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 relacion 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 problematica. 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 internta* y que no pueden ser directamente modificados sino es a traves 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 conmúnmente se denomina *zona de Métodos*.

Lo que se denominao 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 quier construir un sistema, se debe:

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

Por ejemplo se pude 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 determinados 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 implementacion 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 polimosfirmo.
- (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 nº 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 dinamicos que manejan cortos plazos, si se quiere saber lo que se quiere saber lo que se haraá en 6 meses, no se puede con respecto a la tradiciconal. ¿Se aplican para cualquier sistema? No. Hay casos en los que se siguen requiriendo los tradicionales.

Pero es mucho mas 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 dentor 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 hagna 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; todo s 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 ques 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 qu quede un código prolijo y simple.
- Equipos autorganizados, equipos totalmente adaptativos a las circunstancias.
- En las pruebas se puden 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 metodologias 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 preveer lo que va a suceder dentro de un largo plazo de tiempo (Las metodologías tradicionales tienden a gastar mayor cantidad de tiempo en preveer 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

3.3.1. Unarios (sin argumentos)

Operación:

3!

- **Receptor**: 3
- **Mensaje**: Factorial
- **Selector**: Factorial
- **Valor de retorno**: 6

3.3.2. Binarios

Este tipo de operaciones contienen un sólo argumento, siendo de tipo matemático lógicas.
Operación:

$$3 < 5$$

- **Receptor**: 3
- **Selector**: ¡
- **Argumento**: 5
- **Valor de retorno**: True

3.3.3. De palabra clave

Operaciones que pueden tener uno o más argumentos.
Operación: #(3 5 7 1) at: 2 put:9

- **Receptor**: #(3 5 7 1)
- **Selectores**: at y put
- **Argumentos**: 2 y 9
- **Mensaje**: at: 2, put:9
- **Valor de Retorno**: #(3 9 7 1)

Lo que realiza esta operación es poner el valor 9 en la posición nº 2 del array creado.

3.3.4. Orden de Resolución de los mensajes

1. Los paréntesis (si hay paréntesis siempre se resuelve primero eso)
2. Los mensajes unarios, de izquierda a derecha
3. Los mensajes bianrios, de izquierda a derecha
4. Los mensajes de palabra clave, de izquierda a derecha

Debe existir un orden para realizar las operaciones ya que en una misma línea, es decir en un mensaje anidado, el lenguaje debe saber cómo resolver primero qué parte, por ello requiere tener una lógica; hay que saber cómo SmallTalk lo resuelve. El lenguaje requiere que cada linea termine con un "."(punto)

3.3.5. Ejemplo de mensajes y operaciones

Unarios:

- 5!.
- 19,75 rounded.
- "abcd" size.

Binarios:

- 'abc' n = 'def'
- 3 ¿5
- true and false

De Palabra clave:

- #(3 5 7 1) at: 2 put: 9
- 5 between: 8 and 10.

3.4. Mensajes anidados

Ejemplo de un mensaje anidado: *2 factorial negated*. En este ejemplo se puede observar que hay dos mensajes unarios integrados, anidados.

Lo primero que se resuelve es *2 factorial* porque se resuelve de izquierda a derecha.

Orden de resolución:

1. **Receptor**: 2
2. **Mensaje**: factorial
3. **Valor de retorno**: 2

Se resuelve segundo: "2 nagated"

1. **Receptor**: 2
2. **Mensaje**: negated
3. **Valor de Retorno**: -2

Ejercicio de la práctica:
Operación:

$$3 + 4 * 6 + 3$$

Contiene varios mensajes binarios.

Resolución del primer mensaje:

$$3 + 4$$

1. **Receptor**: 3
2. **Mensaje**: + 4
3. **Selector**: +
4. **Argumento**: 4
5. **Valor de Retorno**: 7

Resolución del segundo mensaje (de izquierda a derecha):

$$7 * 6$$

1. **Receptor**: 7
2. **Mensaje**: * 6
3. **Selector**: *
4. **Argumento**: 6
5. **Valor de Retorno**: 42

Resolución del tercer mensaje:

$$42 * 3$$

1. **Receptor**: 42
2. **Mensaje**: + 3
3. **Selector**: +
4. **Argumento**: 3
5. **Valor de Retorno**: 45

Recordar que Smalltalk NO tiene prioridad sobre los operadores como puede ser C o Pascal, siempre resuelve las cosas de izquierda a derecha. Por lo que el programador debe poner paréntesis en las operaciones a realizar.

Ejercicio de la práctica

Operación:
5 between: 1 and: 3 squared + 2 factorial
Observar que la operación completa contiene 3 mensajes:

1. **1er mensaje**: 3 squared = 9
2. **2do mensaje**: 2 factorial = 2
3. **3er mensaje**: 5 between: 1 and: 1er + 2do
4. **Resultado**: 5 between: 1 and: 9 + 11 = true

Estructuras de control

Condicionales

La estructura sería la siguiente: (expresión booleana) *iftrue:* [Bloque verdadero]
iffalse: [Bloque falso]
Explicación de lo anterior: Se resuelve primero la expresión booleana que está en entre paréntesis, que devueve true o false los cuals son una instancia de la clase booleana. Esta instancia de booleana se le está enviando un mensaje *iftrue, iffalse* ¿Dónde está implementado ese mensaje? Está definido en su protocolo de mensaje en la clase o superclase, porque hay herencia, lo puede tomar porque lo entiende.

Bloques es todo lo que está entre corchetes. Si el que recibe es una instancia verdadera, entonces recibe el mensaje bloque verdadero, si el que recibe es la instancia falso, entonces recibe el mensaje bloque falso.

¿Qué diferencia hay entre una función y los bloques? La función puede devolver un resultado, puede tener argumentos, tiene un nombre y necesitan ser invocados. En cambio los bloques son anónimos y se activan cuando el control pasa por ellos, no requieren ser invocados. Recordar que son los que están entre corchetes.

1. La expresión booleana es evaluada, dará como resultado un objeto true o false.
2. Si el resultado es true el mensaje iftrue: con sus argumentos serán enviados al objeto true.
3. El objeto true responde el mensaje evaluando el [bloque verdadero].
4. Si el resultado es false el mensaje iffalse: con sus argumentos serán enviados al objeto false.
5. El objeto false responde el mensaje evaluando el [bloque falso].

Repetición Condicional

La estructura sería la siguiente: [expresión booleana] *WhileTrue:* [Cuerpo del loop]
[expresión booleana] *WhileFalse:* [Cuerpo del loop]
iffalse: [Bloque falso]
Interpretación del whileTrue:
No se tiene paretensis, por lo que el mensaje whiltrue se envia a la expresión booleana, se evalúa, si el objeto es true entonces se evalua el cuerpo del loop.

1. El mensaje whileTrue es enviado al bloque 'expresión booleana'.
2. Como respuesta, se evalúa el bloque.
3. Si el bloque retorna el objeto true entonces se evalúa el cuerpo del Loop y el mensaje whileTrue es nuevamente enviado al bloque de la expresión booleana.
4. Luego se repiten los pasos 1, 2, 3.

Repetición de Longitud física

Estructura de la condición:
ValorInicial to: *valorFinal* do: [:variable del loop — cuerpo del loop]
El mensaje to: do: evalúa el argumento en bloque para cada entero que esté en el intervalo dado por el valor del receptor, hasta el valor final incluido

Ejercitación de mensajes en Smalltalk

1. 'casa' isNil.
2. 9 + 3 * 2.
3. true is false.
4. #(12 65 'olas' true) includes: 'viento'.
5. 3 * 2 squared
6. 4 + 2 negated between: 3 + 4 * 5 and: 'hello' size * 10

Resolución:
Operación: "casa" isNil
Tipo de mensaje: Unario

- **Receptor**: "casa"
- **Mensaje**: isNil
- **Selector**: isNill
- **Argumento**: -
- **Valor de Retorno**: false

Operación: $9 + 3 * 2$
Tipo de mensaje: Varios mensajes Binarios
Primer mensaje de Izquierda a Derecha: $9 + 3$

- **Receptor**: 9
- **Mensaje**: + 3
- **Selector**: +
- **Argumento**: 3
- **Valor de Retorno**: 12

Segundo mensaje de Izquierda a derecha:
Operación: $12 * 2$

- **Receptor**: 12
- **Mensaje**: * 2
- **Selector**: *
- **Argumento**: 2
- **Valor de Retorno**: 24

Operación: true is false
Tipo de mensaje: Binario

- **Receptor**: true
- **Mensaje**: if falase
- **Selector**: is
- **Argumento**: false
- **Valor de Retorno**: false

Operación: #(12 65 "olas" true) includes: "viento"
Tipo de mensaje: de Palabra Clave, ya que lleva el caracter ":"

- **Receptor**: #(12 65 "olas" true)

- **Mensaje**: includes "viento"
- **Selector**: include
- **Argumento**: "viento"
- **Valor de Retorno**: false

Operación: 3 * 2 squared
Segundo mensaje realizado:
Operación: 3 * 4

- **Receptor**: 3
- **Mensaje**: * 4
- **Selector**: *
- **Argumento**: 4
- **Valor de Retorno**: 12

4. Clase nº 4

4.1. Programación extrema

En la Programación Extrema (XP) el cliente se encuentra presente en TODO el proceso, y tiene una participación activa (es lo que diferencia esta metodología con respecto a otra). En otras metodologías el cliente es pasivo, porque no tiene incidencia en los desarrollos.

Las **Historias de usuario**: son escrita por el cliente, y los desarrollos. Ademas asigna la prioridad porque es el que más entiende.

Planificación: el cliente negocia lo que se entrega porque entiende lo que es la funcionalidad. Y sabe cuándo está completo el dsarrollo porque sólo él tiene el conocimiento completo del negocio.

¡Los roles dentro de XP son muy importantes! Las tareas que requiere cada uno de las partes del equipo.

Estandares de código: se tiene que mantener fácil para realizar refactorización. Las convenciones ayudan a entenderse mjor entre todo el desarrollo del equipo.

Implementación dirigida por prubas: característica muyu importante en XP, ya que en otras meto se prueban todo al final, en XP se desarrollan los test desde el comienzo. Se define antes de codificar cuáles son los resultados a los que se quiere llegar.

Gracias a esta forma de desarrollar los test, se dicen que los test dirigen el proceso (implementación dirigida por pruebas). Se apuntan a esos desarrollos. A cada modificación de la funcionalidad, también se requiere que se modifiquen los test, ya que van de la mano.

Programación en pareja: dos personas en la MISMA computadora, lo que se apunta son a códigos más eficientes, se detectan los errores antes de que surgan, se a la implementación mayor calidad en el código.

Ventaja:

- Los errores se detectan antes, los diseños mejores y código más corto, el equipo resuelve más rápido las dificultades, se termina con más personas conociendo el código a detalle. A diferencia de las metodologías anterires donde cada programador sólo hacía una parte del sistema.
- Otra Ventaja es que las personas aprenden a trabajar en grupo, es más dinamico, las personas disfrutan más de su trabajo.

Integración secuencial: todos tienen que trabajar con la última versión. Todo lo que se sube tiene que estar probado y todo tiene que estar funcionando correctamente.

Propiedad colectiva: animar a todos a proponer nuevas ideas.

Ritmo constante: beneficioso mantner un ritmo en conjunto. Se define cuantas historias entras en una realease. todo esto para no sobrecargar al equipo, para tner un poco de estimar los tiempo de entregas; cuando se encuentra un problema en la estimación, se alarga la entrega. No se hace hors extra ni se llama a nadie mas.

Características importantes:

- Simplicidad: no hacer más de lo que se tiene que hacer
- Comunicación: el cara a cara.
- Retroalimentación: retro, mejorar el equipo para procesos futuros

- Coraje: responsabilidad en las tareas que se asumen.

Roles (importantes):

- **Cliente**: responsable de conducir el proyecto, lo define y también sus objetivos. Tiene que saber determinar el valor d las historias de usuario en el sistema. El cliente representa al usuario final y a los intereses económicos de la emprsa. Ver bien los Derechos y sus responsabilidades.
- **Programador**: responsablidad de tomar decisiones técnicas. Estiman las historias de usuario. Ver bien los Derechos y responsabiidades.
- **Encargado de pruebas (tester)**: Ayuda al cliente para escribir los test de las historias de usuario, informa al equipo de ello.
- **Encargado de seguimiento (tracker)**: Hace el seguimiento de acuerdo de la planifiicación. Pregunta uno por uno a los usuarios qué onda con las cosas con las que se comprometieron. La velocidad ayuda a mejorar el
- **Entrenador (coach)**: hace que cada persona dé lo mejor de sí para la empresa. No se entrena para un trbajo específico, lo que hace es prerpara al equipo en las prácticas. Se entrena en la metodología que se irá a aceptar.
- **Gestor (big boss)**: El big boss es el gerente del proyecto. Debe tener una idea generalizada del proyecto y esar relacionado con su estado.

5. Clase nº 5

5.1. Scrum

IMPORTANTE: De scrum lo más importante son las etapas y lo que salen de estas: el gráfico, las etapas, la relación con el cliente y las etapas.

No hay que darle pelota a la historia. Hay que hacer algún tipo de apunte con lo que tiene que ver con los roles, y demás cosas. Lo que interesa las fases que son propias de Scrum.

El cliente no participa activamente durante la iteración ¿por qué? Participa en la planificación de la iteración, XP es más flexible, scrum NO. En scrum debe decirir el cliente lo que requiere para definir, por lo que lo compromete de forma más enérgica para poder terminar de entregar tangible, para que pida las cosas que realmente necesite.

Planificación de la iteración: Lista de requisitos del proyecto lo da el cliente. El equipo toma las cosas y compromete a realizarlas. Se llega a un acuerdo entre lo que se pide y lo que se va a desarrollar.

De la segunda parte de la reunión sale la *Lista de tareas* de iteración. Se realiza una estimación conjunta del esfuerzo necesario. Cada uno se compromete a las tareas que SOLO PUEDE realizar. Es una diferencia con respecto a otras metodlogías, ya que aquí cada miembro se tiene un mayor compromiso con el proyecto en general. El Equipo asume la responsablidad, cada persona se responsabliliza las tareas que se asigna.

Iteración: Ejecución del sprint (iteración): 2 a 4 semanas(sin diferencia de XP). Se entrega un todo como XP.

Las **reuniones de sincronización/dailys** son iguales en XP, son como las diarias en XP. Cada uno dice lo que hizo, lo que le falta hacer, etc. Algunas empresas son más estrictos en aquellas personas que no logran los objetivos a los que se comprometieron. El tiempo máximo es de 15 minutos (todo esto es igual en XP).

Facilitador /scrum master: quita obstáculos para que elequipo trabaje más rápido y sin problemas, sin conflictos externos ni tampoco internos. De que se no merme su productividad.

Restricciones: No se pueden cambiar los pedidos. ¿Cuándo se puede cambiar? Cuando ya no tiene sentido; cuando por ejemplo se cambia el contexto, cuando es más perjudicial seguir trabajando en la iteración que cambiar de rumbo (terminación anormal). Comprometerse al cliente facilita que se cumpla con su responsablidad de conocer qué es lo más prioritario.

Lo característico de Scrum es que permite visualizar el avance del equipo mediante el gráfico **burndown chart**. Es un gráfico que permie saber dónde se está posicionado, hacia dónde se puede entregar, cuál es la fecha que uno se comprometió, qué se puede hacer para cumplir con las fechas estipuladas, se pueden hacer todas estas simulaciones. Sólo mirando el gráfico se puede saber en qué etapa estás, si se está atrasado con los que se comprometió. El grafico permite hacer las correciones necesarias.

Responsabilidades:

- **Lista de requisitos** la define el cliente.
- **Lisa de tareas** la define el equipo.

Demostracion de requisitos: se le muestra los resultados obtenidos de la iteración al cliente, a muy alto nivel. El cliente en esta etapa realiza las modificaciones pertinentes que él requiere. ESE es el momento que el cliente puede realizar alguna planificación en los cambios y esos cambios serán realizados en la proxima iteración.

Retrospectiva: intervienen el equipo realizando una crítica para mejorar en las siguientes etapas. El facilitador se encarga de ser una persona que permite limar asperezas.

Replanificación del proyecto: En un principio se puede haber pensado determinadas pero en este momento el cliente puede replanificar todos los requisitos.

Si en el medio de la iteración cambió algo, no se puede cambiar, sino hasta el final. Sólo en el final se puede re-planificar, se puede abortar cuando no tiene sentido seguir.

El cliente puede cancelar todo el proyecto en caso de perder dinero si siguen en pie los desarrollos.

Estimación: El equipo debe tener una estimación de lo que lleva cada tarea, de cada uno delos requisitos.