**Ingeniería de software 2 Práctico**

Alumno: Santiago Vietto

Docente: Gerardo Fernando Córdoba

DNI: 42654882

Institución: UCC

Año: 2022

**Modelado y diagrama de casos de uso**

**Introducción**

\_ La materia puntualmente se basa en los requerimientos, entonces como base principal los diagramas de caso de uso, ya sean utilizados o no en empresas, son la base que vamos a tener para poder encontrar los requerimientos. Tratamos dominios o contextos complejos como por ejemplo un sistema relacionado a la pandemia, para tratar de generar una mentalidad de modelado. Cualquiera sea el área en la que nos especialicemos (base de datos, gestión de proyectos, programación, etc), siempre nos vamos a encontrar con requerimientos, por eso es importante poder saber identificarlos, poder saber describirlos, porque es gran parte de la base de la ingeniería de software.

\_ Es importante la cuestión de los requerimientos ya que hoy la tendencia es que todo sea más ágil, es decir, encontrar los requerimientos de forma más ágil, poder desarrollar de forma más ágil y poder probar el proyecto rápido para ver si funciona o no y en base a eso aplicar los métodos que tenemos en mente.

**Modelo**

\_ Un modelo es básicamente una representación en un cierto medio de algo. A continuación vemos algunas características:

* Lo más importante es que el modelo puede ser visual, es decir, ayuda a visualizar el sistema en el rubro de la ingeniería nos vamos a encontrar con un cliente o stakeholder que nos va a comentar un problema que tenga y nosotros deberíamos tratar de darle una solución, y la idea del modelo es que pueda resolver la ambigüedad del lenguaje natural que tenemos en algo un poco más visual.
* El modelo es mucho más fácil de entender, como futuros ingenieros debemos transmitir ideas a través de un modelo indistintamente si se sigue algún patron, si se usa UML o no, o algún otro lenguaje, ya que lo importante es que se pueda transmitir algo a partir de un dibujo sin importar como sea. Tratamos de captar lo importante, el modelo no busca tener todo el detalle, ya que en caso de tener todo en detalle iría en contra de lo que dijimos de que si hacemos un modelo demasiado grande se pierde el concepto de lo importante. Definimos un punto de vista, cada uno tiene una forma de pensar diferente, hay modelos que tienen diferentes perspectivas, por ejemplo un diagrama de secuencia es un punto de vista, el diagrama de casos de uso es otro punto de vista. Pero la importancia de todos los modelos es simplificar, es decir, simplificamos la realidad.
* Lo más importante de los modelos es que buscan reducir y atacar la complejidad inherente en los sistemas. Si hacemos un modelo más complejo que la forma en la que nos lo están contando, claramente vamos en el camino equivocado.

¿Por qué modelamos?

\_ La idea es tener conocimiento del dominio, captar requisitos, diseñar, captar decisiones, usabilidad, explorar soluciones, organizar el desarrollo, definir estructura, definir comportamiento, etc. Pero analizamos lo siguiente:



\_ Con abstraer hablamos de sacar lo que no es importante, es decir, nos abstraernos un poco del contexto sin dar tanto detalle, luego con especificar hace referencia a la comunicación de lo que estamos haciendo y ser bastante concreto con lo que queremos en el modelo, y después describir el mismo. Tenemos que ser directos cuando representamos un modelo.

Herramientas para modelar

\_ La idea es usar UML (Lenguaje Unificado de Modelado), este lenguaje tiene muchos tipos de modelos, diagramas y demás, pero puntualmente vamos a ver el diagrama de casos de uso. UML es uno de los lenguajes para modelar ya que hay varios. Lo importante es tener la mentalidad de modelar, indistintamente de la herramienta, donde lo que podamos dibujar tenga sentido con lo que estamos tratando de resolver. De este lenguaje tenemos en cuenta la semántica, notación y principios.

\_ Algunas características de UML:

* Visualizar, especificar, construir y documentar sistemas orientados a objetos.
* Bloques de construcción: elementos, bloques, diagramas.
* Representación gráfica: se utilizan grafos con nodos y arcos.
* Diferentes diagramas, diferentes aspectos del sistema. Dentro de los tipos de modelos que hay, vamos a encontrar dos grandes divisiones
* Diagramas estructurales: representa la parte estática con sus elementos y sus relaciones. En esto modelos el tiempo no entra en juego, es todo más estático, es decir, de la estructura de lo que estamos intentando representar. Un ejemplo de estos es el diagrama de casos de uso.
* Diagramas Comportamiento: representa la parte dinámica. En estos el tiempo entra el juego, y un ejemplo sería el diagrama de secuencia.

\_ Info extra: <https://www.youtube.com/watch?v=zid-MVo7M-E>

**Modelado de requerimientos funcionales**

Diagrama de casos de uso

\_ Lo más importante que tenemos que saber es que los diagramas de casos de uso representan requerimientos funcionales (no representan ninguna otra cosa que no sean requerimientos). Estos diagramas pueden ser a nivel de sistema o de negocio. Está dirigido al proceso de desarrollo, es decir, cada caso de uso representa un requerimiento en sí. El diagrama trata de identificar las siguientes cuatro sencillas cosas pero a su vez problemáticas:

Actores: son quiénes interactúan con el sistema, ya sea de forma directa o indirecta es decir, son todos aquellos que dan algún estimulo o impulso al sistema y desencadenan o disparan algún requerimiento. No necesariamente un actor tiene que estar relacionado con una persona física, ya que puede ser otro sistema, algún componente de hardware, etc.

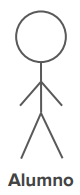
Casos de uso: representan la funcionalidad o parte de la funcionalidad del sistema.

Relaciones entre actores y casos de uso: la relacion, entre los que impulsan los requerimientos o funcionalidad con los cosas de uso, determina funcionalidades que pueden realizar.

Relaciones entre casos de uso y casos de uso: en este caso tenemos que un requerimiento discuta otro requerimiento, para expandir las funcionalidades del sistema.

Identificación de actores

\_ Como venimos diciendo, el actor tiene un rol dentro de nuestro sistema, este interactúa directamente con el sistema y ejecuta las funcionalidades. Hablando un poco más de la sintaxis, el nombre del actor debe describir el rol que desempeña, y en nuestro diagrama se representan de la siguiente manera:



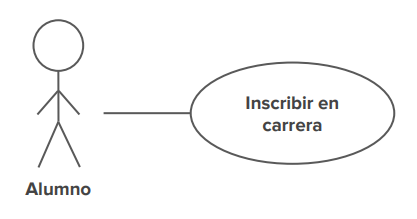
\_ Existen dos tipos de actores:

* Actores principales: son los que directamente se relacionan son el sistema.
* Actores secundarios: hay algunas funcionalidades que requieren más de un actor, por ende los actores secundarios ayudan al principal a complementar esas funcionalidades. Por ejemplo si estamos haciendo una transacción bancaria, nosotros seriamos el actor principal y el actor secundario seria algún sistema que complemente la funcionalidad para que todo el modelo del sistema funcione.

\_ Tomamos como ejemplo la inscripción a la facultad. En este caso uno de los actores podría ser el alumno. Si pensamos en Instagram los actores serían los usuarios de ocio que a su vez son dos, los usuarios registrados y no registrados, y ademas tenemos los usuarios de Marketplace.

Identificación de casos de uso

\_ Básicamente los casos de uso describen la funcionalidad y los requerimientos del sistema. En el diagrama de casos de uso se describen requerimientos funcionales, y los no funcionales no son parte de este modelado. Los casos de uso deben ser descriptos con una acción en infinitivo al principio, se trabaja con un verbo. Deben ser equilibrados entre el tamaño y su descripción, por ejemplo para este caso tranquilamente podríamos decir que una funcionalidad del sistema de inscripción de la universidad sea inscribirse a una carrera, ahora el hecho de inscribirse a una carrera implica muchas otras cosas que se tienen que cumplir en el sistema tanto en el mundo digital como el físico, para que eso se pueda dar, es decir, cuando hablamos de equilibrio en términos de tamaño hacemos referencia a todo lo que hace el requerimiento. Ahora tampoco tenemos que ir al extremo de ser muy minimalista (por ejemplo, ingresar texto) o muy abarcativos. Un requerimiento o funcionalidad se puede descomponer y luego relacionar para formar un todo.

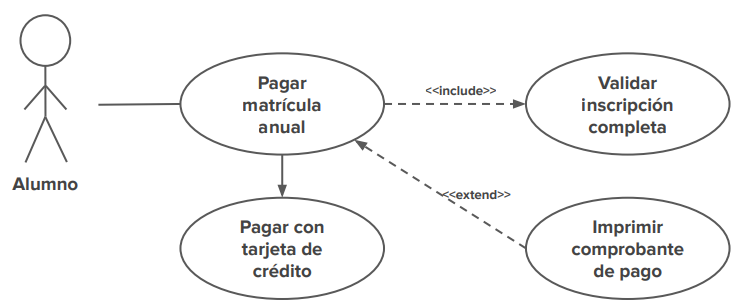


\_ Dicho esto, volviendo al caso de Instagram, tenemos algunos casos de usuario como seguir usuario, registrase, comentar, publicar, buscar usuario, etc.

Identificación de relaciones

\_ Entre el actor y el caso de uso, la relacion es directa. Pero después tenemos tres tipos de relaciones:

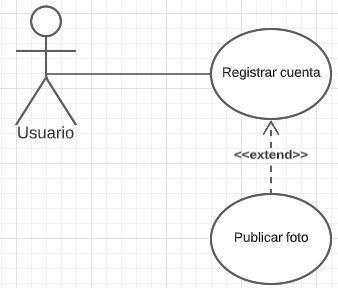
* Include: estas son relaciones entre casos de uso e indican que si se da un caso de uso, entonces el otro también se tiene que dar. Debe ejecutarse siempre y ayudan a complementar la función de quién lo llama. Siempre se ejecutan los dos casos de uso.
* Extend: esta relacion entre casos de uso indica que si se da un caso de uso, se puede o no dar el otro. Es como un condicional que define si se da o no. Agrega funcionalidad a otro caso de uso, que puede o no ejecutarse siempre, es optativo. Esta es una decisión muy del actor.
* Generalización: permite especificar comportamiento partiendo de la funcionalidad base. Podríamos tener un caso de uso bastante más abstracto o genérico, y alguna generalización o especificación que nos dé más contexto del mismo.



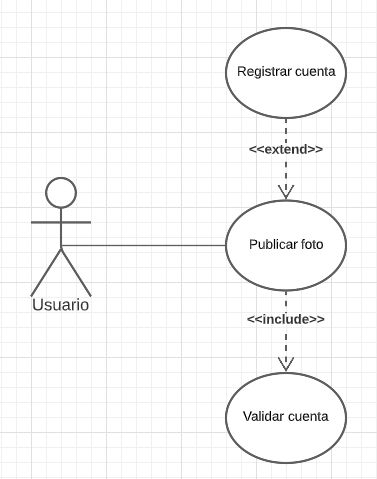
\_ En el include de este ejemplo tenemos que el alumno al ejecutar el caso de uso pagar matricula anual, antes de poder pagar la matricula el sistema debería primero validar si la inscripción esta completa, entonces solo si cumplió este requerimiento, recién ahí va a poder pagar la matricula. Siempre se van a ejecutar los dos. En el caso del extend, por ejemplo si pagamos la matricula anual, si queremos podemos imprimir el comprobante de pago, ya que no es obligatorio pero esta la opción. En el caso de la generalización, tenemos que podríamos pagar la matricula anual pero no aclaramos como, pero sabemos que puede haber algo más específico y que sea pagar la matricula anual con tarjeta de crédito por ejemplo.

\_ Puede haber más de una relacion y pueden combinarse entre sí pero siempre y cuando el modelo no quede muy encadenado y sea fácil de leer y entender.

\_ Volviendo al caso de Instagram, si casi todas las funcionalidades o el 70% de las funcionalidades solicitan un registro, nos preguntamos si sería un buen modelo que un solo caso de uso tenga 70 extends a 70 funcionalidades, no es que este mal pero el modelo queda raro.

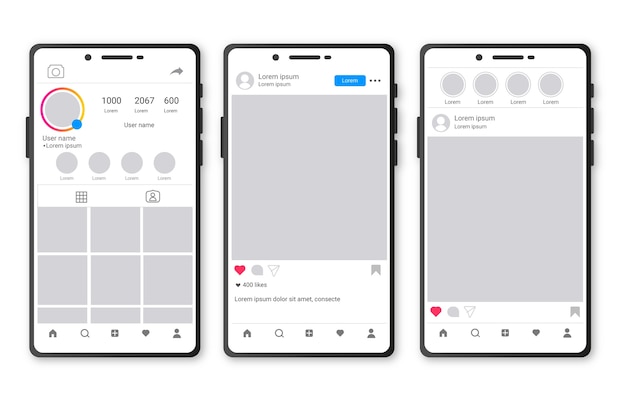


\_ En este caso los requerimientos pueden tener precondiciones, como en el caso de registrarse, en donde se da por hecho que los requerimientos que puede hacer el actor se dan ya con un registro previo. Entonces por ejemplo, como usuario podemos tener el caso de uso publicar foto, ahora para publicar una foto tenemos que tener una cuenta por ende agregamos un extend, en el caso de estar registrado podemos publicar, caso contrario no podemos salvo que nos registremos. Podemos tener otro caso de uso que sea validar cuenta que va a ser un include, en donde si queremos publicar foto siempre validamos la cuenta, si la cuenta esta creada esto se ejecuta y caso contrario habrá que registrarse.

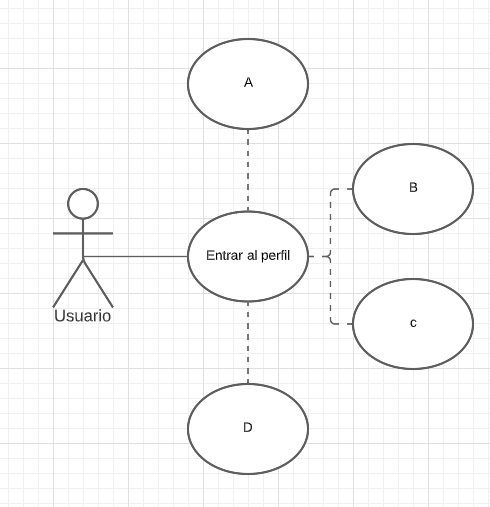


Errores comunes

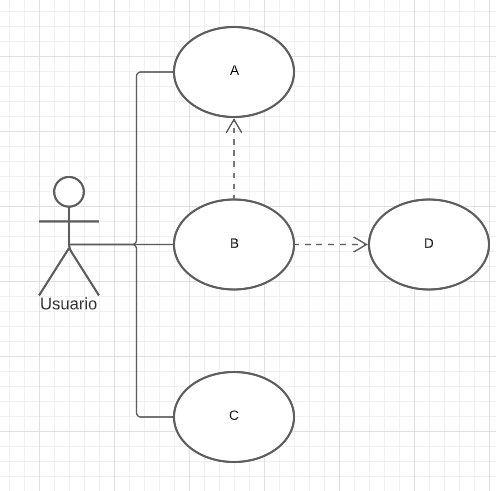
\_ Tenemos que tratar de no relacionar el modelo de casos de uso, directamente con la pantalla o la usabilidad que tiene el sistema, porque no son comunes. Acá estamos modelando requerimientos con sus relaciones y no el comportamiento de una interfaz o frontend como vemos en la siguiente imagen:



\_ Tenemos que buscar funcionalidades, no siempre una pantalla tiene todas las funcionalidades del sistema. Si tenemos que describir la interfaz del perfil de usuario como vemos en la imagen, de todos los requerimientos que esta sección tiene, generalmente tendemos a extender de una funcionalidad todas las otras funcionalidades y esto es lo que no debemos hacer, como por ejemplo entrar al perfil y de ahí hacer todas las funcionalidades:



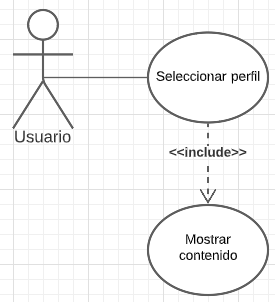
\_ Pero en realidad por ejemplo podríamos hacer cualquiera de las funcionalidades de manera independiente y separadas, y esto sería lo más adecuado, en donde también pueden relacionarse entre sí:



Ejemplo 1: selección de usuario para ingresar a la pantalla inicial en Netflix:

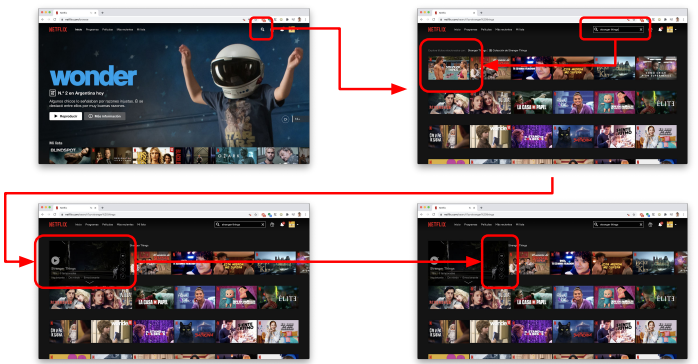


\_ Nosotros no podemos ver contenido de Netflix gratis, ya que debemos estar registrados, pero de los usuarios registrados tenemos los subscritos que puede hacer todo, y los no subscritos que pueden tener algunas funcionalidades. Entonces determinamos como actor un usuario que engloba un usuario registrado y subscripto o no, luego tenemos nuestro primer caso de uso que es seleccionar perfil, en donde una vez seleccionado el perfil, el sistema nos muestra contenido de acuerdo al mismo y para esto usamos una relacion de include ya que no podemos acceder a un contenido sin seleccionar un perfil.

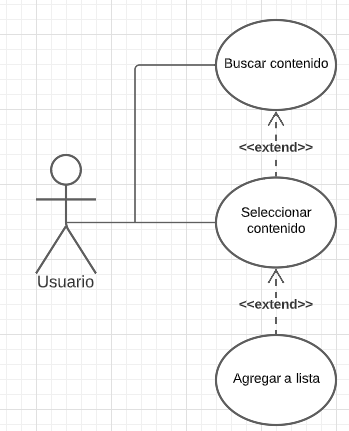


\_ Otra forma de verlo es, si nos logueamos y tenemos solo un perfil, esto se valida y directamente mostrará el contenido.

Ejemplo 2: buscar contenido para agregar a “mi lista”



\_ Arrancamos con determinar el actor que es el usuario registrado. Ahora respecto a los casos de uso tenemos uno que será buscar contenido y luego seleccionar el mismo contenido, o directamente seleccionar contenido en el caso de encontrarlo de una, estas dos se relacionan por medio de un extend. Entonces para este modelo se cumple el caso de buscar y encontrar de una, para el caso de buscar y no encontrar el contenido que queremos también se cumple ya que mostrara opciones que se relacionan. El seleccionar contenido, representaría el tercer paso en donde de nuestra lista encontramos por ejemplo una película y la seleccionamos para verla, quitarla, etc. Luego sacamos un extend de agregar a lista hacia seleccionar contenido.



\_ Este modelo cumple para el caso de representar por ejemplo el escenario donde buscamos una película y nos olvidamos de que ya la teníamos en la lista de favoritos y no la queremos agregar, ya que el hecho de agregar es opcional.

**Descripción de casos de uso utilizando trazo grueso y trazo fino**

**Trazos finos**

\_ Un trazo fino es una descripción detallada de la funcionalidad de uno o varios casos de uso. Recordamos que en el diagrama de casos de uso no hay descripciones de que es lo que hace el mismo, hay una aproximación de lo que es el caso de uso y cuál es el requerimiento o funcionalidad pero no damos el detalle de todo lo que pasa dentro de ese requerimiento, por ende con el trazo fino lo que logramos tener esa descripción en detalle. Como características tenemos que:

* Brindan un contexto: complementan a un caso de uso brindando un contexto de lo que hace el mismo. Leyendo el caso de uso por sí solo, puede que le falte contexto para entender cuál es el objetivo que persigue, cuáles son las pre condiciones que se tienen que dar para que se ejecute, como es la salida de ese caso de uso, entonces esto le da como un marco o contexto agregando muchos más detalles mediante el lenguaje natural para describir esa funcionalidad.
* Define las pre condiciones para que un caso de uso pueda darse. De los ejercicios que vimos antes un ejemplo de pre condición puede ser en el caso del supermercado (ejercicio 2) en el que para cambiar la forma de envio una pre condición puede ser que ya se armó el carrito de compra. Otro ejemplo puede ser en el sistema de peaje (ejercicio 1) donde una pre condición para llamar a la policía puede ser que la patente este dentro del domino de los autos buscados. Entonces una pre condición es algo que se tiene que dar para que el requerimiento se ejecute.
* Define las acciones que realiza un actor y el sistema como un flujo continuo de pasos. El trazo fino lo que describe es paso a paso lo que va a hacer su funcionalidad, donde acá si ya entra en detalle como es el comportamiento del sistema, cosa que no veíamos en el diagrama. Sería como contar una historia de que hace el actor y que devuelve el sistema, también puede ser entre sistemas, y se cuenta el paso a paso de cómo se tienen que ir dando las cosas. Volviendo al ejemplo del supermercado tenemos el caso en el que el actor selecciona un producto, luego el sistema en base al producto seleccionado busca el precio del mismo, el sistema multiplica la cantidad del producto seleccionado por el precio, y después el sistema suma ese monto al monto total de la compra.
* Describen el camino feliz y los flujos alternativos de la interacción actor-sistema. El camino feliz es el camino en el que se dan todas las condiciones para que el sistema siga tal cual. Y los flujos alternativos son aquellos lugares donde puede haber un punto de falla ante alguna de las acciones que haga que se abra la funcionalidad. Acá es donde entra lo de conexiones o relaciones de extensión, porque tenemos el camino que se sigue si todo sale bien y el camino que no. En el ejercicio del supermercado, teníamos una pre condición que era que si ingresamos una dirección que este por fuera del lugar habilitado, entonces no podría haber entrega a domicilio y de ahí pasaba algo, donde a ese algo que pasa le podemos definir una condición donde si esta todo bien avanza y si da algún error o camino alternativo, podemos describir que va a hacer el sistema.
* Define las post condiciones que un casos de uso realiza al finalizar. Básicamente hacen referencia a como queda el sistema luego de ejecutarse todas sus funcionalidades. Si volvemos al ejemplo del supermercado tenemos algunas post condiciones como que se actualice la orden de compra, puede cerrarse la orden de trabajo, en el ejercicio del peaje podemos tener que la pasada en el mismo termino y se cobró tanto en base a la hora. Es decir, la post condición hace referencia a como queda el sistema y que entidades cambian dentro del sistema para que o finalice un requerimiento o le de pie a otro requerimiento.
* Ayuda a ver las relaciones entre los diferentes casos de uso que definen una funcionalidad. En el trazo fino cuando solamente describimos un solo caso de uso que a su vez puede tener relacion con otros, nosotros vamos a poder especificar bien esas relaciones, es decir, cuando se dispara ese caso de uso y bajo qué condiciones, que devuelve el otro caso de uso y podemos dejar mucho más claras esas relaciones entre los diferentes requerimientos y funcionalidades.

\_ Tenemos que aprender que es un trazo fino y como se complementan con los diagramas de casos de uso. Por ende debemos escribir el trazo fino de una funcionalidad. Una de las desventajas de los trazos finos es que dependiendo del tamaño del modelado, va a ser la magnitud del trazo fino. No necesariamente todo lo que modelemos en el diagrama de caso de uso requiere detalle. Generalmente se van a hacer trazos finos de las cosas más importantes o de las cosas que tengan una ambigüedad y las queramos resolver de alguna manera. Entonces como ventajas los trazos finos le agregan el modelo el detalle que justamente no tiene y abstrae. El trazo fino describe lo que pasa en el sistema y no lo que pasa en el mundo real.

\_ En el flujo de la descripción del caso de uso con el trazo fino, podemos volver a un paso anterior o llevarlo a otra parte de la descripción, o directamente salir (el sistema o caso de uso finalizo).

**Trazos gruesos**

\_ Lo único que hacen es dar una descripción de la funcionalidad que se está describiendo, o sea el caso de uso. Simplemente es una subparte de un trazo fino que se enfoca en la descripciones detallada de la funcionalidad de un caso de uso.

**Diagrama de clases de dominio**

**Modelo de dominio**

Perspectiva conceptual

\_ Un modelo de dominio es una vista más general de los conceptos del negocio de más alto nivel. Es una representación visual de las clases conceptuales del mundo real en un dominio de interés. Es un diccionario visual de abstracciones. Cuando tengamos que definir un modelo de dominio vamos a definir:

Clases: formalmente una clase representa un concepto dentro del sistema que se está modelando. Podemos decir que una clase es una idea, una cosa o un objeto, como por ejemplo una mascota, un producto o un contrato. Estas tienen dos compartimientos en el primero se pone el nombre de la clase en singular y capitalizado (primer letra en mayúscula) y los atributos comienzan con letra en minúscula, si está compuesto por más de una palabra, por convención en las siguientes se capitaliza la primer letra.

Asociaciones: estas representan relaciones entre instancias de las clases, por ejemplo, tenemos cuatro clases que son empleado, empresa, alumno y materia, donde para una la clase materia podemos definir o decir que tenemos las instancias algebra, lengua e inglés, otra forma de pensar las instancias es si hiciéramos por ejemplo un listado de materias y pensarlas como las distintas filas que conforman una tabla. Entonces podemos decir que un empleado trabaja en una empresa o que un alumno cursa una materia.

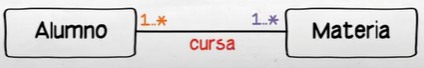


\_ A continuación vamos a aplicar adornos:

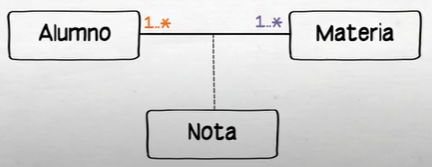
* Multiplicidad: para responder a la pregunta ¿cuántas materias cursa el alumno Santiago?, indicamos que cursa una o más materias (1..\*). Entonces la multiplicidad determina cuantas instancias, de en este caso materias, se asocian con alguna instancia de alumnos. Viéndolo al revés, es decir, ¿Cuántos alumnos cursan inglés?, podemos decir que uno o muchos (1..\*), haciendo referencia a cuantas instancias de alumnos se asocian con una instancia de materia. Es obligatorio indicar la multiplicidad en los atributos cuando esta difiera de 1 (por ejemplo 0..1, 1..\*, etc).



* Nombre de asociación: viendo el ejemplo anterior, la asociación tiene de nombre “cursa”.



* Clase de asociación: dado el ejemplo anterior, cambiamos el nombre de asociación “cursa” por el de “rinde”. Cuando los alumnos rinden materias, estos obtienen una nota como resultado y esta nota no es propia del alumno ni de la materia, sino que es algo que se obtiene, genera o gana a partir de que un alumno rinde dándose la asociación que un alumno rinde una materia. Lo representamos de la siguiente manera, donde la nota va a bajar como clase y la unimos con línea punteada de la asociación anterior. No es obligatorio modelar usando clases asociativas salvo que se lo solicite expresamente. La asociación no lleva nombre pero si lleva multiplicidad.

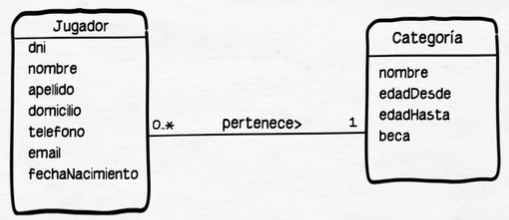


Atributos: son un valor de datos lógico de un objeto, es decir, son donde se modela o representa el dato para luego producir la información que nuestros sistemas bajo estudio gestionan. Ademas, los atributos son la información que necesitamos para registrar ciertos requisitos y teniendo en cuenta siempre el dominio que se está analizando. Por ejemplo la clase “materia” podría tener atributos totalmente diferentes, si estamos analizando el dominio “facultad” o el dominio “físico-química”.

\_ Como ejemplo, tenemos como dominio de problema a un club de fútbol. Este club cuenta con jugadores o categorías. Las reglas de negocio son las siguientes:

* De un jugador se conoce: DNI, nombre, apellido, teléfono, mail y fecha de nacimiento.
* De las categorías se conoce: nombre, rango de edad y si se puede asignar una beca económica.
* Un jugador pertenece a una única categoría.

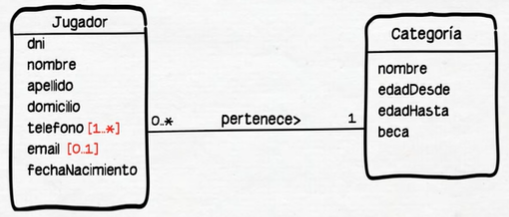
\_ Al momento de graficar esto, los atributos se colocan en el segundo compartimiento del rectángulo de la clase, su tipo se puede mostrar opcionalmente. La última regla de negocio nos hace pensar que existe una relacion entre negocio y categoría. No es necesario agregar un atributo “categoría” al jugador para indicar justamente a que categoría pertenece, así como tampoco es necesario agregar en categoría un atributo para todas las instancias de jugador que pertenecen a ella. Por ende simplemente agregamos la asociación:



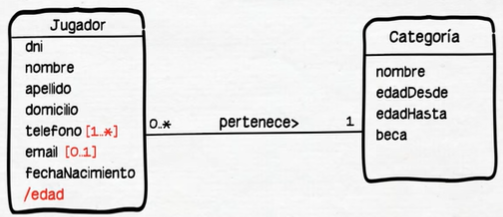
\_ Supongamos que se agrega una nueva regla de negocio y es:

* Un jugador puede tener varios números telefónicos y puede ser que no tenga mail.

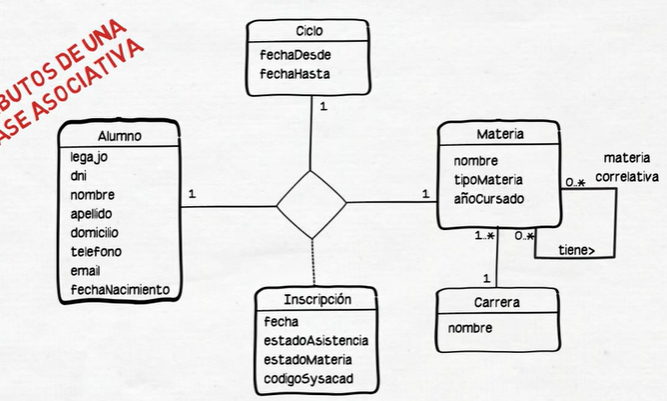
\_ A continuación hacemos el modelado:



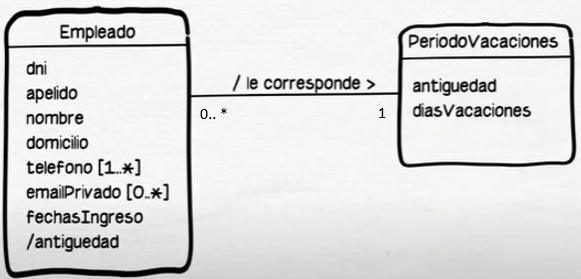
* Atributos derivables: estos son atributos cuyo valor puede ser calculado partiendo del valor de otros atributos de la misma instancia o de otras instancias de la misma clase, o de otras clases. No es necesario modelar estos atributos pero si lo hacemos debemos anteponer una barra / y siguiente el nombre del mismo.



* Atributos de una clase asociativa: como vemos en el ejemplo tenemos un alumno, tenemos materia y ciclo, donde un alumno se inscribe en una materia para un ciclo, esta relacion genera una clase asociativa que es inscripción donde los atributos de esta son todos datos que no pertenecen al resto de clases.



* Asociación derivable: son asociaciones que pueden inferirse a través del valor de un atributo. En el siguiente ejemplo, para poder determinar el periodo de vacaciones que le corresponde al empleado, primero tenemos que obtener el atributo derivable “antigüedad”. Como vemos hay una / antes del nombre de la asociación.



\_ Cuando armamos el diagrama de clases de dominio, al no existir el tiempo no hay una instancia, es decir, no hay un valor para los atributos. Nosotros definimos los atributos y las entidades, pero no tienen un valor asociado, ya que el tiempo es el que les da un valor dependiendo el estado del sistema.

\_ El diagrama de casos de uso nos puede dar una aproximación de las entidades, por lo cual, para encontrar las entidades tenemos que combinar la descripción del dominio, el diagrama de casos de uso y la filosofía de que mas poner, es decir, pensar más allá que es lo que hace falta para que el modelado tenga sentido.

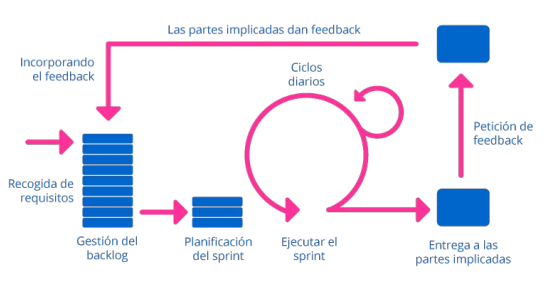
**Metodologías Ágiles: Historias de Usuarios**

**Introducción**

\_ Es en lenguaje natural lo que le usuario puede hacer o no en el sistema. Su estructura está compuesta por la descripción de la historia de usuario y los criterios de aceptación. Dentro de la descripción se definen tres cosas, que serían para quien está destinado, luego que es lo que va a hacer el sistema o lo que desea el actor y por último cual es la ganancia, donde en base a lo que la ganancia va a ser para el actor, se define la prioridad. Ahora con los criterios de aceptación lo que buscamos es, con la descripción que se hace de esa historia de usuario, poder saber si es satisfactorio o no cuando lo implementemos de alguna manera.

**Flujo de desarrollo ágil**

\_ En toda metodología ágil no necesariamente tiene que haber historias de usuario pero estas si ayudan. Podemos usar un trazo fino en vez de una historia de usuario, es menos ágil pero no estamos obligados a usar historias de usuario. Analizando el grafico, nosotros estaríamos del lado de los requerimientos (izquierda), donde tenemos las cosas que hay que hacer, quitamos la ambigüedad, damos contexto y de alguna forma priorizamos lo importante con el equipo para arrancar a trabajar pero no nos dice que tan complejo es. Entonces el diagrama el flujo básico de una metodología ágil.



**Objetivos del modelo ágil**

* Ser capaz de adaptarse, cambiar y crecer.
* Mejorar la satisfacción del cliente.
* Ser sea adaptable y abierto al cambio: lo importante es que se pueda adaptar la metodología o el proceso de desarrollo a lo que estamos tratando de desarrollar o lograr un fin, pero nunca al revés, o sea tratar de hacer encajar el producto a la metodología ya que podría no resultar
* Cooperación en equipos de desarrollo.
* Creación eficiente de software.
* Racionalización del proceso de desarrollo (optimización de tiempo, costo y esfuerzo).
* Lanzamiento de productos más rápido: generalmente la metodología ágil esta orientada a productos que tienen que salir rápido al mercado teniendo en cuenta las ganancias posibles.

**Historia de usuario**

Definición

\_ Las historias de usuario son pequeñas descripciones de los requerimientos que tenemos del cliente. Son mecanismos que constituyen entendimiento, colaboración y discusión utilizando el lenguaje natural, sin tecnicismo y buscando el desacoplamiento. Algo clave de las historias de usuario es que no hay que darle mucha complejidad ni entrar mucho en detalle, ya que son simples, a estas debería escribirla el product owner que no necesariamente debe saber con mucho detalle ni tecnicismo. Hay que buscar desacoplar las historias de usuario unas de otras, estas tienen que tener relacion pero no muy acopladas. No necesariamente metodología ágil es sinónimo de historia de usuario.

\_ Compuesto por:

* Rol: definir quién utilizará la funcionalidad a desarrollar (¿Quién?).
* Funcionalidad: especificar qué producto quiere el usuario (¿Qué?).
* Resultado esperado: para qué utilizará el producto, o también es el valor agregado (¿Qué se espera?).
* Criterios de aceptación: verificar el cumplimiento de las expectativas (¿Cómo lo validamos?).

Como se escriben las historias de usuario

**Como un** < usuario | rol que interactúa > …

**Quiero** < acción | funcionalidad | lo que debe suceder > ...

**Para** < objetivo | valor agregado | que resuelve > …

\_ A continuación tenemos un ejemplo:

* Como Vendedor, quiero registrar los productos y cantidades que me solicita un cliente para crear un pedido de venta.

\_ No nos tenemos que olvidar del valor, porque este define la funcionalidad. Si no ponemos el valor y dejamos la funcionalidad sola, no podemos discernir cuan critica es respecto a otra.

Conclusión

\_ Una historia de usuario está totalmente centrada en el usuario, y no en el producto o la organización. Tenemos que tener cuidado al momento de pensar la historia de usuario muy enfocada al producto o la organización, ya que tenemos que pensarla respecto al usuario ya que este tendrá un rol clave a la hora de saber el valor.

**Criterios de aceptación**

\_ Sin criterios no hay historias de usuario. Cuando escribimos una historia de usuario con el formato que vimos, hay un tema que no nos permite saber la complejidad de lo que el cliente nos está pidiendo y no nos da los escenarios para poder saber que nos va a dar una cosa u otra, por ejemplo el cliente quiere poner productos en cantidades negativas, podemos pensar o no si esto sería un escenario valido, pero podríamos tener esa información, también podría darse el escenario de poner cantidades de productos iguales a cero y superior. Entonces los criterios de aceptación complementan el contexto que no lo determina la historia de usuario en la descripción, por ende ayuda a verificar el contexto y a entender si esa funcionalidad que estamos desarrollando, diseñando y demás, pasa o no los test para que sea integra. Ayudan a planificar el contexto. Nos ayudan también a saber si la funcionabilidad está terminada o no, criterios de aceptación tenemos que poner uno o N tanto funcionales como no funcionales, donde si ponemos un solo criterio o ninguno, nunca podríamos validar, con lo cual la historia de usuario estaría mal. Si ponemos pocos escenarios, no estaría mal pero podrían aparecer escenarios que no están contemplados, por lo que no podríamos saber si la historia de usuario esta completa o no. Los criterios de aceptación cumplen dos funciones, la primera es clarificar el contexto en el que ocurre la historia de usuario y facilitar saber cuándo están realmente terminadas. Pueden, además, contener requerimientos no funcionales y usarse ejemplos. Ejemplos:

* Puedo crear un pedido de venta con un solo producto.
* Puedo crear un pedido de venta con más de un producto.
* Agregar un producto al pedido no debe demorar >10ms.
* No puedo crear un pedido de venta de un producto con cantidad cero.

\_ Tenemos que ser bien específicos al momento de escribir los criterios de aceptación y siempre ir de lo mas amplio a lo más sencillo. Entre requerimientos funcionales y no funcionales, con diez criterios de aceptación una historia de usuario va a estar cubierta, siempre hablando como super complejo.

\_ Las historias de usuario no tienen todo el detalle, con lo cual nosotros como analistas de requerimientos no vamos a saber todo lo que tiene que tener para poder ser 100% valida. Lo que buscan las historias de usuario es como abrir un poco el juego para el dialogo, y por eso se vuelve ágil, porque si tuviéramos que explicar todo lo que hace la funcionalidad, el cliente no va a entender y entra la ambigüedad de lo que hay que escribir.

**Las “Cs” de las historias de usuario card (tarjeta)**

* Card: recordatorio de la funcionalidad a hablar y discutir antes de comenzar el desarrollo → Ayuda a la conversación.
* Conversación: todo lo que no se puede especificar o aclarar se DEBE conversar → Se busca aclaración y completitud, presencial preferentemente.
* Confirmación: criterios de aceptación que se deben validar antes de entregar la historia de usuario.
* Construcción: desarrollo de cada card de acuerdo a lo conversado y pactado.
* Resultados: se evalúa cada card con los diferentes interesados teniendo presente los criterios de aceptación como punto de validación además de los hallazgos en las demos/pruebas.

**Modelo INVEST para definir una historias de usuario**

\_ Algunas características de las historias de usuario:

Independiente: no necesite de otra para poder funcionar o, al menos, no planificables en una misma iteración si son dependientes.

Negociable: historias pequeñas, no completas al máximo, y conversables de modo que se pueden acordar algunas cosas y reducir alcances. Si una historia de usuario es muy grande va a ser muy difícil hacer la negociación en vez de si es una más pequeña.

Valiosa: que una historia de usuario no dependa de otra para generar valor.

Estimable: poder definir la dificultad comparada con otra para saber cuánto se puede comprometer.

Pequeña: debemos tratar de que entre en una iteración y que al mismo tiempo no consuma el tiempo total de una persona al límite de la iteración.

Comprobable: que una persona pueda decir cuando está lista y que pueda verificarla.

Rol del Product Owner: ayuda a crear las historias de usuarios, las mantiene claras, define buenos criterios de aceptación, está presente para conversarlas. Responsable de conjugar las cualidades INVEST para generar las mejores historias de usuarios mediando entre equipo y cliente. No necesariamente el product owner tiene que estar como entidad, ya que puede ser una entidad muy de scrum y no todos lo tienen.

**Diferencias entre historia de usuario y trazo fino**

Historias de usuario:

* Presentes en backlog del producto.
* Se descomponen del producto a realizar.
* Funcionalidad: visible para el usuario.
* Complejo.
* Colaborativa.
* Conjunción de trabajo.
* Tienen un valor, riesgo y estimación.

Trazo fino (tareas):

* Presente en backlog del sprint (iteración).
* Se descomponen de historia de usuarios.
* Organización: visible para el equipo.
* Ejecutable.
* Individual.
* Un solo tipo de trabajo.
* Descomponen valor, riesgo y estimación.