Conceptos de Análisis y Diseño. Programación Orientada a Objetos

Introducción a UML

¿Qué es UML? ¿Qué es UP?

Unified Model Language o lenguaje de modelos unificado es un herramienta que nos provee una serie de convenciones, estructuras, estándares, reglas para el análisis y diseño orientado a objetos. Es una herramienta fundamental para los programadores orientados a objetos, que nos permite pensar en los objetos de una forma ordenada y metodológica. Larman los introduce de forma muy pedagógica y útil en su libro UML y patrones.

Unified Process o proceso unificado es un forma de construir un sistema que introduce una serie de disciplinas y fases en ese proceso y que usa a UML como standard en la elaboración de los artefactos(es cualquier documento diagrama dibujo estandarizado por UML y que se usa en el proceso de desarrollo del software) y entregables que este proceso crea y utiliza. En pocas palabras UP es una metodología de desarrollo de software que nos propone una forma de hacer software y utiliza UML para elaborar sus diagramas y documentos
El UP fomenta muchas buenas prácticas pero una destaca sobre las demás: el desarrollo iterativo. En este enfoque el desarrollo se organiza en una serie de mini proyectos cortos de duración fija (por ejemplo de cuatro semanas) llamados iteraciones, el resultado de cada uno es un sistema (software o no) que puede ser probado, integrado y ejecutado. Cada iteración incluye sus propias actividades de análisis de requisitos, diseño, implementación y pruebas.

Fases y Disciplinas

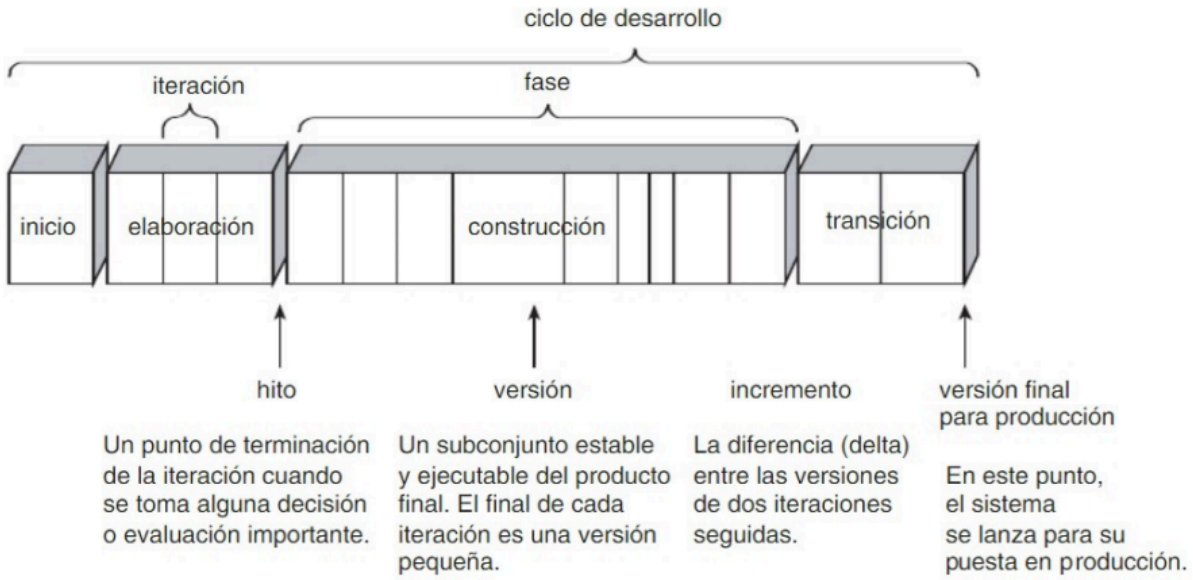
El UP hace uso de una serie de conceptos para explicar su metodología, dos de ellos son las Fases y las Disciplinas. Para UP las Fases son aquellas que están asociadas a una transición temporal, en cambio las disciplinas están asociadas al tipo de trabajo que en ellas se realiza. Si bien algunas disciplinas están asociadas a algunas fases, ya que es lógico que ciertos tipos de trabajo se hagan en cierta fase de la elaboración del software, es importante entender que no son lo mismo ni tampoco su diferencia es redundante ya que la relación entre fases y disciplinas es bastante dinámica.

Fases

- Inicio: es una visión aproximada de él/los problemas a resolver en ella se hace análisis del negocio, se determina el alcance a abarcar, se realizan estimaciones indefectiblemente imprecisas. Es una fase de viabilidad

- Elaboración: visión refinada, implementación iterativa del núcleo central de la arquitectura, resolución de riesgos altos, identificación de más requerimientos o requisitos. Se logran estimaciones más realistas. Es una fase de implementación

- Construcción: se resuelven de forma iterativa los demás riesgos, se prepara para el despliegue
- Transición: Se hacen pruebas beta (pruebas en un entorno relativamente real, con algunos usuarios finales o bien casos de usos factibles en el negocio). Se realiza el despliegue del producto elaborado



Disciplinas

A continuación se presentan las principales disciplinas de UP. Las disciplinas marcadas en azul están presentes en todas las iteraciones.

- Planificación

  En esta fase se incluyen tareas como la determinación del ámbito del proyecto, un estudio de viabilidad, análisis de riesgos, costes estimados, asignación de recursos en las distintas etapas, etc.
  Son tareas que influyen en el éxito del proyecto, por eso es necesaria una planificación inicial.

- Análisis

  Proceso en el que se trata de descubrir lo que se necesita y cómo llegar a las características que el sistema debe poseer.

- Diseño

  Se estudian las posibles implementaciones que hay que construir y la estructura general del software.
  Es una etapa complicada, y si la solución inicial no es la más adecuada, habrá que redefinirla.

- Implementación

Se trata de elegir las herramientas adecuadas, un entorno de desarrollo que haga más sencillo el trabajo y el lenguaje de programación óptimo.
Esta decisión va a depender del diseño y el entorno elegido.
Es importante tener en cuenta la adquisición de productos necesarios para que el software funcione.

- Pruebas

  Conseguiremos detectar los fallos que se hayan cometido en etapas anteriores, para que no repercuta en el usuario final.
  Esta fase del ciclo de vida del software hay que repetirla tantas veces como sea necesaria, ya que la calidad y estabilidad final del software dependerá de esta fase.

- Despliegue
  En esta fase pondremos el software en funcionamiento.

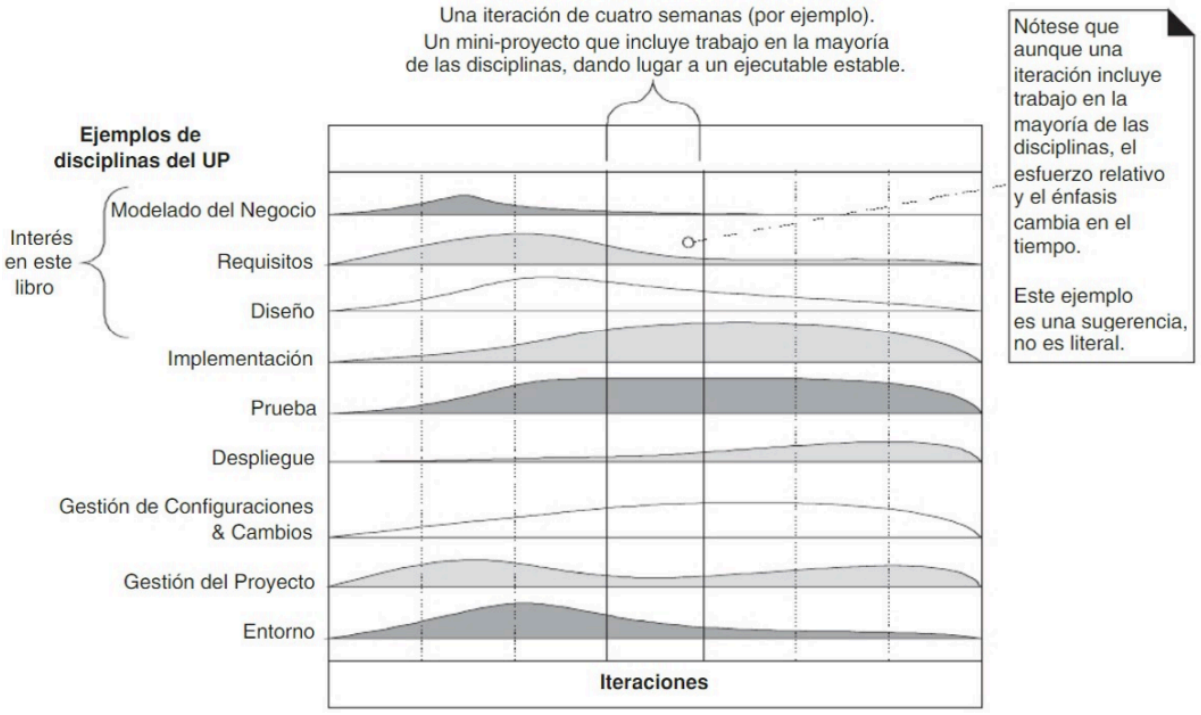


Figura 2.4. Disciplinas del UP[5].

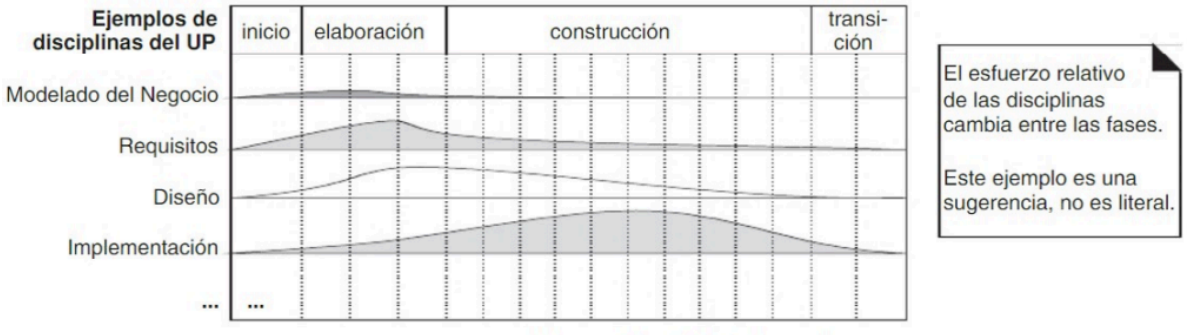
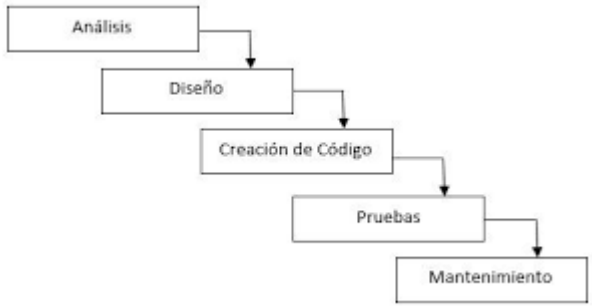


Figura 2.5. Disciplinas y fases.

Ciclos de vida del proceso de desarrollo

Modelo en cascada

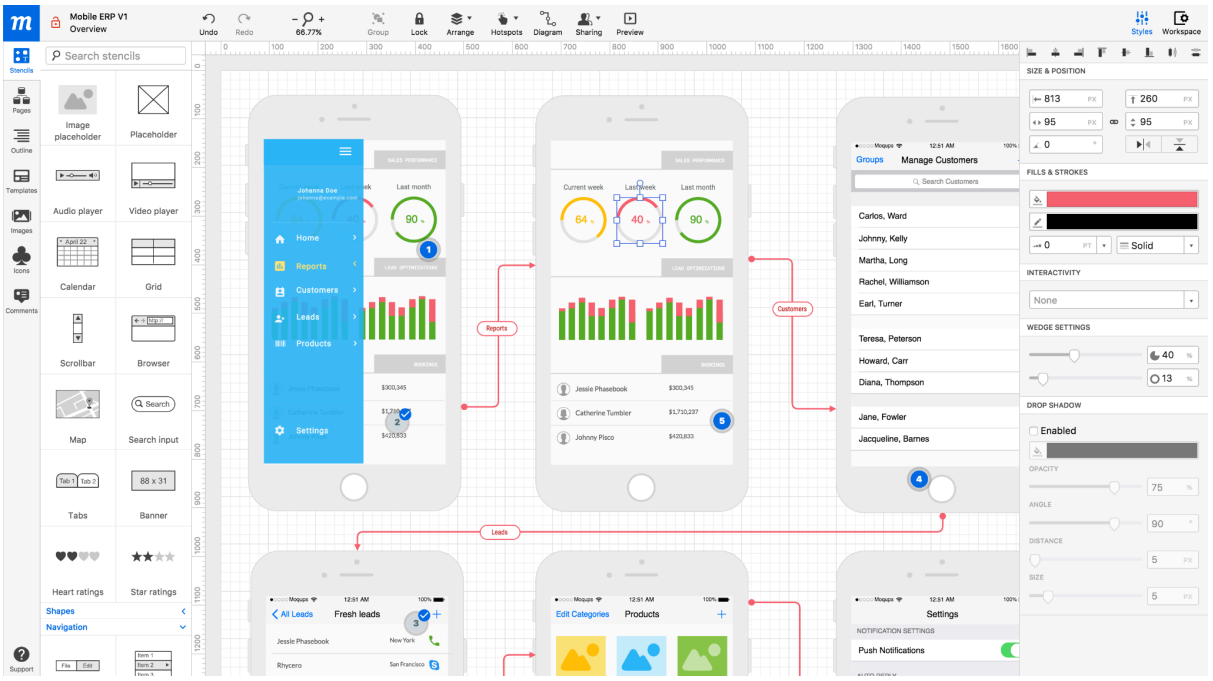
En este modelo del ciclo de vida de un software, se espera a finalizar una etapa para comenzar con la siguiente.
Es un proceso secuencial en el que el desarrollo va fluyendo de arriba hacia abajo.
Aunque en ocasiones ha sido criticado debido a su rigidez, sigue siendo el más seguido a día de hoy.



(En la imagen creación de código se refiere a la etapa de implementación)

Modelo de prototipos

Comienza con la recolección de requisitos y definición de objetivos globales, llevando a un diseño rápido y a un prototipo.
El prototipo es evaluado por el cliente, y nos permite refinar los requisitos hasta llegar a lo que el cliente espera.



Modelo de desarrollo iterativo e incremental

Se basa en la ampliación y refinamiento sucesivos del sistema mediante múltiples iteraciones, con retroalimentación cíclica y adaptación como elementos principales. El sistema crece incrementalmente a lo largo del tiempo, iteración tras iteración, y por ello su nombre

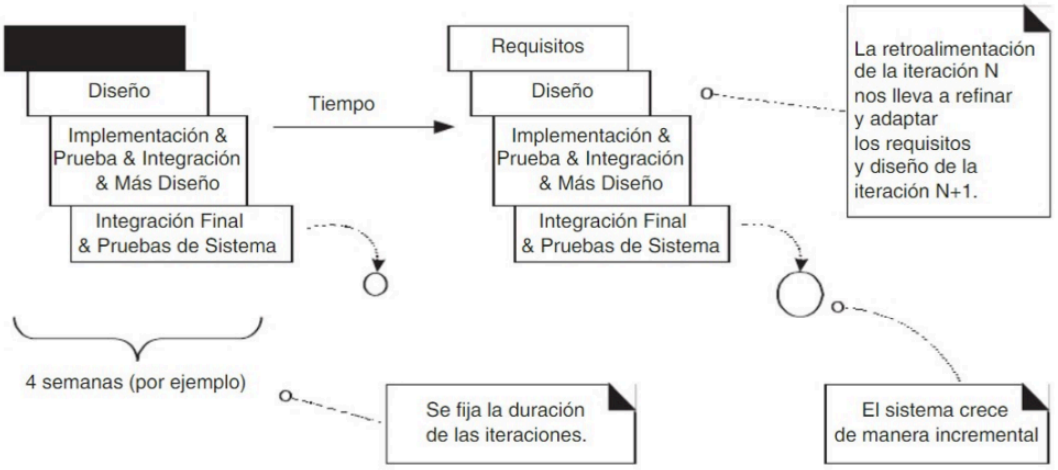


Figura 2.1. Desarrollo iterativo e incremental.

Programación Orientada a objetos y principios fundamentales

Encapsulación

El concepto de encapsulación es el más evidente de todos. Pero, precisamente por su sencillez, a veces pasa inadvertido.

La encapsulación es la característica de un lenguaje POO que permite que todo lo referente a un objeto quede aislado dentro de éste. Es decir, que todos los datos referentes a un objeto queden "encerrados" dentro de éste y sólo se puede acceder a ellos a través de los miembros que la clase proporciona (propiedades y métodos).

Por ejemplo, en el caso de las personas, toda la información sobre éstas está circunscrita al ámbito de dicha persona.

Así, internamente tenemos un dato que es el nombre de la persona y accedemos a él a través de la propiedad email que define la clase que representa a las personas. De este modo damos acceso sólo a lo que nos interese y del modo que nos interese. En un lenguaje no orientado a objetos tendríamos que montar alguna estructura global para almacenar esa información y luego acceder ordenadamente a ella, sin mezclar los datos de una persona con los de otra.

Gracias a la encapsulación, toda la información de un objeto está contenida dentro del propio objeto.

Abstracción

Este concepto está muy relacionado con el anterior.

Como la propia palabra indica, el principio de abstracción lo que implica es que la clase debe representar las características de la entidad hacia el mundo exterior, pero ocultando la complejidad que llevan aparejada. O sea, nos abstrae de la complejidad que haya dentro dándonos una serie de atributos y comportamientos (propiedades y funciones) que podemos usar sin preocuparnos de qué pasa por dentro cuando lo hagamos.

Así, una clase (y por lo tanto todos los objetos que se crean a partir de ella) debe exponer para su uso solo lo que sea necesario. Cómo se haga "por dentro" es irrelevante para los programas que hagan uso de los objetos de esa clase.

En nuestro ejemplo(última imagen de esta unidad) el diagnóstico puede tener un método que devuelve el monto a cobrar, sin embargo el objeto abstrae la implementación interna de este método y el usuario del sistema no sabe que el método getPrice(int IdDiagnostico) devuelve el precio teniendo en cuenta si el cliente (Persona) es un cliente antiguo y tiene un descuento especial o si es deudor y se le computará también su deuda.

La abstracción está muy relacionada con la encapsulación, pero va un paso más allá pues no sólo controla el acceso a la información, sino también oculta la complejidad de los procesos que estemos implementando.

Herencia

Dado que una clase es un patrón que define cómo es y cómo se comporta una cierta entidad, una clase que hereda de otra obtiene todos los rasgos de la primera y añade otros nuevos y además también puede modificar algunos de los que ha heredado.

A la clase de la que se hereda se le llama clase base o padre, y a la clase que hereda de ésta se le llama clase derivada o hija.

En el diagrama final podemos observar un ejemplo de herencia: La clase Persona es una clase padre que tiene dos clases hijas, Fisica y Juridica. Esas dos clases heredan todos los métodos y atributos definidos en la clase Padre y a su vez le añaden un atributo cada una, en el caso de la Física añade el atributo “DNI” y en la clase Jurídica añade “CIF” (código de identificación fiscal).

La herencia es una de las características más potentes de la POO ya que fomenta la reutilización del código permitiendo al mismo tiempo la particularización o especialización del mismo.

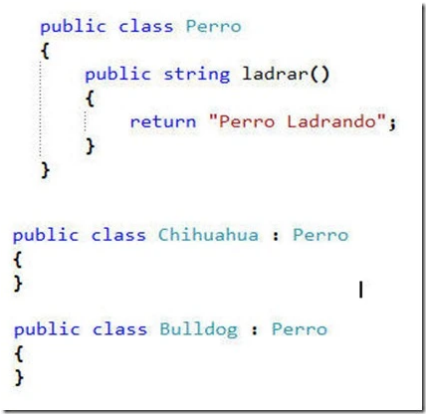
Polimorfismo

La palabra polimorfismo viene del griego "polys" (muchos) y "morfo" (forma), y quiere decir "cualidad de tener muchas formas".

En POO, el concepto de polimorfismo se refiere al hecho de que varios objetos de diferentes clases, pero con una base común, se pueden usar de manera indistinta, sin tener que saber de qué clase exacta son para poder hacerlo.

El polimorfismo permite que una clase hija puede redefinir un método de la clase padre. En el caso de que la clase hija tuviese un método definido con el mismo nombre pero con el atributo “override”, la que es válida para objeto instancia de la clase hija es el método redefinido en la clase hija.

Ejemplo:



```
public class Perro
{
    public string ladrar()
    {
        return "Perro Ladrando";
    }
}

public class Chihuahua : Perro
{
}

public class Bulldog : Perro
{
}
```

En la clase perro tenemos un método que solo nos devuelve un string. Por otra parte las clases chihuahua y bulldog serán clases que heredarán el comportamiento de la clase perro.

Ahora supongamos que estos dos perros son callejeros y viene el control de animales y los meta a la perrera. Esta perrera será un arreglo o Array en nuestro program, de la siguiente manera:

```
static void Main(string[] args)
{
    Chihuahua _Chihuahua1 = new Chihuahua();
    Chihuahua _Chihuahua2 = new Chihuahua();
    Bulldog _Bulldog1 = new Bulldog();
    Bulldog _Bulldog2 = new Bulldog();

    Perro[] _Perrera = { _Chihuahua1, _Chihuahua2, _Bulldog1, _Bulldog2 };
    foreach (Perro item in _Perrera)
    {
        Console.WriteLine(item.ladrar());
    }

}
```

Resultado:

Ahora aplicaremos polimorfismo:

Cambiemos el comportamiento de los hijos; veamos supongamos que no todos los perros ladran de la misma forma el chihuahua ladra distinto que el bulldog. Para lograr esto la clase perro tiene que dejar o permitir que los hijos modifiquen el comportamiento; miremos como quedan las clases ya modificadas:

```
public class Perro
{
    public virtual string ladrar()
    {
        return "Perro Ladrando";
    }
}
public class Chihuahua : Perro
{
    public override string ladrar()
    {
        return "Chihuahua Ladrandando";
    }
}
public class Bulldog : Perro
{
    public override string ladrar()
    {
        return "Bulldog Ladrandando";
    }
}
```

Primero el método Ladrar es de tipo virtual esto me permitirá sobreescribir este método en los hijos, en las clases hijos usaremos la palabra clave override para sobrescribir el método virtual del padre. Si corremos la aplicación de consola nos dará esto:

De hecho, el polimorfismo puede ser más complicado que eso ya que se puede dar también mediante la sobrecarga de métodos y, sobre todo, a través del uso de interfaces.

El polimorfismo nos permite utilizar a los objetos de manera genérica, aunque internamente se comportan según su variedad específica.

Artefactos UML fundamentales

Minuta de relevamiento

Este artefacto no está definido por el estándar UML pero es por el cual se parte para realizar todos los demás. Normalmente hecho por el analista del equipo, una minuta de relevamiento es una narración descriptiva de él/los procesos de negocio de la empresa que son relevantes para el sistema que se precisa desarrollar.

Este relevamiento no es para nada sencillo de realizar y de hacer, ya que requiere captar de manera objetiva, sencilla y metódica todos los pasos del proceso de negocio así como de sus alternativas. El analista tiene que escribir esta minuta pensando en aquellos que después la leerán para obtenerlos demás artefactos.

Se suele escribir con oraciones cortas en presente simple, siendo consistente con los términos utilizados (Ej al cliente no se le dice de distintas formas como consumidor o comprador)

Lista de requerimientos

Los requerimientos son capacidades y condiciones con las cuales debe ser conforme el sistema y más ampliamente el proyecto que se desea construir. Por lo tanto el primer desafío con el que nos encontramos a la hora de desarrollar un sistema es con encontrar comunicar y registrar los requerimientos.
Normalmente son definidas por el cliente, sin embargo esto es todo un proceso porque usualmente el cliente no sabe lo que quiere, e incluso aunque lo sepa sus necesidades pueden cambiar a lo largo del desarrollo del sistema. El negocio, la presión del mercado, innovaciones, cambio de necesidades en sus clientes, diferentes formas de comercialización,etc son algunos de los motivos por lo que debemos considerar a los requerimientos como un conjunto dinámico que debe ser revisado periódicamente.
Para UML estos requerimientos se explicitan en un artefacto denominado “Lista de requerimientos”

Tipos de requerimientos

UML define una clasificación que es muy útil debido a que los tipos que se definen permiten dividir el trabajo para resolución de requisitos similares. El clasificación de UML se denomina modelo FURPS+:

F Funcional: Características, capacidades, seguridad. Deben ser cumplidas para que el sistema se pueda usar.
U Facilidad de uso (Usability): factores humanos, ayuda, documentación.
R Fiabilidad (Reliability): frecuencia de fallos, capacidad de recuperación de un fallo, grado de previsión.

P Rendimiento (Performance): tiempo de respuesta, productividad, precisión, disponibilidad, uso de recursos
S Soporte (Supportability): adaptabilidad, facilidad de mantenimiento, internacionalización, configurabilidad.
+ Implementación, Interfaz, Legales, Operaciones, Empaquetamiento, etc

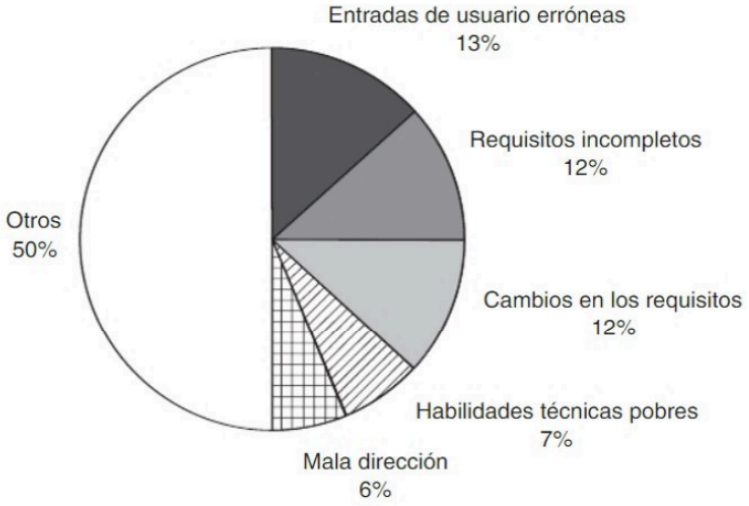


Figura 5.1. Factores del coste en proyectos software reales.

Casos de uso

En primer lugar algunas definiciones informales, un actor es algo con comportamiento, como una persona (identificada por un rol), sistema de informatizado y organización; por ejemplo, un cajero.

Un escenario es una secuencia específica de acciones e interacciones entre los actores y es el sistema objeto de estudio. Es una historia particular del uso de un sistema, o un camino a través del caso de uso; por ejemplo, el escenario de éxito de compra de artículos con pago en efectivo, o el escenario de fallo al comprar el debido rechazo de la transacción de un pago con tarjeta de crédito.

Un caso de uso es una colección de escenarios con éxito y fallo relacionados, que describe los actores utilizando el sistema para satisfacer un objetivo.

Un caso de uso esta compuesto por:

- Actor primario: es el que usa el sistema
- Actor/es secundarios: de haberlos son aquellos que solicitan o proveen alguna información necesaria para que el caso de uso se pueda realizar
- Acción disparadora del caso de uso: un evento que hace que el caso de uso inicie
- Estado de éxito: es el escenario que resulta de finalizar el caso de uso con éxito
- Estado de fracaso:es el escenario que resulta de finalizar el caso de uso con fracaso

- Un camino básico en donde se describe el paso a paso de la utilización del sistema por un actor.
- Caminos alternativos: son las variantes que se pueden dar si el escenario no transcurre por el camino básico.

  Es importante destacar que para cada requerimiento de tipo funcional se debe corresponder con un caso de uso que narra cómo se lo satisface.

Tipo de caso de uso

Difieren en la perspectiva y el nivel de detalle que cada uno contempla.

- Casos de uso resumen de negocio(CURN).: son los casos de uso desde el punto de vista del proceso de negocio antes de tener el sistema. Es uno por requerimiento funcional.
- Caso de uso resumen de sistema(CURS): es uno por cada caso de uso resumen de negocio pero desde el punto de vista del proceso con el sistema hecho
- Caso de uso usuario(CUU): estos casos de uso son producto de partir los CURS en pequeñas partes enfocándose en uso especifico de sistema y corto en el tiempo. Cada CUU se corresponde solo a un CURS

Diagrama de Clases

El diagrama de clases es uno de los diagramas incluidos en UML 2.5 clasificado dentro de los diagramas de estructura y, como tal, se utiliza para representar los elementos que componen un sistema de información desde un punto de vista estático.

El diagrama de clases es un diagrama puramente orientado al modelo de programación orientado a objetos, ya que define las clases que se utilizarán cuando se pase a la fase de construcción y la manera en que se relacionan las mismas. Se podría equiparar, salvando las distancias, al famoso diagrama de modelo Entidad-Relación (E/R), no recogido en UML, tiene una utilidad similar: la representación de datos y su interacción. Ambos diagramas muestran el modelo lógico de los datos de un sistema.

El diagrama UML de clases está formado por dos elementos: clases, relaciones e interfaces.

Clases

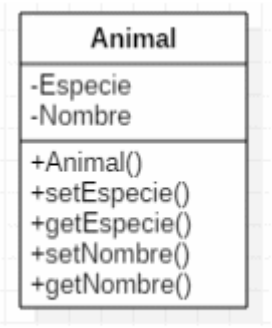
Las clases son el elemento principal del diagrama y representa, como su nombre indica, una clase dentro del paradigma de la orientación a objetos. Este tipo de elementos normalmente se utilizan para representar conceptos o entidades del “negocio”. Una clase define un grupo de objetos que comparten características, condiciones y significado. La manera más rápida para encontrar clases sobre un enunciado, sobre una idea de negocio o, en general, sobre un tema concreto es buscar los sustantivos que aparecen en el mismo. Por poner algún ejemplo, algunas clases podrían ser: Animal, Persona, Mensaje,

Expediente… Es un concepto muy amplio y resulta fundamental identificar de forma efectiva estas clases, en caso de no hacerlo correctamente se obtendrán una serie de problemas en etapas posteriores, teniendo que volver a hacer el análisis y perdiendo parte o todo el trabajo que se ha hecho hasta ese momento.

Bajando de nivel una clase está compuesta por tres elementos: nombre de la clase, atributos, funciones. Estos elementos se incluyen en la representación (o no, dependiendo del nivel de análisis).

Para representar la clase con estos elementos se utiliza una caja que es dividida en tres zonas utilizando para ello líneas horizontales:

- La primera de las zonas se utiliza para el nombre de la clase.
- La segunda de las zonas se utiliza para escribir los atributos de la clase, uno por línea.
- La última de las zonas incluye cada una de las funciones que ofrece la clase. De forma parecida a los atributos. De la misma manera que con los atributos, se suele simplificar indicando únicamente el nombre de la función y, en ocasiones, el tipo devuelto.



Ejemplo de una clase

Relaciones

Una relación identifica una dependencia. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina dependencia reflexiva. Las relaciones se representan con una línea que une las clases, esta línea variará dependiendo del tipo de relación

Las relaciones en el diagrama de clases tienen varias propiedades, que dependiendo la profundidad que se quiera dar al diagrama se representarán o no. Estas propiedades son las siguientes:

- Multiplicidad. Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango… Se utiliza n o * para identificar un número cualquiera.
- Nombre de la asociación. En ocasiones se escribe una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como por ejemplo: “Una empresa contrata a n empleados”

Tipos principales de relaciones

- Asociación

Este tipo de relación es el más común y se utiliza para representar dependencia semántica. Se representa con una simple línea continua que une las clases que están incluidas en la asociación.

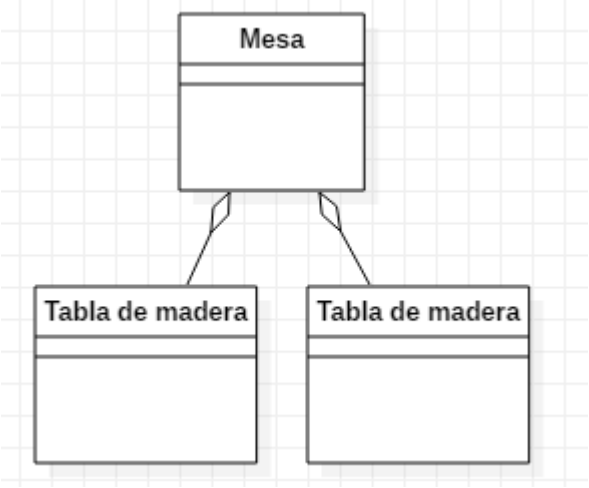
Un ejemplo de asociación podría ser: “Una mascota pertenece a una persona”.

- Agregación o composición

Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que un objeto es parte de otro, pero aun así debe tener existencia en sí mismo.

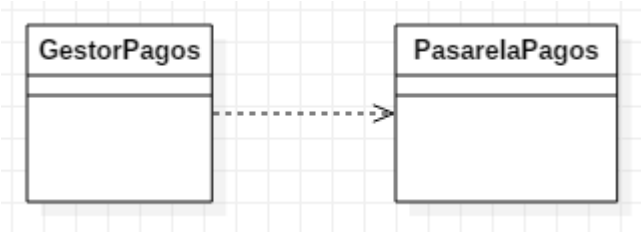
Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Un ejemplo de esta relación podría ser: “Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa”. Cómo ves, el tornillo podría formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.



- Dependencia

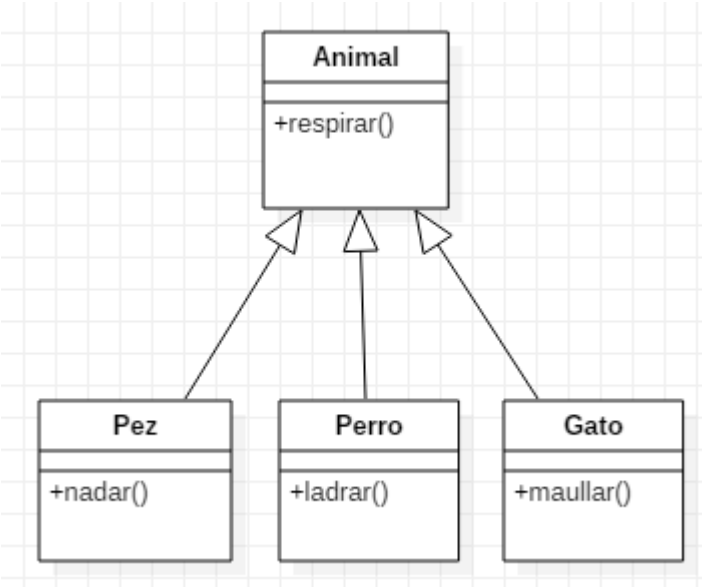
Se utiliza este tipo de relación para representar que una clase requiere de otra para ofrecer sus funcionalidades. Es muy sencilla y se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.



- Herencia

Otra relación muy común en el diagrama de clases es la herencia. Este tipo de relaciones permiten que una clase (clase hija o subclase) reciba los atributos y métodos de otra clase (clase padre o superclase). Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma. Se utiliza en relaciones “es un”.

Un ejemplo de esta relación podría ser la siguiente: Un pez, un perro y un gato son animales.



Cómo dibujar un diagrama de clases

Los diagramas de clase van de la mano con el diseño orientado a objetos. Por lo tanto, saber lo básico de este tipo de diseño es una parte clave para poder dibujar diagramas de clase eficaces.

Este tipo de diagramas son solicitados cuando se está describiendo la vista estática del sistema o sus funcionalidades. Unos pequeños pasos que puedes utilizar de guía para construir estos diagramas son los siguientes:

- Identifica los nombres de las clase

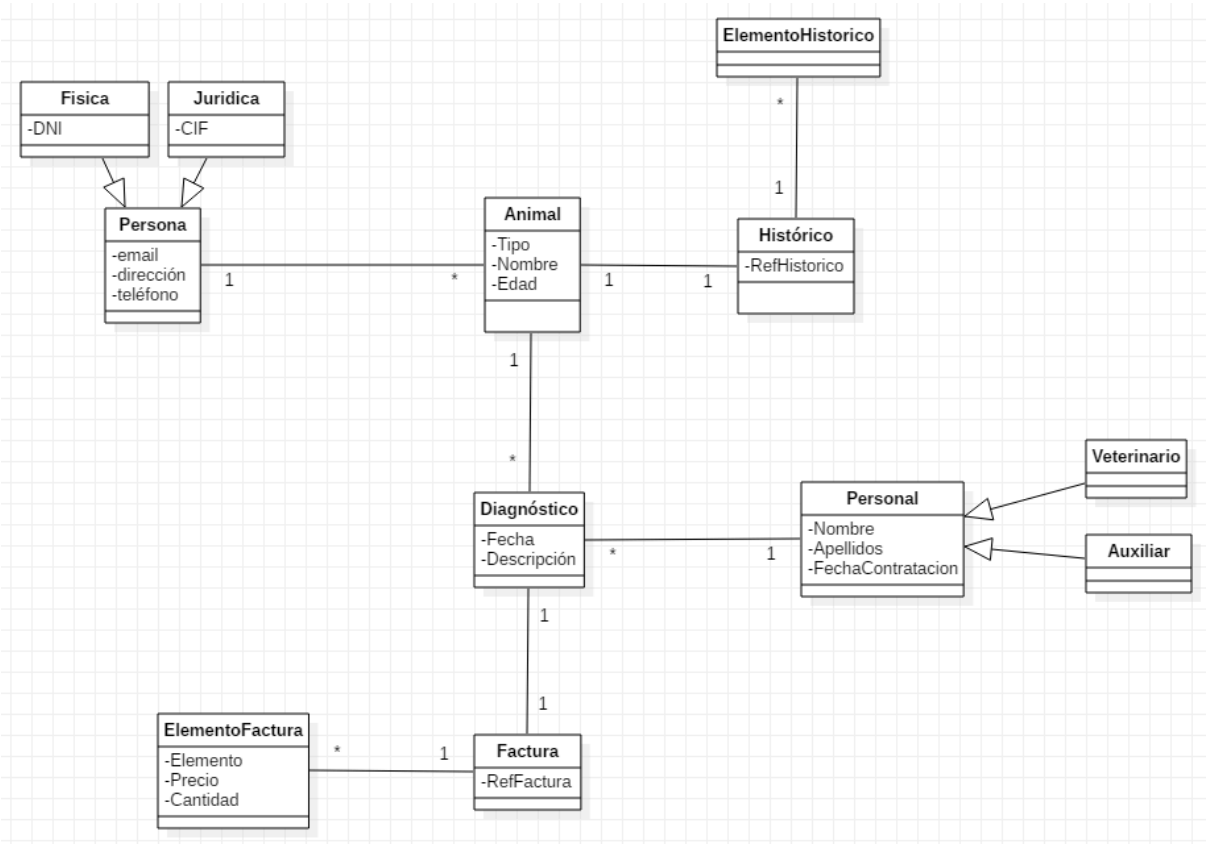
El primer paso es identificar los objetos primarios del sistema. Las clases suelen corresponder a sustantivos dentro del dominio del problema.

- Distingue las relaciones

El siguiente paso es determinar cómo cada una de las clases u objetos están relacionados entre sí. Busque los puntos en común y las abstracciones entre ellos; esto le ayudará a agruparlos al dibujar el diagrama de clase.

- Crea la estructura

Primero, agregue los nombres de clase y vincule con los conectores apropiados, prestando especial atención a la cardinalidad o las herencias. Deje los atributos y funciones para más tarde, una vez que esté la estructura del diagrama resuelta.



Anexo Ejemplos de Artefactos

Minuta de relevamiento

Previo al sistema

Actualmente el paciente se acerca al laboratorio en dos circunstancias: cuando el médico se lo pide (a través de una receta médica donde especifica que análisis deben hacerse) o cuando por su propia cuenta va a hacerse un análisis de forma particular. En ambos casos, el bioquímico le indica el día y la hora de la toma de muestra, así como los requisitos que tiene que tener en cuenta previamente para ese día (Ej: no comer carnes, no comer lácteos, no haber orinado, etc).

En el caso en el que paciente se acerque con intención de hacerse una extracción inmediata sin turno previo, si cumple con los requisitos para la toma de muestra en base a los análisis que desea hacerse, espera en la sala de espera hasta que no haya ninguna persona con turno en ese momento, y luego se procede a la toma de muestra. Caso contrario se le asigna un turno indicando los requisitos respectivos.

Al momento de asignarle un turno se le comenta al paciente cuando estarán disponibles los resultados para la entrega (generalmente es en el dia o al dia siguiente pero puede haber hasta una semana de demora). En el caso de que se haya realizado una toma de muestra sin turno previo se le comunica esto antes de proceder a la toma de muestra.

Con sistema

Cuando un paciente desea solicitar un turno para hacerse un análisis en un laboratorio específico, se dirige al portal web del laboratorio, accede al sitio con su mail y contraseña (que fueron previamente cargados al sistema por la administración del laboratorio), y luego ingresa a la sección de turnos. En esta sección el paciente selecciona día y horario conveniente y agrega una foto de la receta médica o, en su defecto, una descripción de los estudios que desea realizarse.

Luego de que el paciente envía la solicitud de turno, el personal administrativo revisa la receta médica o la descripción y agrega los estudios específicos a realizar. A medida que se agregan estudios, el sistema propone automáticamente las recomendaciones que debe seguir el paciente previo a la toma de la muestra. Todas las recomendaciones se agregan como texto en un mail de confirmación de turno. Una vez que el personal administrativo termina de cargar todos los estudios necesarios, confirma la solicitud del turno y el sistema envía el mail de confirmación a la casilla de correo del respectivo paciente.

Caso de uso ejemplo

CURN - Previo sistema

CURN 01: Registrar toma de muestra

Nivel de meta: **Resumen** Alcance del Caso de Uso: **Negocio**

META: Registrar turno y datos del análisis de un paciente.

ACTORES: Primario: Paciente Secundario: Bioquímico
Administrativo

PRECONDICIONES: <vacío>
DISPARADOR: El paciente desea realizarse un análisis
FLUJO DE SUCESOS:

Camino básico:

1. El paciente solicita realizarse un análisis indicando los detalles del mismo.
2. El administrativo indica al paciente instrucciones relativas al análisis
3. El paciente selecciona fecha y hora del turno.
4. El administrativo registra fecha y hora del turno.
5. El paciente se presenta en el turno indicado.
6. El bioquímico realiza la toma de la muestra.
7. El bioquímico registra los datos del análisis en el cuadernito.
8. El administrativo indica al paciente fecha y hora de disponibilidad de resultados.

Camino alternativo:

2a. El paciente desea realizarse un análisis inmediato:
2a.a: El paciente cumple con las instrucciones:
2a.a.1**: Ir al paso 6**
2a.b. El paciente no cumple con las instrucciones:
2a.b.1**: Ir al paso 3**

POSTCONDICIONES: Éxito: Se ha registrado el turno y los datos del análisis.
Éxito Alternativo: Se han registrado los datos del análisis.
Fracaso: <vacío>

CURS - Con sistema

CURS 01: Registrar toma de muestra

Nivel de meta: **Resumen** Alcance del Caso de Uso: **Sistema**

META: Registrar turno y datos del análisis de un paciente.

ACTORES: Primario: Paciente Secundario: Administrativo

PRECONDICIONES: El paciente se encuentra registrado en el sistema.
DISPARADOR: El paciente desea realizarse un análisis
FLUJO DE SUCESOS:

Camino básico:

1. El paciente inicia sesión en el sistema y solicita un turno en el portal de turnos. El sistema registra el turno solicitado
2. El administrativo confirma el turno en el sistema. El sistema envía al paciente las instrucciones relativas al análisis.

Camino alternativo: <vacío>

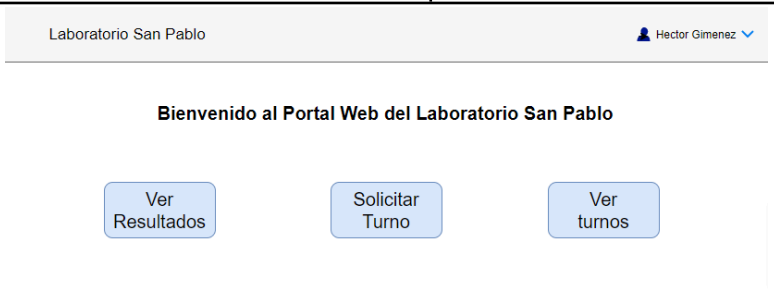
POSTCONDICIONES: Éxito: El sistema registró el turno confirmado.
Éxito Alternativo: <vacío>
Fracaso: <vacío>

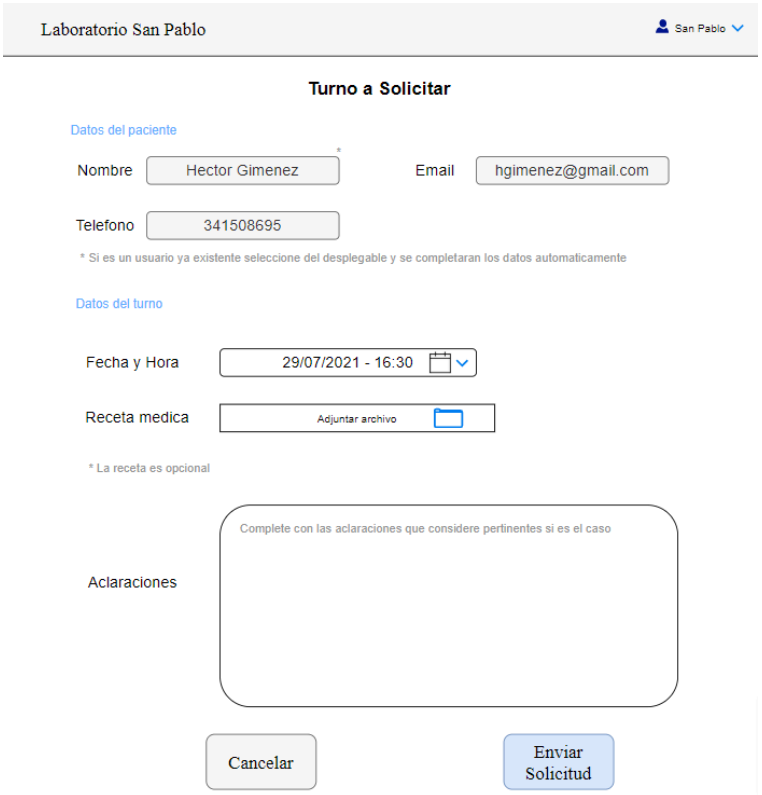
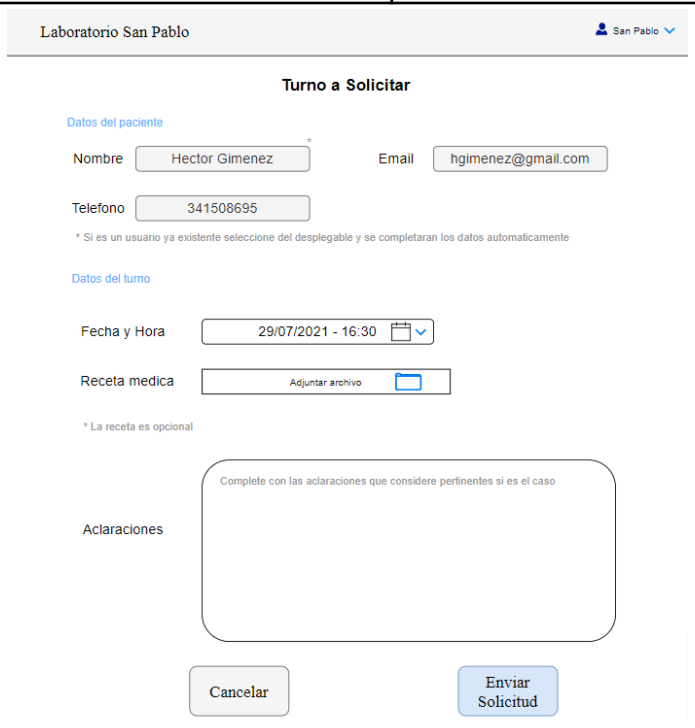
CUU - Reserva Turno del paciente

Caso de Uso: 007 – Reserva de turno del paciente

Nivel de meta**: Usuario** Alcance del Caso de Uso: **Sistema** Caja: **Negra**

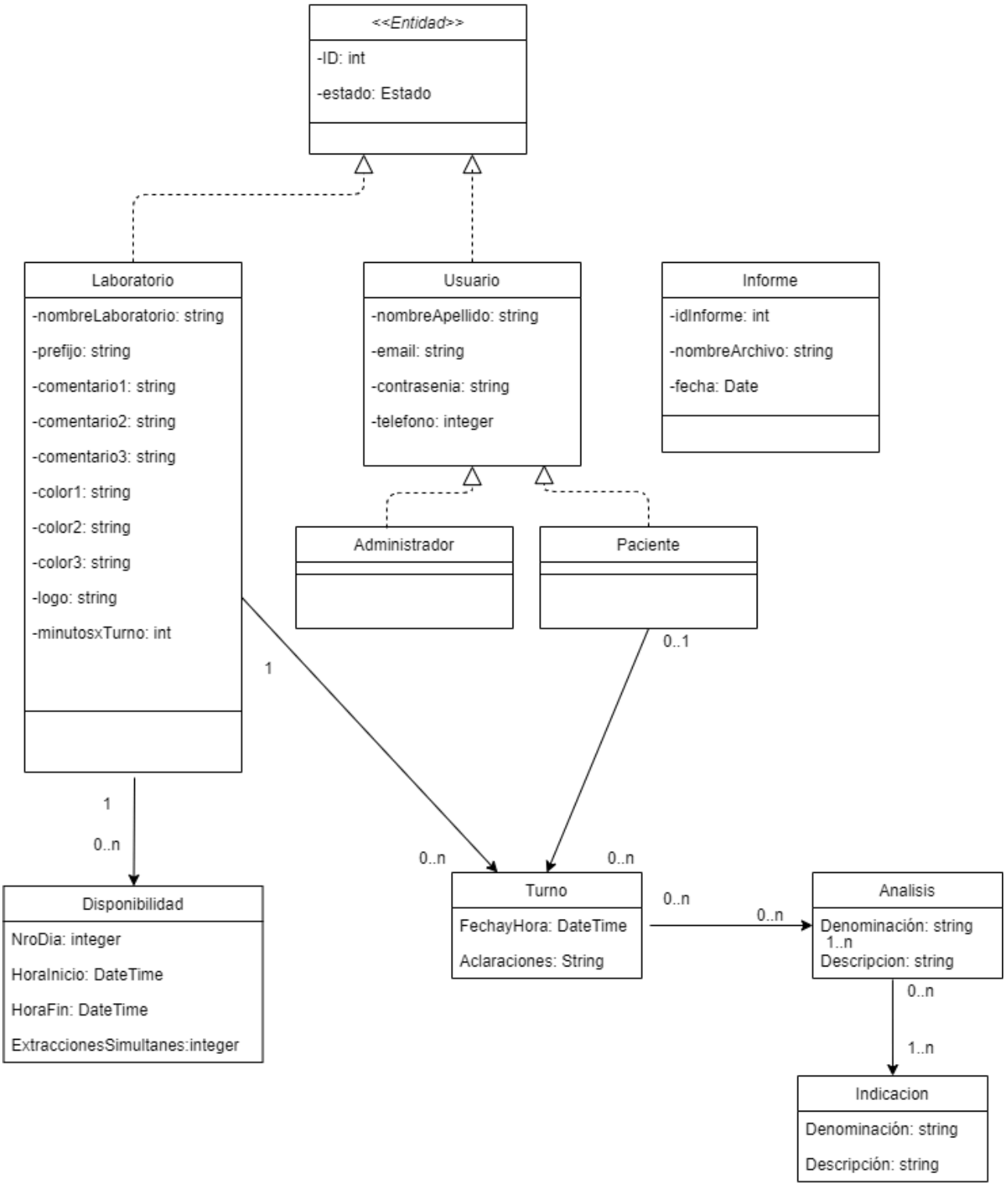
META: Reservar turno del paciente
ACTORES Primario: Paciente Secundario: <vacío>
PRECONDICIONES: Paciente tiene la sesión iniciada en PWR.
DISPARADOR: El paciente desea reservar un turno.
POSTCONDICIONES: Éxito: El paciente logró reservar el turno.
Éxito Alternativo: <vacío>
Fracaso: <vacío>

Paso		Usuario/Condición	Sistema
1		El paciente clickea el botón “Solicitar turno”.	El sistema muestra los datos para solicitar un turno con los datos del paciente
	Usu		

	Sist		
2		El paciente clickea el botón “Enviar Solicitud”.	El sistema registra el turno con estado de “Pendiente” y confirma el éxito de la operación al usuario
	Usu		

	Sist	 Laboratorio San Pablo San Pablo Turno a Solicitar Datos del paciente Nombre Hector Gimenez Email hgimenez@gmail.com Telefono 341508695 * Si es un usuario ya existente seleccione del desplegable y se completaran los datos automaticamente Datos del turno Fecha y Ho Receta med * La receta es Turno solicitado con éxito. Se le informara cuando este confirmado Aceptar Complete con las aclaraciones que considere pertinentes si es el caso Aclaraciones Cancelar Enviar Solicitud

Diagrama de clases



Actividad: Realizar un diagrama de clase más adecuado para la siguiente minuta:

Contexto

Un empresa de transporte envía camiones a todo el país para ello posee dos tipos de empleados, camioneros y logística, los empleados de logística se encargan de ubicar a los camioneros, trazar las rutas y documentar los viajes, por su parte los de transporte eligen si quieren realizar el viaje o no.

Proceso actual: Logística contacta por whatsapp al camionero y le pregunta si está disponible para el viaje, una vez aceptado el viaje, logística envía la carta de porte a través de whatsapp para el camionero y documenta el viaje en papel. Con esta metodología se vuelve difícil contactar a los camioneros, al pasar la carta de porte por whatsapp se vuelve susceptible a robos o falsificaciones, y la documentación muchas veces se pierde.

Con el sistema: Logística carga un viaje al sistema y asigna uno o más camioneros, una vez un camionero acepta el viaje, este cambia de estado. Al aceptar el viaje le llega una notificación a logística y este sube un pdf con la/s carta/s de porte, una vez terminado el viaje el camionero lo marca como Finalizado y este cambia de estado una vez más.

Domain Driven Design (DDD)

Conceptoos fundamentales

- Se popularizó en 2003 cuando Eric Evans publicó el libro Domain Driven Design.

- La idea es que se escriba código de tal forma que los expertos en el negocio/dominio puedan leerlo y verificar que es correcto.

- Tener un lenguaje ubicuo es fundamental, y esto se alcanza de tal forma que los términos utilizados en el código sean los mismos términos que se utilizan para hablar con los expertos del negocio.

- Se comienza con el diseño extragégico, donde se exploran los diferentes subdominios que tiene el sistema.
Luego se continúa con el diseño táctico, donde se entra en detalle sobre como se compone cada subdominio, identificando cada objeto de los mismos, principalmente las entidades y sus relaciones.

- El modelo de dominio es un artefacto fundamental. El mismo se crea utilizando un diagrama de clases, donde las clases son entidades del dominio del sistema.

- Patrones de diseño recomendados
    - Repository
    - Service

Video
[Domain-Driven Design in 150 Seconds]