

Metodología de la Programación II

Resumen Primera Parte

Emiliano Salvatori

Agosto 2019

1. Clase nº 1

1.1. Características de la Programación Orientada a Objetos (pantallazo)

Es un tipo de programación el cual el sistema de software se describe en relación a los objetos involucrados. Un objeto tiene atributos y comportamientos. Por ejemplo:

Objeto: Alumno

- *Atributo:* nombre, legajo, etc
- *Comportamiento:* mensajes que puede contestar el objeto ¿cual es el legajo?, ¿el nombre? etc. Lo que serian los setters y getters.

Todo se puede representar como un objeto y con un comportamiento.

Cuando se debe construir un sistema se deben dejar fuera muchos objetos. Sólo se tienen en cuenta los objetos que son los del sistema de software que se quiere dar solución, con lo que tiene que ver con lo que se debe estar dentro del diseño de la solución. Lo que compete a la problemática. Se deben definir los objetos que estarán involucrados en la solución.

Los objetos modelados se comunican entre sí enviándose mensajes.

Todos los objetos que tienen un mismo comportamiento se agrupan en clases. La clase se puede ver como un molde, algo general.

Para la representación en UML de cada objeto se dibuja mediante un rectángulo. El nombre de la clase en mayúscula. En la parte superior del rectángulo se listan sus datos miembro, cuya parte se denomina *vista interna*, *vista del implementador*, *estructura interna* y que no pueden ser directamente modificados sino es a través de sus funciones. Esto se denomina *encapsulamiento*: se denomina así al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.

La parte inferior del rectángulo se denomina *Protocolo de Mensajes* que NO se debe confundir con lo que comúnmente se denomina *zona de Métodos*.

Lo que se denomina Protocolo de Mensaje son los mensajes que entiende el objeto, el método es la IMPLEMENTACION del mensaje, de la función en sí. El objeto sólo puede responder al protocolo de mensajes que está definido en su clase.

Cuando se quiere construir un sistema, se debe:

1. Se definen los objetos y
2. Luego ver quién conoce a quién, las relaciones de conocimiento que existen entre ellos.

Por ejemplo se puede tener distintos tipos de alumnos: los de grado y los de postgrado, y eso se denomina *especialización*.

La especialización es cuando se va complejizando el objeto a definir. Cuando dos objetos comparten atributos, entonces esos atributos comunes se generalizan en una superclase, y de esa manera se puede construir de abajo para arriba del diagrama de clases.

Se denomina *instancia* a un objeto particular de la clase.

Tipo de datos: valores posibles de los datos, que a su vez condiciona las operaciones posibles que pueden manipular estos los datos.

Por ejemplo, los datos predefinidos pueden ser: int, char, double. Al mismo tiempo al elegir el tipo de dato, también se definen las operaciones de encapsulamiento: sumar, restar, etc. Se encapsulan determinadas funciones para determinados tipos de datos.

Se debe tener en cuenta que se denomina *TAGS* a las Estructuras de datos como ser: array, árboles, y demás estructuras.

Polimorfismo: dos objetos pueden entender un mismo mensaje de manera diferente, porque el método es diferente (la implementación del mensaje es diferente).

Delegación de mensaje: cuando un objeto transmite el mismo mensaje entre objetos.

Autodelegación: cuando el mismo objeto se delega a sí mismo.

Herencia: La Ventaja que tiene este tipo de técnicas: aminorar la cantidad de código. La Herencia se produce desde la Superclase hacia abajo. *Herencia Simple*: Lenguajes con herencia simple (como Smalltalk por ejemplo), se refiere cuando se hereda de una sola clase. *Herencia Múltiple*: cuando una clase hereda de dos o más superclases.

1.1.1. Características generales de POO

Algunas características que se pueden detallar son:

- El polimorfismo.
- (falta una)
- La herencia.
- El encapsulamiento.
- Binding dinámico o ligadura.
- Ocultamiento de la información (ocultamiento de la estructura interna). Es decir, desde un objeto de afuera no modifique directamente otro objeto sino a través de sus protocolo de mensajes.

Binding dinámico o ligadura

- **Ejemplo**: `x := 5;`
- *Estatico*: cuando se define en tiempo de compilación (y algo más).
- *Dinámico*: cuando se define en la ejecución. En ambos casos lo define el lenguaje.

Esto depende del lenguaje: Python es dinámico, Smalltalk es dinámico, pero el lenguaje C es estático. Si se define `x` como entero, no se le puede asignar un número flotante, porque da error.

1.2. Formas de encarar una solución de Software

Cuando se quiere comenzar a realizar las soluciones:

1. Se definen los objetos
2. Luego ver quién conoce a quién, las relaciones de conocimiento que existen entre ellos. Sólo se conocen aquellos objetos que pueden responder mensajes según su protocolo de mensajes.

En el desarrollo de una solución existen varias formas de representar las relaciones existentes entre objetos. Una de ellas son los Diagramas de Clases, como también los Diagramas de Secuencia. Éstos últimos son de colaboración en el que se dibuja para un mensaje todos los objetos que intervinieron para responder ese mensaje.

Un mensaje complejo es por ejemplo: comprar un boleto de avión. Para esta operación, intervienen miles de mensajes de miles de objetos por lo que se dificulta realizar un Diagrama de Secuencia abarcativo de ese mensaje

2. Clase n° 2

2.1. Metodologías ágiles

Las Metodologías ágiles son una forma de encarar un proyecto de software con respecto a las tradicionales. Es un conjunto de métodos. Las metodologías ágiles se aplican a proyectos dinámicos.

Estas metodologías al ser sistemas dinámicos que manejan cortos plazos, si se quiere saber lo que se quiere saber lo que se hará en 6 meses, no se puede con respecto a la tradicional. ¿Se aplican para cualquier sistema? No. Hay casos en los que se siguen requiriendo los tradicionales.

Pero es mucho más que una técnica ya que involucra un equipo multidisciplinario.

Las tradicionales se aplican cuando los sistemas de software son más estáticos. En las tradicionales se tienen más acento en la planificación.

2.1.1. Historia de las Metodologías ágiles

Las mismas tienen *Principios son 12* que son super generales. Si uno no se cumple, no se considera ágil, pero no se considera si se aplican todas las demas.

Aparecen a partir de los 70, pero la metodología ágil se diseño en el 2001, pero hubo antes algunos esbozos de este tipo de prácticas.

En la época de antaño se invertía mucho tiempo en la planificación y en la documentación, diagramas, refinamiento, etc. Luego de todo esto se comenzaba a codificar. Para este punto las modificaciones fueron tantas que lo entregado no era satisfactorio para el cliente.

Comenzaron a surgir prototipos, un estilo de maqueta para el cliente. Sobre el final se hacían las pruebas, por lo que si se quería hacer un cambio no se tiene en cuenta, por lo que es much más difícil.

En cambio los entregables son más flexibles, por lo que permite avanzar más rápidamente. Algunas metodologías son más flexibles a los cambios que pueda producir el cliente, aun dentro de la misma iteraciones y otros no (cuando no se aceptan más cambios en las iteraciones Se utiliza para comprometer más al cliente, para que defina más lo que requiere). Siempre el cliente está, algunas veces participa activamente y en otras no.

En comparación con la ingeniería de software tradicional, el desarrollo ágil se dirige principalmente a los sistemas complejos y proyectos con características dinámicas, donde las estimaciones precisas, los planes estables y las predicciones son a menudo difíciles de conseguir en las primeras etapas, y grandes... (de la filmina) La tradicional permite saber lo que se llegará a hacer dentro de 6 meses.

2.1.2. Particularidades

- Iteraciones deben durar de 1 a 4 semanas, y se produce TODO el entregable: Diseño, planificación, implementación, prueba, etc. Y se entrega todo funcionando.
- Cliente siempre presente, depende de la metodología depende de si forma parte activa o más pasiva. En el momento de la iteración algunas veces está para que le haga consulta y NO para modificar lo solicitado.
- Se trata de trabajar en el mismo lugar que esté el cliente. Lo mejor es que sea entre medio de 9 personas, todas las mañanas se hacen las reuniones de sincronización para explicar lo que está haciendo cada uno; todos saben lo que hace cada uno, qué problemas tuvo, qué va a hacer.
- Si se dividen en iteraciones, entonces se puede ir midiendo a medida que estas evolucionan y se entregan. Si hay una entrega que es difícil de llegar a tiempo, NO se toma personal extra, y TAMPOCO se hacen horas extras, sino que se subdividen los problemas, y se extienden los plazos.
- Diseños simples, sin complejizar las cosas, y refactorizar el código: para que quede un código prolijo y simple.
- Equipos autorganizados, equipos totalmente adaptativos a las circunstancias.
- En las pruebas se pueden hacer concurrentes a cada iteración. Se documenta lo mínimo e indispensable para que se lleve a cabo la iteración. Pero tampoco es bueno documentar tan poco.

Algunos consideran que las metodologías ágiles son muy extremas por lo que no se implementa, pero según el mercado como evoluciona rápidamente, se implementa de a poco en otros tipos de proyectos.

2.2. Diagramas

Diagrama de clases: Si dentro de una clase Auto, tengo un tipo de dato "motor" y esta tiene muchos datos que se le pueden atribuir, entonces acá se debe poner una relación de *composición*

Diagrama de Secuencia: Se debe tener en cuenta que los objetos de tipo "tabla de cuentas" sólo sirven para ser como paso intermedio antes de comunicarse con el objeto particular en sí mismo. Por ejemplo en el caso de que lo que se quiera modelar sea una entidad bancaria, y se quiera representar la evaluación de una cuenta, primero se comunica con la "tabla de cuentas" para saber si es que existe propiamente la cuenta y luego si es que existe, entonces se diagrama la relación del objeto llamador con la clase "cuenta" propiamente.

3. Clase nº 3

3.1. Programación Extrema (XP)

Programación Extrema: pone mayor énfasis en la adaptabilidad que en la previsibilidad (como lo suelen hacer las metodologías tradicionales). Se prefiere invertir tiempo en los cambios a medida que surgen que a generar planes. Con esto se ahorra de gastar energías en prever lo que va a suceder dentro de un largo plazo de tiempo (Las metodologías tradicionales tienden a gastar mayor cantidad de tiempo en prever el futuro).

3.2. Fases de la XP

Las Fases adoptadas por la Programación Extrema son las siguientes:

- **Exploracion:** se determina el alcance del proyecto se recaudan las historias de usuario y se determina cuánto tiempo va a durar el proyecto en base a las historias de usuario. La fase dura: aproximadamente 2 semanas.
- **Planificacion:** cliente, gerentes, desarrolladores, TODOS según la profesora. En otras metodología no se incluye el cliente para esta fase. Se determina el orden en que se va a ejectura, a qué historia se le da más bola, y a partir de ahí se determina la entrega. Una vez que se deterina la entrega comienza la fase de Iteración.
- **Iteración:** se implementan una parte de la historias de usuario. Algunas veces se redefine el orden de las historias. En una iteración se implementa las hitistirias y se entregan un software funcionando. Partes de la iteración: Analisis, diseño, programación pruebas. Cuando se hace cada itreación se hacen pruebas, todo el tiemp ose está probando lo nuevo y lo que ya se había probado. Iteración permite medir el avance del proyecto.
- **Historia de usuario:** tiene que tener la información necesaria para que se pueda estimar el tiempo que va a llevar desarrollarlo. No se documenta todo, lo menos posible como para que se entienda lo que se está haciendo. Cuando se implementa una historia de usuario el cliente tiene que estar activamente en cada implementación, ya que da ayuda. Ayuda activamente en la iteración el cliente. Las historias las escriben los usuarios en un lenguaje lo más sencillo posible. En ésta etapa Se analizan y se generan un plan de entregas. Recordar que las Historias de usuario sustituyen los casos de usos. Debe durar 1 y 3 semanas, si lleva más entonces se dividen, y si es menos se agregan nuevas a la iteración. Las historias de usuario se ordenan según el criterio del cliente. El soporte también lo dan el otro equipo, pero predomina el criterio del cliente.

¿Cuándo un sistema debería pasar a producción? Cuando está completo. No es lo que pasa habitualmente, a veces hay etapas donde se hace una puesta en desarrollo; en otros lugares directamente lo pasan a producción sin tener una mínima requisito y los errores se corrigen en producción. Como por ejemplo en software de liquidación de sueldos que se hacen sobre la marcha y la problemática lleva a que los sistemas se degeneren.

Cuando hay dificultades de definir los tiempos, hay programas que se denominan "Spikes" que sirven para ello.

La metodología XP se diferencian de las tradicionales por: Todas las personas trabajando en el mismo lugar, no hacer horas extras, tampoco incluir otras personas temporalmente porque se pierde más tiempo. Xp apunta a la simplicidad y a su funcionamiento.

Características fundamentales del código:

- Testeable
- Legible
- Comprensible
- Explicable

Refactorizacion: código simple y funcional. Reescribir las cosas que quedaron poco claras. XP sugiere refactorizar cuando sea necesario. Es para que quede más ordenado y legible.

3.3. SmallTalk y POO: tipos de mensajes