## INDICE

## UML y el desarrollo de software Orientado a Objetos

Esta claro a esta altura de las circunstancias de los tiempos que corren que es imposible pensar en el desarrollo de sistemas informáticos en al actualidad sin una metodología de modelado que avale y sustente dichos desarrollos.

La complejidad y magnitud de los mismos (desarrollos de sistemas para Intranet empresariales con cientos o miles de terminales interconectadas, desarrollos de sistemas para Internet con un numero elevadísimo de accesos simultáneos distribuidos a lo largo del mundo, etc, etc, etc…) implica o conlleva que si uno desea llegar a buen término con los mismos debe tener como soporte primario una metodología de modelado.

Es bueno aclarar y no quiero que se forme un concepto erróneo que para sistemas informáticos de poca complejidad también es necesario contar con una metodología de modelado que los sustente, pero es más difícil de demostrar ya que si usted esta en su día de suerte y cree que los planetas se alinearon para que a usted le salgan las cosas bien y es un buen programador por una cuestión meramente azarosa podría desarrollar un sistema informático pequeño de principio a fin sin “esbozar formalmente” en papel ninguna metodología de modelado, sentarse directamente a programar y llegar a buen término en el desarrollo en cuestión, pero es importante resaltar un par de cosas.

1. Usted, como dije antes no formalizó en papel la metodología de modelado, pero mentalmente tenia claro como iba a desarrollar su sistema informático.
2. Pudo trabajar de esta manera por la baja complejidad y magnitud del sistema
3. Seguramente si el día de mañana la empresa para la que desarrollo el sistema posee un crecimiento importante, su sistema informático no le servirá o el costo de adaptarlo para las nuevas circunstancias será mayor que desarrollar uno nuevo.

4. En caso de lograr adaptarlo, la performance del mismo será menor que el de un sistema desarrollado con una metodología de modelado debido a que:
    a. Su sistema tendrá más líneas de código de las necesarias para realizar la gestión del sistema en cuestión
    b. Por lo citado anteriormente el mismo será un parche adaptado a las circunstancias del momento y no una solución real a las necesidades de la empresa.
    c. Será muy poco probable que soporte futuras adecuaciones.
    d. Tiene altas probabilidades que el sistema en cuestión colapse de forma inesperada.

Quiero ser claro con lo citado anteriormente, no es que este desmereciendo su trabajo, sino que deseo que entienda que esa no es la forma correcta de trabajar.

Además usted se preguntará que tiene que ver todo lo anterior con UML, pues bien UML (Unified Modeling Lenguaje, es un lenguaje unificado modelado de propósito general orientado a objetos, pero es importante que entienda que UML **"no es una metodología de desarrollo de software"**en si mismo).

Hechas estas aclaraciones empecemos a ver que características debe cumplir una metodología de modelado de forma general.

## *Por que debemos modelar*

Quiero hacer una aclaración a esta altura que me parece pertinente. Nosotros, los que trabajamos en sistema no somos los únicos que necesitamos y realizamos metodologías de modelado, de echo todas las demás ciencias o carreras también poseen sus propias metodologías de modelados. Por citar algunos ejemplos Los arquitectos y los ingenieros civiles antes de realizar una obra realizan los planos, maquetas y demos en 3D realizadas en computadora para mostrar como va a ser la obra y como quedará la misma una vez terminada.

Los diseñadores de autos antes de fabricar un modelo especifico realizan también los planos, maquetas, croquis, simulaciones en 3D y Concept para demostrar como será el modelo del auto en cuestión.

Los cineastas antes de empezar a filmar o rodar una película realizan lo que se llama storyboarding (representación de la película en pequeños cuadros o diapositivas dibujados a mano).

Los diseñadores de circuitos electrónicos antes de implementar los mismos también realizan los modelos en papel de los mismos. Y así podría seguir citando infinidad de ejemplos.

Con lo cual con lo dicho hasta este momento la primera pregunta que se me ocurre y que nos va a servir para entender por que aplicamos una metodología de modelado es la siguiente:

¿Qué es un modelo?

Con todo lo que ya dijimos hasta este momento creo que dicha pregunta se auto contesta simplemente diciendo que un modelo es una “simplificación de la realidad”.

Con lo cual de la respuesta anterior surge una segunda pregunta que es la siguiente:

¿Por qué modelamos?

Y con los ejemplos que dimos la respuesta a esta pregunta también es inmediata.

Nosotros construimos modelos para poder comprender más claramente el sistema que estamos desarrollando.

Por medio del modelado logramos “visualizar o imaginar” como queremos que sea nuestro sistema en cuestión.

Además los modelos nos permiten especificar como deseamos que sea la estructura y el comportamiento de nuestro sistema.

Queda absolutamente claro que al realizar diversos tipos de modelo indirecta y directamente estamos documentando las acciones que llevamos a cabo y decisiones que tomamos para realizar dichas acciones.

Y por último al diseñar modelos estamos definiendo plantillas y esquemas en común entendibles y asimilables por todos los integrantes del equipo de desarrollo.

Esta claro a esta altura que diseñamos y construimos modelos para sistemas complejos pues nos es imposible, como humanos que somos, entender el sistema en su totalidad.

Ahora bien, existen un conjunto de pautas o normas que nos ayudaran a construir buenos modelos.

## *Pautas para un buen modelado*

Para realizar un buen modelado es importante que tenga en cuenta lo siguiente. La decisión sobre que tipo de modelo crear tiene una muy fuerte influencia, casi indisoluble, sobre como se resuelve un problema y como se moldea la solución.

Lo que quiero decir con esto es que si el problema es encarado por alguien que trabaja con la metodología de “Análisis Estructurado”, seguramente el modelo que se cree esta orientado hacia el flujo de los datos de un proceso hacia otro haciendo hincapié fuertemente en los algoritmos que sustentan dichos procesos (de echo la metodología de Yourdon es una metodología de modelado orientada a procesos y flujos de datos).

Ahora bien si el problema es encarado por alguien que trabaja en el desarrollo de base de datos seguramente la metodología que utilizará será la de el modelo “Entidad – Relación”.

Pero si la persona que encara el problema en cuestión trabaja utilizando el “AOO y DOO” (análisis y diseño orientad a objetos) seguramente aplicara un modelo el cual se resuelva por medio de objetos y la interacción entre los mismos.

Otro detalle que es importante es que cualquier modelo puede expresarse a diferentes niveles de exactitud en función de las necesidades.

Por ejemplo con los ejemplos que dimos anteriormente tomemos el ejemplo de desarrollo de un nuevo modelo de autos. En este caso a la gerencia de Marketing por ahí le alcanza con ver el croquis del auto para saber si el mismo tendrá una buena penetración en el mercado según los requerimientos actuales sobre gusto o estilo de autos que poseen los clientes. A su vez a los ingenieros aplicados a la aerodinámica del auto lo único que necesitan es el Concept para realizar las pruebas en el túnel de viento y por su parte los ingenieros responsables del diseño del modelo en cuestión querrán todos los planos del modelo y de cada una de las piezas que lo componen para ver si es viable y factible la construcción del mismo.

Otra cuestión a tener en cuenta es que los mejores modelos son lo que están ligados fuertemente a la realidad. Como contraejemplo nadie compraría un auto que tuviera las ruedas en el techo y apoyara su chasis directamente al piso ya que eso no seria un auto y no satisface las funcionalidades que provee un auto.

Con todos los ejemplos dados hasta el momento queda absolutamente claro que una buena metodología de modelado esta compuesta por más de un modelo, mas bien podríamos decir que esta formada por un conjunto de pequeños modelos independientes entre si.

## *Modelado Orientado a Objetos*

La visión del modelado de sistemas orientado a objetos esta basada en tratar de ver el problema en cuestión y por ende su solución como un conjunto de objetos o clases que interactúan entre si.

Aclaremos un poco esto último.

## *Que es un Objeto?*

Un objeto es una unidad atómica que encapsula un estado y un comportamiento. Con lo cual un objeto representa una entidad física (por ejemplo un auto) o

abstracta (por ejemplo un proceso de facturación) del problema o del espacio de la solución.

Los objetos poseen las siguientes características fundamentales:

1. Todo objeto posee identidad
2. Todo objeto posee un estado
3. Todo objeto posee un comportamiento

### Identidad

Cada objeto posee un OID (identificador de objeto) el cual establece la identidad del objeto. Además el OID satisface ciertas características fundamentales:

- Dicho OID es único para cada objeto del sistema y el mismo posee un alcance global dentro del sistema en cuestión. No pueden existir dos objetos con el mismo OID. Si usted elimina un objeto y lo vuelve a crear el OID del nuevo objeto será diferente al OID del objeto eliminado más allá que a simple vista dichos objetos sean idénticos.
- Como acabamos de explicar queda absolutamente claro que el OID de un objeto es generado en el momento de su creación
- El OID es independiente de la ubicación física del objeto en cuestión
- El OID es independiente de las propiedades del objeto, con lo cual esto garantiza la independencia con respecto a la estructura y los valores de la misma.
- El OID persiste durante toda la vida del objeto y si el objeto se elimina el OID del mismo no se vuelve a utilizar.
- Los OID no se pueden controlar ni manipular y la administración de los mismos es transparente.

### Estado

El estado de un objeto queda determinado por los valores que toman los atributos de dicho objeto en un instante dado.

Cada atributo toma un valor de un dominio concreto (se llama dominio por si no lo sabe al conjunto de valores posibles y válidos que puede tomar un atributo. Suponga que tenemos un objeto EMPLEADO y que posee un atributo antigüedad, el dominio de dicho atributo estará dado por los números enteros positivos comprendidos entre 0 y 65, debido que un empleado ingresado este año no posee antigüedad, asumimos que la antigüedad se toma en años, y no puede tener un valor mayor a 65 pues una persona a los 65 años debe jubilarse, con lo cual este número le da un margen para las personas que trabajan un par de años más ya que por las leyes del menor una persona menor a 12 años no debería trabajar con lo cual el dominio podría ser de 0 a 53 si es que uno es estricto con las leyes vigentes).

El estado de un objeto satisface ciertas características:

- Como ya dejamos entrever anteriormente el estado evoluciona con el transcurso del tiempo.
- Algunos atributos pueden ser constantes
- El estado de un objeto se ve afectado por el comportamiento de dicho objeto. El comportamiento describe las acciones y reacciones del objeto en cuestión agrupando las competencias del mismo.
- Las operaciones de un objeto sobre otro también afectan al estado de estos. Las operaciones de un objeto son consecuencia de un estimulo externo representado por medio de un mensaje enviado desde otro objeto.

## Comportamiento

El comportamiento modela la interacción entre los objetos.

La interacción entre los objetos se modela por medio de los mensajes de los objetos.

El comportamiento global del sistema queda representado y se base en la comunicación entre los diferentes objetos que lo componen.

Un sistema informático puede verse como un conjunto de objetos autónomos y concurrentes que trabajan de manera coordinada en la consecución u obtención de un fin específico.

Como dijimos anteriormente el comportamiento agrupa las competencias de un objeto y describe las acciones y reacciones de ese objeto.

### Que es un mensaje?

A la unidad de interacción entre los objetos se la denomina mensaje. El mensaje es el soporte de una comunicación que vincula dinámicamente los objetos que fueron separados previamente en el proceso de descomposición del sistema.

Como dijimos antes un estímulo causará la invocación de una operación, la creación o destrucción de un objeto o la aparición de una señal, etc.

Con lo cual podemos asumir que un mensaje es la especificación de un estímulo dado.

Un mensaje desencadenará una acción en el objeto destinatario. Y dicho objeto debe poseer la inteligencia necesaria para poder interpretar y tratar dicho mensaje.

### Persistencia

Existe una característica más que se puede derivar de las tres características anteriores y que es la de persistencia de los objetos.

La persistencia de los objetos define la capacidad de un objeto de trascender en el espacio/tiempo.

Dicha característica garantiza que podemos materializar cualquier objeto tomado de memorias secundaria para usarlo en la ejecución del sistema y podremos reconstruirlo en el mismo estado que se encontraba antes de dejarlo en dicha memoria.

Esta característica debería ser transparente para el usuario ya que un objeto existe desde el momento de su creación hasta el momento que se decide destruirlo, pero los lenguajes orientados a objetos que existen en la actualidad no proponen un soporte totalmente adecuado o purista sobre dicha característica.

## *Que es una clase?*

Subjetivamente el mundo real puede ser visto desde diferentes tipos de abstracciones de la realidad.

Existen para ello diferentes método de abstracción los cuales pasamos a enumerar.

- Clasificación / Instanciación
- Composición / Descomposición
- Agrupación / Individualización
- Especialización / Generalización

Dentro de estos métodos la clasificación es uno de los métodos más utilizados.

Pues ahora bien existen diferentes maneras de interpretar que es una clase.

Se entiende como clase:

1. A una descripción de un conjunto de objetos similares.
2. Define el ámbito de definición de un conjunto de objetos.

Cabe a esta altura realizar un par de aclaraciones que le pueden ayudar a entender el tema en cuestión.

1. Cada objeto que existe en el sistema pertenece a una clase especifica que dio su origen (podríamos pensar a una clase determinada como la gran fabrica de objetos que poseen las características de esa clase. O también podemos pensar que la clase es una matriz para fabricar dichos objetos).
2. Con lo que acabamos de decir queda como corolario que los objetos se crean por instanciación de una clase.

Las clases poseen una característica fundamental que heredan los objetos y es la característica o propiedad de encapsulación.

## Encapsulación

El encapsulamiento provee al modelo orientado a objetos de ciertas características muy deseables como las que pasamos a describir:

1. Se protegen los datos de accesos indebidos o indeseados.
2. El acoplamiento entre las clases disminuye (se denomina acoplamiento bajo el entorno orientado a objetos a la dependencia funcional existente entre dos clases distintas)
3. Como consecuencia del punto anterior aumenta la cohesión (la cohesión bajo el entorno orientado a objetos implica que una clase u objeto derivado de una clase se ocupa de una funcionalidad especifica, por ejemplo en un entorno ideal de objetos supongamos que tenemos un objeto FACTURA el mismo tendrá toda la lógica necesaria para realizar la funcionalidad correspondiente a la factura, pero no debería ocuparse de la persistencia de la misma, o sea como y donde se guarda y se recupera la factura. Si esto último sucediera la clase que genera la factura y el objeto FACTURA en si no sería cohesivo ya que se esta ocupando de dos funcionalidades diferentes y sobre todo la ultima no es una funcionalidad propia o que haga a la esencia de una factura).
4. Favorece la modularidad y el mantenimiento.

Los atributos de la clase y por ende de los objetos no deberían ser manipulados directamente por el resto de los objetos.

El encapsulamiento se aplica sobre los atributos y métodos o mensajes de una clase y por ende se deriva a los objetos instanciados de la misma.

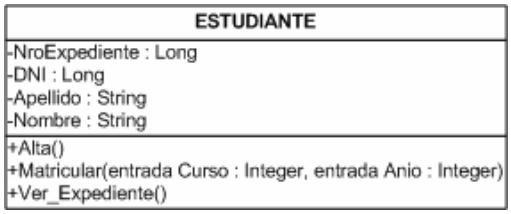
Existen tres niveles de encapsulamiento los cuales pasamos a describir:

**Privado:** Es el más fuerte de todos los atributos y/o métodos que se definan como privados estarán totalmente invisibles para el resto de las clases del sistema.

**Protegidos:** Los atributos y/o métodos que se definan como protegidos únicamente estarán accesibles para las clases derivadas de la original

**Públicos:** Los atributos y/o métodos que se definan como públicos estarán accesibles desde cualquier otra clase del sistema. Ahora vale en este momento hacer una aclaración muy importante. Si define los atributos de una clase como públicos esta rompiendo y transgrediendo el principio básico de encapsulamiento que poseen las clases. Tenga esto siempre bien presente.

Veamos como se representa una clase en UML con sus atributos y métodos o operaciones.




Observe la clase estudiante que posee los atributos privados NroExpediente, DNI, Apellido y Nombre y los métodos públicos Alta(), Matricular(…) y VerExpediente()

## *Relaciones entre clases*

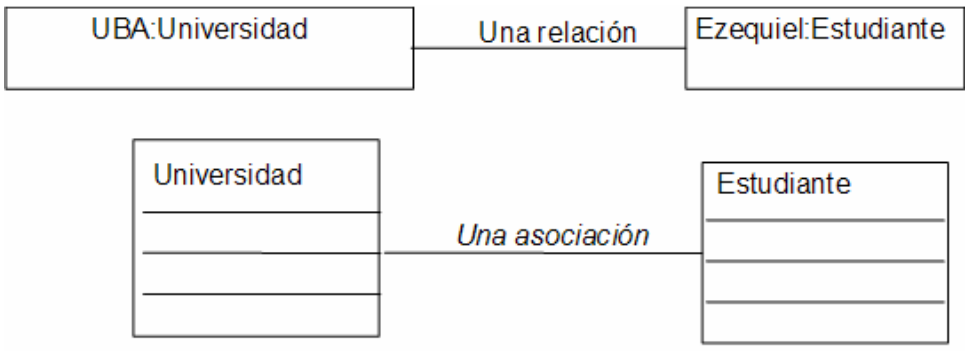
La relación entre objetos en realidad puede representarse como la relación entre las clases de las cuales derivaron los mismos.

**ASOCIACIÓN**

El primer tipo de relación entre clases es la asociación.

La asociación representa la relación bidireccional entre dos clases. Una asociación es una abstracción o generalización de la relación entre dos objetos.

Veamos un ejemplo de lo recién expresado utilizando UML.




La primera parte muestra la relación entre el objeto UBA que es una universidad con el objeto Ezequiel que es un Alumno. Se generaliza esta relación entre ambos objetos y se genera la asociación entre las clases Universidad y Estudiante.

Las asociaciones poseen una especificación de multiplicidad la cual indica cual es el grado de la asociación entre los objetos que la componen bidireccionalmente.

En este caso 1 Universidad puede poseer n alumnos distintos y a su vez 1 alumno puede estar cursando en n universidades distintas.

Tenga cuidado pues esto da la sensación de ser similar al grado de los diagramas de entidad relación pero conceptualmente es totalmente diferente ya que estamos hablando de la relación entre objetos o lo que es peor todavía entre clases. Por lo general se define la multiplicidad mínima y máxima para una clase involucrada en una asociación y la mínima no puede ser igual a 0 (cero), o sea debe ser >=1 debido a que se garantiza la existencia de las clases que componen la asociación.

Además de la asociación entre clases existen otras formas de relacionar las clases como la agregación.

**AGREGACIÓN**

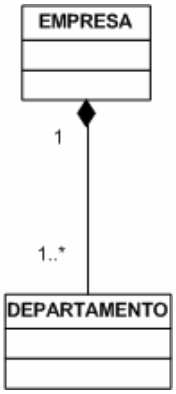
La agregación representa una relación de tipo “parte de” entre objetos.

Con respecto a la agregación existen como subniveles lo cuales se generan en función de la siguiente caracterización.

Si un objeto parte no puede comunicarse directamente con objetos externos distintos al objeto agregado se dice que la misma es una relación de agregación inclusiva. Ahora bien, si el objeto parte puede comunicarse directamente con objetos externos distintos al objeto agregado se dice que dicha relación de agregación no es inclusiva.

Además si la composición del objeto agregado puede cambiar se dice que la relación de agregación es dinámica. Si la composición del objeto agregado no puede cambiar se dice que la relación de agregación es estática.

Veremos como se representa la agregación entre clases en UML.




En este caso es el rombo negro es debido a que la agregación entre departamentos de una empresa y la empresa en cuestión es una agregación inclusiva debido a que no le esta permitido en esta solución a un departamento tener relación por fuera del objeto agregado, si esto no fuera así el rombo que representa la relación de agregación sería de color blanco.

Además dicha relación de agregación es dinámica ya que la empresa puede agregar o quitar departamentos en el momento que lo considere conveniente.

Observe además como se indica la multiplicidad de la relación de agregación donde se indica que un departamento o más de uno son parte de la empresa.

También existe la generalización entre clases la cual pasaremos a explicar.

**GENERALIZACIÓN**

La generalización permite gestionar la relación entre las mismas por medio de un ordenamiento jerárquico. Esto quiere decir que para generalizar lo que se realiza es la factorización de los atributos y métodos en común en clases más generales.

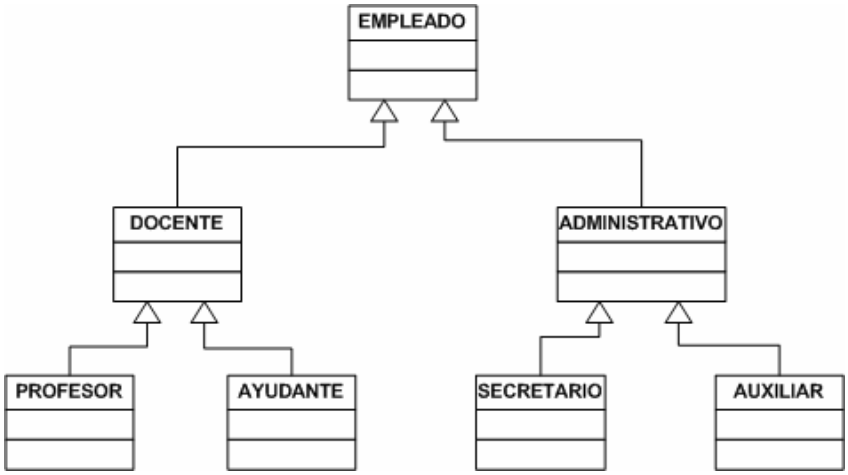
En general son términos usuales al hablar de generalización los términos de clase padre y clase hija o de superclase y subclase o clase base y clase derivada.

La generalización se puede obtener de dos maneras diferentes.

1. Por medio de la abstracción de la generalización lo que implica que de un determinado grupo de clases se factoriza los atributos y métodos comunes de las clases en cuestión agrupándolos en una superclase o clase padre.
2. Por medio de la especialización de una clase padre o genérica se derivan todas las subclases posibles al dominio de estudio del sistema en cuestión. Las subclases encontradas heredan los atributos, métodos y asociaciones de la clase padre o superclase.

Algo que es muy importante resaltar es que no importa por cual de los dos métodos se obtenga la generalización el resultado obtenido es el mismo y ambos métodos son transitivos.

Veamos un ejemplo de generalización realizado en UML para que le quede más claro.




De la generalización se desprende otro concepto o característica muy importante la cual se llama polimorfismo.

**POLIMORFISMO**

La idea de polimorfismo es que una característica de una clase puede tomar formas diferentes, pero seamos más específicos para que la idea quede clara.

El polimorfismo permite que dado un método o mensaje el mismo realice o desencadene la ejecución de operaciones diferentes. Para que esto pueda suceder deberemos tener una estructura jerárquica de clases. Con lo cual lo que estamos pidiendo concretamente es que cuando un mensaje de una superclase o clase padre es heredado por una subclase o clase hija esta última posea la posibilidad de modificar localmente (dentro de la subclase) de modificar las operaciones que realiza dicho mensaje o método.

Por ejemplo supongamos que tenemos una superclase cuenta bancaria la cual posee dos subclases cuenta corriente y caja de ahorro. Podríamos tener un método el cual le pedimos cual es el dinero disponible para gastar. Ahora cuando dicho método se instancia sobre cada una de las subclases en el caso de la caja de ahorro nos dirá que el monto disponible es nuestro saldo menos el tope mínimo que posee la caja de ahorro para no ser cerrada, pero si

instanciamos dicho método en la subclase cuenta corriente nos dirá que el monto disponible para gastar es el saldo de la cuenta corriente más el monto disponible para girar en descubierto.

Asociado al concepto de polimorfismo se encuentra el concepto de sobrecarga. La sobrecarga permite que dado un mensaje en particular según el entorno en el que se ejecuta el mismo permite devolver resultados distintos.

Fíjese que si no existiera el polimorfismo la sobrecarga sería imposible.

Un ejemplo típico de sobrecarga es el operador + dentro de los lenguajes de programación. Si los operandos son números el operador + devuelve la suma de los números sumados. Ahora bien, si los operandos son cadenas de caracteres el operador + concatena ambas cadenas en una cadena de caracteres única.

### Diferencias entre diagramas de clases y objetos

Vale la pena hacer una aclaración especial en este punto sobre los diagramas de clases y los diagramas de objetos.
A saber los diagramas de clases y los diagramas de objetos son dos vistas diferentes y complementarias del modelo.
Mientras que el diagrama de clase muestra la abstracción de una parte del dominio el diagrama de objetos representa una situación concreta del dominio. Esto sucede debido que las clases son abstracciones o generalizaciones de algún tipo de objetos mientras los objetos son representaciones o instancias puntuales de dichas clases. Si esto último no le queda claro vuelva a leer la parte de las relaciones de asociación.
Además las clases son abstractas como ya dijimos antes y no son instanciadas (las clases), en el momento que esto sucede se convierten en objetos, con lo cual los diagramas de clases nos permiten tener una compresión integral del modelo.
Hechas ya todas las aclaraciones y explicaciones necesarias para entender los conceptos básicos del mundo de objetos entremos en el tema que nos interesa.

## *Que es un proceso de desarrollo de software?*

El proceso de desarrollo de software define “quien” debe hacer “que” cosa debe hacer, “cuando” debe hacerla y “como” debe hacerlo.

Como se imaginará no existe un proceso de desarrollo de software universal. Según las características propias de cada proyecto exigen que el proceso sea configurable.

Para ello en el año 1998 los “tres amigos” (Grady Booch, James Rumbauch y Ivar Jacobson, que además son los creadores del UML) definieron un proceso de desarrollo el cual llamaron RUP (Rational Unified Process) que de forma generalizada se denomina proceso unificado el cual permite realizar Ingeniería de Negocio, Gestión de Requerimientos, Ingeniería de Datos, Diseño de interfaces, Pruebas Funcionales, Pruebas de desempeño, Gestión de Cambios y Configuración.

El RUP esta formado por un conjunto de disciplinas las cuales las podemos agrupar en dos grupos bien definidos. Disciplinas primarias y disciplinas de apoyo.

Las disciplinas primarias estan compuestas por el Modelado de Negocio, los Requerimientos, el Análisis y el Diseño, la Implementación, las Pruebas y el Despliegue.

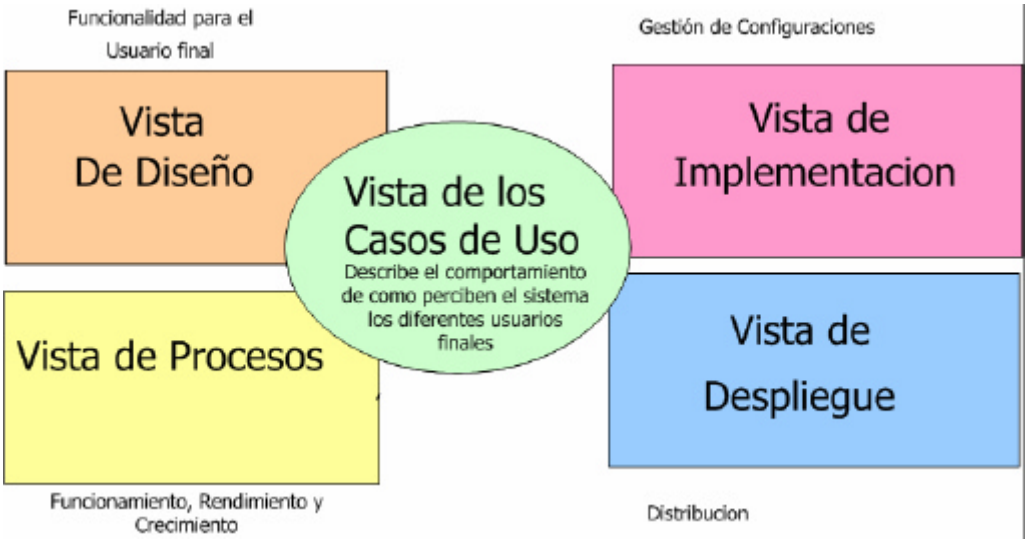
Las disciplinas de apoyo están compuestas por el Entorno, la Gestión del Proyecto y la Gestión de Configuración y Cambios.

El RUP es independiente del proceso, lo que implica que no esta ligado a ningún ciclo de vida de desarrollo de software en particular, sin embargo el RUP posee las siguientes características fundamentales:

- Es un proceso dirigido por Casos de Uso
- Es un proceso centrado en la Arquitectura
- Es un proceso iterativo e incremental

Expliquemos cada uno de estas características.

Dirigido por casos de uso es por que los casos de uso son un artefacto básico para establecer el comportamiento deseado del sistema, para verificar la arquitectura y la comunicación entre las personas involucradas en el proyecto.
Centrado en la arquitectura es por que la misma se utiliza como un artefacto básico para conceptualizar, gestionar, construir y evolucionar el sistema en desarrollo.
Para ello existe un diagrama que visualiza las diferentes vistas arquitectónicas 4+1 (Kruchten 1995)




Iterativo e incremental implica que el ciclo de vida iterativo garantiza la evolución de los prototipos ejecutables que se muestran a los usuarios. Esto implica que en cada iteración se vuelve a realizar todo el ciclo de vida en cascada pero a menor escala.
Además los objetivos de la iteración se establecen en función de la evaluación realizada en las iteraciones anteriores, más específicamente en la anterior a la actual. Esto garantiza un ida y vuelta entre el usuario y el equipo de desarrollo y disminuye la posibilidad de malos entendidos entre las partes.

Cada proceso iterativo esta compuesto de las siguientes etapas en el ciclo de vida en cascada son las siguientes: Análisis, Diseño, Codificación y Pruebas e Integración (estas son las etapas que se iteran n veces).
Cada iteración comprende la planificación de la iteración, esto es lo que se denomina el estudio de riesgo, luego se analizan los casos de uso para los diferentes escenarios. En función de esto se diseñan las opciones arquitectónicas.
Luego se realiza la codificación integrando en la misma el viejo código de las iteraciones anteriores con el nuevo código de la iteración actual en la etapa de construcción de la misma.
Cuando la codificación se encuentra finalizada se realiza la evaluación del ejecutable que será el prototipo a presentar para ver si satisface los criterios definidos para esta iteración.
Y por último se realiza la preparación de la entrega e instalación del prototipo con la documentación correspondiente.
El ciclo de vida esta formado por fases y cada fase esta compuesta por un número de iteraciones. Se define como fase al intervalo de tiempo entre dos hitos importantes del proceso cuando se cumplen un grupo de objetivos bien definidos, se terminan los artefactos y se decide si pasar o no a la próxima fase.
Definimos como Artefacto un resultado parcial o final producido y utilizado durante el proyecto. Los artefactos son las entradas y salidas de las actividades, son ejemplos de artefactos un documento, un modelo o un elemento de un modelo.
Las fases definidas en el ciclo de vida del RUP son las siguientes:

1. Inicio
2. Elaboración
3. Construcción
4. Transición

**Inicio:** Se definen los objetivos del proyecto y la funcionalidad y capacidades del producto.

**Elaboración:** Se estudian en profundidad la funcionalidad como el dominio del problema.

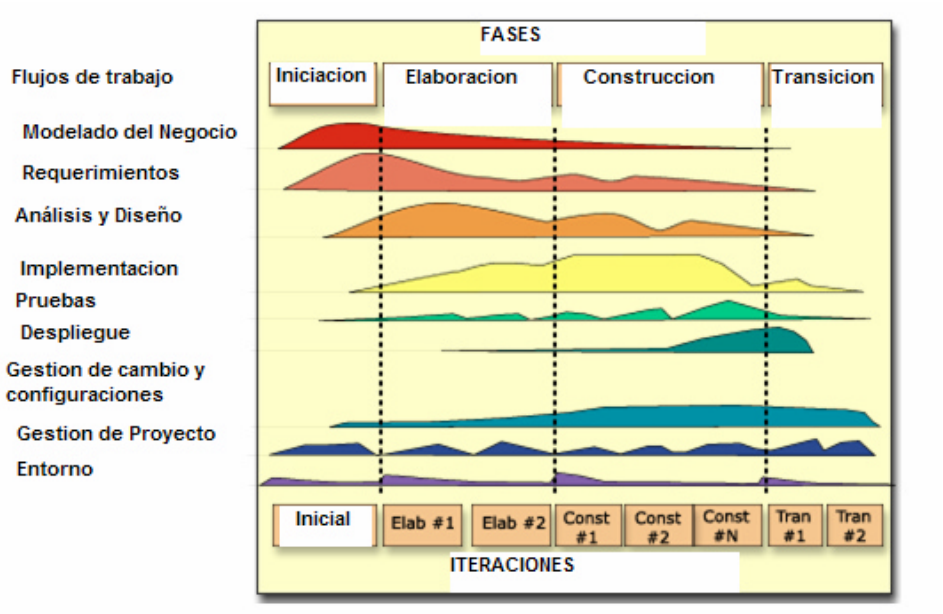
Además se define una arquitectura básica y se planifica el proyecto teniendo en cuenta los recursos disponibles.

**Construcción:** Se desarrolla el producto a través de iteraciones donde en cada iteración se realizan tareas de análisis, diseño e implementación.

En esta etapa se refina y profundiza sobre la arquitectura básica definida en la etapa anterior a medida que se va construyendo la misma. Esto permite realizar los cambios necesarios en la estructura. La mayoría del trabajo de dicha etapa es de programación y realización de pruebas, pero junto con estas tareas se realiza la documentación de las mismas.

**Transición:** En esta etapa se entrega el producto para que sea usado por el usuario real. Además se realizan las tareas de instalación, configuración, entrenamiento, soporte y mantenimiento. Se concluye con los manuales y se refina la información anterior.

Ahora que explicamos el ciclo de vida del desarrollo de software veremos un gráfico que representa la relación entre las diferentes fases del ciclo de vida con las distintas disciplinas del proyecto y muestra el esfuerzo respecto de las fases para cada iteración.




Como conclusión de todo lo expuesto es importante resaltar que para tener un buen desarrollo de los sistemas de información es necesario contar con tres pilares fundamentales. Una notación clara que evite las ambigüedades (nosotros utilizaremos UML como notación), un proceso bien definido (el proceso utilizado es el UP proceso unificado o mejor conocido como RUP) y una herramienta eficiente que potencie y utilice ágilmente el proceso de desarrollo seleccionado codificando con la notación elegida (dicha herramienta en la actualidad puede ser el Racional XDE o el Racional Rose que son las más difundidas , aunque existen herramientas provistas por otros desarrolladores).

# Análisis y Diseño Orientado a Objetos (A/DOO)

Programar es divertido, pero desarrollar software de calidad es difícil. El análisis y el diseño definen cómo solucionar el problema, qué programar y que este diseño de forma sea fácil de comunicar, revisar, implementar y evolucionar.
El Lenguaje Unificado de Modelado (UML) se ha convertido en el lenguaje aceptado universalmente para los planos del diseño software.
En la creación de sistemas complejos es importante la utilización de patrones de diseño que nos permitan describir fragmentos de diseño y reutilizar ideas de diseño.

## *Introducción al A/DOO*

- UML es una notación visual estándar. UML no es A/DOO o un método, es simplemente una notación.
- El A/DOO esta fuertemente relacionado con la actividad que es un requisito previo del análisis de requisitos, que incluye escribir **casos de uso**.
- El análisis de requisitos y el A/DOO requieren que se presenten en el contexto de algún proceso de desarrollo, para ello utilizaremos el **proceso de desarrollo iterativo** conocido como **Proceso Unificado**.

La construcción de software conlleva innumerables habilidades y pasos más allá del análisis de requisitos, el A/DOO y la programación orientada a objetos. Por ejemplo, la ingeniería de usabilidad y el diseño de interfaces de usuario son claves para el éxito, de igual modo que el diseño de base de datos.

## *¿Qué es análisis y diseño?*

El **Análisis** pone énfasis en una investigación del problema y los requisitos, en vez de ponerlo en una solución. (Estudio de los objetos del dominio)
El **Diseño** pone énfasis en una solución conceptual que satisface los requisitos, en vez de ponerlo en la implementación. (Ej.: Descripción del esquema de una base de datos y objetos software). Finalmente, los diseños pueden ser implementados (diseño de objetos o diseño de base de datos.)

## *¿Qué son el análisis y el diseño orientados a objetos?*

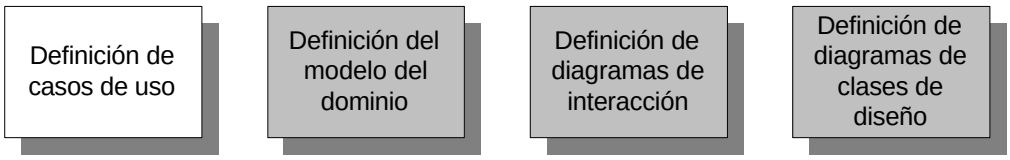
Durante el **análisis orientado a objetos**, se presta especial atención a encontrar y describir los objetos –o conceptos- en el dominio del problema. Por ejemplo, en el caso del sistema de información de la biblioteca, algunos de los conceptos son Libro, Biblioteca y Socio.
Durante el **diseño orientado a objetos**, se presta especial atención a la definición de los objetos software y en cómo colaboran para satisfacer los requisitos. Por ejemplo, en el sistema de la biblioteca, un objeto software Libro podría tener un atributo *Título* y un método *obtenerCapitulo*.
Por último, durante la implementación o programación orientada a objetos, los objetos de diseño se implementan como la clase *Libro*.

## *Ejemplo*

Un *juego de dados* en el que un jugador lanza dos dados. Si el total es siete, gana; en otro caso, pierde.

### Definición de los casos de uso

El análisis de requisitos podría incluir una descripción de los procesos del dominio relacionados, que podrían representarse como **casos de uso**.

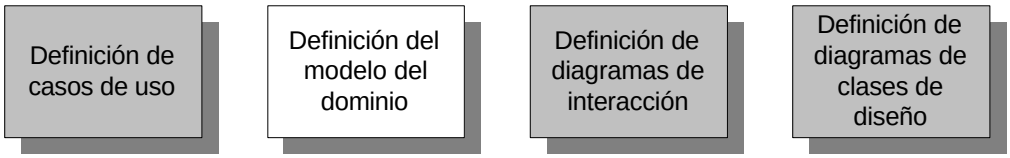



Los casos de uso no son artefactos orientados a objetos -son simplemente historias escritas-. Sin embargo, son una herramienta importante del Proceso Unificado. Por ejemplo, aquí está una visión breve del caso de uso Jugar una Partida de Dados.

**Jugar una partida de dados:** Un jugador recoge y lanza los dados. Si el valor de las caras de los dados suman siete, gana; en otro caso, pierde.

## Definición de un modelo del dominio

La finalidad del AOO es crear una descripción del dominio desde la perspectiva de la clasificación de objetos que conlleva una identificación de los conceptos, atributos y asociaciones que se consideran significativas. El resultado se puede expresar en un **modelo del dominio**, que se ilustra mediante un conjunto de diagramas que muestran los objetos o conceptos del dominio.




**Modelo del dominio parcial del juego de dados**


Este modelo ilustra los conceptos importantes *Jugador, Dado* y *JuegoDados*, con sus asociaciones y atributos. Un modelo del domino es una visualización de los conceptos en el dominio del mundo real.

### Definición de los diagramas de interacción

La finalidad del diseño orientado a objetos es definir los objetos software y sus colaboraciones. El **diagrama de interacción** muestra el flujo de mensajes entre los objetos software y, por tanto, la invocación de métodos.

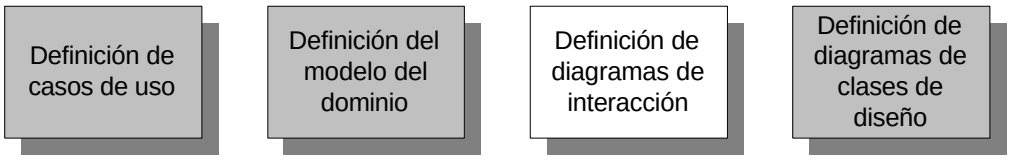
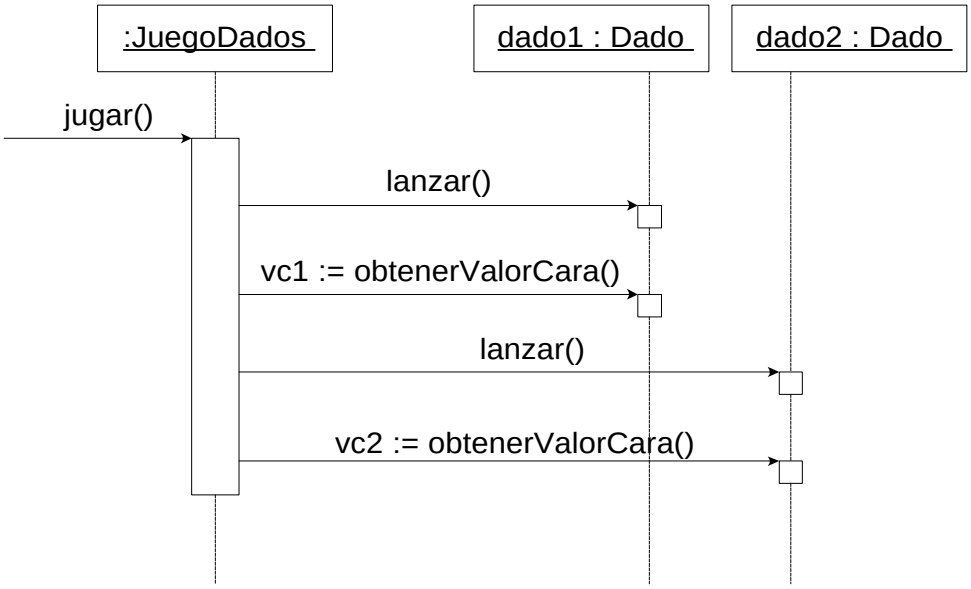
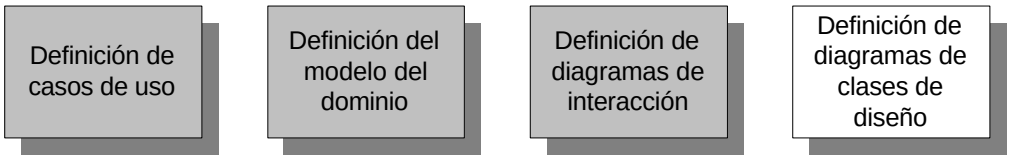



Diagrama de interacción que muestra los mensajes entre los objetos software

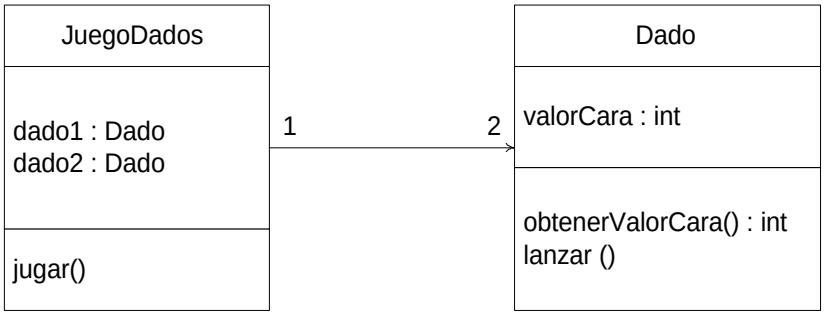



### Definición de los diagramas de clases de diseño

Además de la vista *dinámica* de las colaboraciones entre los objetos que se muestra mediante los diagramas de interacción, es útil crear una vista *estática* de las definiciones de las clases mediante un **diagrama de clases de diseño**.




Por ejemplo, en el juego de dados, un estudio del diagrama de interacción nos conduce al diagrama de clases de diseño parcial que se muestra en la siguiente Figura. Puesto que se envía el mensaje *jugar* al objeto *JuegoDados*, *JuegoDados* requiere un método *jugar*, mientras la clase *Dado* requiere los métodos *lanzar* y *obtenerValorCara*.




### *UML*

*El Lenguaje Unificado de Modelado (**UML**) es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas software, así como para el modelado del negocio y otros sistemas no software.*
UML se ha convertido en la notación visual estándar para el modelado orientado a objetos; se aplica principalmente durante el A/DOO, precedido, normalmente, por el análisis de requisitos.

## Desarrollo Iterativo y el Proceso Unificado

### *Introducción*

El desarrollo iterativo es un enfoque para el desarrollo de software. El Proceso Unificado es un ejemplo de proceso iterativo para proyectos que utilizan el A/DOO.
Un **proceso de desarrollo de software** describe un enfoque para la construcción, desarrollo y, posiblemente, mantenimiento de software.
El **Proceso Unificado de Rational** o **RUP** es un refinamiento detallado del Proceso Unificado.

## *La idea más importante del UP: desarrollo iterativo*

El UP fomenta el **desarrollo iterativo**. En este enfoque, el desarrollo se organiza en una serie de mini-proyectos cortos, de duración fija llamados **iteraciones**; el resultado de cada uno es un sistema que puede ser probado, integrado y ejecutado. Cada iteración incluye sus propias actividades de análisis de requisitos, diseño, implementación y pruebas.

El ciclo de vida iterativo se basa en la ampliación y refinamiento sucesivos del sistema mediante múltiples iteraciones, con retroalimentación cíclica y adaptación como elementos principales que dirigen para converger hacia un sistema adecuado. El sistema crece incrementalmente a lo largo del tiempo, iteración tras iteración, y por ello, este enfoque también se conoce como **desarrollo iterativo e incremental**.

El resultado de cada iteración es un sistema ejecutable, pero incompleto. La salida de una iteración *no* es un prototipo experimental sino un subconjunto con calidad de producción del sistema final.

En general, cada iteración aborda nuevos requisitos y amplía el sistema incrementalmente.

### Beneficios del desarrollo iterativo

La retroalimentación iterativa y la adaptación nos conducen hacia el sistema deseado. La inestabilidad de los requisitos y el diseño disminuyen a lo largo del tiempo.

### Longitud de una iteración y fijación de la duración

El **UP** recomienda que la longitud de una iteración sea de dos a seis semanas. Una idea clave es que se **fija la duración** de las iteraciones.

## *Conceptos del UP*

La idea fundamental para apreciar y utilizar el **UP** es el desarrollo iterativo, fijando iteraciones cortas, y adaptable. Otra idea del UP implícita, es el uso de las tecnologías de objetos, entre las que se encuentra el A/DOO y la programación orientada a objetos.

Algunos conceptos claves son:

- Abordar cuestiones de alto riesgo y muy valiosas en las primeras iteraciones.
- Involucrar continuamente a los usuarios para evaluación, retroalimentación y requisitos.
- Construir en las primeras iteraciones una arquitectura que constituya un núcleo central consistente.
- Verificar la calidad continuamente; pruebas muy pronto, con frecuencia y de manera realista.
- Aplicar casos de uso.
- Modelar software visualmente (con UML).
- Gestionar los requisitos con cuidado.
- Manejar peticiones de cambio y gestión de configuraciones.

## *Las fases del UP*

1. **Inicio:** fase de viabilidad, donde se lleva a cabo sólo el estudio suficiente para decidir si continuar o no. Visión aproximada, análisis del negocio, alcance, estimaciones imprecisas.
2. **Elaboración:** fase donde se implementa, de manera iterativa, la arquitectura que constituye el núcleo central y se mitigan las cuestiones de alto riesgo. Visión refinada, implementación iterativa del núcleo central de la arquitectura, resolución de los riesgos altos, identificación de más requisitos y alcance, estimaciones más realistas.

3. **Construcción:** implementación iterativa del resto de requisitos de menor riesgo y elementos más fáciles, preparación para el despliegue.
4. **Transición:** pruebas beta, despliegue.

### *Las disciplinas del UP*

El UP describe actividades de trabajo, como escribir casos de uso, en **disciplinas**. Una disciplina es un conjunto de actividades (y artefactos relacionados) en un área determinada. En el **UP**, un **artefacto** es el término general para cualquier producto del trabajo.
Disciplinas en el UP:

- **Modelado del Negocio**. Esto incluye el modelado de los objetos del dominio.
- **Requisitos**. Análisis de los requisitos para una aplicación, como escritura de casos de uso e identificación de requisitos no funcionales.
- **Diseño**. Todos los aspectos de diseño, incluyendo la arquitectura global, objetos, bases de datos, red y cosas parecidas.

## Inicio

La fase de inicio consiste en vislumbrar el alcance del producto, visión y análisis del negocio.
El propósito de la fase de inicio es establecer una visión común inicial de los objetivos del proyecto, determinar si es viable y decidir si merece la pena llevar a cabo algunas investigaciones serias en la fase de elaboración.

### *¿Qué artefactos podrían crearse en la fase de inicio?*

| *Artefacto* | *Comentario* |
|---|---|
| Visión y Análisis del Negocio | Describe los objetivos y las restricciones de alto nivel, el análisis del negocio y proporciona un informe para la toma de decisiones. |
| Modelo de Casos de Uso | Describe los requisitos funcionales y los no funcionales relacionados. |
| Especificación Complementaria | Describe otros requisitos. |
| Glosario | Terminología clave del dominio |
| Lista de Riesgos & Plan de Gestión del Riesgo | Describe los riesgos del negocio, técnicos, recursos, planificación, y las ideas para mitigarlos o darles respuesta. |
| Prototipos y pruebas-de-conceptos | Para clarificar la visión y validar las ideas técnicas. |
| Plan de Iteración | Describe qué hacer en la primera iteración de la elaboración. |

## Comprensión de los requisitos

Los **requisitos** son capacidades y condiciones con las cuales debe ser conforme el sistema. El primer reto del trabajo de los requisitos es encontrar, comunicar y recordar (registrar) lo que se necesita realmente, de manera que tenga un significado claro para el cliente y los miembros del equipo de desarrollo.

La respuesta iterativa es utilizar un proceso que acepte el cambio y la retroalimentación como motores centrales en el descubrimiento de los requisitos.

### *Tipos de requisitos*

- **Funcional (Functional):** características, capacidades y seguridad.
- **Facilidad de uso (Usability):** factores humanos, ayuda, documentación.
- **Fiabilidad (Reliability):** frecuencia de fallos, capacidad de recuperación de un fallo y grado de previsión.
- **Rendimiento (Performance):** tiempos de respuesta, productividad, precisión, disponibilidad, uso de los recursos.

- **Soporte (Supportability):** adaptabilidad, facilidad de mantenimiento, internacionalización, configurabilidad.

Requisitos adicionales:

- **Implementación:** limitación de recursos, lenguajes y herramientas, hardware.
- **Interfaz:** restricciones impuestas para la interacción con sistemas externos.
- **Operaciones:** gestión en su puesta en marcha.
- **Empaquetamiento**
- **Legales:** licencias, etc.

Lo normal es dividir los requisitos en **funcionales** (comportamiento) y **no funcionales** (todos los demás).
El artefacto Visión resume los requisitos de alto nivel. El Glosario agrupa y clarifica los términos que se utilizan en los requisitos (**diccionario de datos**).
Los prototipos son un mecanismo para clarificar qué es lo que se quiere o es posible.

## Modelo de Casos de Uso: Escritura de Requisitos en Contexto

### *Introducción*

Los casos de uso son un mecanismo utilizado para descubrir y registrar los requisitos (especialmente los funcionales).
El UP define el Modelo de Casos de Uso en la disciplina Requisitos.
Básicamente, es el conjunto de todos los casos de uso; es un modelo de la funcionalidad y entorno del sistema.

### *Objetivos*

Los clientes y los usuarios finales tienen objetivos (también conocidos como *necesidades*) y quieren sistemas informáticos que les ayuden a conseguirlos.

Hay varias formas de capturar estos objetivos y requisitos del sistema; las mejores son simples y familiares, porque esto hace que sea más fácil - especialmente para clientes y usuarios finales – contribuir a su definición o evaluación. Eso reduce el riesgo de perder el hilo.
Los casos de uso son un mecanismo para ayudar a mantenerlo simple y entendible para todo el personal involucrado. De manera informal, son historias del uso de un sistema para alcanzar los objetivos.

## *Casos de uso*

En primer lugar, algunas definiciones informales: un **actor** es algo con comportamiento, como una persona (identificada por un rol), sistema informatizado u organización; por ejemplo, un cajero.
Un **escenario** es una secuencia específica de acciones e interacciones entre los actores y el sistema objeto de estudio; también se denomina **instancia de caso de uso**. Es una historia particular del uso de un sistema; por ejemplo, el escenario de éxito de compra de artículos con pago en efectivo, o el escenario de fallo al comprar debido al rechazo de la transacción de pago con la tarjeta de crédito. Informalmente entonces, un **caso de uso** es una colección de escenarios con éxito y fallo relacionados, que describe a los actores utilizando un sistema para satisfacer un objetivo.
Una actitud clave en el trabajo con casos de uso es centrarse en la pregunta: "¿Cómo puedo, utilizando el sistema, proporcionar un valor observable al usuario, o cumplir sus objetivos?"

## *Casos de uso y requisitos funcionales*

Los casos de uso son requisitos funcionales que indican qué hará el sistema; no describen el funcionamiento interno del sistema, sino que se describe el sistema en base a las *responsabilidades* que tiene.
A través de la definición de las responsabilidades del sistema con casos de uso, es posible especificar *qué* debe hacer el sistema (los requisitos funcionales) sin

decidir *cómo* lo hará (el diseño). De hecho, la definición de “análisis” frente al “diseño” se resume como el “qué” frente al “cómo”.

### Tipos de formalidad

Los casos de uso se escriben como formatos diferentes, dependiendo de la necesidad:

- **Breve**: resumen conciso de un párrafo, normalmente del escenario principal con éxito.
- **Informal**: formato de párrafo en un estilo informal. Múltiples párrafos que comprenden varios escenarios.
- **Completo**: el más elaborado. Se escriben con detalle todos los pasos y variaciones, y hay secciones de apoyo como precondiciones y garantías de éxito. (Ver www.usecases.org)

### *Ejemplo de este estilo de plantilla:*

Caso de uso…. : …….

**Actor principal:** …….

**Personal involucrado e intereses:**

- ……
- ……

**Precondiciones:**

**Garantías de éxito (Postcondiciones):**

**Escenario principal de éxito (o Flujo Básico):**

**Flujos Alternativos:**

**Extensiones:**

**Requisitos especiales:**

- …
- …

**Lista de tecnología y variaciones de datos**

- …
- …

**Frecuencia:**

**Temas abiertos:**

### La variación de dos columnas

Algunos prefieren el formato en dos columnas o conversacional, que destaca el hecho de que se establece una interacción entre los actores y el sistema.

Caso de uso…..: ……

**Actor principal:**

**Escenario principal de éxito:**

| Acción del actor (o intención) | Responsabilidad del Sistema |
|---|---|
| 1. ……. | |
| 2. ……. | |
| 3. ……. | |
| | 4. ……. |
| | 5. ……. |
| 6. ……. | |
| 7. ……. | |
| | 8. …….. |
| | 9. …….. |

## *Explicación de las secciones*

**Actor principal:** El actor principal que recurre a los servicios del sistema para cumplir un objetivo.

### Personal involucrado y lista de intereses

Esta lista sugiere y delimita qué es lo que debe hacer el sistema. Citando a Cockburn: "*El [sistema] funciona siguiendo un contrato entre el personal involucrado, donde los casos de uso detallan la parte de comportamiento del contrato(…) El caso de uso, como contrato de comportamiento, captura* ***todo*** *y* ***sólo*** *el comportamiento relacionado con la satisfacción de los intereses del personal involucrado.*"

¿Qué debería estar en un caso de uso?

Lo que satisfaga los intereses de todo el personal involucrado.

**Personal involucrado e intereses:**

- Cajero: quiere entradas precisas, rápidas, y sin errores de pago, ya que las pérdidas se deducen de su salario.
- Vendedor: quiere que las comisiones de las ventas estén actualizadas.
- ……..

### Precondiciones y garantías de éxito (postcondiciones)

Las **precondiciones** establecen lo que *siempre debe* cumplirse antes de comenzar un escenario en el caso de comenzar un escenario en el caso de uso. Las precondiciones *no* se prueban en el caso de uso, sino que son condiciones que se asumen que son verdad.
Las **garantías de éxito** (o **postcondiciones**) establecen qué debe cumplirse cuando el caso de uso se completa con éxito.

**Precondiciones:** El cajero se identifica y autentica.
**Garantías de éxito (Postcondiciones):** Se registra la venta. El impuesto se calcula de manera correcta. Se actualizan la Contabilidad y el Inventario. Se registran las comisiones. Se genera el recibo.

### Escenario principal de éxito y pasos (o Flujo Básico)

Describe el camino de éxito típico que satisface los intereses del personal involucrado.

Escenario principal de éxito (o Flujo Básico):

1. El Cliente llega a una terminal con mercancías para comprar.
2. El Cajero comienza una nueva venta.
3. El Cajero introduce el identificador del artículo.
4. ……
El Cajero repite los pasos 3-4 hasta que se indique.
5. ……

### Extensiones (o Flujos Alternativos)

Indican todos los otros escenarios o bifurcaciones, tanto de éxito como de fracaso. En una extensión se etiqueta con el nº de paso del escenario principal de éxito, primero identifica la condición y después la respuesta.

**Extensiones (o Flujos Alternativos):**

3 a. Identificado no válido:

1. El Sistema señala el error y rechaza la entrada…

## Bibliografía

UML y Patrones – Una introducción al análisis y diseño orientado a objetos y al proceso unificado –, Craig Larman