEE 830-1998, busca eliminar la ambigüedad y representar claramente las acciones del escenario.
6	b	La especificación de casos de uso incluye precondiciones, flujos principales, flujos alternativos y postcondiciones para cada caso de uso.
7	b	El diagrama de actividades se utiliza para modelar y representar procesos de negocio.
8	a	El flujo de control se representa con un rombo y se utiliza para evaluar condiciones y definir caminos de ejecución.
9	b	El diagrama de secuencias se utiliza para modelar el comportamiento dinámico de un sistema.
10	a	La línea de vida en un diagrama de secuencias representa una instancia específica de una clase que participa en el intercambio de mensajes entre objetos.

[Ir a la autoevaluación](#)



## Autoevaluación 4

Pregunta	Respuesta	Retroalimentación
1	b	La vista lógica se enfoca en la estructura interna del sistema y cómo se organizan los elementos para cumplir con los requisitos funcionales, lo cual es crucial para el desarrollo efectivo del software.
2	c	Un diagrama de clases en UML se centra en representar la estructura estática del sistema utilizando clases para definir atributos y métodos, lo cual facilita la comprensión de la organización interna del sistema.
3	a	En UML, la diferencia clave entre la agregación y la composición radica en la permanencia de la relación “todo/parte”. En la composición, las partes no pueden existir sin el todo, mientras que en la agregación pueden existir independientemente.
4	b	Las interfaces en el diseño de software permiten separar cómo se proporciona un servicio (definición) de cómo se implementa internamente (implementación), promoviendo la modularidad y la flexibilidad en el desarrollo de sistemas.
5	c	Los paquetes en el desarrollo de software permiten organizar componentes relacionados lógica y físicamente.
6	b	Las interfaces provistas en el diagrama de componentes proveen funcionalidades específicas que un componente habilita para que otros utilicen.
7	b	Los puertos en el diagrama de componentes son puntos de conexión que habilitan la comunicación entrante y saliente entre componentes.
8	b	El diagrama de despliegue se utiliza para mostrar cómo se distribuyen físicamente los componentes del software en los nodos de hardware.
9	b	Los nodos computacionales en el diagrama de despliegue representan los dispositivos físicos o virtuales en los que se ejecutan los artefactos de software.
10	c	El diagrama de despliegue permite representar como los componentes de software se comunican entre sí a través de conexiones físicas o virtuales.

[Ir a la autoevaluación](#)



## Autoevaluación 5

Pregunta	Respuesta	Retroalimentación
1	c	El método C4 es utilizado para descomponer sistemas complejos en diferentes niveles de especificación.
2	c	El diagrama de contexto (C1) es fundamental para delimitar el alcance del sistema y facilitar la comprensión de las interacciones clave sin entrar en detalles técnicos.
3	d	El diagrama de contenedores (C2) muestra cómo diferentes aplicaciones o servicios interactúan entre sí y con usuarios externos, proporcionando una vista detallada de la arquitectura del sistema.
4	b	Los métodos ágiles, como el modelado ágil con C4, promueven la iteración rápida, la colaboración y la entrega continua de valor, aspectos clave en el desarrollo de software moderno.
5	b	Los diagramas jerárquicos permiten representar desde una vista general del sistema hasta detalles específicos de implementación.
6	c	Utilizar el modelo C4 facilita la gestión del cambio, el tiempo de diseño y la adaptación de nuevos requisitos.
7	c	El nivel C3 descompone los componentes individuales dentro de cada contenedor, permitiendo brindar una vista detallada de la estructura y responsabilidades de cada componente.
8	d	Un diagrama de código (C4) ofrece una imagen completa de cómo fue ejecutada esta implementación, permitiendo a los ingenieros entender aspectos específicos del diseño y la operación del programa.
9	d	Al desglosar el sistema en componentes individuales, el nivel C3 permite a los equipos de desarrollo concentrarse en partes específicas del sistema, lo cual es fundamental para mejorar su mantenibilidad.
10	d	Utilizar el nivel C4 permite a los desarrolladores comprender completamente cómo están implementados los componentes del sistema, facilitando así la depuración, optimización y evolución del software.

[Ir a la autoevaluación](#)



## 5. Glosario

**Abstracción:** representación de los aspectos esenciales de un objeto o clase sin mostrar los detalles.

**Actividad:** tarea ejecutable dentro del sistema que produce un resultado.

**API (*Application Programming Interface*):** componente de *software* que facilita la interoperabilidad entre sistemas.

**Arquitectura distribuida:** diseño de sistema donde los componentes físicos están distribuidos en múltiples computadoras interconectadas.

**Componente de *software*:** unidad modular e independiente de *software* que realiza una función específica dentro de una aplicación.

**Diagrama UML (*Unified Modeling Language*):** lenguaje estándar para visualizar, especificar, construir y documentar los artefactos de un sistema de *software*.

**DevOps:** práctica que integra los equipos de desarrollo de *software* (*Dev*) y operaciones (*Ops*).

**Escalabilidad:** capacidad de un sistema para gestionar el aumento y disminución en las cargas de trabajo.

**Esterotipo:** extiende la semántica de los elementos UML, añadiendo características específicas de dominio.

**Estado:** condición o situación en la que se encuentra un objeto en un momento dado.

**Herencia:** mecanismo que permite que una clase herede atributos y métodos de otra clase.

**IaaS (Infrastructure as a Service):** modelo de servicio *cloud* que proporciona acceso a recursos de infraestructura como servidores virtuales, almacenamiento y redes.



**Interacción:** acción entre objetos o clases en el sistema, representada en diagramas de secuencia y colaboración.



**Microservicios:** estilo arquitectónico en la que la aplicación se descompone en servicios pequeños e independientes y se disponibilizan a través de API.



**Middleware:** software que actúa como intermediario entre diferentes aplicaciones, permitiendo la comunicación y la gestión de datos entre ellas.



**Modelado de procesos:** representación visual de los procesos de negocio y su interacción dentro de un sistema u organización.



**Multiplicidad:** especifica las instancias de una clase pueden estar asociadas con otra clase.



**Orquestación de contenedores:** gestión automatizada de la configuración, implementación, escalado y operaciones de contenedores dentro de un clúster.



**Paquete:** agrupación lógica de elementos UML relacionados para organización y modularidad.

**PaaS (Platform as a Service):** modelo de servicio *cloud* que proporciona plataformas y herramientas para el desarrollo, prueba y gestión de aplicaciones.



**Patrón de arquitectura:** estructura de alto nivel que define la organización fundamental de un sistema, guiando su diseño y evolución.

**Patrón de diseño:** solución reutilizable a un problema común de diseño de software, que promueve buenas prácticas y estructuras.

**Patrón de diseño SOLID:** conjunto de cinco principios de diseño orientado a objetos para facilitar el mantenimiento y extensión del software.

**Polimorfismo:** capacidad de los objetos de una clase para ser tratados como objetos de su clase base.

**Realización:** indica que una clase implementa una interfaz especificada por otra clase o componente.

**Relación de agregación:** indica que una clase contiene a otra clase como parte de su estructura, pero esta puede existir de manera independiente.

**Relación de asociación:** conecta clases con roles específicos y multiprocesos que muestran cómo los objetos se relacionan entre sí.

**Relación de composición:** indica una relación más fuerte que la agregación, donde un objeto no puede existir sin el otro.

**SaaS (Software as a Service):** modelo de servicio *cloud* que permite acceder a aplicaciones de software a través de *Internet* bajo demanda.

**SDN (Software Defined Networking):** gestión centralizada y programática de la red mediante software para mejorar la agilidad y eficiencia en redes *cloud*.

**Virtualización:** creación de versiones virtuales de recursos de *hardware*, como servidores o almacenamiento, para optimizar la utilización de recursos físicos.

**Visibilidad:** controla el acceso a los atributos, métodos y otras características de una clase.



## 6. Referencias Bibliográficas

Rumbaugh, J. B. (1991). *Object-oriented modeling and design*. Englewood Cliffs: NJ: Prentice-hal.

Cockburn, A. (2000). *Writing Effective Use Cases*. Addison-Wesley.

Booch, G. R. (2005). *El lenguaje unificado de modelado (UML)*. Pearson.

Martin, R. C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.

Fowler, M. L. (2008). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley.

Kurose, J. F. (2017). *Redes de Computadoras: Un Enfoque Descendente*. Pearson Educación.

Humble, J. &. (2010). *ontinuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. E. (1991). Object-oriented modeling and design (Vol. 199, No. 1). Englewood Cliffs, NJ: Prentice-hall.

Booch, G. (1990). Object-oriented design with applications. Benjamin-Cummings Publishing Co., Inc.

Jacobson, I., Booch, G., & Rumbaugh, J. (2000). UML: el proceso unificado de desarrollo de software. Addison-Wesley.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. IEEE software, 12(6), 42-50.



---

## 7. Anexos

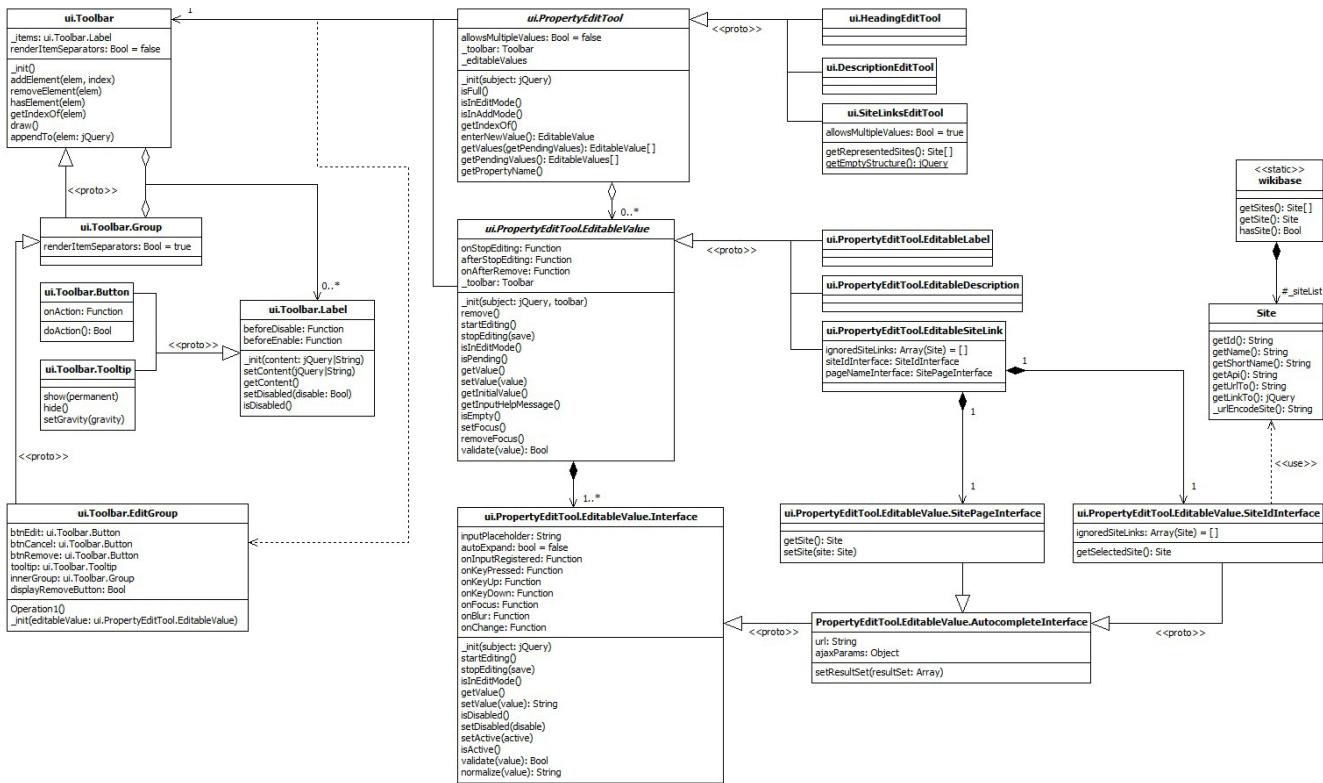
---

## Anexo 1. Representación UML del Modelado estructural de objetos

La figura representa un diagrama UML de clases de referencia.

**Figura 1**

Representación UML del Modelado estructural de objetos.



Nota. Tomado de [File:Wikibase JavaScript ui diagram.jpg](#) [Ilustración], por Daniel Werner, 2012, [wikimedia](#), CC BY 4.0.