

1. Metodologías de desarrollo de Sistemas de Información

*Un Sistema de Información es un conjunto integrado de **personas, procedimientos, información y equipos diseñados**, contruidos, operados y mantenidos para recoger, registrar, procesar, almacenar, recuperar y visualizar información.*

Los sistemas de información tienen muchas cosas en común, la mayoría de ellos están formados por:

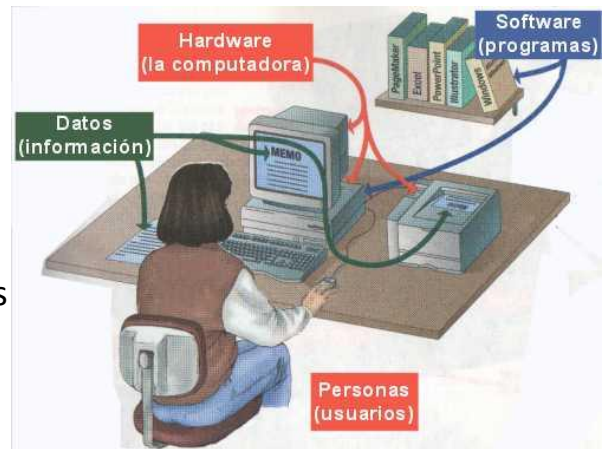
- **Personas** son un componente esencial en cualquier sistema de información, producen y utilizan la información de sus actividades diarias para decidir lo que se debe hacer. Las decisiones pueden ser rutinarias o complejas.

- **Procedimientos**, los sistemas de información deben soportar diversas clases de actividades del usuario, por eso han de establecerse procedimientos que aseguren que los datos correctos llegan a las personas adecuadas en su momento justo.

- **Equipo**, es decir los ordenadores y todos los dispositivos necesarios, para la entrada, el procesamiento, el almacenamiento y la comunicación de información.

- **Información**, es el componente fundamental.

EL **objetivo de un sistema de información** es, ayudar al desempeño de las actividades productivas y de decisión en todos los niveles de la organización, mediante el suministro de la información adecuada, con la calidad suficiente, a las personas apropiadas, en el momento y lugar oportunos, y con el formato más útil para el receptor.



Estructura de la información en un sistema de información es la siguiente:

A continuación se describen las diferentes partes de la pirámide:

Operaciones y transacciones: Incluye el procesamiento de las actividades diarias o transacciones, los acontecimientos rutinarios que afectan a la organización (facturación, pagos, entrega de productos, etc.), cuyas características son que hay un gran volumen de transacciones y pocas excepciones a los procedimientos normales. Se emplean procesos interactivos.



Nivel operativo: Se trabaja con información del procesamiento de las actividades diarias o transacciones, para tomar decisiones a corto plazo y de consecuencias limitadas. Las características de esta información son:

- Es repetitiva, informes periódicos.
- Con datos originados internamente.
- Gran volumen de datos.
- Los datos cuentan con un formato bien estructurado, son detallados y precisos.

Se utilizan procesos batch.

Nivel táctico: Se ocupa de la asignación efectiva de los recursos a medio plazo para mejorar el rendimiento de la empresa. Se basa en análisis de informes:

- Resúmenes con medidas estadísticas, con medias, desviaciones, etc.
- De excepciones, por ejemplo, centros con pérdidas.
- Específicos, que no se han pedido antes, y que los directivos necesitan con rapidez para resolver un problema muy concreto. A nivel táctico no es necesario que los datos estén actualizados instantáneamente, se pueden utilizar los de pocos días antes, lo cual permite realizar las operaciones paralelamente a las de, por ejemplo, el nivel operativo.

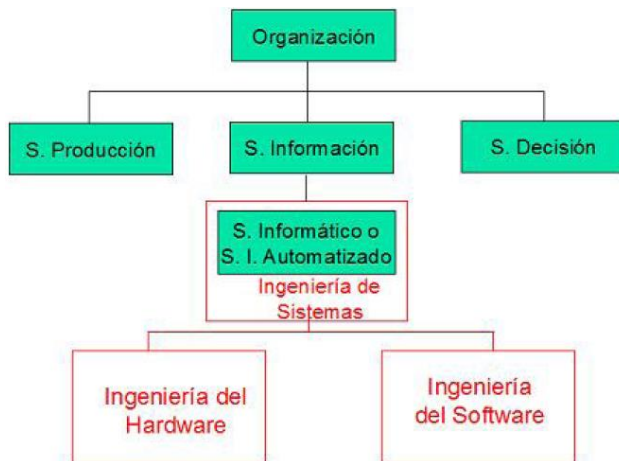
Nivel estratégico: Trabaja con plazos largos para acometer la difícil tarea de decidir las líneas maestras que debe seguir la empresa en el futuro. Se trabaja con información del tipo:

- En formato muy resumido.
- En formatos muy variables y procedente de las fuentes externas más inesperadas.
- Las decisiones están poco formalizadas y con un fuerte componente subjetivo.

Sistemas de Información Automatizados (SIA)

Un Sistema de Información (SI) no necesita estar obligatoriamente basado en el uso de ordenadores. El sistema de información existe siempre, esté mecanizado o no. Cuando para recoger, registrar, procesar, almacenar, recuperar y visualizar información se utilizan ordenadores estamos ante Sistemas de Información Automatizados (SIA).

Una organización incorpora un sistema de información de la siguiente manera:



*Un **sistema de información** se centra en tratamiento de datos para empresas. La **ingeniería de software** es dirigido a programas informáticos, buscando mayor eficiencia y seguridad, no es tanto tratamiento de datos.*

El ingeniero de software está más enfocado al desarrollo de las soluciones de software mediante el uso de las ciencias de la computación, mejor dicho, a echar código. El informático por su parte debe estar en capacidad de analizar los problemas que tienen que ver con el manejo de la información y diseñar una solución para los mismos.

Para aclarar lo anterior veamos un ejemplo: En una empresa que acaba de surgir necesitan vender sus productos por internet, y no tienen un sistema de información adecuado para el manejo interno de la empresa. El informático debe estar en capacidad de analizar la cadena de producción de dicha empresa, saber donde se genera la información y el flujo de la misma, comprender como hace la empresa para saber cuánto le costó cada producto, a quienes les vende, cuanto está perdiendo o ganando, cuando tiene que generar nuevas órdenes de producción, que procesos hacen que se genere nueva información o que se modifique la ya existente, y así mismo diseñar una solución que permita el funcionamiento óptimo de la empresa. Esto por ejemplo puede llevar a la conclusión de que necesita una base de

datos, que se comunica con una aplicación interna para el manejo de las cuestiones de la empresa y con un sitio web desde el cual se generan los pedidos y se le informa al cliente cuando serán despachados. El ingeniero de software es el encargado de hacer esta implementación (la aplicación interna, la base de datos, la pagina web).

Para el desarrollo, operación, mantenimiento de un sistema es necesario el uso de la Ingeniería de Software ya que “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 de software”.

El **proceso de ingeniería de software** se define como un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad. El **proceso de desarrollo de software** es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos que son transformados en diseño e implementado en código, el código es probado y documentado para su uso operativo, y se implementan los procesos del desarrollo de software.

Para la creación del **diseño de software** es necesario realizarlo a tres niveles: Conceptual, Lógico y Físico. El proceso de diseño de software unido a un **Análisis Orientada a Objetos** nos brinda el beneficio de seguir un mecanismo para formalizar un modelo de la realidad.

En la presente asignatura, la modelación del funcionamiento de cada uno de los subsistemas se realiza con el lenguaje Unificado de Modelamiento (UML). Los diagramas de UML serán utilizados para el análisis y diseño del hardware y software. Para la simulación del sistema se realiza, la parte de la programación de los PIC'S (Peripheral interface Controller), con el entorno del desarrollo de la firma Atmet utilizando Proteus Lite – Isis. Mientras que el programa de control se empleará Visual Basic.

Sistemas de control automatizado

Desde el principio de las civilizaciones el hombre ha tratado de simplificar tareas de la vida cotidiana por medio de aparatos o sistemas que funcionen de manera automática.

La ingeniería de control es una ciencia interdisciplinar relacionada con muchos otros campos, principalmente las matemáticas y la informática. Las aplicaciones son de lo más variado: desde tecnología de fabricación, instrumentación médica, Subestación, ingeniería de procesos, robótica hasta economía y sociología. Aplicaciones típicas son, por ejemplo, el piloto

automático de aviones y barcos y el ABS de los automóviles. En la biología se pueden encontrar también sistemas de control realimentados, como por ejemplo el habla humana, donde el oído recoge la propia voz para regularla. El sistema de control se encuentra basado en dos partes:

- ✚ Software.

- ✚ Hardware.

Con respecto al Hardware se considera: **Procesador, Memoria, Dispositivos de E/S**

- **Sensores:** Un sensor es un elemento que está en contacto directo con una variable física que se desea medir, y que provee una salida de acuerdo al valor de dicha variable. Existen distintos tipos de sensores, pero pueden considerar principalmente dos: pasivos y activos. En los pasivos hay que provee una tensión para obtener una respuesta. Los sensores activos autogeneran tensión. Los sensores que utilizan capacitores se llaman capacitivos, y pueden usarse para medir presión, nivel, desplazamiento, deformación y humedad. Los sensores inductivos se basan en cambios de inductancia, y suelen utilizarse para medir desplazamiento, deformación, velocidad o aceleraciones. Hay otros tipos de sensores como por ejemplo: sensores de presión, fotoeléctricos, de ultra sonido, para la detección de fuego, de calor.
- **Actuadores:** Un actuador es un dispositivo físico que provoca una alteración del ambiente con el cual interactúa, produciendo una reacción física o química en el mismo. Estas reacciones se producen ante la llegada de una señal eléctrica o mecánica. Los actuadores convierten una forma de energía a otra, que además, permite controlar una variable física. Pueden haber actuadores de diferentes tipos, como por ejemplo: actuadores eléctricos, neumáticos, actuadores de flujo, y mecánicos.
- **Transductores:** Son dispositivos que traducen los valores de una variable física a una señal eléctrica. Suelen estar compuestas de un sensor y un acondicionador de señal. Se utilizan para contar eventos o para proveer una salida digital. A veces constan de un transductor analógico asociado con un conversor de energía alterna / continua en la misma unidad.
- **Interfaces para dispositivos de intercambio con el ambiente:** Las interfaces digitales permiten el intercambio de señales digitales, por lo que se deben usar señales de entrada normalizadas. Hay una gran variedad de interfaces digitales, pero, básicamente todas constan de un registro digital que se conecta con los dispositivos de entorno.

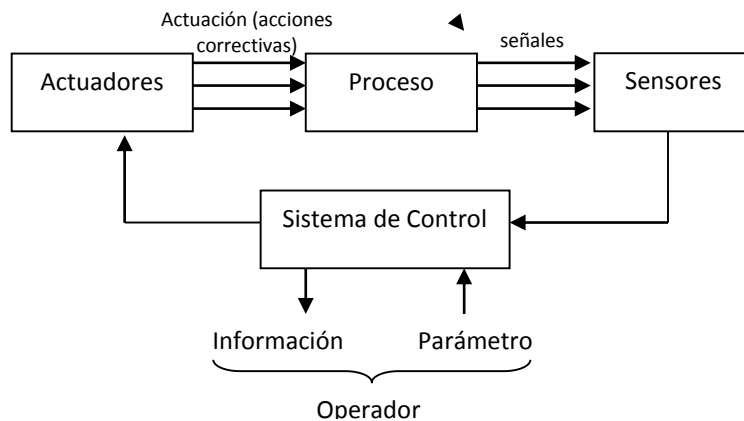
CONTROL DE PROCESOS

Puede definirse como la aplicación de una acción planificada para que el objeto controlado satisfaga ciertos objetivos. Control implica generar una señal de salida en respuesta a la entrada de datos. Un controlador acepta la información desde los sensores a intervalos regulares (o es manejado por eventos), la procesa, y envía los resultados al objeto controlado a través de

actuadores. Por ende la información de salida influye sobre el objeto controlado.

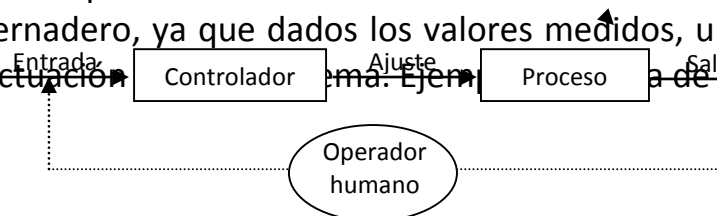
Los elementos básicos de cualquier control de procesos son:

- ✚ El valor medido. sensores
- ✚ Mecanismos de control.
- ✚ Actuación. actuadores



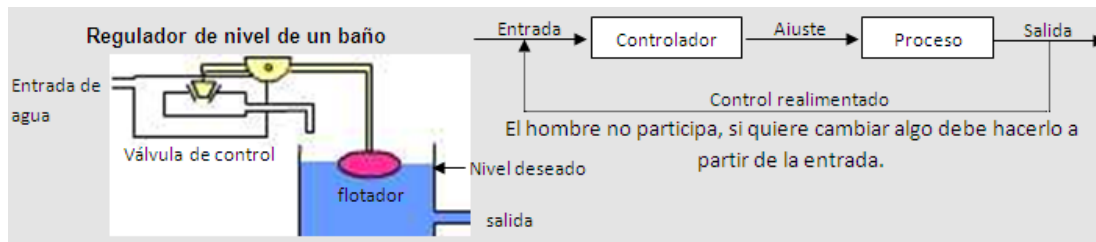
Configuraciones de control

De lazo abierto: Son aquellos en los que el **operador** debe calcular una acción correctiva a partir de un valor observado en la variable de salida. Ejemplo 1: Invernadero, ya que dados los valores medidos, un hombre es el que realiza la actuación. Ejemplo 2: Regulador de nivel de un baño.



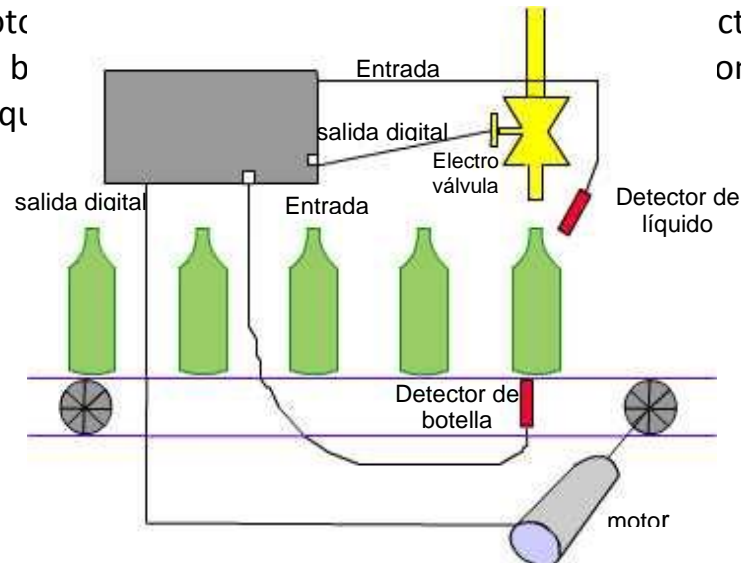
El hombre interviene en el control

De lazo cerrado: La idea es observar el resultado obtenido y con ese resultado generar una modificación de las variables del proceso. En este caso es el propio sistema de control el que lo hace. Ejemplo 1: Un termo. Ejemplo 2: Regulador de nivel de un baño.



Cuando se aplican en sistemas de control computarizados. Se usa una computadora digital como dispositivo de control y si se la conecta directamente con el proceso se dice que está haciendo CDD (Control Digital Directo). Veamos un ejemplo en el embotellamiento de gaseosas cuando hay una secuencia de pasos controlados por sensores.

El sistema de control pone en marcha el motor hasta que se activa el detector de botellas, lo cual indica que hay una botella en la posición de llenado. En ese momento se para el motor. El sistema de control activa el detector de líquido (la botella llena) y pone en marcha el motor hasta que







En aplicación de Redes Neuronales: Como forma de hacer control, se encuentra el uso de redes neuronales artificiales. El formalismo de redes neuronales artificiales permite rescatar algunas propiedades estudiadas en

los sistemas nerviosos de los seres vivos. Manejan arquitecturas con interconexión paralela masiva de procesadores simples (neuronas), que se intercalan, definiendo una arquitectura. Cada conexión (sinapsis) tiene un peso (eficacia sináptica) asociado. Normalmente son empleados para aprender el comportamiento del proceso y hacer ajuste automáticos de parámetros. Ejemplo: aplicaciones de robótica (control de manipulación, locomoción, etc.) en navegación automática, en aeronáutica.

Entre **las funciones básicas** de cualquier sistema de control se cuenta con:

Registro de Datos Históricos: Permite mantener una historia del proceso, permitiendo identificar las causas de un error en el sistema o los equipamientos. Se registran las operaciones del sistema y se usa ese registro para evitar fallas futuras. También puede usarse con el fin de analizar el proceso, y optimizar su funcionamiento. Entre los datos que se registran están:

-  Cambios de estados.
-  Registros de errores.
-  Registro de operaciones.
-  Registro de datos.

Creación de Informes: Permiten recuperar datos del archivo de registro del sistema y crear informes de acuerdo con los propósitos del usuario. Si el informe va dirigido a un ingeniero, deberá disponer de información sobre la cual basar decisiones de planes de mantenimiento de instrumental, de reemplazo de actuadores y componentes de la planta. Si va dirigido a un gerente, contendrá información sobre el desempeño de la planta desde un punto de vista económico.

Las **metodologías de desarrollo de sistemas** deben definir: objetivos, fases, tareas, productos y responsables, necesarios para la correcta realización del proceso y su seguimiento.

Los principales **objetivos de una metodología de desarrollo** son:

- Asegurar la uniformidad y calidad tanto del desarrollo como del sistema en sí.
- Satisfacer las necesidades de los usuarios del sistema.
- Conseguir un mayor nivel de rendimiento y eficiencia del personal asignado al desarrollo.
- Ajustarse a los plazos y costes previstos en la planificación.
- Generar de forma adecuada la documentación asociada a los sistemas.
- Facilitar el mantenimiento posterior de los sistemas.

Una **Metodología para el Desarrollo de Sistemas de Información** es un conjunto de actividades llevadas a cabo para desarrollar y poner en marcha un Sistema de Información.

Los **Objetivos de las Metodologías de Desarrollo de Sistemas de Información** son:

- Definir actividades a llevarse a cabo en un Proyecto de S.I.
- Unificar criterios en la organización para el desarrollo de S.I.
- Proporcionar puntos de control y revisión

Independientemente de la Metodología de Desarrollo de Sistemas de Información que se siga, varios autores sugieren distribuir el tiempo de desarrollo de acuerdo a los siguientes porcentajes:

Distribución del Tiempo (en %) para un Proyecto de S.I.			
Autor	J Senn	M Zelkowitz	J Montilva
Fases			
Estudio de Factibilidad	35%	20%	40%
Análisis			
Diseño		15%	
Programación	25%	20%	20%
Prueba	35%	45%	40%
Documentación	5%		

METODOLOGIA DE DESARROLLO

Uno de los aspectos más discutidos e indeterminados de las metodologías de desarrollo es precisamente su definición. No hay dos metodologías que

abarquen exactamente las mismas responsabilidades, ni una definición precisa y aceptada que delimite cuales deben ser cubiertas por una metodología de desarrollo de software para poder denominarse como tal.

DEFINICION Y CONCEPTOS

El primer problema que se debe solventar para comprender el concepto de metodología o proceso de desarrollo es precisamente alcanzar una definición del mismo. En consecuencia, se debe comenzar por delimitar el alcance de los distintos términos que se manejan en torno a la ingeniería del software, sus métodos y herramientas.

Los elementos clave de la Ingeniería del software son los métodos, las herramientas y los procedimientos, procesos o metodologías. Mientras que los **métodos** de la ingeniería de software indican cómo debe construirse el software desde un punto de vista técnico, las **herramientas** suministran soporte automático o semiautomático para estos métodos, y los **procedimientos o procesos** determinan la forma de aplicar los métodos y emplear las herramientas para facilitar el desarrollo racional y oportuno del software.

METODOLOGÍAS O PROCESOS DE DESARROLLO

El término metodología más adecuado dentro del contexto de este estudio de los presentes en el diccionario de la Real Academia de Lengua Española es: *"Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal"*.

Rumbaugh aportó la siguiente definición: *"Una metodología de ingeniería software es un proceso para la producción organizada del software, empleando una colección de técnicas predefinidas y convenciones en las notaciones. Una metodología se presenta normalmente como una serie de pasos, con técnicas y notaciones asociadas a cada paso...Los pasos de producción del software se organizan normalmente en un ciclo de vida consistente en varias fases de desarrollo."* [Rumbaugh91]¹.

Algunos autores emplean el término proceso de desarrollo de software para denominar, sino exactamente, prácticamente el mismo concepto. Para Pressman, se trata de *un marco de trabajo de las tareas que se requieren para construir software de alta calidad* [Pressman91]². Según Sommerville, se trata del *conjunto de actividades y resultados asociados que producen un producto*

¹ [Rumbaugh91] Object-Oriented Modeling and Design. J. Rumbaugh, M. Blaha, W. Premerlani, and V.F. Eddy. Prentice-Hall, 1991.

² [Pressman91] Ingeniería de Software. Un enfoque práctico. Tercera Edición. Roger S. Pressman. McGraw Hill.

de software [Sommerville02]³. Estas actividades son llevadas a cabo por los denominadas ingenieros de software.

A grandes rasgos, si tomamos como criterio las notaciones utilizadas para especificar artefactos producidos en actividades de análisis y diseño, podemos clasificar las metodologías en dos grupos: Metodologías Estructuradas y Metodologías Orientadas a Objetos. Por otra parte, considerando su filosofía de desarrollo, aquellas metodologías con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales (o peyorativamente denominada Metodologías Pesadas, o Peso Pesado). Otras metodologías, denominadas Metodologías Ágiles, están más orientadas a la generación de código con ciclos muy cortos de desarrollo, se dirigen a equipos de desarrollo pequeños, hacen especial hincapié en aspectos humanos asociados al trabajo en equipo e involucran activamente al cliente en el proceso.

No existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable.

A continuación se revisan brevemente cada una de estas categorías de metodologías.

Metodologías estructuradas

Los métodos estructurados comenzaron a desarrollarse a fines de los 70's con la Programación Estructurada, luego a mediados de los 70's aparecieron técnicas para el Diseño (por ejemplo: el diagrama de Estructura) primero y posteriormente para el Análisis (por ejemplo: Diagramas de Flujo de Datos). Estas metodologías son particularmente apropiadas en proyectos que utilizan para la implementación lenguajes de 3ra y 4ta generación.

Ejemplos de metodologías estructuradas de ámbito gubernamental: MERISE (Francia), MÉTRICA 3 (España), SSADM (Reino Unido). Ejemplos de propuestas de métodos estructurados en el ámbito académico: Gane & Sarson , Ward & Mellor , Yourdon & DeMarco e Information Engineering . Sus características son:

- Se maneja como proyecto
- Gran volumen de datos y transacciones
- Abarca varias áreas organizativas de la empresa
- Tiempo de desarrollo largo

³ [Sommerville02] Ingeniería de Software. Sexta Edición. Ian Sommerville, Addison Wesley 2002.

- Requiere que se cumplan todas las etapas, para poder cumplir las siguientes (progresión lineal y secuencial de una fase a la otra)

Metodologías orientadas a objetos

Su historia va unida a la evolución de los lenguajes de programación orientada a objeto, los más representativos: a fines de los 60's SIMULA, a fines de los 70's Smalltalk-80, la primera versión de C++ por Bjarne Stroustrup en 1981 y actualmente Java o C# de Microsoft. A fines de los 80's comenzaron a consolidarse algunos métodos Orientadas a Objeto.

En 1995 Booch y Rumbaugh proponen el Método Unificado con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta a un objetivo más modesto, para dar lugar al Unified Modeling Language (UML) , la notación OO más popular en la actualidad.

Algunos métodos OO con notaciones predecesoras de UML son: OOAD (Booch), OOSE (Jacobson), Coad & Yourdon, Shaler & Mellor y OMT (Rumbaugh).

Algunas metodologías orientadas a objetos que utilizan la notación UML son: Rational Unified Process (RUP) , OPEN , MÉTRICA 3 (que también soporta la notación estructurada). Sus características son:

- No modela la realidad, sino la forma en que las personas comprenden y procesan la realidad
- Es un proceso ascendente basado en una abstracción de clases en aumento
- Se basa en identificación de objetos, definición y organización de librerías de clases, y creación de macros para aplicaciones específicas
- Utiliza menor cantidad de código
- Es más reutilizable

Metodologías tradicionales (no ágiles)

Las metodologías no ágiles son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema.

-Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales. Aunque en el caso particular de RUP, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del

proyecto (mediante su configuración previa a aplicarse), realizando una configuración adecuada, podría considerarse Ágil.

Metodologías ágiles

Un proceso es ágil cuando el desarrollo de software es incremental (entregas pequeñas de software, con ciclos rápidos), cooperativo (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), sencillo (el método en sí mismo es fácil de aprender y modificar, bien documentado), y adaptable (permite realizar cambios de último momento). Entre las metodologías ágiles identificadas en:

- Extreme Programming.
- Scrum.
- Familia de Metodologías Crystal.
- Feature Driven Development.
- Proceso Unificado Rational, una configuración ágil.
- Dynamic Systems Development Method.
- Adaptive Software Development.
- Open Source Software Development.

Metodologías Estructuradas y orientadas a objetos

Las metodologías estructuradas se basan en la estructuración y descomposición funcional de problemas en unidades más pequeñas interrelacionadas entre sí. Representan los procesos, flujos y estructuras de datos, de una manera jerárquica y ven el sistema como entradas-proceso-salidas.

Existe una gran cantidad de proyectos implementados utilizando estas metodologías, generalmente orientados a la manipulación de datos (persistentes en ficheros o bases de datos) y gestión. Estas metodologías funcionan muy bien con los lenguajes de programación estructurados.

Las metodologías estructuradas hacen fuerte separación entre los datos y los procesos. Producen una gran cantidad de modelos y documentación y se basan en ciclos de vida en cascada.

Al hacer una comparación entre ambas metodologías con respecto a la descomposición, se puede mencionar lo siguiente:

Descomposición Estructurada.

- La descomposición algorítmica se aplica para descomponer un gran problema en pequeños problemas.
- La unidad fundamental de este tipo de descomposición es el subprograma.

- El programa resultante toma la forma de un árbol, en el que cada subprograma realiza su trabajo, llamando ocasionalmente a otro programa. También se le conoce con el nombre de diseño estructurado top-down.

Descomposición orientada a objetos.

- El mundo es visto como un conjunto de entidades autónomas, que al colaborar muestran cierta conducta.
- Los algoritmos no existen de manera independiente, estos están asociados a los objetos.
- Cada objeto exhibe un comportamiento propio bien definido, y además modela alguna entidad del mundo real.
- La programación tradicional (estructurada) separa los datos de las funciones, mientras que la programación orientada a objetos define un conjunto de objetos donde se combina de forma modular los datos con las funciones.

Al comparar ambas metodologías se puede mencionar como aspectos relevantes lo siguiente:

- La forma algorítmica muestra el orden de los eventos.
- La forma orientada a objetos enfatiza las entidades que causan una acción.
- La forma orientada a objetos surge después de la algorítmica.
- El análisis y diseño estructurado se concentra en especificar y descomponer la funcionalidad del sistema total. Los cambios a los requisitos cambiarán totalmente la estructura del sistema.
- Ambas metodologías cuentan con modelos similares: estático, dinámico y funcional.
- Las metodologías de objetos están dominados por el modelo estático, en el modelo estructurado domina el funcional y el estático es el menos importante.
- El modelo estructurado se organiza alrededor de procedimientos. Los modelos orientados a objetos se organizan sobre el concepto de objetos.
- Comúnmente los requisitos cambian en un sistema y ocasiona cambios en la funcionalidad más que cambios en los objetos. El modelo estructurado puede ser útil en sistemas en que las funciones son más importantes y complejas que los datos.
- En el modelo estructurado la descomposición de un proceso en subprocesos es bastante arbitraria. En el modelo de objetos diversos desarrolladores tienden a descubrir objetos similares, incrementando la reusabilidad entre proyectos.

- Lo más recomendable es aplicar una descomposición orientada a objetos, la cual arrojará una estructura, para continuar con una descomposición algorítmica.
- El enfoque orientado a objetos integra mejor las bases de datos con el código de programación.
- En los métodos estructurados el criterio de modularización es la funcionalidad. La pregunta básica que se realiza en este caso, es:

¿Qué hace el sistema?

Como respuesta a esta pregunta aparecen las acciones que se identifican con los módulos que son de naturaleza funcional. La estrategia es típicamente Top-down. Se parte del programa principal y luego se modulariza.

- En los métodos orientados a objetos, el criterio de modularización son los componentes del problema. La pregunta básica que se realiza es:

¿Qué componentes tiene el problema?

Como respuesta se identifican los objetos que intervienen en el problema y la forma que interactúan entre sí. Como consecuencia de su naturaleza y de las interacciones que se producen, resulta la funcionalidad. El programa principal, si existe, es lo último que se aborda.

1.2. Patrones y UML

Los patrones son:

- Arquitecturas comprobadas para construir software orientado a objetos que sea flexible y se pueda mantener.
- Proveen la reutilización del diseño en sistemas posteriores.
- Ayudan a identificar los errores y obstáculos comunes que ocurren al crear sistemas.

*“Cada patrón describe un **problema** que ocurre una y otra vez en nuestro **entorno**, para describir después el núcleo de la **solución** a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma”* Christopher Alexander

Patrón = <**problema, contexto, solución**> +<recurrencia, enseña, nombre>

- **Solucionar un problema:** Los patrones capturan soluciones, no sólo principios o estrategias abstractas
- **Ser un concepto probado:** capturan soluciones demostradas, no teorías o especulaciones

- **La solución no es obvia:** los mejores patrones generan una solución a un problema de forma indirecta
- **Describe participantes y relaciones entre ellos:** describen módulos, estructuras del sistema y mecanismos complejos
- **El patrón tiene un componente humano significativo:** todo software proporciona a los seres humanos confort y calidad de vida (estética y utilidad)
- Los diseñadores expertos no resuelven cada problema desde el principio
- Se pueden encontrar patrones de clases y comunicaciones entre objetos en muchos sistemas orientados a objetos
- El objetivo de los patrones es guardar la **experiencia** en diseños y arquitecturas, ayudando a elegir entre diseños alternativos, haciendo a un sistema reutilizable de programas orientados a objetos

Cuando se desarrolla la arquitectura, se escogen componentes o patrones de diseño de un catálogo, los cuales fueron creados para ser reutilizados.

Algunas soluciones ya probadas y eficaces de los problemas de diseño pueden expresarse (y se han plasmado) como un conjunto de principios, con heurística o **patrones** (fórmulas de solución de problemas que codifican los principios aceptados del diseño). En este libro enseñamos los patrones y así favorecemos el aprendizaje rápido y la utilización hábil de los tecnicismos fundamentales de esta tecnología.

Patrones. Los diseñadores expertos en orientación a objetos (y también otros diseñadores de software) van formando un amplio repertorio de principios generales y de expresiones que los guían al crear software. A unos y a otras podemos asignarles el nombre de patrones, si se codifican en un formato estructurado que describe el problema y su solución, y si se les asigna un nombre. A continuación ofrecemos un ejemplo de patrón.

Nombre del patrón: Experto.

Solución: Asignar una responsabilidad a la clase que tiene la información necesaria para cumplirla.

Problema que resuelve: ¿Cuál es el principio fundamental en virtud del cual asignaremos las responsabilidades a los objetos?

En la terminología de objetos, el **patrón** es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas.¹ Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas.

Expresado lo anterior con palabras más simples, el patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.

Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería del software. Todo lo contrario: intentan codificar el conocimiento, las expresiones y los principios ya existentes: cuanto más trillados y generalizados, tanto mejor.

En teoría, todos los patrones poseen nombres muy sugestivos. El asignar nombre a un patrón, a un método o a un principio ofrece las siguientes ventajas:

- Apoya el agrupamiento y la incorporación del concepto a nuestro sistema cognitivo y a la memoria.
- Facilita la comunicación.

Darle nombre a una idea compleja -digamos a un patrón- es un ejemplo de la fuerza de la abstracción: convierte una forma compleja en una forma simple con sólo eliminar los detalles. Por tanto, los patrones GRASP poseen nombres concisos como Experto, Creador, Controlador.

Resumimos a continuación la introducción anterior:

- Asignar correctamente las responsabilidades es muy importante en el diseño orientado a objetos.
- La asignación de responsabilidades a menudo se asignan en el momento de preparar los diagramas de interacción.
- Los patrones son parejas de problema/solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades.

Dicho esto, podemos proceder a examinar los patrones GRASP

Pregunta: ¿Qué son los patrones GRASP?

Respuesta: Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Es importante entender y poder aplicar estos principios durante la preparación de un diagrama de interacción, pues un diseñador de software sin mucha experiencia en la tecnología de objetos debe dominarlos cuanto antes: constituyen el fundamento de cómo se diseñará el sistema. GRASP es un acrónimo que significa General Responsibility Assignment Software Patterns (patrones generales de software para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

Los **primeros cinco patrones de GRASP** son:

- Experto
- Creador
- Alta Cohesión
- Bajo Acoplamiento
- Controlador

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen.

El patrón Experto -como tantas otras cosas en la tecnología de objetos- ofrece una analogía con el mundo real. Acostumbramos asignar responsabilidad a individuos que disponen de la información necesaria para llevar a cabo una tarea. Por ejemplo, en una empresa quién será el encargado de preparar un estado de pérdidas y ganancias? El empleado que tiene acceso a toda la información necesaria para elaborarlo: quizá el director financiero. Y del mismo modo que los objetos del software colaboran porque la información está esparcida, lo mismo sucede con el personal de una empresa. El director financiero podrá pedir a los encargados de cuentas por cobrar y de cuentas por pagar que generen reportes individuales sobre créditos y deudas.

Beneficios

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. (Bajo Acoplamiento es un patrón GRASP que examinaremos más adelante.)

- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase "sencillas" y más cohesivas que son más fáciles de comprender y de mantener. **Así** se brinda soporte a una **alta** cohesión (patrón del que nos ocuparemos luego).

El **patrón Creador** guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.

El Agregado **agrega** la Parte, el Contenedor **contiene** el contenido, el Registro **registra**. En un diagrama de clases se registran las relaciones muy frecuentes entre las clases. El patrón Creador indica que la clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear la cosa contenida o registrada. Desde luego, se trata tan sólo de una directriz. En ocasiones encontramos un patrón creador buscando la clase con los datos de inicialización que serán transferidos durante la creación. Este es en realidad un ejemplo del patrón Experto. Los datos de inicialización se transmiten durante la creación a través de algún método de inicialización, como un constructor en Java que cuenta con parámetros.

El **Bajo Acoplamiento** es un principio que debemos recordar durante las decisiones de diseño: es la meta principal que es preciso tener presente siempre. Es un *patrón evaluativo* que el diseñador aplica al juzgar sus decisiones de diseño.

Beneficios

- no se afectan por cambios de otros componentes
- fáciles de entender por separado
- fáciles de reutilizar

Como el patrón **Bajo Acoplamiento**, también Alta Cohesión es un principio que debemos tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Grady Booch señala que se da una alta cohesión funcional cuando los elementos de un componente (clase, por ejemplo) "colaboran para producir algún comportamiento bien definido" [Booch94].

El patrón **Alta Cohesión** -como tantas otras cosas en la tecnología de objetos- presenta semejanzas con el mundo real. Todos sabemos que, si alguien asume demasiadas responsabilidades -sobre todo las que debería delegar-, no será eficiente. Esto se observa en algunos gerentes que no han aprendido a delegar. Muestran baja cohesión; prácticamente ya están "desligados"

Beneficios

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Controlador La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario (IGU) operado por una persona.

Otros medios de entrada son los mensajes externos - entre ellos un conmutador de telecomunicaciones para procesar llamadas- o las señales procedentes de sensores como sucede en los sistemas de control de procesos.

En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan.

La misma clase controlador debería utilizarse con todos los eventos sistémicos de un *caso* de uso, de modo que podamos conservar la información referente al estado del *caso*.

Esta información será útil -por ejemplo- para identificar los eventos del sistema fuera de secuencia (entre ellos, una operación ***efectuarPago*** antes de ***terminarVenta***).

Puede emplearse varios controladores en los casos de uso.

Un defecto frecuente al diseñar controladores consiste en asignarles demasiada responsabilidad. Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. Favor de consultar la sección Problemas y soluciones donde se dan más detalles al respecto.

Beneficios

- Mayor potencial de los componentes reutilizables
- Reflexionar sobre el estado del caso de uso.

Los últimos **cuatro patrones GRASP** son:

- Polimorfismo
- Fabricación Pura
- Indirección
- No Hables con Extraños

Como el patrón Experto, el uso del **patrón Polimorfismo** está acorde al espíritu del patrón "Lo hago yo mismo" [Coad95]. En vez de operar externamente sobre un pago para autorizarlo, el pago se autoriza a sí mismo; esto constituye la esencia de la orientación a objetos.

Para diseñar una **Fabricación Pura** debe buscarse ante todo un gran potencial de reutilización, asegurándose para ello de que sus responsabilidades sean pequeñas y cohesivas.

Estas clases tienden a tener un conjunto de responsabilidades de ***granularidad fina***.

Una fabricación pura suele partirse atendiendo a su funcionalidad y, por lo mismo, es una especie de objeto de función central. Generalmente se considera que la fabricación es parte de la capa de servicios orientada a objetos de alto nivel en una arquitectura.

Muchos patrones actuales del diseño orientado a objetos constituyen ejemplos de Fabricación Pura: Adaptador, Observador, Visitante y otros más.

Beneficios

- Se brinda soporte a una Alta Cohesión porque las responsabilidades se dividen en una clase de granularidad fina que se centra exclusivamente en un conjunto muy específico de tareas afines.
- Puede aumentar el potencial de reutilización debido a la presencia de las clases de Fabricación Pura de granularidad fina, cuyas responsabilidades pueden utilizarse en otras aplicaciones.

"La mayoría de los problemas de la computación puede resolverlos otro nivel ***de indirección***", reza un adagio muy acorde a los diseños orientados a objetos.

Del mismo modo que muchos patrones actuales de diseño son especializaciones de la Fabricación Pura, también muchos son especialización de Indirección; por ejemplo, Adaptador, Fachada y Observador. Además, muchas clases de Fabricación Pura se generan a causa del patrón Indirección. El motivo de la Indirección casi siempre es el Bajo Acoplamiento; se agrega un intermediario con el fin de desacoplar otros componentes o servicios. Se asigna la responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios, y éstos no terminen directamente acoplados. El intermediario crea una ***indirección*** entre el resto de los componentes o servicios.

No Hables con Extraños se refiere a no obtener una visibilidad temporal frente a objetos indirectos, que son de conocimiento de otros objetos pero no del cliente. La desventaja de conseguir visibilidad ante extraños es la siguiente: la solución se acopla entonces a la estructura interna de otros objetos. Ello origina un alto acoplamiento, que hace el diseño menos robusto y más propenso a requerir un cambio si se alteran las relaciones estructurales indirectas. El código se desplaza por las conexiones estructurales saltando de un objeto al siguiente.

Reflexione sobre la fragilidad del código: está estrechamente acoplado al conocimiento de muchas relaciones estructurales indirectas.

