1 Introducción a la Ingeniería de la Programación

Si un antropólogo del futuro analizase los hábitos de las sociedades occidentales de principios del siglo XXI, observaría un entusiasmo desaforado de las hoy llamadas tecnologías de la información. Muchas de éstas se basan en la producción de software. Estos ciudadanos esperan nuevas prestaciones de sus teléfonos móviles, de sus diagnósticos médicos, de sus coches, de sus electrodomésticos,... En fin, hasta los medios de comunicación se hacen eco de la presentación de los nuevos sistemas operativos, de aplicaciones 3D o de videojuegos.

Tras estos deseos, se encuentran millones de desarrolladores intentando introducir más inteligencia a los artefactos del siglo XXI. Las abigarradas comunicaciones entre personas o entre máquinas, los controles de cientos de millones de procesadores sobre los sistemas productivos, las millares de aplicaciones de ocio, las transacciones económicas y un sin fin de aplicaciones están basadas en la introducción de conocimiento a través del software.

Pero la producción de software es una disciplina de la Ciencia muy reciente, de unas pocas décadas. Todavía no ha habido una estructuración formal de este conocimiento humano. A diferencia con otras ingenierías no hay una metodología que tenga procedimientos matemáticos que indique el camino más correcto a seguir en la elaboración del software.

Recientemente, los investigadores de las Ciencias de la Computación han decantado una metodología que ayuda a la creación del software. Durante los últimos treinta años ha habido múltiples experiencias cuantitativas (nuevos lenguajes de

programación, método de representación y procedimientos de producción software) que han dado paso a un salto cualitativo produciéndose un nuevo paradigma en esta disciplina. Para mostrar este salto y ver su cercanía en el tiempo, sólo es necesario indicar tres referencias bibliográficas: en 1995 la Pandilla de los Cuatro (*Gang of Four, GoF*) publicaron su famoso libro de "*Patrones de diseño*", editado en castellano en el 2003; UML 1.0 aparece en 1997 y el Proceso Unificado en 1998.

Este curso trata precisamente de dar la metodología necesaria para la producción del software de principios del siglo XXI. Los siguientes capítulos mostrarán cómo hacer software siguiendo el Proceso Unificado, abarcando los aspectos desde la recogida de los requisitos hasta las etapas de análisis, diseño e implementación.

Se inicia este temario analizando qué es el software, cuales son sus características y cómo se ha evolucionado hacia el paradigma orientado a objetos, para encuadrar definitivamente el ámbito del curso: el Proceso Unificado, las herramientas de UML y el diseño con patrones.

1.1 Qué es el software y la ingeniería del software

Según la definición de la IEEE, un sistema software es:

"la suma total de los programas, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo, esto es, un conjunto integrado de programas que en su forma definitiva se pueden ejecutar, pero comprende también las definiciones de estructura de datos (p.ej. tipos de ficheros, acceso bases de datos, interacción con otros componentes) que utilizan estos programas y también la documentación referente a todo ello"

El Software no es una obra de arte, sino un producto de consumo utilitario y masivo. Es un producto industrial. Pero tiene características especiales. No es una producción en serie, es un producto singular y extremadamente complejo. Muchos autores dicen que la producción del SW se parece a la construcción de edificios. Cada uno requiere su propio proyecto.

Otra característica del SW es que no se *"estropea"* por el uso ni por el paso del tiempo. Aunque hay problemas relacionados con el mantenimiento y también está la obsolescencia del producto SW por el avance de estas tecnologías.

Resumiendo, los productos de programación se hacen a medida, representan el conocimiento introducido a la máquina, no se rompen aunque resultan difíciles de mantener y son un producto industrial que no se fabrica pero que se desarrolla.

A cada tipo de producto industrial le corresponde un tipo de ingeniería, entendida como el conjunto de procedimientos, técnicas y herramientas, que se emplean para desarrollar el producto. Una técnica es la manera preestablecida en la que se lleva a término un paso en la elaboración del producto. Un procedimiento es la manera de aplicar varias técnicas sucesivamente. Por último, una herramienta es un instrumento de cualquier tipo que se utiliza en la aplicación de una técnica.

En este contexto, la Ingeniería del Software es la rama de la Ingeniería que aplica los principios de la Ciencia de la Computación y las Matemáticas para lograr soluciones costo-efectivas a los problemas de desarrollo del software. El proceso de la

Ingeniería del Software se define como un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un software de calidad. Este desarrollo se traduce en requerimientos del SW, estas especificaciones son transformadas en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo.

En la tabla adjunta se compara, a modo de ejemplo, dos tipos de Ingenierías diferentes. Una la aplicada en proyectos de Control y la otra la que se empleará en la construcción de sistemas orientado a objetos (OO), la cual será el objetivo de este curso.

Tabla 1.1 Comparación entre las Ingenierías de Control y las de Sistemas orientado a objetos

Control	Ingeniería SW (AOO/D)
Especificaciones del Diseño	Captura de los requisitos
Identificación y modelado	AOO (investigación del dominio)
Diseño (técnicas de compensación)	DOO (Definición de los diagramas de clases de diseño y de los diagramas de interacción)
Simulación e implementación	Modelo de implementación (Programación eXtrema)

Entre las categorías de tipos de software, en una primera aproximación, se podrían clasificar en: a) Sistemas Operativos, b) Tiempo real, c) Gestión, d) Empotrados y e) Ingeniería y científico. Sin embargo, en este temario lo que interesa es el desarrollo y proceso de la construcción de los sistemas orientados a objetos, los cuales pueden englobar a la mayoría de tipos de SW. Los sistemas operativos actuales son OO, muchas aplicaciones en tiempo real están basadas en programación OO y así podría citarse cada una de las categorías referenciadas. Este salto se debe a que actualmente el paradigma de programación se basa en tecnologías orientadas a objetos. Las razones de esta elección se presentan en el siguiente apartado.

1.1.1 Los grandes problemas de la ingeniería del SW

La calidad y la productividad del SW todavía no han alcanzado niveles comparables con otras tecnologías. En cuanto a la calidad, la causa principal es la gran complejidad. La falta de modelos formales para saber cómo introducir conocimiento en las máquinas es un gran inconveniente. Respecto a la productividad, el nivel más alto en la fabricación se da en la producción en serie y no en la generación de un producto singular. Además, los proyectos SW suelen empezar de cero. Si se piensa en la construcción de un edificio, ejemplo típico de producto singular, hay muchos elementos que son comunes: materiales, instalaciones, planos, cálculos, ... Una de las razones de haberse decantado por la tecnología OO es por haber empezado a utilizarse software prefabricado. Las vías ofrecidas por la POO para la reutilización del SW son:

1. Componentes: los lenguajes OO ofrecen servicios de alto nivel mediante un universo computacional repletos de objetos servidores (ActiveX, COM, ...).

2. Diseño con patrones: recetas para solventar problemas que aparecen en muchos proyectos.

3. Marcos de trabajos (*Frameworks*): Bibliotecas de servicios disponibles contrastadas y de calidad para enchufar a otros SW (STL, Qt, .NET, ..).

Se deja al lector que resuelva las siguientes cuestiones:

1. ¿Por qué lleva tanto tiempo programar?
2. ¿Por qué es tan caro?
3. ¿Por qué no se puede hacer sin errores?
4. ¿Por qué no se puede planificar?
5. ¿Por qué no hace lo que se esperaba?
6. ¿Por qué es difícil modificar el SW?
7. ¿Por qué un aumento de los programadores no ayuda?

1.2 El paradigma orientado a objetos

El concepto de *paradigma* en la Ciencia fue dado por Kuhn en *La estructura de las revoluciones científicas* (1962):

> *Un paradigma es aquello que los miembros de una comunidad científica, y sólo ellos, comparten y a la inversa, es la posesión de un paradigma común lo que constituye a un grupo de personas en una comunidad científica [...]. Una teoría científica se declara inválida sólo cuando se dispone de un candidato alternativo para que ocupe su lugar. La decisión de rechazar un paradigma es siempre, simultáneamente, la decisión de aceptar otro.*

La llegada a la programación orientada a objetos proviene de múltiples intentos de mejorar la producción del SW. La Programación Orientada a Objetos (POO) como paradigma es una forma de pensar, una filosofía, de la que surge una cultura nueva que incorpora técnicas y metodologías diferentes. La POO percibe el universo computacional como poblaciones de objetos, cada uno responsabilizándose de sí mismo, y comunicándose con los demás por medio de mensajes.

El paradigma OO se basa en el concepto de objeto. Un objeto es aquello que tiene estados (atributos y su evolución en el tiempo), comportamientos (acciones y reacciones a los mensaje) e identidad (propiedad que lo distingue de los demás objetos). La estructura y comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables. Una clase es un conjunto de objetos que comparten una estructura y comportamiento común. La diferencia entre un objeto y una clase es que un objeto es una entidad concreta que existe en el tiempo y en el espacio, mientras que las clases son representaciones de unas abstracciones, las esencias de los objetos.

Estas ideas de clase y objetos provienen del espíritu cognitivo de los seres humanos. Así, al nombrar la clase "perro", la mente humana asocia a un conjunto de objetos caracterizados por ser animales mamíferos, con cuatro patas, un rabo, … Uno de los triunfos de las tecnologías OO es su correspondencia con la biología evolutiva.

Pero antes de entrar en la tecnología OO, se verán las razones de haber llegado a este punto de la evolución en los lenguajes informáticos.

1.2.1 Programación estructurada y programación orientada a objetos

Una de las cuestiones más importantes que hay que tener en cuenta al construir un programa es el control de su ejecución. Son raros los programas que constan de un conjunto de instrucciones que se ejecutan sucesivamente una tras otra. En la mayor parte de los casos, diversas partes del programa se ejecutan o no, dependiendo de que se cumpla alguna condición. Además, hay instrucciones (los bucles) que deben efectuarse varias veces, ya sea en número fijo o hasta que se cumpla una condición determinada. Los lenguajes más antiguos (como Fortran) se apoyaban en una sola instrucción para definir el control de los programas: la instrucción *GOTO* (del inglés *go to*, que significa *ir a*). Ésta corresponde con la transferencia incondicional de los lenguajes simbólicos. También, se le puede combinar con alguna expresión condicional, permitiendo bifurcar el orden de ejecución de las instrucciones.

Pero las etiquetas numéricas y las transferencias arbitrarias hacen los programas muy poco legibles y difíciles de comprender. Como primer intento de resolver el problema, en lenguajes más modernos se recurrió a utilizar etiquetas no numéricas, igual que se hace en los lenguaje simbólicos de segunda generación.

Sin embargo, a finales de los años sesenta surgió una nueva forma de programar que reduce a la mínima expresión el uso de la instrucción *GOTO* y la sustituye por otras más comprensibles.

Un famoso teorema, demostrado por Edsger Dijkstra (*http://www.cs.utexas.edu/users/UTCS/notices/dijkstra/ewdobit.html*) en los años sesenta, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:

1. El bloque secuencial de instrucciones: instrucciones ejecutadas sucesivamente.

2. La instrucción condicional alternativa, de la forma "IF condición THEN instrucción-1 ELSE instrucción-2". Si la condición se cumple, se ejecutará "instrucción-1". En caso contrario se ejecuta "instrucción-2". Abreviadamente, esta instrucción se suele llamar IF-THEN-ELSE.

3. El bucle condicional "WHILE condición DO instrucción", que ejecuta la instrucción repetidamente mientras la condición se cumpla. En su lugar, se puede utilizar también la forma "UNTIL condición DO instrucción", que ejecuta la instrucción hasta que la condición se cumpla. Los dos bucles se diferencian entre sí porque en la forma WHILE la condición se comprueba al principio, por lo que es posible que la instrucción no se ejecute ni una sola vez. En cambio, en la forma UNTIL la condición se comprueba al final del bucle, por lo que la instrucción se ejecuta siempre al menos una vez.

Los programas que utilizan sólo estas tres instrucciones de control básicas o sus variantes (como los bucles FOR o la instrucción condicional CASE), pero no la instrucción GOTO, se llaman **estructurados**. La **programación estructurada** (llamada también "programación sin GOTO") se convirtió durante los años setenta en la forma de

programar más extendida. A pesar de todo, la mayoría de los lenguajes conservan la instrucción *GOTO* y las etiquetas de las instrucciones, para utilizarla en casos muy especiales, aunque normalmente se desaconseja su uso.

Entre los lenguajes de alto nivel, Pascal, C y Ada pueden considerarse especializados en programación estructurada, y aunque todos ellos permiten utilizar la instrucción *GOTO*, se desaconseja su uso (aunque en C es una práctica común utilizarlo para manejo de errores).

Los métodos estructurados provienen de la programación estructurada. Se caracterizan por:

- La especificación de los procesos y la de las estructuras de datos generalmente quedan bastante diferenciadas.
- Las técnicas de análisis y diseño o bien pasa de lo general a lo particular o a la inversa.

Las técnicas más usadas son los diagramas de entidad-relación y de flujos de datos. Los primeros se refieren a los datos y los segundos a los procesos.

Diferente a la **programación estructurada** es la programación orientada a objetos. Esta última se enfoca en reducir la cantidad de estructuras de control y reemplazarlo con el concepto de polimorfismo. Aún así los programadores todavía utilizan las estructuras de control (*if, while, for,* etc...) para implementar sus algoritmos, porque en muchos casos es la forma más natural de hacerlo.

Resumen

Algoritmos + datos = programa (programación estructurada)

Servicios + atributos = objeto (programación orientada a objetos)

Pero estas diferencias en sus apariencias son mucho más profundas y de un gran calado. Resulta fácil de citar los conceptos de abstracción, ocultación de la información, reutilización del software, polimorfismo y otras palabras claves que todos los autores de programación orientada a objetos comentan. Pero éstas quedarán en el olvido si sólo se pasa de codificar de un lenguaje estructurado a otro orientado a objeto.

Para entender qué es la programación orientada a objetos hay que tratarla desde el punto de vista del método y enmarcarse en su paradigma. Por tanto, hay que examinar lo que los investigadores acaban de destilar después de más de treinta años y que sólo desde hace unos pocos de años está algo más claro.

1.2.2 Programación orientada a objetos

El proverbio “tener un martillo no te hace ser un arquitecto” es especialmente cierto con respecto a las tecnologías de objetos. Una cosa es conocer C++, Java o C# y otra es “pensar en objetos”. De hecho, muchos saben codificar en un lenguaje OO, pero

no emplean POO. Justamente, el objetivo del curso es saber POO, para ello se requiere de:

1. Un marco para el análisis y el diseño OO (A/DOO) aplicando el Lenguaje Unificado de Modelado (UML).
2. La utilización de patrones de diseño y
3. El Proceso Unificado.

La producción del software requiere seguir un conjunto de etapas que van desde la recogida de los requisitos, el análisis y diseño de la aplicación hasta la implementación. Por tanto, en el paradigma orientado a objetos se requiere de disciplinas y artefactos para realizar estas etapas, a las que se describirán mediante el acrónimo de AOO/D (Análisis y diseño orientado a objetos).

En el análisis se pone énfasis en una investigación del problema y de los requisitos, en vez de poner una solución. En el AOO se presta atención a encontrar los objetos en el dominio del problema. Por tanto, el AOO es un método de análisis que examina los requerimientos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema.

El diseño pone énfasis en una solución conceptual que satisface los requisitos, en vez de ponerlo en la implementación. En DOO se presta atención a la definición de los objetos software y en cómo colaboran para satisfacer los requisitos.

UML es sólo una notación, capaz de hacer los planos del SW. Últimamente han aparecido muchos cursos de UML, pero UML es una herramienta, no un fin en sí mismo. A nadie se le ocurriría dar un curso exclusivo de PSPICE, ya que es sólo una herramienta de simulación de circuitos electrónicos, habrá que darlo dentro del contexto del análisis de los circuitos eléctricos-electrónicos. Lo mismo pasa con UML, sólo se entiende si se está dentro de un contexto y éste es el de la producción del software.

Como se verá a lo largo del curso, la POO se basa en la asignación de responsabilidades, pero ¿cómo se debería asignar las responsabilidades a los objetos? ¿cómo deberían interactuar los objetos?. Ciertas soluciones contrastadas a problemas de diseño se pueden expresar mediante patrones o buenas prácticas. A éstas se las conocen como patrones de diseño. Una habilidad clave y fundamental en el A/DOO es la asignación cuidadosa de responsabilidades a los componentes software. En consecuencia, este curso se centra en los pasos a seguir para asignar responsabilidades a los objetos. Se emplearán los patrones GRASP y GoF.

La captura de los requisitos, el A/DOO y la implementación requieren moverse dentro de un proceso. En este caso, se seleccionará mediante un proceso de desarrollo iterativo llamado Proceso Unificado. La figura adjunta muestra un gráfico que sintetiza los temas y habilidades en el que se va a centrar el curso.

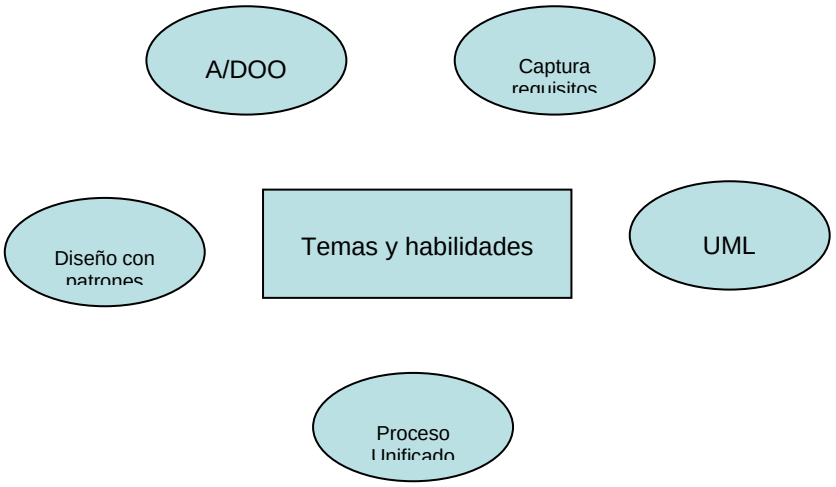


Figura 1. 1 Habilidades para la programación orientada a objetos

1.3 El ciclo de vida SW

El problema del software se reduce a la dificultad que afrontan los desarrolladores para coordinar las múltiples cadenas de trabajo de un gran proyecto. La comunidad de desarrolladores necesita una forma coordinada de trabajar. Necesita un proceso que integre las múltiples facetas del desarrollo. Necesita un método común, un proceso que:

- Proporcione una guía para ordenar las actividades del equipo.
- Dirija las tareas de cada desarrollador por separado y del equipo como un todo.
- Especifique los artefactos que deben desarrollarse.
- Ofrezca criterios para el control y la medición de los productos y actividades del proyecto.

La presencia de un proceso bien definido y bien gestionado es una diferencia esencial entre proyectos productivos y otros que fracasan. A continuación se muestran diferentes tipos de procesos de producción del software.

1.3.1 Modelo lineal secuencial (Ciclo de vida clásico)

Enfoque sistemático y secuencial del desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento.

Actividades:

Ingeniería y modelado de Sistemas/Información

Ubicación del software en el ámbito donde va a funcionar.

Análisis de requisitos:

Se deben conocer los aspectos relacionados con la información a tratar, la función requerida, comportamiento, rendimiento, etc.

El cliente debe dar el visto bueno.

Diseño:

Estructura del programa y arquitectura del software.

Representaciones de la Interfaz.

Detalle Procedimental (algoritmo).

Generación de código o Implementación:

Puede automatizarse si el diseño está bien detallado.

Pruebas:

De *Caja Blanca*: Análisis de los distintos caminos de ejecución de los algoritmos.

De *Caja Negra*: Análisis de los procesos externos funcionales.

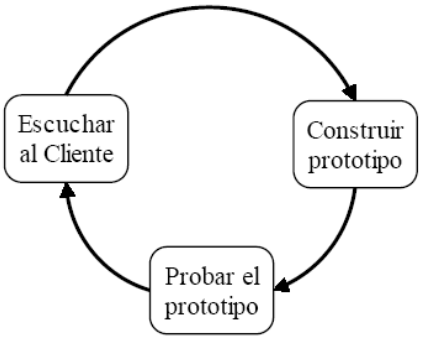
Mantenimiento:

Gestión de cambios en el software debidos a:

- Errores durante el desarrollo.
- Adaptación a nuevos entornos (e.g. sistemas operativos)
- Mejoras funcionales o de rendimiento.

1.3.2 Modelo de construcción de prototipos

Se da un conjunto de objetivos generales de entrada, procesamiento y salida. Con ellos se desarrolla un prototipo inicial que sirve para detallar los objetivos (requisitos) más concretos del producto final. El prototipo se suele desechar.



1.3.3 Desarrollo rápido de aplicaciones.

Es una adaptación a "alta velocidad" del modelo lineal secuencial en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Puede permitir el desarrollo de un sistema completamente funcional en periodos cortos de tiempo (de 60 a 90 días). Los componentes que se desarrollen se pueden reutilizar en posteriores proyectos, llamados repositorio de componentes.

El sistema se descompone en un conjunto de bloques que se pueden desarrollar de manera independiente por distintos equipos de desarrollo.

Sólo puede aplicarse cuando se cumplen una serie de condiciones:

- Se comprenden muy bien los requisitos del sistema a desarrollar; ya sea porque los conoce el propio desarrollador o porque se tiene una experiencia previa en un sistema similar.
- Se delimita muy bien el ámbito del problema.
- La interacción del software con el nuevo sistema no es complicada o se utilizan nuevas tecnologías que son dominadas por el equipo de desarrollo.

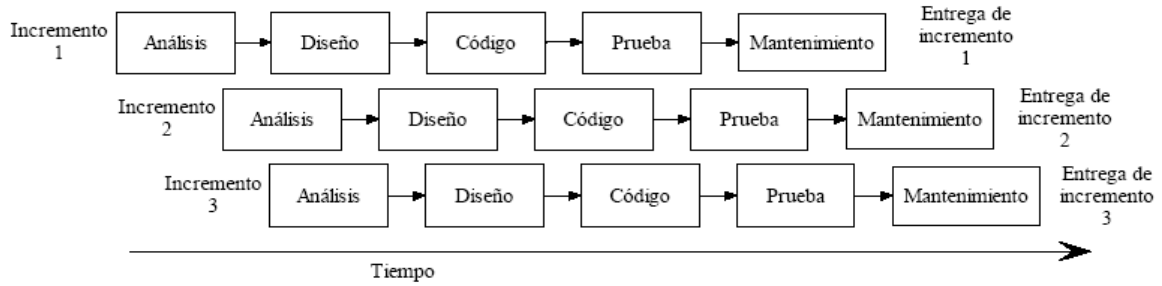
Inconvenientes:

- Debe haber un compromiso por parte del equipo de desarrollo y del cliente en el desarrollo rápido de actividades.
- Requiere recursos suficientes para crear el número de equipos necesarios.

1.3.4 Modelos Evolutivos

El software, al igual que el resto de los sistemas, evoluciona con el tiempo. Se necesita de procedimientos que permitan una evolución permanente.

Modelo Incremental

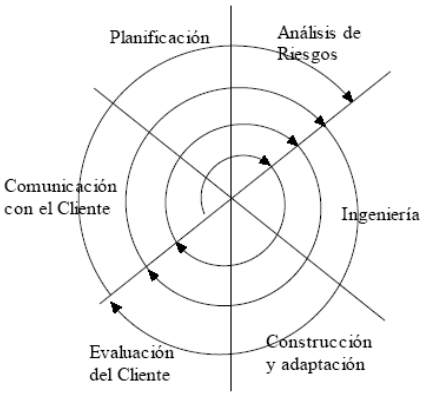


- Combina elementos del modelo lineal secuencial con la filosofía interactiva de construcción de prototipos.
- Entrega por incrementos.
- Fácil adaptación a requerimientos temporales de entrega.

Modelo en Espiral

Combina el modelo lineal secuencial y la construcción de prototipos, proporcionando el potencial necesario para el desarrollo rápido de versiones incrementales del software.

Se debe establecer el número de interacciones.



1.4 El Proceso Unificado

El Proceso Unificado (*Unified Process*, UP) es una solución al problema de producción del software. En primer lugar el Proceso Unificado es un proceso de desarrollo del software. Un proceso de desarrollo del SW es el conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema SW. UP se ha convertido en el procedimiento de gran éxito para la construcción de sistemas orientado a objetos.

UP está basado en componentes, lo cual quiere decir que la aplicación está constituida por componentes software interconectados entre ellos a través de sus interfaces bien definidas. UP utiliza UML para preparar todos los esquemas de un sistema SW. De hecho, UML es una parte esencial del Proceso Unificado; sus desarrollos fueron en paralelo.

No obstante, los verdaderos aspectos definitorios del Proceso Unificado se resumen en tres frases claves: a) dirigido por los casos de uso, b) centrado en la arquitectura y c) iterativo e incremental.

Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un beneficio. Los casos de uso representan los requisitos funcionales de la aplicación. Todos los casos de uso juntos constituyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema. Todos estos aspectos serán tratados en el siguiente capítulo.

El concepto de arquitectura del SW incluye los aspectos estáticos y dinámicos más significativos del sistema. No sólo refleja la solución a los distintos casos de uso, sino que también muestra los bloques de construcción reutilizables, los marcos de trabajo para los interfaces, los sistemas de gestión de bases de datos, consideraciones de implementación, etc. La arquitectura debe diseñarse para permitir que el sistema evolucione, no sólo en su desarrollo inicial, sino también a lo largo de las futuras generaciones. Estos conceptos serán el núcleo del temario.

UP emplea el ciclo de vida iterativo y desarrollo dirigido por el riesgo. El desarrollo iterativo se organiza en una serie de mini-proyectos, de duración fija (aproximadamente 4 semanas) llamadas iteraciones. El ciclo de vida iterativo se basa en la ampliación y refinamiento sucesivos mediante iteraciones, con retroalimentación cíclica y adaptación.

Ejemplo de actividades en una iteración

- Clarificar tareas y requisitos de la iteración
- Ingeniería inversa (código de iteración anterior a UML)
- Discusión de incremento de diseño
- Implementar y probar

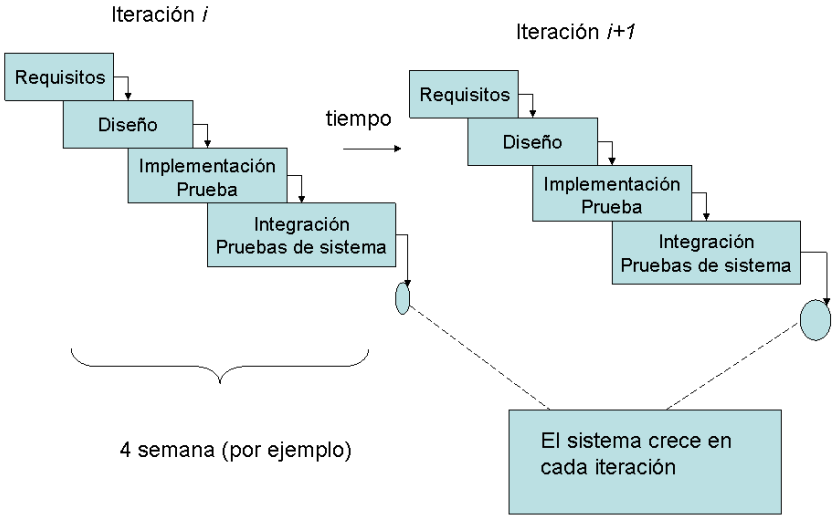


Figura 1. 2 UP y las iteraciones

El desarrollo iterativo es Aceptar el Cambio. En lugar de luchar contra el inevitable cambio, el desarrollo iterativo se basa en una aptitud de aceptación del cambio y la adaptación como motores inevitables. Se buscará el equilibrio entre estabilizar un conjunto de requisitos y por otro la realidad cambiante de los requisitos. Tener realimentación en una etapa temprana vale su peso en oro.

"Sí, esto es lo que pedí, pero ahora que lo pruebo, lo que realmente quiero es algo un poco distinto".

En consecuencia, el trabajo se desarrolla a lo largo de una serie de ciclos estructurados de construir-realimentar-adaptar.

Beneficios del desarrollo iterativo:

- Mitigar tan pronto como sea posible los altos riesgos.
- Progreso visible en las primeras etapas.
- Retroalimentación con los usuarios.
- Gestión de la compatibilidad. No se abruma por la parálisis del análisis muy largo.
- El propio conocimiento en cada iteración se puede utilizar para mejorar el proceso del desarrollo.

1.4.1 Las fases del UP

Un proyecto UP organiza el trabajo y las iteraciones en cuatro fases fundamentales:

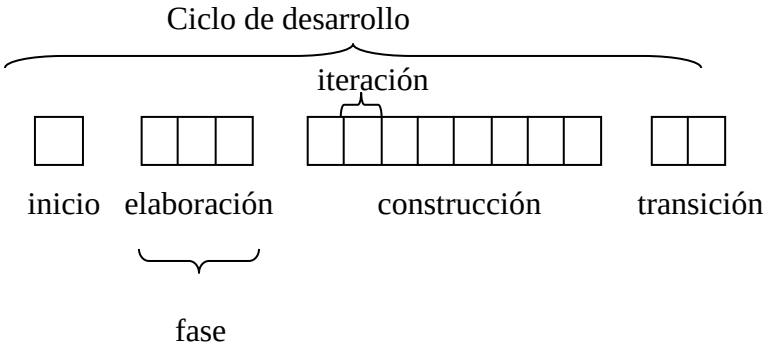
1. Inicio: Visión aproximada, análisis del negocio, alcance, estimaciones imprecisas.

2. Elaboración: visión refinada, implementación iterativa del núcleo, resolución de los altos riesgos.

3. Construcción: implementación iterativa del resto de requisitos de menor riesgo.

4. Transición: pruebas beta, despliegue.

Esto no se corresponde con el antiguo ciclo de vida "en cascada" o secuencial. La fase de inicio no es una fase de requisitos; sino una especie de fase de viabilidad. En la fase de elaboración no sólo se hace requisitos y diseño, sino que es una fase de implementación del núcleo central y se mitigan las cuestiones de alto riesgo.



En UP se describe las actividades de trabajo en disciplinas. Una disciplina es un conjunto de actividades o artefactos relacionados dentro de un área determinada. En UP, un artefacto es el término general para cualquier producto del trabajo: código, documentos, gráficos, diagramas, etc.

En UP hay varias disciplinas (ver tabla 1.2), aunque en este temario se centrará en:

- Requisitos: Sus artefactos son Casos de Uso, Visión, Especificaciones Complementarias y Glosario.

- Modelo del negocio: El artefacto más empleado en una aplicación única es el Modelo del Dominio. A nivel más grande se hace ingeniería inversa de todos los procesos del negocio de toda la empresa.

- Diseño: Modelo del Diseño, documentos de Arquitectura SW, Modelo de Datos.

En este temario se hace especial hincapié en las fases de Inicio y Elaboración, cuyos artefactos están unidos con las disciplinas de Requisitos, Modelado de Negocio y Diseño, donde se aplica el análisis de los requisitos, el AOO/D, loa patrones y su notación mediante UML.

Tabla 1.2 Disciplinas del UP. c= comenzar, r=refinar

Disciplina	Artefacto	Inicio	Elaboración	Construcción	Transición
Requisitos	Modelo de Casos de Uso Visión Especificaciones Complementarias Glosario	c c c c	r r r r		
Modelado del Negocio	Modelo del Dominio		c		
Diseño	Modelo de Diseño Documento de Arquitectura SW Modelo de Datos		c c c	r r	
Implementación	Modelo de implementación		c	r	r
Gestión del Proyecto	Plan de Desarrollo SW	c	r	r	r
Pruebas	Modelo de Pruebas		c	r	
Entorno	Marco de Desarrollo	c	r		

Es importante saber que en UP todas las actividades y artefactos son opcionales. Dependiendo del tipo de proyecto se escogen los artefactos y las disciplinas necesarias.

1.5 Herramientas CASE

CASE es el acrónimo de *Computer-Aided Software Engineering*. Son herramientas de apoyo al desarrollo, mantenimiento y documentación informatizada del SW. No son herramientas CASE:

a) Las que no tienen esa finalidad

b) Las que se utilizan para codificar SW.

Tipos de herramientas:

- Herramientas diagramáticas (*Rational Rose*, *BoUML*)
- Herramientas de gestión de cambios (CVS)
- Herramientas para la documentación (*Doxygen*)

1.6 Problemas

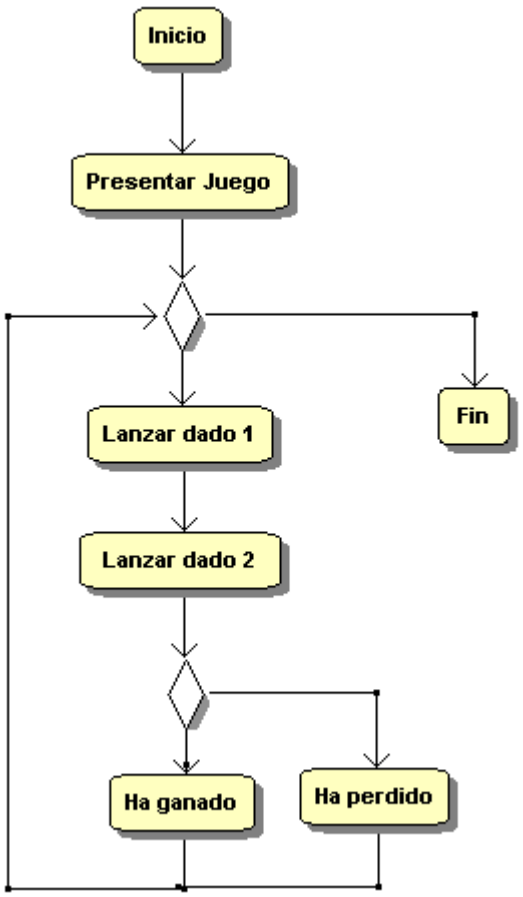
1. Objetivos del curso AOO/D.
2. Características del producto SW.
3. ¿Qué es la Ingeniería del SW?.
4. El paradigma OO.
5. Diferencias entre los métodos estructurados y los métodos OO.
6. Habilidades necesarias para la POO.
7. Enfoques sobre el proceso de desarrollo del SW.
8. ¿Qué es el UP?
9. Fases, iteraciones, disciplinas y artefactos en UP.

Ejercicio 1

Diseñar un juego de lanzar dos dados, si la suma de las caras es siete se gana en caso contrario se pierde. Emplear métodos estructurados y métodos orientado a objetos.

METODOLOGÍA ESTRUCTURADA

1. Definición del flujograma e implementación:

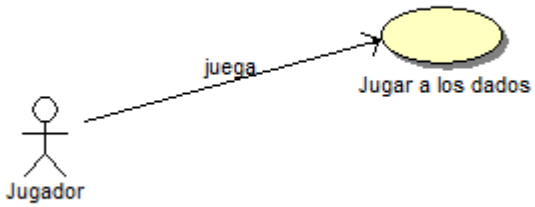


```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int lanzarDado ( void );

void main (void)
{
        int caraDado1, caraDado2;
        int continuarJuego = 1;
        char letra;

        printf("\nJuego de dados,lanzar dos dados y sumar 7 ganar\n");


        while(continuarJuego)
        {
                puts("Pulsar cualquier tecla para lanzar dado 1");
                getch();
                caraDado1 = lanzarDado();
                puts("Pulsar cualquier tecla para lanzar dado 2");
                getch();
                caraDado2 = lanzarDado();
                if( (caraDado1 + caraDado2) == 7 )
                        puts ("Ha ganado");
                else
                        puts("Ha perdido");
                puts("Si desea jugar otra vez, pulse C");
                letra = getch();
                if ((letra == 'c' )||(letra == 'C'))
                        continuarJuego = 1;
                else
                        continuarJuego = 0;
        }

        puts("Se termino el juego de dados. Un saludo");
}
int lanzarDado ( void )
{
        int valorCara;
        valorCara=(int)((rand()%6)+1);
        printf("\n%d\n",valorCara);
        return (valorCara);
}
```

MÉTODO PROCESO UNIFICADO

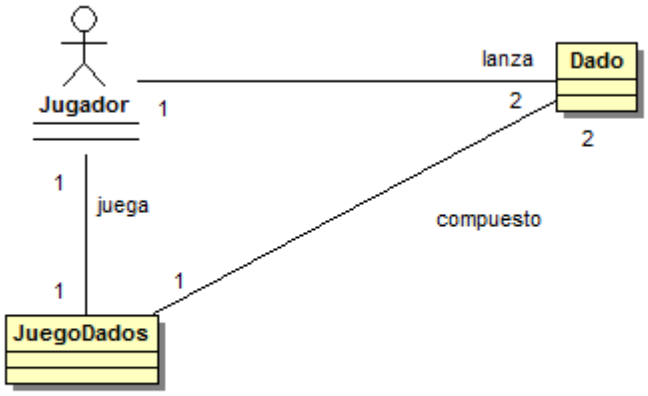
1. Definición de casos de uso:

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



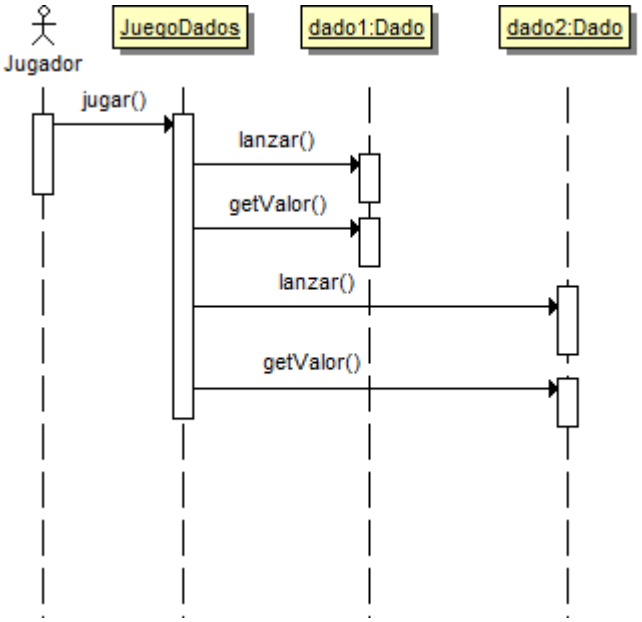
2. Definición del modelo del dominio

El modelo del dominio se ilustra mediante un diagrama que muestran el vocabulario del problema mediante clases conceptuales.



3. Definición de los diagramas de interacción

La finalidad del DOO es definir los objetos SW y sus colaboraciones. Las colaboraciones se expresan mediante diagramas de interacción.



4. Definición de los diagramas de clase de diseño.

A diferencia del modelo del dominio, este diagrama no muestra conceptos del mundo real, sino clases de diseño que no de SW.

5. Implementación

Utilizando la programación extrema, XP, se empieza a escribir el código de prueba y luego se construyen desde las clases menos acopladas a las más acopladas. En este caso se codificará en C++.

```
#include <iostream>
#include <conio.h>
#include "JuegoDados.h"

void main ()
{
        bool ganarPartida, continuarJuego = true;
        char tecla;
        JuegoDados elJuego;

        std::cout << "\nJuego de dados, lanzar dos dados y sumar 7 para ganar\n";

        while (continuarJuego)
        {
                std::cout << "\nPulsar cualquier tecla para lanzar dados\n";
                getch();

                ganarPartida = elJuego.jugar();

                if ( ganarPartida )
                        std:: cout << "Ha ganando.\n";
                else
                        std:: cout << "Ha perdido.\n";

                std::cout<<"\nSi desea jugar otra vez, pulse C\n";
                std::cin>> tecla;
                continuarJuego = (tecla == 'c')||(tecla == 'C')?true:false;
        }

        std::cout << "Se termino el juego de dados. Un saludo\n";
}
```

```
class Dado
{
public:
        void lanzar();
        int getValorCara();

private:
        int valorCara;

}; /* Dado.h */

ç
```

```
#include "Dado.h"
#include <stdlib.h>

void Dado::lanzar()
{
        valorCara = ((rand() % 6) +1);
}

int Dado::getValorCara()
{
        return valorCara;;

} /* Dado.cpp */
```

```
#include "Dado.h"

class JuegoDados
{
public:

        bool jugar();

private:
        Dado dado1;
        Dado dado2;

}; /* JuegoDados.h */
```

```
#include "JuegoDados.h"



bool JuegoDados::jugar()
{
        dado1.lanzar();
        dado2.lanzar();

        return((dado1.getValorCara()+
        dado2.getValorCara())==7?true:false);

} /* JuegoDado.cpp */
```

Derecho de Autor © 2014 Carlos Platero Dueñas.

Permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, Versión 1.1 o cualquier otra versión posterior publicada por la Free Software Foundation; sin secciones invariantes, sin texto de la Cubierta Frontal, así como el texto de la Cubierta Posterior. Una copia de la licencia es incluida en la sección titulada "Licencia de Documentación Libre GNU".

La Licencia de documentación libre GNU (GNU Free Documentation License) es una licencia con ***copyleft*** para contenidos abiertos. Todos los contenidos de estos apuntes están cubiertos por esta licencia. La version 1.1 se encuentra en http://www.gnu.org/copyleft/fdl.html. La traducción (no oficial) al castellano de la versión 1.1 se encuentra en http://www.es.gnu.org/Licencias/fdles.html