

Componente formativo

**Análisis, diseño y arquitectura de software**

**Breve descripción:**

En este componente formativo se tratarán los temas de metodologías de desarrollo de software tradicional, ágil y elementos de estructura arquitectónica.

**Área ocupacional:**

Ciencias naturales, aplicadas y relacionadas

**Noviembre 2022**

Tabla de contenido

[Introducción 3](#_Toc142487993)

[1. Metodologías de desarrollo de software tradicionales 4](#_Toc142487994)

[2. Metodologías de desarrollo de software ágiles 11](#_Toc142487995)

[3. Elementos del modelo objetos 22](#_Toc142487996)

[4. Elementos de la estructura arquitectónica 23](#_Toc142487997)

[Síntesis 34](#_Toc142487998)

[Material complementario 35](#_Toc142487999)

[Glosario 36](#_Toc142488000)

[Referencias bibliográficas 37](#_Toc142488001)

[Créditos 38](#_Toc142488002)

Introducción

Apreciado aprendiz, bienvenido a este componente formativo, donde el contenido a estudiar es muy importante para gestar las buenas prácticas de diseño de un producto software.

En el siguiente video conocerá, de forma general, la temática que se estudiará a lo largo del componente formativo.

1. Introducción - Análisis, diseño y arquitectura de software



[**Enlace de reproducción del video**](https://www.youtube.com/watch?v=gomlgvATlPU&t=2s)

|  |
| --- |
| **Síntesis del video: Introducción - Análisis, diseño y arquitectura de software** |
| Este componente formativo brinda conocimientos fundamentales sobre metodologías tradicionales y ágiles en desarrollo de software para realizar una adecuada comprensión sobre sus conceptos, tipos, características principales y forma de implementarse en un ambiente empresarial y productivo.  Los proyectos de software están conformados por complicados procesos que involucran a diferentes participantes, conocimientos y procedimientos que incluyen reunir requisitos, pruebas y mantenimiento acordes con metodologías que ayudan a concretar el producto planificado.  En la industria del software con frecuencia los proyectos fracasan, debido a las malas prácticas en el proceso de su desarrollo o la gestión del proyecto, pero esta tecnología se encuentra en cambio y avance permanente, son la experiencia y la práctica las que permiten administrarlos de manera eficiente.  Es vital tener claridad de lo que se va a desarrollar, y saber qué es lo que el cliente desea para plasmarlo de la forma correcta. La comunicación es muy importante; por eso las formas correctas de llevarla a cabo son definitivas.  En este componente se brindan las herramientas fundamentales a través del lenguaje UML y de gráficos o diagramas para que su comunicación sea asertiva, sin dejar nada a la incertidumbre. Se utilizará un carácter teórico-práctico; por lo tanto, se llevará una secuencia de teorías muy importantes para la conceptualización sobre las metodologías de desarrollo de software, del modelo de objetos y de sus diagramas arquitectónicos. Con la comprensión de estas teorías se podrán entender e interpretar correctamente los ejemplos. Las actividades serán un derrotero para gestionar un proyecto de software, esto incluye prácticas para planificar el proyecto. Sea cual sea la metodología que se seleccione, se debe realizar siempre un proceso dedicado y consciente. |

# Metodologías de desarrollo de software tradicionales

Antes de hablar de metodologías de desarrollo de software tradicionales, conozca qué es metodología. Una metodología es un marco de trabajo que se puede aplicar como guía para el desarrollo de las actividades que se van a llevar a cabo. Por tanto, una metodología de desarrollo de software especifica las tareas que se deben seguir, los artefactos que se tienen que construir y las relaciones de los objetos.

Desarrollar un software de calidad depende de un proceso que contiene actividades y etapas donde la elección de la metodología a seguir genera un impacto trascendental en el éxito del proyecto.

Las metodologías de software tradicionales, son aquellas que siguen un proceso lineal donde cada una de las etapas del proceso, debe ejecutarse secuencialmente, seguida una de la otra. Estas metodologías se caracterizan por ser rígidas; al tener que definir todos los requerimientos desde el inicio del proyecto, no son flexibles ni permiten cambios.

**Tipos y características**

Los tipos de metodologías de software tradicionales son:

1. **Cascada**

Esta metodología de software divide el proceso de desarrollo en etapas que se organizan y recorren de arriba a abajo en estricto orden; de ahí se deriva su nombre.

El proceso incluye una serie de pasos, como se presenta en la siguiente figura.

1. Modelo de cascada

La imagen muestra cada una de las etapas del modelo de cascada.
En la imagen se muestran organizadas en forma de escalera lateral o cascada de donde proviene su nombre.

Ampliemos la información sobre cada una de las fases o etapas.

1. **Definición del software:** En esta fase, se define el producto desde una visión de alto nivel con los diferentes aspectos importantes que se deben tener en cuenta.
2. **Análisis de requerimientos:** Esta fase comprende las tareas de entender el negocio en su totalidad, conocer el producto que se quiere desarrollar, sus funciones, características, quiénes lo usarán y las relaciones con los sistemas externos.
3. **Diseño de la arquitectura:** Se realiza el diseño de la arquitectura de la solución que se implementará.
4. **Codificación:** En esta etapa se ejecuta el código de la aplicación, de acuerdo con lo estipulado en las fases anteriores, para cumplir todo lo pactado.
5. **Pruebas:** Fase donde se asegura la calidad del producto de software, se revisa que se satisfagan los requerimientos y se reduzcan los riesgos en la implementación en el entorno real.

Para poder aplicar esta metodología es necesario que los requerimientos del cliente sean conocidos de antemano o se tengan claros, también se recomienda que sea para un número reducido de requerimientos, teniendo presente que las pruebas se hacen cuando se encuentre desarrollado todo el producto software.

Una vez plasmados los requisitos, estos quedan en un estado de congelados, lo que dificulta que se cambien o que se incorpore otro requisito al producto; un ejemplo de esto sería que, al final de todo el desarrollo, cuando se está probando la calidad del producto en la fase de pruebas, el cliente solicite un nuevo requerimiento por un cambio en su negocio, o simplemente porque se le había olvidado. La magnitud de este impacto en el proyecto, genera caos porque la fase de requisitos es prácticamente la primera, lo que hace imposible satisfacer esta nueva necesidad.

Las fases no se pueden solapar, solo se inicia una fase cuando la etapa o fase anterior esté totalmente finalizada y probada. Tampoco son aconsejables el reproceso o la revisión de fases anteriores.

1. **Prototipado**

El prototipado es una metodología de desarrollo iterativa, en la cual se desarrolla una maqueta de la solución. Este prototipo se produce con el fin de mostrarle al cliente si se entendieron los requerimientos iniciales que se hicieron a alto nivel, y si van por buen camino, es decir, se hace el refinamiento con el cliente.

En este proceso, el cliente estará probando cada uno de los prototipos que desarrolle el equipo de proyecto y emitirá su juicio valorativo; dicho juicio determina si el prototipo es desechado o lo continúan para realizar mejoras.

1. Modelo de prototipado



Una de las características que tiene esta metodología, la cual puede ser desfavorable, es que, al presentarle un prototipo funcional al cliente, este tenga la idea errada que ya está listo el software y solo faltan detalles menores.

El usuario / cliente al estar involucrado en el proceso de desarrollo aumenta la aceptación del producto final que será implementado.

Es importante en esta metodología un entendimiento esencial del problema del negocio para evitar generar prototipos y así solucionar problemas incorrectos que provocan gasto de esfuerzo y tiempo de proyecto en prototipos sin fundamento.

1. **Espiral**

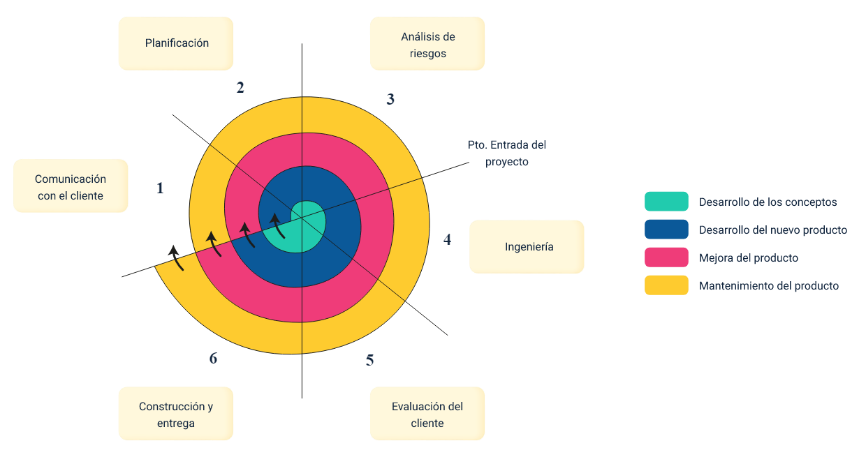
Esta metodología tiene la combinación de los modelos iterativos en su esencia, como el prototipado y el de cascada, por lo tanto, es una metodología de carácter incremental.

El modelo de cascada ofrece un proceso ordenado estricta y estructuralmente para el proyecto, pero las demandas de la industria de tiempos acelerados o reducidos, hace que sus fases sean inapropiadas. Este modelo se basa en ese orden de la cascada volviéndolo iterativo o incremental, teniendo presente que, al ser evolutivo, se aprovechan mejor las funcionalidades del equipo.

En este modelo, el software se divide en mini proyectos, los cuales evolucionan de acuerdo con las fases que sigue el espiral, comenzando con la comunicación con el cliente, para determinar los requisitos iniciales; luego se pasa a la planificación de esos requerimientos donde se incluye la estimación del costo en tiempo y recursos, continuando con el análisis de riesgo; luego empieza la ingeniería del proyecto donde se diseñan las posibles alternativas de prototipo, se presentan al cliente para su evaluación, después se construye, se entrega y se hace la retroalimentación, siempre de la mano del cliente. Si es aceptado, o va por buen camino, se sigue evolucionando; de lo contrario, se descarta y se comienza nuevamente de cero.

La siguiente imagen presenta una explicación gráfica de este proceso.

1. Modelo espiral



**Nota:** tomado y adaptado de Deloitte (2020).

La metodología o modelo en espiral, es un enfoque de desarrollo de software que surgió como respuesta a los inconvenientes presentados por el modelo de cascada. El espiral como se ve en el Figura 3, fundamenta todas sus fases, a través de espirales que se van repitiendo hasta entregar un producto final.

Este modelo es global y se usa junto con otros de desarrollo de software, tanto tradicionales como ágiles.

La característica principal es el control de riesgos que se puede llevar en este modelo porque, como es evolutivo, a medida que se avanza los riesgos se reducen.

Permite una estimación fácil debido a que se hace con mini proyectos donde los requisitos son muy pocos. También, como se evoluciona, hay espacio para atender las retroalimentaciones del cliente.

1. **Incremental**

Esta metodología es otro modelo que tiene en cuenta la evolución del producto en el tiempo, reiterando la funcionalidad del proyecto. La base fundamental de este modelo es el de cascada que se presenta repetidamente e incrementa en cada iteración la funcionalidad del producto, hasta completar, de esta manera, todos los requerimientos del proyecto.

Es bueno recalcar de esta metodología que, en cada iteración, se obtiene un producto operacional y no un prototipo como en el modelo de prototipado. El siguiente gráfico presenta el esquema del modelo.

1. Modelo incremental

La imagen representa los procesos del modelo incremental (Análisis, Diseño, Codificación y Pruebas).
Estos procesos se realizan 3 veces y se denominan Entregables de incremento.

Una de las características de este modelo de desarrollo es que el software se puede empezar a utilizar incluso antes de que se complete totalmente y, en general, es mucho más flexible que las demás metodologías.

Es importante tener claro que, cada iteración debe dar como resultado un producto funcional; encargarse de implementar esto puede requerir de un trabajo extra en cada repetición, lo que es un tiempo y un costo representativos y adicionales al proyecto.

1. **RDA**

Esta metodología, llamada así por sus siglas (RDA – Desarrollo Rápido de Aplicaciones) y también RAD por su origen en inglés (Rapid Application Development), consiste en tener un desarrollo de software lineal secuencial como se estudió en el modelo en cascada, pero llevándolo a un proceso en paralelo con distintos equipos de trabajo.

Estos equipos serán responsables cada uno de una parte de los requerimientos del proyecto. El objetivo de esta metodología es tener en un periodo corto el producto final completo.

Este modelo se caracteriza por sus equipos formados por más o menos seis personas, dentro de las cuales se incluyen desarrolladores y usuarios del sistema de tiempo completo, así como las demás personas involucradas, de acuerdo con los requisitos.

**El esquema de fases o etapas para esta metodología es el siguiente:**

1. **Modelado de gestión:** En esta etapa, se modela la información que gestiona el negocio, la información que el propio negocio genera y el responsable de esta, adónde se lleva, y el responsable de su procesamiento.
2. **Modelado de datos:** En este proceso de modelado se deben tener presente los objetos de datos que el software va a manejar, cuáles son los atributos de esos datos, cómo se componen y se relacionan entre sí y cuáles procesos de negocio impactan. Lo que se busca como objetivo final es modelar una abstracción muy clara del dominio total de los datos relevantes al problema o producto en cuestión.
3. **Modelado de proceso:** Lo que busca este modelado es diseñar el proceso de manipulación de los datos del paso anterior, cómo se agregan, eliminan, modifican o se recuperan.
4. **Generación de aplicaciones:** En esta etapa es donde se desarrolla el código, de acuerdo con el lenguaje de programación seleccionado previamente en los requerimientos, y se tratan de reutilizar, siempre que se pueda, los componentes existentes de los procesos anteriores.
5. **Prueba y entrega:** Fase donde se realizan las pruebas de lo desarrollado; sin embargo, si se reutilizan componentes, se supone que ya fueron probados en los procesos anteriores y solo falta probar la integración con los nuevos componentes.

**Tenga en cuenta:** las etapas anteriores se pueden modificar o variar según los libros, el autor o la página web consultada.

La característica más importante es que se necesita involucrar al cliente. Esta cooperación es necesaria para resolver dudas y poder avanzar.

También se requiere un gran volumen de recursos para poder realizar el trabajo en varios equipos de manera simultánea, además, se requiere que el software a desarrollar pueda ser modularizado.

# Metodologías de desarrollo de software ágiles

Las metodologías de desarrollo de software ágiles tomaron mayor auge en el año 2001, pero se venían revisando y concretando desde los años noventa. En este nuevo siglo tuvo lugar una reunión de representantes de los modelos ágiles que dio origen a lo que hoy en día se conoce como el manifiesto ágil, donde se encuentra plasmado el espíritu de estas metodologías.

Una parte del Manifiesto por el Desarrollo Ágil de Software, dice:

“Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

1. Individuos e interacciones sobre procesos y herramientas.
2. Software funcionando sobre documentación extensiva.
3. Colaboración con el cliente sobre negociación contractual.
4. Respuesta ante el cambio sobre seguir en un plan.

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda. (Beck, K., et al. 2001)

Como se pudo ver en el tema anterior, las metodologías tradicionales son estrictas y rígidas; las ágiles, en cambio, buscan la flexibilidad en el proceso, tratando de llegar a un equilibrio entre restricciones y el libre desarrollo.

Estas metodologías requieren comprensión y aceptación por parte del cliente y donde su papel cambia, se necesita una relación más estrecha. En este tipo de enfoques no existe el contrato permanente, es decir, no se fijan desde el inicio, costo, tiempo, recursos totales, ni alcance. El alcance debe variar de forma controlada para que, durante el proceso, el cliente defina sus requerimientos.

Existe una gran variedad de metodologías ágiles, las cuales son evolutivas, iterativas y tienen tiempos fijos en sus iteraciones.

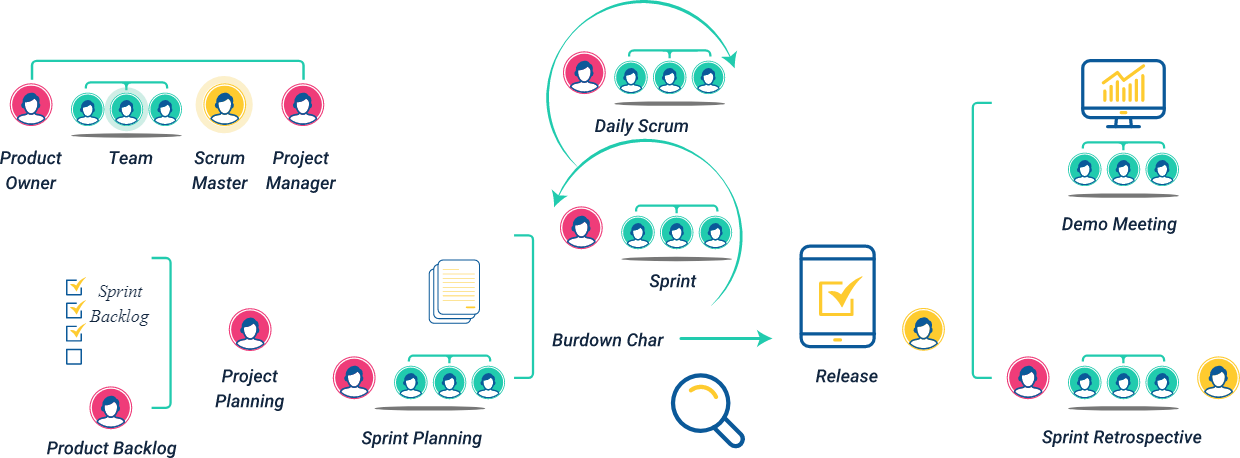
A continuación, conozca cada una de ellas:

1. **Scrum**



Esta metodología ágil está muy empoderada en la industria, su nombre viene del deporte rugby, donde todos los miembros del equipo deben hacer estrategias, unir fuerzas y empujar sus obstáculos para llegar al éxito que, en el caso del juego, serían los jugadores del otro equipo.

Esta metodología tiene una serie de componentes que se pueden agrupar en las categorías roles, activos y flujo de trabajo.



La anterior imagen muestra todo lo que maneja la metodología SCRUM y si es la adecuada para la implementación en el proyecto.

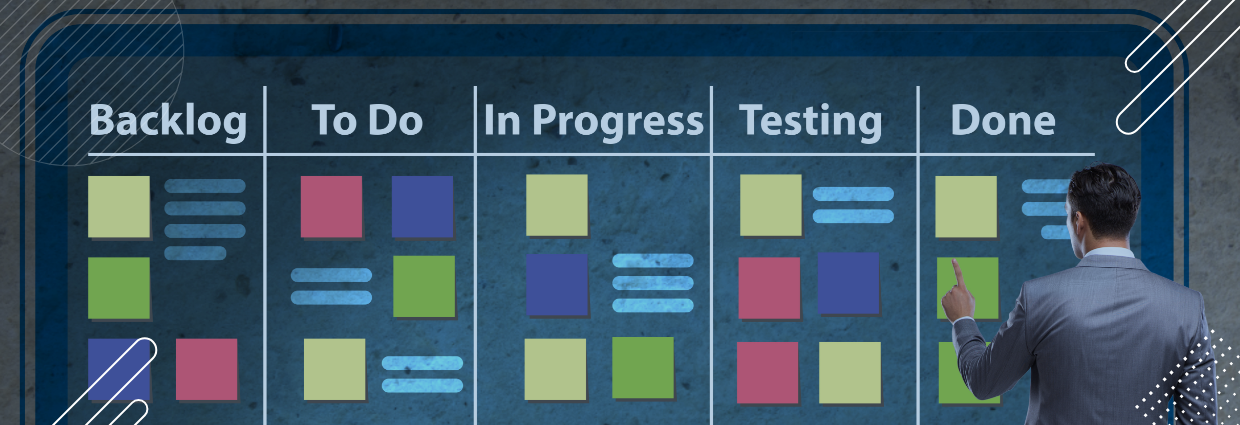
La metodología Scrum contiene los siguientes componentes:

1. **Product Owner:** Es una persona definida por el cliente, tiene que conocer muy bien el negocio y estar en todo el proceso del proyecto como un involucrado encargado de definir el software.
2. **Team:** Equipo de desarrolladores conformado por entre 3 y 7 personas.
3. **Scrum Master:** Es el rol encargado de quitar todos los posibles obstáculos del proyecto y engranar todas las piezas para que funcionen bien; en otras palabras, es un facilitador, y junto con el Product Owner, se ocupa de definir la operatividad del software.
4. **Project Manager:** Integra el proyecto a la organización y es el encargado de los recursos y su aprobación.
5. **Sprint Backlog:** Es el conjunto que se selecciona de requerimientos provenientes del Product Backlog y será desarrollado por el equipo en una iteración llamada Sprint.
6. **Product Backlog:** Es un listado de requerimientos ya priorizados del sistema.
7. **Project Planning:** Proceso donde se planifica todo el proyecto.
8. **Sprint Planning:** Tarea donde se lleva a cabo la planificación de la iteración.
9. **Daily Scrum:** Reunión diaria que se lleva a cabo entre el Scrum Máster y el equipo, para revisar las tareas del día.
10. **Sprint:** Es una iteración de duración fija y dura entre 2 y 4 semanas.
11. **Burdown Char:** Modelo o diagrama que muestra el trabajo sin terminar. Se puede hacer para el Sprint como también para el proyecto completo.
12. **Release:** Versión del código funcional entregado al cliente.
13. **Demo Meeting:** Reunión que se realiza al finalizar el Sprint con el objetivo de mostrarle al cliente la nueva funcionalidad.
14. **Sprint Retrospective:** Como su nombre lo indica, el equipo con el Scrum Máster revisa el Sprint que acaban de terminar y establecen cuáles son las fallas que se tuvieron y cómo mejorar para los próximos.
15. **Kanban**

Esta metodología tiene sus orígenes en los procesos de producción ideados por Toyota, donde se empezó con el uso de tarjetas para tener clara la cantidad de material que necesitaban en cada proceso de la cadena de producción.

Kanban significa en japonés, tarjetas visuales; la palabra kan es visual y la palabra ban es tarjeta. Hace parte de las metodologías ágiles cuyo objetivo es gestionar, de forma óptima, cómo se van completando las tareas y qué recursos tienen. Es así como Kanban es una metodología que busca apoyar a los equipos para encontrar un balance entre las tareas pendientes por realizar y la disponibilidad de cada integrante; su filosofía es la mejora continua a través de la técnica de extracción, es decir, se tiene un repositorio o lista de tareas, de la cual se van tomando y se van anexando al flujo de trabajo, apoyados en los tableros Kanban.

1. Tablero Kanban



La anterior gráfica es el tablero Kanban, el cual se utiliza para organizar el trabajo, cada tarjeta es una tarea y va avanzando en el tablero según su desarrollo.

Kanbantiene cuatro principios que son fundamentales para su implementación:

1. **Empiece con lo que hace ahora:** Esta metodología puede ser implementada en cualquier proceso de trabajo porque es lo suficientemente flexible para adaptarse.
2. **Comprométase a buscar e implementar cambios progresivos y evolutivos:** Los cambios muy grandes pueden impactar negativamente el equipo, por la demora, el esfuerzo y los posibles arreglos que toque realizar; por eso se deben hacer pequeños cambios e ir evolucionando.
3. **Respete los roles, los procesos y las responsabilidades actuales:** Kanban no tiene roles integrados o definidos, puede integrarse y funcionar con la jerarquía de su equipo.
4. **Impulse el liderazgo en todos los niveles:** Se habla de un liderazgo horizontal, y no solo vertical, dentro del proyecto o equipo de trabajo.

Los principios dan una guía para el equipo al momento de comenzar el flujo de trabajo, pero se deben seguir las siguientes prácticas que usan grandes empresas para lograr una mejora continua y un crecimiento progresivo:

1. **Visualizar el trabajo:** Kanban es visualmente muy claro. Una tarjeta de trabajo viajará del lado izquierdo al lado derecho del tablero comenzando por cada una de las columnas que representan un estado, hasta que llegue a estar finalizada.
2. **Limitar el trabajo en curso:** Kanban se basa en un principio de movimiento rápido, es decir, que cada tarjeta o tarea avanza rápido en el tablero y no se queda mucho tiempo en “trabajo en progreso”; no hay una regla, pero se debe poner un indicador limitante para saber cuántas tareas al tiempo pueden estar en progreso.
3. **Gestionar el flujo de trabajo:** Mejorar la gestión del flujo de trabajo ayudará a que se tenga control sobre el avance de las tareas y así se logre cumplir el principio anterior.
4. **Implementar políticas explícitas de procesos:** Las políticas de las tareas y su sintaxis o reglas deben estar claras. Todo el equipo debe conocerlas.
5. **Implementar ciclos de comentarios:** Se deben gestionar muy bien los comentarios, recopilarlos de ambas fuentes: del cliente y del equipo de desarrollo. Sean dudas, preguntas, soluciones y demás comentarios pertinentes.
6. **Mejorar colaborando y crecer experimentando:** Indica que se pueden tener otras metodologías fomentando el trabajo junto con Kanban. Esta colaboración mejora los procesos.

Los anteriores son los conceptos y la información fundamental para saber si el equipo necesita esta metodología y si puede empezar a implementarla.

1. **Lean**

Esta metodología busca la mejora continua y la eliminación del desperdicio. El desperdicio se refiere a todo lo que no aporta o suma al desarrollo del proyecto y, en consecuencia, al cliente.

Lo anterior no se trata de eliminar, como lo indica la palabra, sino de generar valor durante el flujo de trabajo, optimizando así el esfuerzo realizado en cada tarea, para lo cual es necesario siempre estar de la mano del cliente, y escucharlo.

Los fracasos se toman como aprendizajes; del error se aprende para no volverlo a cometer. Cuando una organización implementa Lean se caracteriza por ser metódica y buscar siempre el diferenciador con su competencia.

Fomenta la mentalidad de ensayo y error bajo su pilar de una gestión de riesgo e innovación eficiente.

Se debe respetar el equipo de trabajo sin importar su jerarquía; las ideas innovadoras siempre provienen de las personas que están directamente involucradas con el proceso de fabricación del producto. Este respeto fomenta un liderazgo horizontal y no vertical como las compañías comunes, y ese liderazgo Lean empodera a todos los empleados para la toma de decisiones, el aporte de nuevas ideas al proceso con el objetivo de ejecutar el trabajo siempre de la mejor forma y responder así ante cualquier situación.

1. **Programación Extrema XP**

Por sus siglas en ingles XP (Extreme Programming) fue desarrollada por Kent Beck y busca la disciplina extrema de las personas que conforman el equipo de desarrollo aplicando unas prácticas consideradas esenciales en el proceso. Se fundamenta en valores, principios y prácticas.

Los valores incluyen los de todo el grupo que conforma el proyecto, en este caso se refiere a la forma como toman las decisiones frente a una situación específica; son cinco y se describen a continuación:

1. **Comunicación:** Es el valor clave en el proceso de desarrollo de software y más específicamente en cada equipo de trabajo; junto con este valor se promueven la colaboración y el aprendizaje en equipo.
2. **Simplicidad:** Debido a varias lecciones aprendidas en los proyectos, se pone la simplicidad como base para el diseño y la codificación del software. Esto indica que solo se debe hacer lo que se pida, nada más.
3. **Retroalimentación:** Este valor indica aprender sobre lo ya realizado, es decir, sobre lo que ya se probó y se culminó. Aquí nace una práctica clave llamada refactorización.
4. **Coraje:** Como su nombre lo indica, este valor crea e impulsa la actitud positiva frente a cualquier situación. Siempre se dice la verdad ante cualquier pregunta sobre el proyecto.
5. **Respeto:** Para tener un grupo compenetrado la base de todo es el respeto entre las personas y sus interacciones.

Los valores que se acabaron de revisar no indican cómo se deben llevar a cabo las tareas del proyecto, son las prácticas las que muestran cómo se deben ejecutar en detalle. Pero como hay un espacio muy grande entre los valores y las prácticas, XP dispuso de unos principios que permiten la generación de prácticas concretas a partir de los valores.

En XP existen 14 principios que se describirán brevemente a continuación:

1. **Beneficio mutuo:** todas las tareas o actividades realizadas deben beneficiar a todos los miembros del equipo.
2. **Diversidad:** este principio indica que se debe ser diverso en el proceso de desarrollo de proyecto; todas las opiniones cuentan.
3. **Humanidad:** se deben valorar las necesidades humanas en el ciclo de vida del proyecto.
4. **Responsabilidad aceptada:** la responsabilidad no puede ser impuesta, debe aceptarse.
5. **Economía:** todas las actividades que se realicen deben aportar valor al negocio.
6. **Calidad:** el equipo debe trabajar con calidad y sentirse orgulloso de cada producto que desarrolle.
7. **Mejora:** se debe velar siempre por una mejora continua en los procesos.
8. **Flujo:** hay que asegurar el transcurrir normal de los flujos y la entrega de productos.
9. **Reflexión:** siempre se debe tener un proceso de reflexión ante cada trabajo realizado. Cómo y por qué.
10. **Pequeños pasos:** siempre se debe avanzar de manera constante y con pequeños pasos.
11. **Auto semejanza:** aprender de las experiencias pasadas y lecciones aprendidas.
12. **Falla:** no quebrarse ante una falla, sino utilizar ese caso como parte del aprendizaje.
13. **Oportunidad:** los problemas que se presenten en el desarrollo del proyecto hay que transformarlos en oportunidades de mejora.
14. **Redundancia:** tomar los problemas, buscar las diferentes soluciones y probar varias al tiempo.

XP, aparte de ofrecer valores y principios, tiene prácticas que proponen una mejor organización, planificación, comunicación y permiten generar una cultura de excelencia.

Las prácticas más relevantes son:

1. **El juego de la planificación:** Esta práctica es muy importante y requiere una participación activa del cliente en la planificación de los requerimientos junto con el equipo en busca de la cohesión entre el conocimiento del negocio y los requerimientos por parte del cliente, y el de costos y desarrollo del equipo.
2. **Pequeños releases:** El equipo de desarrollo debe velar por entregar con frecuencia al cliente versiones de las iteraciones funcionales.
3. **Metáforas:** Capacidad de explicar el sistema a personas nuevas con base en el concepto de metáforas con el fin de evitar el tiempo que gastarían leyendo la documentación.
4. **Diseño simple:** La idea es hacer solo lo que se solicita y mantener todo con simplicidad.
5. **Desarrollo guiado por pruebas:** Es un enfoque evolutivo que permite el desarrollo de actividades con calidad. Para ello se crea primero la prueba con la cual se va a testear y después se hace la mejora.
6. **Refactorización:** Consiste en mejorar el código ya realizado sin alterar su comportamiento.
7. **Programación en parejas:** Es una técnica de trabajo que reúne a un equipo de dos personas en un solo computador para que ambas trabajen resolviendo el mismo problema.
8. **Propiedad colectiva del código:** Esta práctica indica que el código no se privatiza o ciertas partes no son de equis desarrollador; cualquier desarrollador del equipo puede desarrollar o mejorar cualquier funcionalidad.
9. **Integración continua:** Indica que se requiere que todo se vaya integrando y probando de forma continua, sin esperar hasta el final para unir todas las partes.
10. **Historias de usuario:** Es un documento importante donde se describen las funcionalidades del proyecto.
11. **Estándares de codificación:** Consiste en producir código con un estilo estandarizado para todo el proyecto y no que cada desarrollador lo haga a su estilo.
12. **RUP**

El Proceso Unificado de Rational o Rational Unified Process (RUP), es una metodología que combina buenas prácticas de gestión de proyectos ágiles y tradicionales. La política de RUP se basa en desarrollar el proyecto en iteraciones que van evolucionando gradualmente. Cada iteración se compone de cuatro fases que forman su ciclo de vida y son guiadas por casos de estudio. El inicio se da con requerimientos genéricos y se avanza, hasta finalizar en los más especializados.

Existe otro método llamado AUP (proceso unificado ágil), el cual es una versión simplificada de RUP. Se basa en el desarrollo de aplicaciones empresariales y en él se usan técnicas ágiles llamadas TDD (desarrollo basado en pruebas), MDD (desarrollo basado en modelos) y gestión de software.

**Las cuatro fases en las que se divide esta metodología son:**

1. **Launch:** Es el inicio del proyecto donde se identifica el alcance, la arquitectura a alto nivel, se vinculan todos los interesados del proyecto y se elabora el presupuesto.
2. **Design:** Se diseña todo el software, incluyendo la arquitectura detallada.
3. **Realization:** Proceso de codificación del software que se hace de forma gradual con priorización de los requerimientos.
4. **Delivery:** Implementación y pruebas del software.

**Las metodologías ágiles tienen las siguientes características en común:**

1. **Menos costos:** Siempre se trata de mejorar la calidad del producto y que se haga solo lo que le aporte valor al cliente, con la finalidad de maximizar las ganancias.
2. **Mejores interacciones con el cliente:** Se busca optimizar la interacción con el cliente, por este motivo la comunicación debe ser fluida y en ambas direcciones. Se deben tener unos tiempos eficientes de respuesta.
3. **Mayor calidad:** Lean indica que se debe optimizar cada uno de los procesos y tareas prestando mucha atención en las especificaciones y sus detalles con el objetivo de reducir los defectos y retrasos. Con esto se optimizan los costos y se impacta en una mejoría en tiempos durante el proyecto.
4. **Crea cultura de mejora:** En una empresa, un equipo de proyecto debe tener cultura de la comunicación, por dar un ejemplo; pero, en este caso, se contextualiza en la importancia de la calidad, que se vuelve cultura, al crear estrategias, prácticas y formas de trabajo, que se llevan a cabo, en cada tarea realizada, para que el trabajo sea más eficiente.
5. **Mayor moral de los empleados:** Esto se da gracias al liderazgo horizontal, donde los jefes toman en cuenta las opiniones de los trabajadores y sus ideas se valoran, se revisan, se analizan y, de ser necesario, se implementan.

Para finalizar esta temática, encontramos las siguientes ventajas y desventajas de estas metodologías.

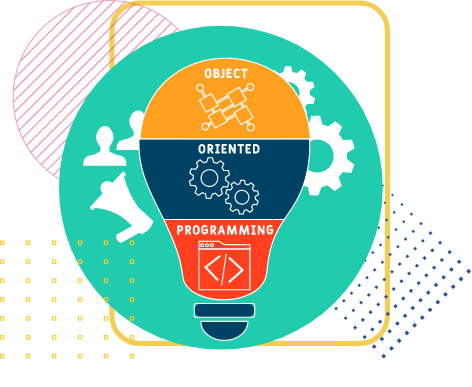
**Ventajas**

1. Incentiva la creatividad.
2. Entregas continuas.
3. Mejora continua del flujo de trabajo.
4. Flexibilidad ante los cambios.
5. Se eliminan los cambios posteriores a los proyectos ya terminados.
6. Mejora la comunicación del equipo e involucra más estrechamente al cliente.

**Desventajas**

1. La revisión excesiva, puede llevar a retrasos.
2. Sin un equipo de trabajo calificado, las entregas se pueden retardar.
3. Sin información detallada de los cambios, se puede llegar a operaciones improductivas.

# Elementos del modelo objetos



Cuando se habla del modelo de objetos se hace referencia a los modelos del paradigma de programación orientado a objetos; es decir, los conceptos, sintaxis, fundamentos y pilares que lo componen.

El paradigma de programación orientado a objetos es un estándar en la industria del software y es implementado en diferentes lenguajes de programación como Java, Net, PHP, C#, entre otros. La programación orientada a objetos se centra en las clases y en los objetos como su core de programación.

**Objeto**

Primero se debe definir qué es un objeto. En diferentes libros hay definiciones de este concepto, pero se dirá lo siguiente:

Un objeto es una entidad del mundo real según el contexto - problema que se esté trabajando. Tiene características (atributos) y comportamientos (métodos) específicos.

Por ejemplo, si se está trabajando un software para una universidad, el universo de dominio del problema trae varios objetos: los docentes, los estudiantes, los administrativos, etc.

Si se analiza, el estudiante tiene características que lo definen; lo que se llama atributos: código, nombre, dirección, número de documento, teléfono, etc.; y tendrá un comportamiento específico dentro del problema, como ver notas, matricularse, materias, etc.

**Clase**

Después de tener clara la idea de un objeto se debe revisar qué es una clase en este paradigma. La clase es como una plantilla. Si se hace una analogía, es como el plano de una casa, de ese plano se pueden crear muchas casas. En este caso, se podrían crear muchos objetos con ese plano llamado clase.

La clase es una entidad que reúne los objetos con atributos y métodos en común, y de la cual van a crearse los objetos. Un objeto es una instancia de una clase: vive y muere en memoria, en tiempo de ejecución del programa o software.

Más adelante se revisará cómo es una clase visualmente en los diagramas UML que se realizan en el diseño del software.

Pilares de la programación orientada a objetos: [En este enlace se muestran los pilares de la programación orientada a objetos:](https://sena-externos.github.io/CF6_228125_ANALISIS_DISENO_ARQUITECTURA_DE_SOFTWARE/downloads/Infografia_pilares_de_la_programacion_orientada_a_objetos.pdf)

# Elementos de la estructura arquitectónica

Los elementos de la estructura arquitectónica son los elementos de diseño del proceso de desarrollo de software. Para empezar, se debe indicar que un artefacto es un producto que resulta del mismo proceso de desarrollo y puede ser un documento, un modelo o un elemento en particular.

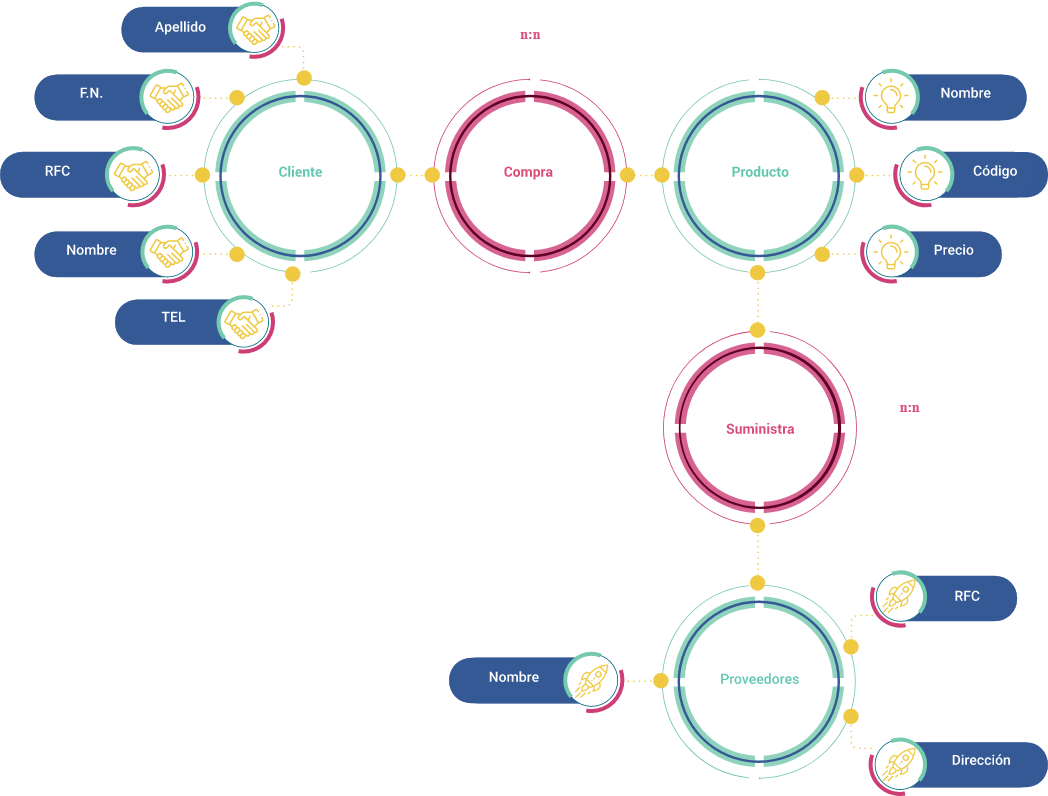
**Diagrama de base de datos**

El diagrama de base de datos se refiere al modelo entidad relación, también llamado modelo conceptual de datos. Fue desarrollado por Peter Chen en 1976 y consiste en una práctica dinámica que se caracteriza por una representación gráfica que incorpora información relativa a los datos y sus respectivas relaciones, usado como una vista general de los datos a implementar y la estructura en la organización.

El modelo entidad / relación sirve como guía para el paso siguiente que es la construcción de la base de datos en un sistema gestor de bases de datos.

El diseño del modelo entidad-relación, corresponde a la segunda fase de la creación de una base de datos, que se realiza una vez se ha recopilado toda la información por parte del cliente. Los elementos básicos de un modelo entidad-relación son: las entidades, las relaciones, los atributos y las cardinalidades, que pueden observarse en la Figura 6.

1. Elementos de un modelo de entidad / relación



A continuación, se detallan los elementos básicos de un modelo entidad - relación.

1. **Entidades:** Las entidades se representan gráficamente mediante un rectángulo que en el interior lleva el nombre correspondiente. El nombre de la entidad no se debe repetir, cada entidad lleva su nombre.

**Ejemplos de entidades:** profesores, empleados, estudiantes, clientes, proveedores, etc.

1. **Relaciones:** Las relaciones son las asociaciones que se describen entre dos atributos (campos) de dos tablas para compartir información. Esas respectivas relaciones entre las entidades se representan por medio de un rombo y describiendo en el interior la acción de la relación.
2. **Atributos:** Una entidad está representada por un grupo de atributos, los cuales son los encargados de describir una entidad específica. Como los atributos se encargan de describir cada una de las características o propiedades de la entidad, se representan por medio de una elipse o un círculo con su nombre correspondiente en el interior.

Por ejemplo, la entidad vehículo se puede describir con los atributos marca, modelo, color, placa, número de pasajeros, etc.

1. **Cardinalidad:** Esta cardinalidad dentro del modelo se representa en los extremos de cada relación. La cardinalidad expresa cuántas entidades de un extremo de la relación están relacionadas con cuántas entidades del otro extremo. Pueden ser “uno a uno”, “uno a varios” o “varios a varios”.
2. **Clave en una entidad:** Dentro del modelo entidad / relación están representadas:
   1. **La clave primaria:** es aquel atributo único para cada registro. Se representa por medio de una línea continua que subraya el nombre del atributo correspondiente.
   2. **La clave foránea:** es la encargada de demostrar la respectiva relación que se tiene en dos entidades; se representa marcando una línea subrayada de forma punteada debajo del nombre del atributo - candidato de la relación.

**Diagrama de clases**

Antes de iniciar a hablar sobre el diagrama de clases ―que muestra la estructura estática del sistema, sus clases y sus relaciones― se debe revisar qué es una clase y cómo se ve.

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

La siguiente imagen presenta un modelo de una clase.

1. Modelo de una clase



Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellas.

Son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Describe gráficamente las especificaciones de las clases de software y de las interfaces (por ejemplo, las de Java) en una aplicación.

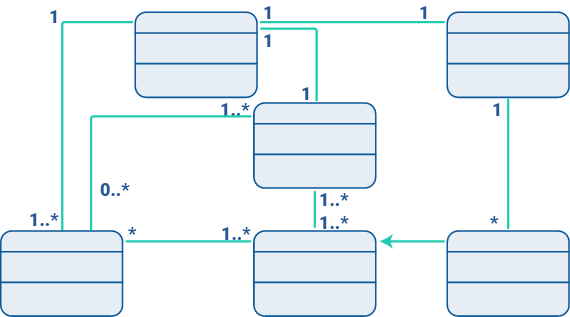
Contiene la siguiente información:

1. Clases, asociaciones y atributos.
2. Interfaces, con sus operaciones y constantes.
3. Métodos.
4. Información sobre los tipos de los atributos.
5. Navegabilidad.
6. Dependencias.

Para elaborar un diagrama de clases orientado al diseño se debe tener en cuenta lo siguiente:

1. Identifique todas las clases que participan en la solución del software.
2. Dibújelas en un diagrama de clases.
3. Identifique los atributos provenientes de los conceptos asociados del modelo conceptual.
4. Agregue los nombres de los métodos.
5. Incorpore la información sobre los tipos a los atributos y los métodos.
6. Agregue las asociaciones necesarias para dar soporte a la visibilidad requerida de los atributos.
7. Agregue flechas de navegabilidad a las asociaciones, para indicar la dirección de la visibilidad de los atributos.

Los datos que componen la clase son:



1. **Atributo:** Son valores que corresponden a un objeto; como color, material, cantidad, ubicación. Generalmente se conoce como la información detallada del objeto. Suponiendo que el objeto es una persona, sus propiedades serían: nombre, edad, sexo, etc.

**Tipo:** puede llegar a depender del lenguaje de programación a utilizar (public, protected, private).

**Valor inicial:** valor que poseerá el atributo al crear un objeto.

**Visibilidad:** está relacionada con el encapsulamiento.

**Multiplicidad:** determina si un atributo debe estar o no y si posee un único valor o una lista de valores.

**1->** El atributo debe tener un único valor.

**0…1->** El atributo puede o no tener un valor.

**0…\* ->** El atributo puede tener varios valores o ninguno.

**1…\*->** El atributo puede tener varios valores, pero debe tener al menos uno.

**\* ->** El atributo puede tener varios valores.

**m…n ->** El atributo puede tener entre m y n valores.

1. **Visibilidad:** El encapsulamiento presenta las siguientes ventajas básicas:
2. Se protegen los datos de accesos indebidos.
3. Favorece la modularidad y el mantenimiento.

Los atributos de una clase no deberían ser manipulables directamente por el resto de los objetos.

1. **Niveles de encapsulamiento:**

**(-) Privado:** es el más fuerte. Esta parte es totalmente invisible desde fuera de la clase.

**(~) Package:** solo es visible dentro del mismo package.

**(#) Los atributos** / métodos protegidos están visibles para las clases amigas y para las clases derivadas de la original.

**(+) Los atributos** / métodos públicos son visibles a otras clases.

1. **Método:** Una operación (método) es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento. Es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase.

Para cada operación debe especificarse:

1. **Tipo retornado:** puede llegar a depender del lenguaje de programación a utilizar.
2. **Parámetros:** especificación del tipo de datos y la información que determina el funcionamiento de la clase.
3. **Visibilidad:** está relacionada con el encapsulamiento (-, #, +).

**Diagrama de casos de uso**

El diagrama de casos de uso se utiliza para modelar todas las funcionalidades del sistema. Se basa en casos de uso y utiliza tres elementos básicos:

1. **Actores:** Representa los diferentes papeles que una persona u otro sistema pueden tener en la interacción con el sistema en construcción.
2. **Casos de uso:** Es el elemento que representa la funcionalidad que el actor puede realizar con el sistema.
3. **Relaciones:** Son las formas como se asocian los casos de usos, los actores y casos de uso y actores.



**Caso de uso**

Un caso de uso es una secuencia de acciones que ejecuta un actor que dan un resultado de valor y observable ―incluyendo sus variantes―y que el sistema puede realizar.

Los casos de uso son para modelar las funcionalidades del sistema, el qué hace el actor con el sistema y no el cómo. El nombre de un caso de uso debe comenzar con un verbo para especificar que se trata de un proceso.

Los requerimientos funcionales están naturalmente estructurados como casos de uso.

¿Cómo identificar casos de uso? Hay que preguntarse:

1. ¿Cuáles son las tareas y responsabilidades de cada actor con el sistema?
2. ¿Algún actor creará, almacenará, cambiará, borrará o leerá información del sistema?
3. ¿Qué casos de uso crearán, almacenarán, cambiarán, borrarán o leerán información?

**Actores**

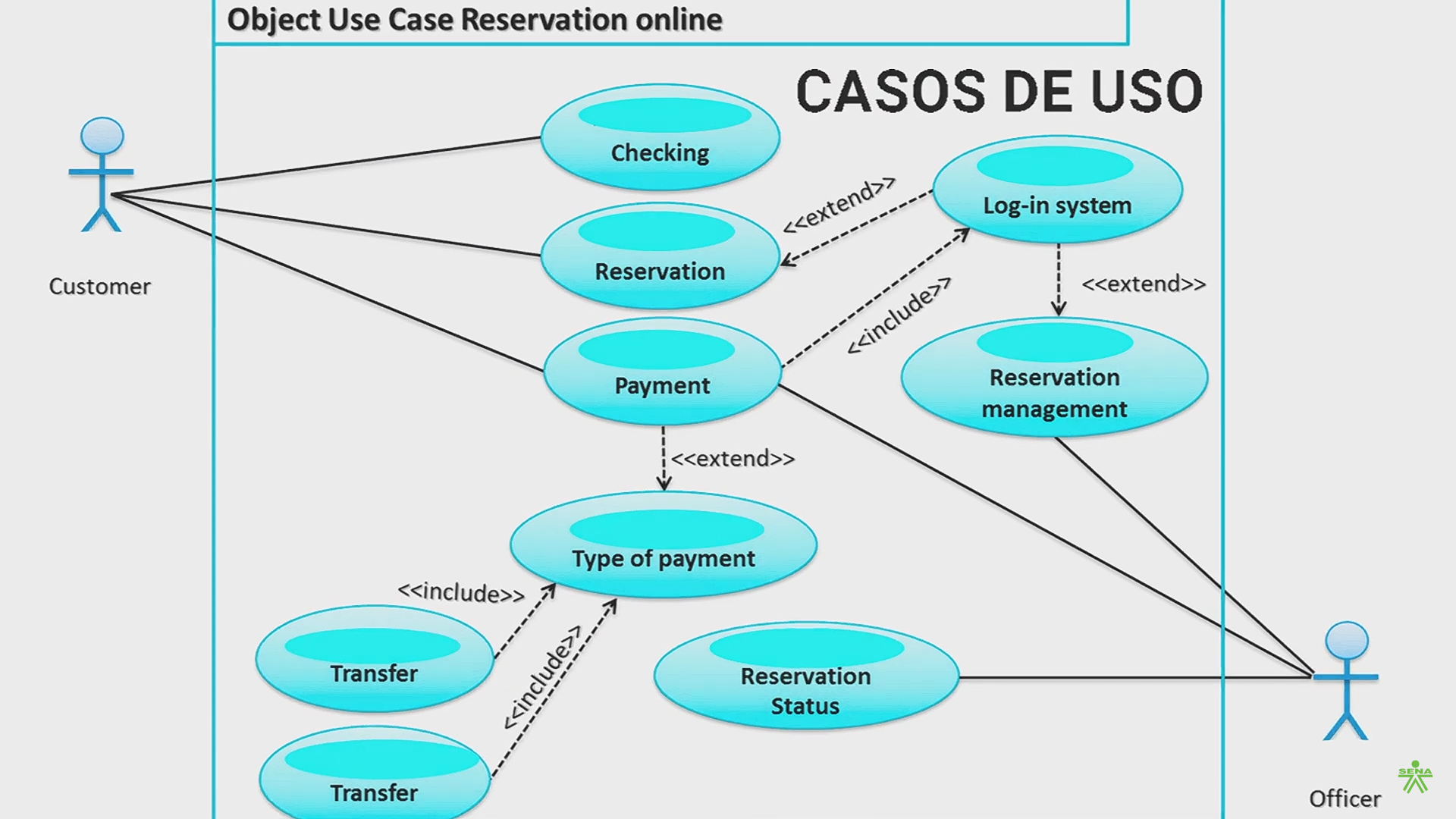
Especifica un conjunto de roles que los usuarios de los casos de uso representan al interactuar con estos.

Un actor puede ser el rol que desempeña una persona o un sistema externo que interactúa con nuestro software.

Los actores son los que ejecutan los casos de uso.

Para complementar el tema se le invita a ver el siguiente video.

1. Diagrama de casos de uso



[**Enlace de reproducción del video**](https://www.youtube.com/watch?v=RuIPx_fWYsU)

|  |
| --- |
| **Síntesis del video: Diagrama de casos de uso** |
| El video nos permite evidenciar un Diagrama de casos de uso, realizado con el ejemplo del sistema de un Cajero Automático, entre el actor y los casos de uso como: Retirar dinero, consultar dinero, consignar dinero, transferir dinero.  Ejercicio realizado en la herramienta StarUML. |

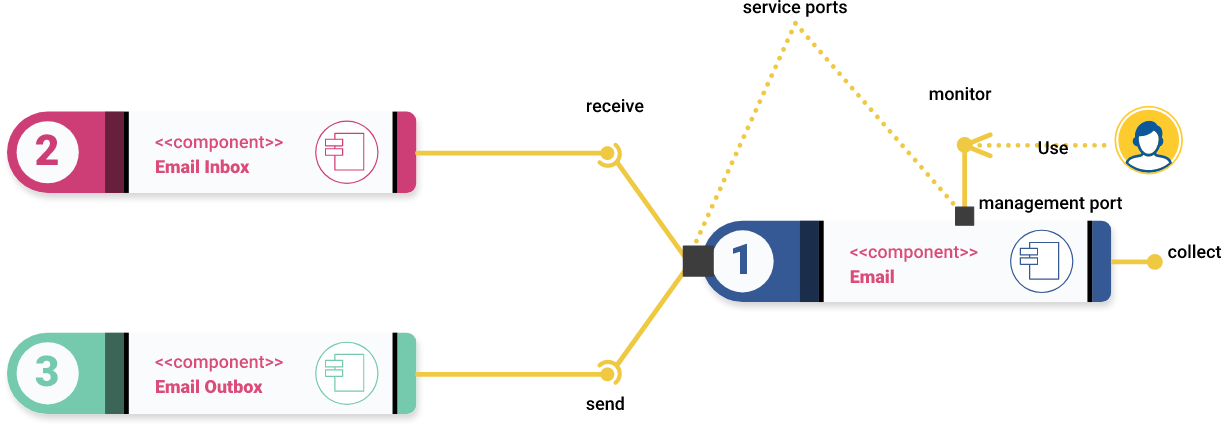
**Diagrama de componentes**

El diagrama de componentes modela los artefactos físicos o lógicos del sistema, en UML representa los módulos.

Los componentes encapsulan clases del sistema y por eso se les conoce como subclases, tienen estructura compuesta y define los mismos contenidos de la clase.

La siguiente imagen presenta un ejemplo de un diagrama de componentes de correo electrónico.

1. Ejemplo diagrama de componentes de correo electrónico

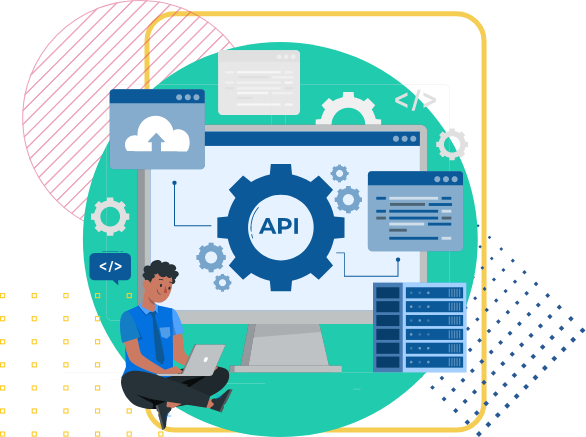


Nota. Tomado y adaptado de IONOS (2020).

**API**

Las API son componentes que permiten que dos software se comuniquen entre sí a través de un protocolo de comunicación; tienen diferentes definiciones y restricciones. Por ejemplo, un software puede crear una API para compartir la información de sus productos a otro servicio de software, para su insumo y trabajo.

Su sigla significa interfaz de programación de aplicaciones. En este caso, la palabra aplicación hace referencia a cualquier sistema y la interfaz sería el protocolo de comunicación entre dos servicios de dos sistemas.



La arquitectura de una API normalmente es cliente – servidor. La solicitud es enviada por el cliente hacia la API que se encuentra en el servidor y puede funcionar de cuatro formas diferentes:

1. API SOAP
2. API RPC
3. API WebSocket
4. API Rest

Actualmente en la industria las API están cargadas en la nube, por ejemplo, en la nube de Amazon, Azure o cualquier otro servicio en donde el usuario hace las peticiones. Tenerlas en la nube hace que la empresa solo se preocupe por el desarrollo de la API y la comunicación, la seguridad y el control de tráfico.

Los diagramas UML se caracterizan porque permiten una comunicación fluida con el cliente, comunican visualmente las ideas y, sobre todo, son precisos y eliminan la ambigüedad.

Existen diagramas que modelan la estructura y otros que modelan el comportamiento del software. A través de este modelamiento se comunican las ideas de forma certera; el cliente aprueba y se puede avanzar de forma correcta en el proyecto.

Recuerde explorar los demás recursos que se encuentran disponibles en este componente formativo; para ello, diríjase al menú principal, donde encontrará la síntesis, una actividad didáctica para reforzar los conceptos estudiados, material complementario, entre otros.

Síntesis

En la siguiente imagen podrá observar un resumen de la temática estudiada en este este componente formativo.

Análisis, diseño y arquitectura de software.
1. Análisis: metodologías de desarrollo de software, tradicionales y ágiles. 
2. Diseño: diseño orientado a objetos como los diagramas UML 
3. Arquitectura: diagrama de base de datos y API

Material complementario

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material | Enlace del Recurso o  Archivo del documento material |
| Elementos de la estructura arquitectónica | Lucid Software Español. (2019). *Tutorial – Diagrama de Clases UML* (video). YouTube | Video | <https://www.youtube.com/watch?v=Z0yLerU0g-Q> |

Glosario

**API:** interfaz de programación de aplicaciones.

**Atributo:** característica relevante de un objeto.

**Clase:** entidad principal del mundo real que agrupa varios objetos con atributos en

común y métodos en común.

**Método:** es una función de código que realiza un comportamiento especifico de un

objeto.

**Objeto:** entidad del mundo real que representa la instancia de una clase.

**Parámetro:** especifica el tipo de dato que recibirá un método.

**UML:** lenguaje unificado de modelado.

**Visibilidad:** valor que tendrá un atributo para conocer quién puede alcanzarlo en el

sistema.

Referencias bibliográficas

Beck, K., et al. (2001). *Manifiesto por el Desarrollo Ágil de Software*. <https://agilemanifesto.org/iso/es/manifesto.html>

Carrizo, D., & Alfaro, A. (2018). Método de aseguramiento de la calidad en una metodología de desarrollo de software: un enfoque práctico. *Ingeniare. Revista chilena de ingeniería*, 26(1), p. 114-129.

Guedes, G. T. (2018). UML 2-*Uma abordagem prática*. Novatec Editora.

Jabangwe, R., Edison, H. & Duc, A. N. (2018). Software engineering process models for mobile app development: A systematic literature review. *Journal of Systems and Software*, 145, p. 98-111.

Montero, B. M., Cevallos, H. V., & Cuesta, J. D. (2018). Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software. *Espirales revista multidisciplinaria de investigación*, 2(17), p. 114-121.

Silva, A., Ledezma, E., Castorena, J., Domínguez, A. & Riojas, A. (2018). Utilidad del Lenguaje Unificado de Modelado (UML) en el desarrollo de software profesional dentro del sector empresarial y educativo. *Ciencia Cierta revista de divulgación científica*, 56.

Velásquez, S. M., Montoya, J. D. V., Adasme, M. E. G., Zapata, E. J. R., Pino, A. A. & Marín, S. L. (2019). Una revisión comparativa de la literatura acerca de metodologías tradicionales y modernas de desarrollo de software. *Revista Cintex*, 24(2), p. 13-23.

Zumba, J. (2018). Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software. *INNOVA Research Journal*, 3(10), p. 20-33.

Créditos

Elaborado por Innovative Education

