INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

M. en C. Martha Rosa Cordero López M. En C. Marco A. Dorantes Glez.

CONCEPTOS

La Ingeniería de Software implica seguir en cualquier proyecto de software una metodología de desarrollo y la utilización de distintas técnicas y herramientas. Los diferentes procedimientos a seguir en cualquier proyecto de Ingeniería de software son: Definición de requerimientos, Análisis, Diseño, Verificación y Validación (Pruebas de calidad del Software), Pruebas y Mantenimiento.

Así pues, definimos **Diseño de Software** como la acción de construir soluciones que satisfagan los requerimientos del cliente. Existen varias etapas en el proceso de diseño de software, a saber son:

- ✓Entendimiento del problema
- ✓Identificar una o más soluciones
- ✓Describir abstracciones de la solución
- ✓Repetir el proceso para cada abstracción identificada hasta que el diseño este
- ✓expresado en términos sencillos

El **Diseño Orientado a Objetos** se define como un diseño de sistemas que utiliza objetos auto-contenidos y clases de objetos.

Clases.

Hay dos grandes categorías de objetos: clases e instancias. Los usuarios de la POO piensan en las clases como contenedores de la información necesaria para crear instancias, estos son, la estructura y las capacidades de una instancia están determinadas por su correspondiente clase. Hay tres definiciones válidas de clase:

1. Una clase es un patrón, una plantilla para una categoría de objetos con la misma estructura. Los ítems creados con la clase se llaman instancias. Es como si la clase fuera un molde y la instancia un pastel.
2. Una clase consiste en un patrón y en el mecanismo de crear ítem basado en ese patrón. A esta clase se le puede ver como a una fábrica de instancias. Las instancias son ítem manufacturado usando el mecanismo de creación de ítem de la clase.
3. Una clase es un conjunto de ítems creados usando un patrón específico. Dicho de otra manera, la clase es el conjunto de todas las instancias de ese patrón.

Superclases.

Una superclase es una clase cuyas instancias también son clases. Esto significa que cuando se use el mecanismo de creación de instancias en una metaclase, la instancia creada será una clase.

CONCEPTOS

Una definición que también usaremos es la de instanciación que tiene dos significados:

- Como verbo, instanciación es el proceso de creación de la instancia de una clase, y
- Como nombre, una instanciación es una instancia de una clase.

OBJETO **D**entro del software orientado a objeto, un objeto es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos.

TIPO DE OBJETO

Los conceptos que poseemos se aplican a tipos determinados de objetos. Por ejemplo, empleado se aplica a los objetos que son personas empleadas por alguna organización. Algunas instancias de empleado podrían ser Juan Pérez, José Martínez, etc. En el análisis orientado a objetos, estos conceptos se llaman tipos de objetos; las instancias se llaman objetos. Así, un tipo de objeto es una categoría de objeto, mientras que un objeto es una instancia de un tipo de objeto.

MÉTODOS

Los métodos especifican la forma en que se controlan los datos de un objeto. Los métodos en un tipo de objeto sólo hacen referencia a la estructura de datos de ese tipo de objeto. No deben tener acceso directo a las estructuras de datos de otros objetos. Para utilizar la estructura de datos de otro objeto, deben enviar un mensaje a éste. El tipo de objeto empaca juntos los tipos de datos y su comportamiento.

ENCAPSULADO El encapsulado oculta los detalles de su implantación interna a los usuarios de un objeto. Los usuarios se dan cuenta de las operaciones que puede solicitar del objeto, pero desconocen los detalles de cómo se lleva a cabo la operación. Todos los detalles específicos de los datos del objeto y la codificación de sus operaciones están fuera del alcance del usuario.
Así, encapsulado es el resultado (o acto) de ocultar los detalles de implantación de un objeto respecto de su usuario.

MENSAJES
Para que un objeto haga algo, le enviamos una solicitud. Esta hace que se produzca una operación. La operación ejecuta el método apropiado y, de manera opcional, produce una respuesta. El mensaje que constituye la solicitud contiene el nombre del objeto, el nombre de una operación y, a veces, un grupo de parámetros.

HERENCIA
Un tipo de objeto de alto nivel puede especializarse en tipos de objeto de bajo nivel. Un tipo de objeto puede tener subtipos. Por ejemplo, el tipo de objeto persona puede tener subtipos estudiante y empleado. A su vez, el tipo de objeto estudiante puede tener como subtipo estudiante de pregrado y estudiante de postgrado, mientras que empleado puede tener como subtipo a académico y administrativo. Existe de este modo una jerarquía de tipos, subtipos, subsubtipos, etc.

CONCEPTOS

ATRIBUTO

- Propiedad del objeto
- Determinan el estado del objeto
- Son los datos que contiene y encapsula una clase(ocultas al exterior) y se acceden por medio de algún método
- Es un valor de un dato que está almacenado en los objetos de una clase
- Los atributos son llamados variables de instancia
- Podrían verse como una variable global a toda la clase.

POLIMORFISMO

Es la propiedad que tienen los objetos de permitir invocar genéricamente un comportamiento (método) cuya implementación será delegada al objeto correspondiente recién en tiempo de ejecución. El polimorfismo tiende a existir en las relaciones de herencia, pero no siempre es así.

Características del paradigma Orientado a Objetos

La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de los años 1990. En la actualidad, existe variedad de lenguajes de programación que soportan la orientación a objetos.

La programación orientada a objetos, intenta simular el mundo real a través del significado de objetos que contiene características y funciones. Los lenguajes orientados a objetos se clasifican como lenguajes de quinta generación.

Como su mismo nombre indica, la programación orientada a objetos se basa en la idea de un objeto, que es una combinación de variables locales y procedimientos llamados métodos que juntos conforman una entidad de programación.

Características del paradigma Orientado a Objetos

Existe un acuerdo acerca de qué características contempla la "orientación a objetos", las características siguientes son las más importantes:

Abstracción: denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar *cómo* se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real.

Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

Características del paradigma Orientado a Objetos

Modularidad: Se denomina Modularidad a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

Principio de ocultación: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una *interfaz* a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado.

Polimorfismo: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado.

Herencia: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Cuando un objeto hereda de más de una clase se dice que hay *herencia múltiple*.

Recolector de basura: la recolección de basura o *garbage collector* es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando. En la mayoría de los lenguajes híbridos que se extendieron para soportar el Paradigma de Programación Orientada a Objetos como C++ u Object Pascal, esta característica no existe y la memoria debe desasignarse manualmente.

Metodologías Orientado a Objetos

El análisis y diseño orientado a objetos constituye una nueva forma de pensar acerca de problemas empleando modelos que son útiles para comunicarse con expertos en esa aplicación, modelar empresas, preparar documentación, diseñar programas y bases de datos.

Un modelo es una abstracción de algo, cuyo objetivo es comprenderlo antes de construirlo. Dado que los modelos omiten los detalles no esenciales, es más sencillo manipularlos que manipular la entidad original. La abstracción es una capacidad humana fundamental que nos permite enfrentarnos a la complejidad. Los ingenieros, artistas y artesanos han estado construyendo modelos durante miles de años para probar los diseños antes de ejecutarlos. El desarrollo de sistemas hardware y software no es una excepción. Para construir sistemas complejos, el desarrollador debe abstraer distintas vistas del sistema, construir modelos utilizando notaciones precisas, verificar que los modelos satisfacen los requisitos del sistema y añadir, gradualmente, detalles para trasformar los modelos en una implementación.
(Rumbaugh, 1996)

OMT (OBJECT MODELING TECHNIQUE)

La técnica de modelado de objetos (OMT) es un lenguaje de modelado de objetos para software de modelado y diseño. Se desarrolló alrededor de 1991 por Rumbaugh, Blaha, Premerlani, Eddy y Lorensen como un método para desarrollar sistemas orientados a objetos y apoyar orientada a objetos Modelo de objetos programming.Describes o estructura estática del sistema.

OMT se desarrolló como un enfoque para el desarrollo de software. Los propósitos de modelado de acuerdo con Rumbaugh son:

- probando entidades físicas antes de su construcción (simulación),
- comunicación con los clientes,
- visualización (presentación alternativa de la información), y
- reducción de la complejidad.

OMT ha propuesto tres tipos principales de modelos:

Modelo del objeto: El modelo de objetos representa los fenómenos estáticos y estables en el dominio de modelado principales conceptos son clases y asociaciones con atributos y operaciones. Agregación y generalización (con herencia múltiple) son relaciones predefinidas.

Modelo dinámico: El modelo dinámico representa una vista de estado / transición del modelo. Principales conceptos son estados, transiciones entre estados y eventos para desencadenar transiciones.

Modelo funcional: El modelo funcional se encarga de la perspectiva de proceso del modelo, que corresponde aproximadamente a los diagramas de flujo de datos. Principales conceptos son procesar, almacenar datos, flujo de datos, y los actores.

OMT es un predecesor del Lenguaje de Modelado Unificado (UML). Muchos elementos de modelado OMT son comunes a UML.

UNIFIED PROCESS (UP)

El Proceso Unificado no es simplemente un proceso, sino más bien un marco extensible que debe ser personalizado para las organizaciones o proyectos específicos. El Proceso Unificado Racional es, asimismo, un marco personalizable.

Características Unified Process

El Proceso Unificado es un proceso iterativo e incremental de desarrollo. Las fases de elaboración, construcción y transición se dividen en una serie de iteraciones timeboxed. (La fase inicial también se puede dividir en iteraciones para un proyecto grande.) Cada uno de los resultados de iteración en un incremento, que es una versión del sistema que contiene funcionalidad ni mejorado en comparación con la versión anterior. En el Proceso Unificado, casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. Cada iteración toma un conjunto de casos de uso o escenarios de los requisitos de todo el camino hasta la implementación, prueba y despliegue.

Ciclo de vida del proyecto

El Proceso Unificado divide el proyecto en cuatro fases:

- comienzo
- elaboración
- construcción
- transición

RATIONAL UNIFIED PROCESS (RUP)

Según Jacobson, I., Booch, G., Rumbaugh J. (1998)1 El nombre Proceso Unificado se usa para describir el proceso genérico que incluye aquellos elementos que son comunes a la mayoría de los refinamientos existentes. También permite evitar problemas legales ya que Proceso Unificado de Rational o RUP son marcas registradas por IBM (desde su compra de Rational Software Corporation en 2003).

El Proceso Unificado de Rational (Rational Unified Process en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software desarrollado por la empresa Rational Software, actualmente propiedad de IBM. Junto con el Lenguaje Unificado de

El RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

Fases:

- Establece oportunidad y alcance
- Identifica las entidades externas o actores con las que se trata
- Identifica los casos de uso

Principales características

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo)
- Pretende implementar las mejores prácticas en Ingeniería de Software
- Desarrollo iterativo
- Administración de requisitos
- Uso de arquitectura basada en componentes
- Control de cambios
- Modelado visual del software
- Verificación de la calidad del software

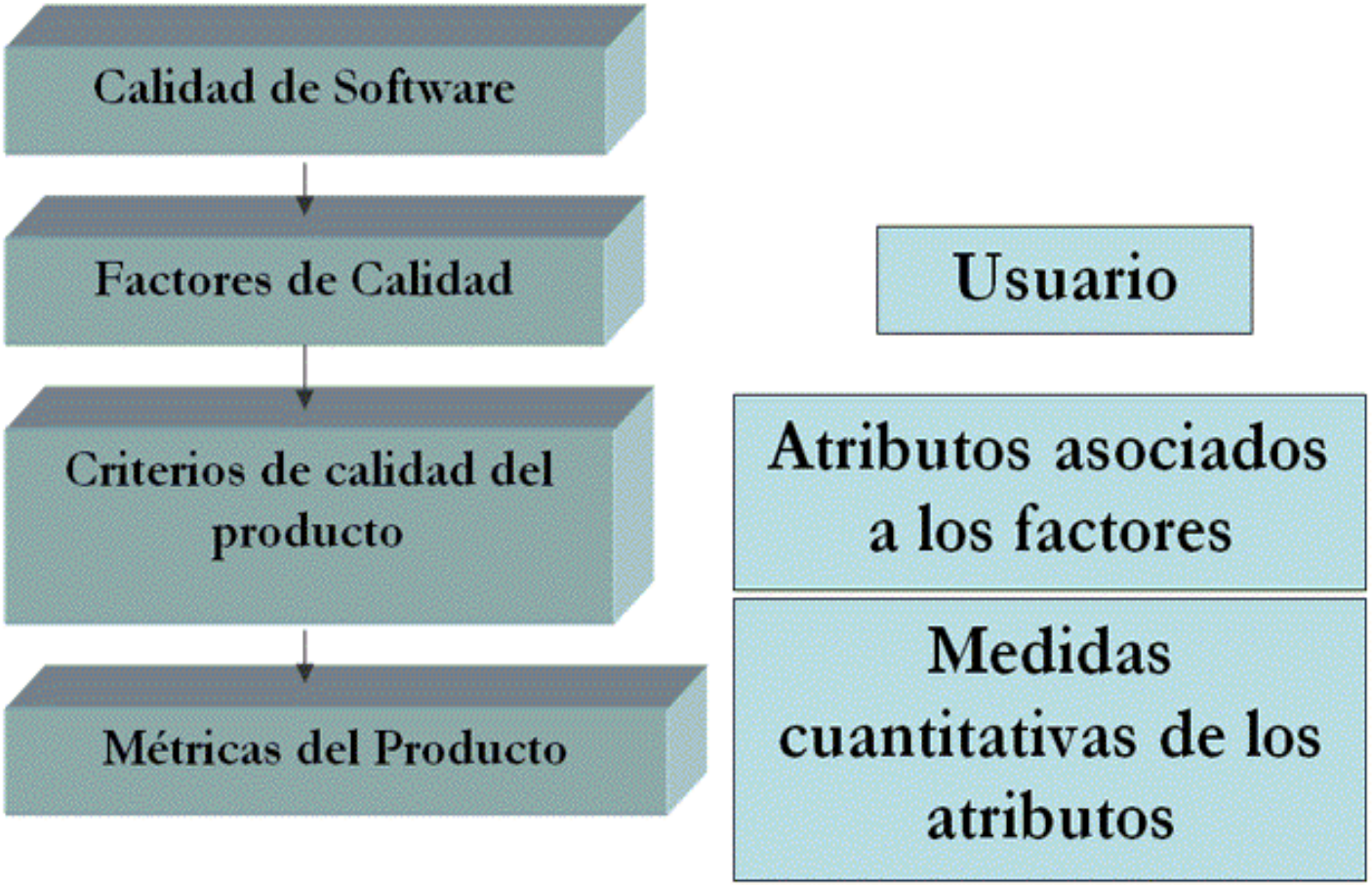
LA CALIDAD DE LOS SISTEMAS SOFTWARE

La calidad del software. La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo y de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.
Los requisitos del software son la base de las medidas de calidad. La falta de concordancia con los requisitos es una falta de calidad.

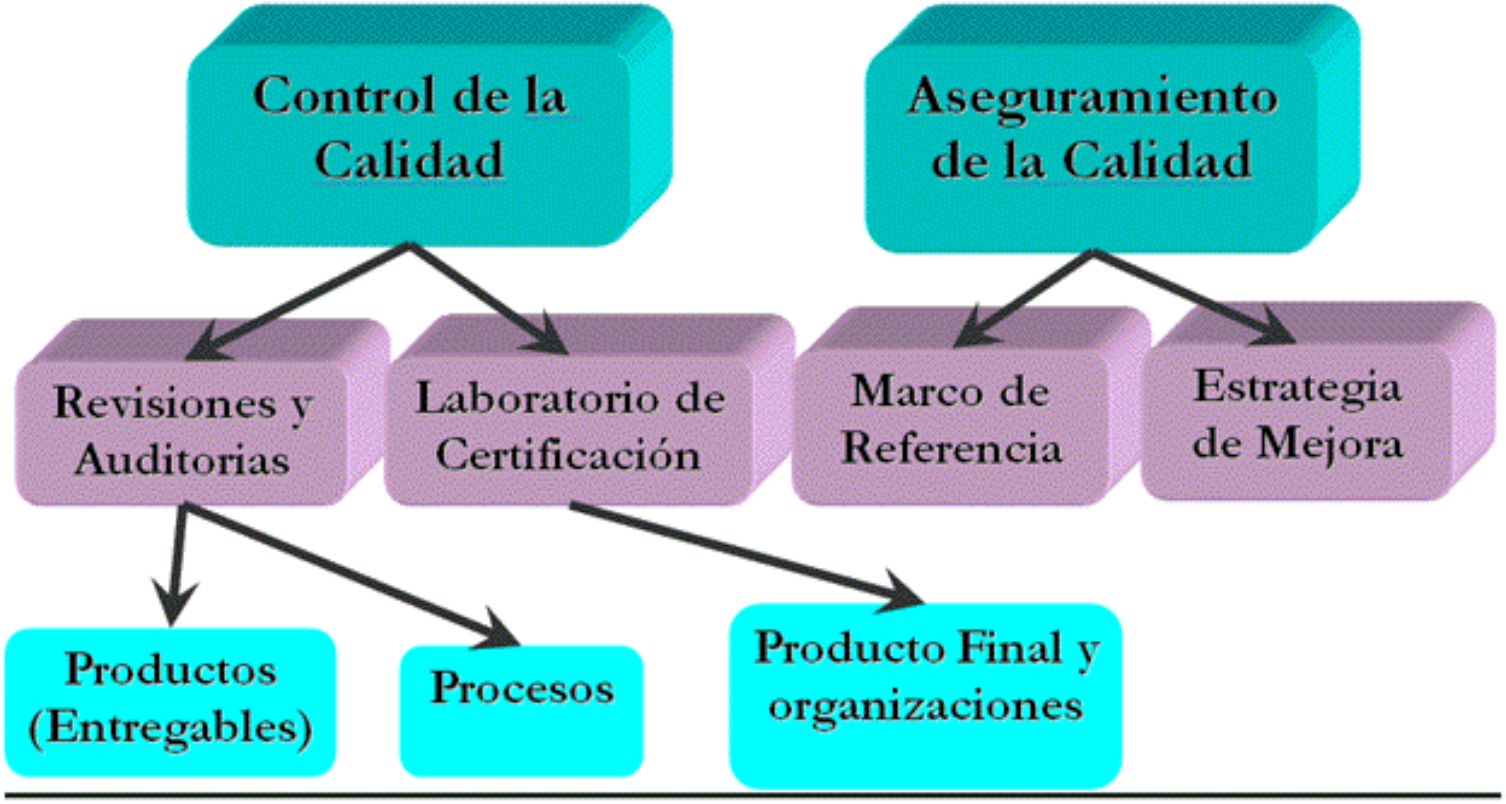
Los estándares o metodologías definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software. Si no se sigue ninguna metodología siempre habrá falta de calidad.

La política establecida debe estar sustentada sobre tres principios básicos: tecnológico, administrativo y ergonómico.
El principio tecnológico define las técnicas a utilizar en el proceso de desarrollo del software.

LA CALIDAD DE LOS SISTEMAS SOFTWARE



LA CALIDAD DE LOS SISTEMAS SOFTWARE



Atributos de calidad y entidades de diseño

En primer lugar debemos acotar el contexto en el que nos movemos, ya que la calidad, en el software, se puede entender como calidad de proceso o de producto. La calidad de proceso ha sido objeto de mucho interés en las últimas décadas (Humphrey,1989) y su desarrollo ha concienciado a los profesionales de la necesidad de aplicar la calidad a otros aspectos del software. En nuestro caso la única norma que hemos encontrado en la literatura que da una definición de calidad de producto software en base a atributos es ISO 9126 (ISO/IEC, 2001).

Dicha norma establece que la medición de la calidad de un producto software debe hacerse en base a sus atributos, siendo éstos internos, propiedades características de cómo se estructura el software, o externos, cualidades observables aún sin conocer cómo está construido.

De hecho no se habla de calidad, en general, sino de calidad interna y calidad externa, afirmándose que la calidad interna de un producto software influye directamente en su calidad externa.

Fiabilidad:
La capacidad del software de mantener un nivel específico de rendimiento bajo determinadas condiciones de uso.

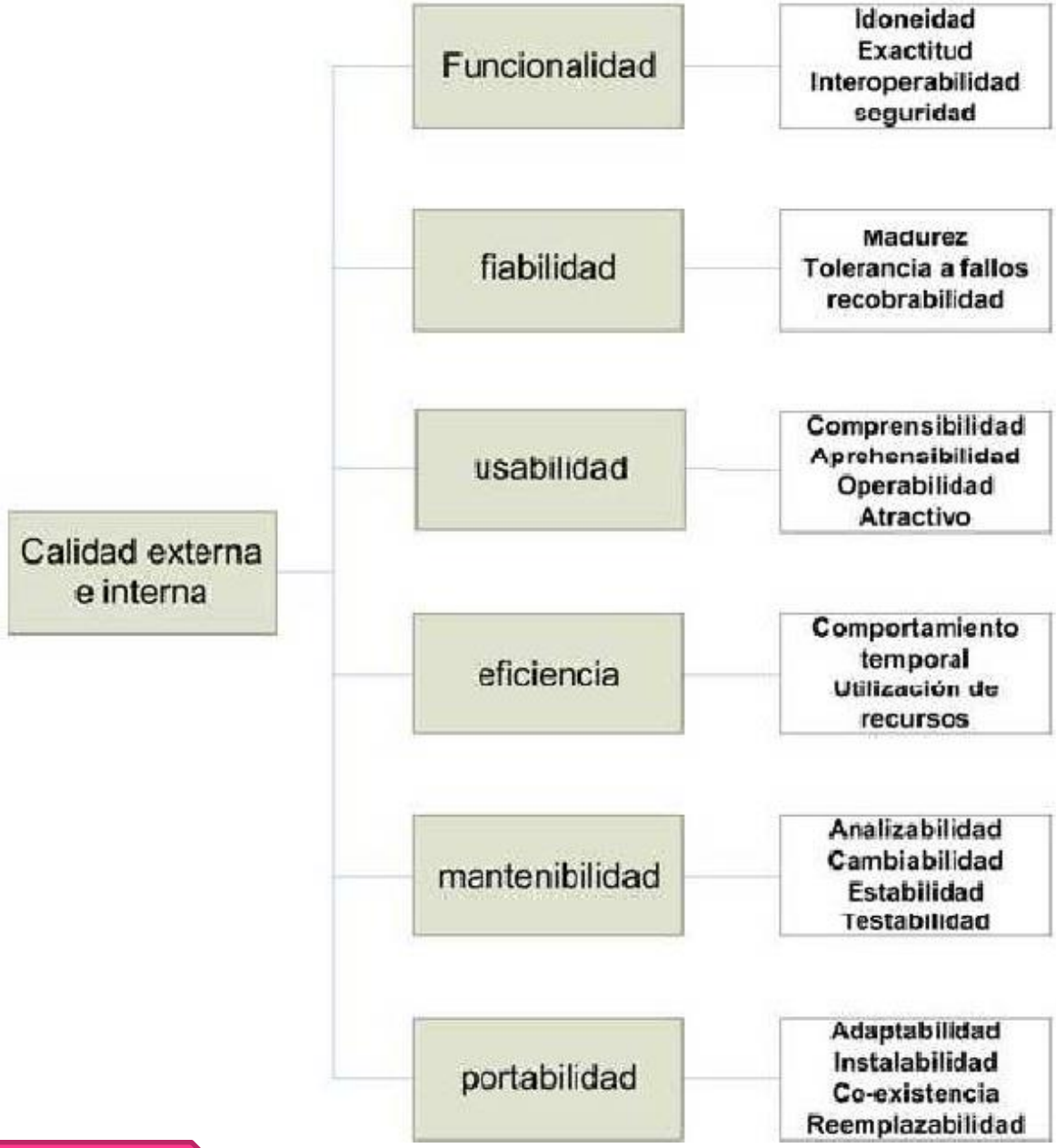
Usabilidad:
La capacidad del producto software de ser entendido, aprendido, usado y atractivo al usuario, cuando se usa bajo ciertas condiciones.

Eficiencia:
La capacidad del software de ofrecer el rendimiento apropiado con respecto a la cantidad de recursos utilizados, bajo condiciones prefijadas.

Mantenibilidad:
La capacidad del producto de ser modificado. Dichas modificaciones pueden incluir correcciones, mejoras o adaptaciones a cambios en el entorno y en los requisitos y especificaciones funcionales.

Portabilidad:
La capacidad del software de ser trasladado de un entorno (informático) a otro. Estas características son generales para cualquier tipo de programa informático o software independientemente de si el paradigma empleado para construirlo es el Orientado a Objetos, el estructurado u otro cualquiera, sin embargo sí que afectará a la manera de medirlas. Dichas características se dividen a su vez en subccaracterísticas tal como se puede ver en el siguiente esquema.

LA CALIDAD DE LOS SISTEMAS SOFTWARE



MÉTRICAS

Una Métrica de un proyecto es la medida de alguna propiedad de un entregable del proyecto o del proceso de administración de proyectos, efectuada para conocer el avance o los desvíos al plan original.

Si se definen métricas acerca de un entregable específico, estas métricas son particulares al proyecto. Las métricas relacionadas al proceso de administración de proyectos pueden usarse en todo tipo de proyectos. Las métricas pueden ser usadas para medir el estado, efectividad o progreso de las actividades de un proyecto y así contribuir a tomar decisiones estratégicas ante los desvíos, incidentes o diferentes problemas que surgen en la ejecución.

¿Para qué sirven las Métricas?

Identifican eventos y tendencias importantes en los proyectos y otorgan a la organización la información necesaria para la toma de decisiones.

Sirven como vocabulario común entre el grupo de personas que participa de la implementación de los proyectos, y el grupo que los patrocina (Sponsors, Stakeholders).

Sirven como motivación para el equipo, porque relacionan en esfuerzo personal de los miembros con los resultados generales del proyecto.

MÉTRICAS

Hay distintos tipos de MOO, como por ejemplo:

- Métricas orientadas a clases
- Métricas orientadas a operaciones
- Métricas para pruebas orientadas a objetos
- Métricas para proyectos orientados a objetos

Algunos métodos de este tipo de métricas son:

- ✓ Métodos ponderados por clase (C&K)
- ✓ Árbol de profundidad de herencia (C&K)
- ✓ Número de Descendientes (C&K)
- ✓ Tamaño de Clase (Lorenz y Kidd)
- ✓ Índice de Especialización (Lorenz y Kidd)

Métricas CK Chidamber y Kemerer

Son métricas orientadas a clases: clases individuales, herencia y colaboraciones. Es uno de los conjuntos de métricas más referenciado.

a) Métodos ponderados por clase (WMC: Weighted Methods per Class). Calcula la suma de la complejidad ciclo matica de los métodos de una clase:

WMC=∑i..n mci , siendo mci la complejidad ciclo matica del método i.

Las principales interpretaciones de esta métrica son las siguientes:

El número y la complejidad de los métodos son indicadores del tiempo necesario para desarrollar/mantener la clase.

Cuanto mayor sea el nº de métodos mayor impacto potencial tendrá en los hijos, sus herederos potenciales.

Las clases con gran nº de métodos serán de aplicación específica, y por lo tanto más difíciles de reutilizar

b) Profundidad en el árbol de herencia (DIT: Depth Inheritance Tree). Es la distancia desde una clase a la raíz del árbol de herencia. Cuanto más alto es el mayor valor de DIT, mayor complejidad hay en el diseño y cuanto más alto sea el valor de DIT de una clase más posibilidades existen de que reutilice/refine métodos heredados.

Las principales interpretaciones de esta métrica son las siguientes:

A mayor profundidad de la clase, más métodos puede heredar y es más difícil de explicar su comportamiento. A mayor profundidad de una clase, mayor posibilidad de reutilización de métodos heredados.

MÉTRICAS

c) Número de hijos inmediatos en el árbol de herencia (NOC: Number Of Children). En principio, cuanto más alto es el valor de NOC, una clase es más reutilizable pero también la probabilidad de que se hayan hecho extensiones no apropiadas de la clase es mayor.

Las principales interpretaciones de esta métrica son las siguientes:

Cuanto mayor sea NOC más reutilización habrá por herencia.

Si NOC es muy grande hay un fallo en la abstracción de la clase padre, falta algún nivel intermedio. NOC da una idea del peso que la clase tiene en el diseño, y de los recursos que se deben dedicar a probar sus métodos.

d) Acoplamiento entre clases (CBO: Coupling Between Object Classes). Es el número de clases acopladas a una clase. Dos clases están acopladas cuando los métodos de una de ellas usan variables o métodos de una instancia de la otra clase. Si existen varias dependencias sobre una misma clase es computada como una sola. No es deseable que CBO > 14. Cuanto más alto es el más dificil será el mantenimiento y el reuso y en general el código será más propenso a fallos.

Las interpretaciones de esta métrica son las siguientes:

Cuanto mayor es CBO, peor es la modularidad y la reutilización.

Cuanto mayor es CBO, peor es el encapsulamiento y más cuesta mantenerlo.

e) Respuesta para una clase (RPC: Response for a Class). Es el número de métodos que pueden ser ejecutados en respuesta a un mensaje recibido por un objeto de esa clase. Cuanto mayor sea RPC, mayor esfuerzo se requiere para su comprobación, y más complejo es el diseño. Existen dos variaciones de esta métrica:

RPC: Sólo considera el primer nivel del árbol de llamadas: número de métodos de una clase más el número de métodos remotos llamados directamente por una clase.

Las principales interpretaciones de esta métrica son las siguientes

a. Cuanto mayor sea la respuesta de una clase, más complicadas serán las pruebas y el mantenimiento.

b. Cuanto mayor sea la respuesta de una clase, mayor será la complejidad de la clase.

f) Carencia de cohesión (LCOM: Lack of Cohesion). Cada método de una clase tiene acceso a uno o más atributos. LCOM calcula el conjunto de atributos comunes en los métodos de una clase. Dos métodos son similares si comparten al menos un atributo de la clase. A mayor número de atributos similares, mayor cohesión hay en la clase.

Métricas de Li-Henry

a) APM Acoplamiento por paso de mensajes: número de mensajes o métodos invocados enviados por una clase a otras clases del sistema.

b) AAD Acoplamiento por abstracción de datos: (se considera una definición ambigua con dos posibles interpretaciones):

I. Número de tipos de datos abstractos definidos en una clase.

II. Número de atributos de una clase que se referencian desde otra clase.

c) NML Número de métodos localmente definidos en la clase.

d) Tamaño: número de atributos y métodos locales de una clase.

Métrica	Definición	Propiedades
Message passing coupling (MPC)	MPC = número de métodos invocados en un clase.	Acoplamiento/Comunicación
Data abstraction coupling (DAC)	El número de atributos en una clase que tienen como tipo otra clase.	Acoplamiento/Abstracción
SIZE1	Es una variación de la tradicional LOC (Lineas de Código) definida específicamente para el lenguaje Ada. Obviamos la definición	Tamaño del diseño
SIZE2	SIZE2 = número de atributos + número de métodos locales.	Tamaño del diseño

TABLA III: Métricas de Li y Henry

Métricas de Bansiya y Davis

Métrica	Definición	Propiedades
Design size of classes (DSC)	Número total de clases en el diseño	Tamaño del diseño
Number of Hierarchies (NOH)	Número de jerarquías de clases	Jerarquías
Average number of ancestors (ANA)	Número medio de ancestros	Abstracción
Data access Metric (DAM)	Relación entre el número de atributos privados (protegidos) y el número total de atributos de la clase.	Encapsulamiento
Direct class coupling (DCC)	Número de clases diferents con las que una clase está relacionada. Esta métrica incluye clases que están relacionadas directamente mediante la declaración de atributos y el paso de parámetros de los métodos.	Acoplamiento
Cohesion among methods of class (CAM)	Esta métrica calcula la relación entre métodos de una clase basándose en la lista de parámetros de los métodos. Se calcula utilizando la suma de la intersección de parámetros de un método con el conjunto máximo independiente de todos los tipos de parámetros de la clase.	Cohesión
Measure of aggregation (MOA)	Mide la extension de la relación parte/todo, mediante el uso de atributos. Es el número de declaraciones de variables cuyo tipo es definido por el usuario.	Composición
Measure of functional abstraction (MFA)	Relación del número de métodos heredados por una clase y el número total de métodos accesibles por un método miembro.	Herencia
Number of polymorphic methods (NPM)	Número de métodos que pueden mostrar comportamiento polimórfico (virtual en C++ y no final en Java)	Polimorfismo
Class interface size (CIS)	Número de métodos públicos en una clase	Comunicación
Number of methods (NOM)	Número de métodos definidos en una clase	Complejidad

TABLA IV: Métricas de Bansiya y Davis

MODELO 4+1

Un gran problema que sucede cuando se intenta adoptar una arquitectura para modelar un sistema. Pero sucede que los diagramas que se utilizan para esto intentan agrupa diferentes estilos de arquitectura en un solo modelo. Al mismo tiempo puede ocurrir que se pone demasiado énfasis en alguno de los aspectos de ingeniería de software. El modelo de 4+1 vistas fue desarrollado para remediar estos problemas. El modelo 4+1 describe la arquitectura del software usando cinco vistas concurrentes. Cada vista se refiere a un conjunto de intereses de diferentes stakeholders del sistema.

La arquitectura del software se trata de abstracciones, de descomposición y composición, de estilos y estética. También tiene relación con el diseño y la implementación de la estructura de alto nivel del software. Los diseñadores construyen la arquitectura usando varios elementos arquitectónicos elegidos apropiadamente. Estos elementos satisfacen la mayor parte de los requisitos de funcionalidad y performance del sistema, así como también otros requisitos no funcionales tales como confiabilidad, escalabilidad, portabilidad y disponibilidad del sistema.

MODELO 4+1

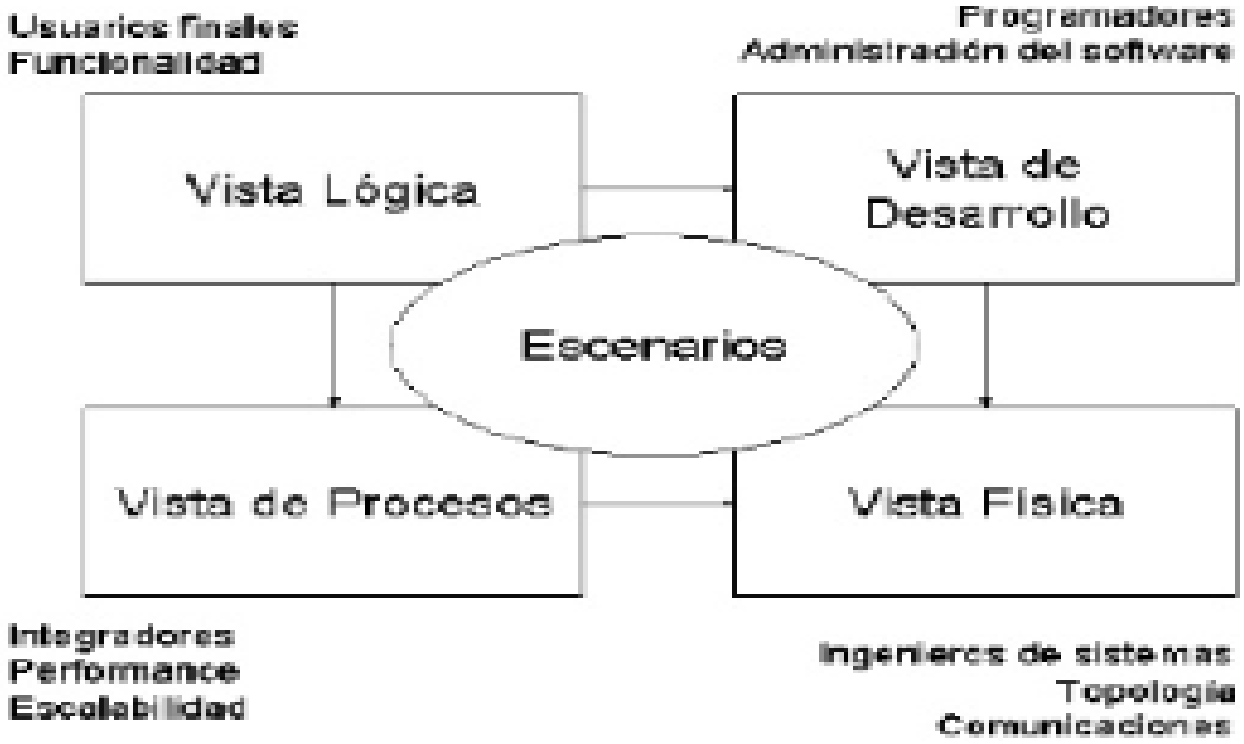


Figura 1: Modelo de "4+1"

La Arquitectura Lógica

La arquitectura lógica apoya principalmente los requisitos funcionales lo que el sistema debe brindar en términos de servicios a sus usuarios. El sistema se descompone en una serie de abstracciones clave, tomadas (Principalmente) del dominio del problema en la forma de objetos o clases de objetos. Aquí se aplican los principios de abstracción, encapsulamiento y herencia. Esta descomposición no sólo se hace para potenciar el análisis funcional, sino también sirve para idénticar mecanismos y elementos de diseño comunes a diversas partes del sistema. Usamos el enfoque de Booch/Rational para representar la arquitectura lógica, mediante diagramas de clases y templetes de clases. Un diagrama de clases muestra un conjunto de clases y sus relaciones lógicas: asociaciones, uso, composición, herencia y similares.

Notación.

La notación para la lógica se deriva de la notación de Booch. Esta se simplifica considerablemente de tal modo de tener en cuenta solamente los ítems relevantes para la arquitectura. En particular, los numerosos adornos disponibles son bastante inútiles a este nivel de diseño. Usamos Rational Rose para apoyar el diseño lógico de la arquitectura.

MODELO 4+1

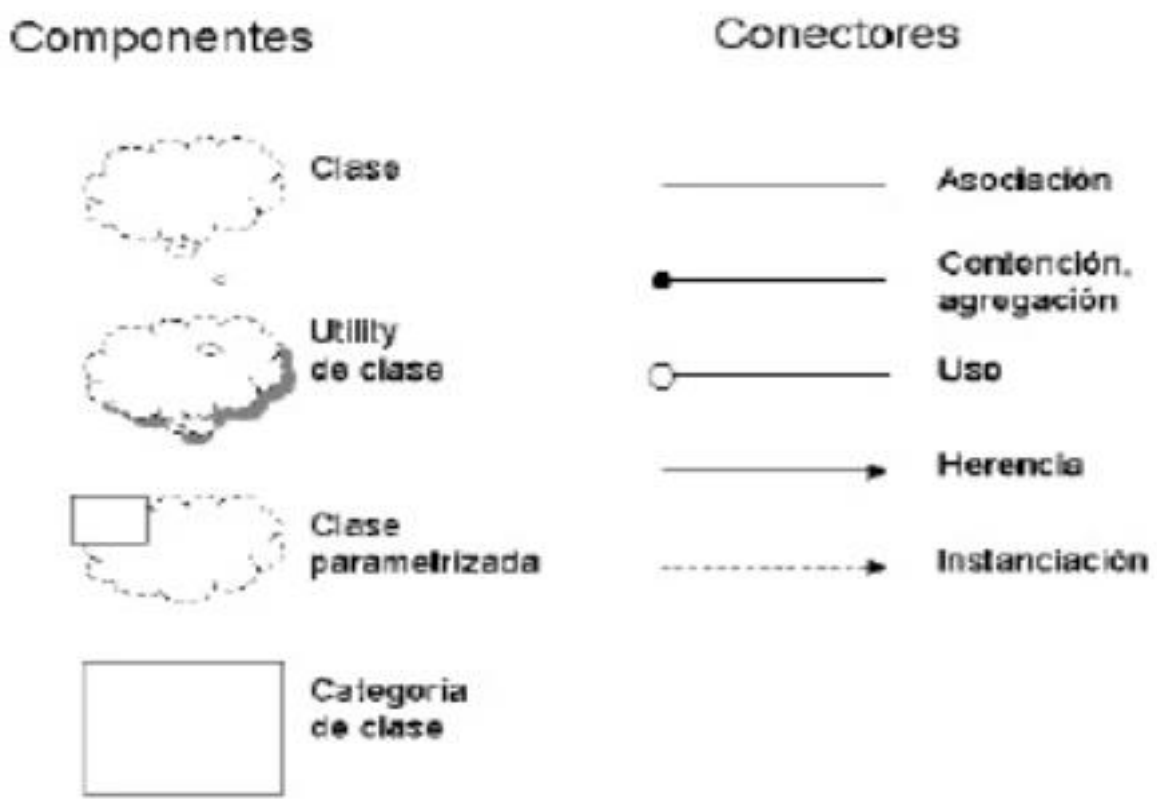


Figura 2: Notación para la lógica

La Vista de Procesos

La arquitectura de procesos toma en cuenta algunos requisitos no funcionales tales como la performance y la disponibilidad. Se enfoca en asuntos de concurrencia y distribución, integridad del sistema, de tolerancia a fallas. La vista de procesos también especifica en cuál hilo de control se ejecuta efectivamente una operación de una clase identificada en la vista lógica. La arquitectura de procesos se describe en varios niveles de abstracción, donde cada nivel se refiere a distintos intereses. El nivel más alto la arquitectura de procesos puede verse como un conjunto de redes lógicas de programas comunicantes (llamados “procesos”) ejecutándose en forma independiente, y distribuidos a lo largo de un conjunto de recursos de hardware conectados mediante un bus, una LAN o WAN. Además, los procesos pueden replicarse para aumentar la distribución de la carga de procesamiento, o para mejorar la disponibilidad.

Notación.

La notación que usamos para la vista de procesos se expande de la notación originalmente propia por Booch para las tareas de Ada y se centra solamente en los elementos arquitectónicamente relevantes (Figura 3).

MODELO 4+1

Figura 3: Notación para el diagrama de procesos

Vista de Desarrollo

La vista de desarrollo se centra en la organización real de los módulos de software en el ambiente de desarrollo del software. El software se empaqueta en partes pequeñas bibliotecas de programas o sub sistemas que pueden ser desarrollados por uno o un grupo pequeño de desarrolladores. Los subsistemas se organizan en una jerarquía de capas, cada una de las cuales brinda una interfaz estrecha y bien definida hacia las capas superiores.

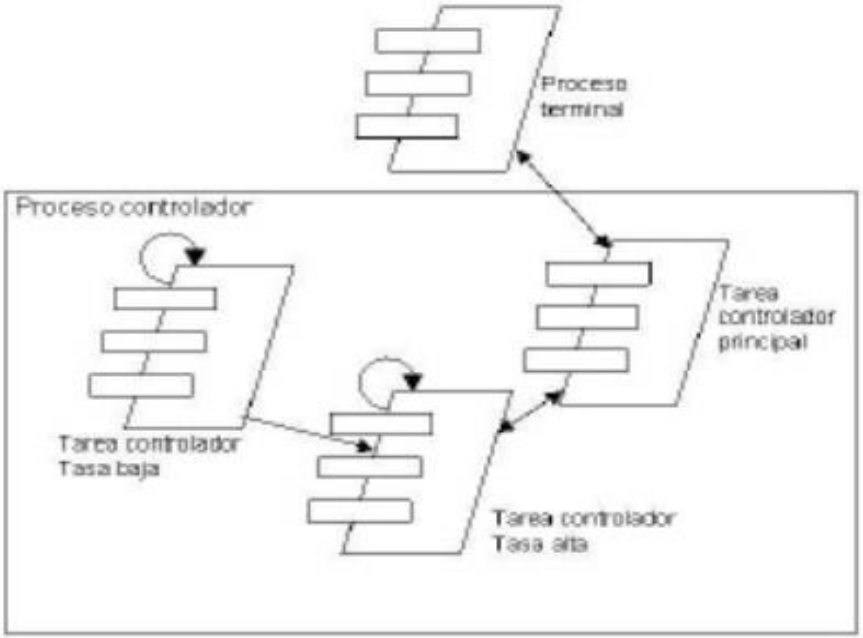


Figura 4: Diagrama (parcial) de procesos para Télic PBX

MODELO 4+1

La vista de desarrollo de un sistema se representa en diagramas de módulos o subsistemas que muestran las relaciones exporta e importa. La arquitectura de desarrollo completa sólo puede describirse completamente cuando todos los elementos del software han sido identificados. Sin embargo, es posible listar las reglas que rigen la arquitectura de desarrollo partición, agrupamiento, visibilidad antes de conocer todos los elementos.

Notación.

Tal como se muestra en la Figura 5, usamos una variante de la notación de Booch limitándonos a aquellos items relevantes para la arquitectura.

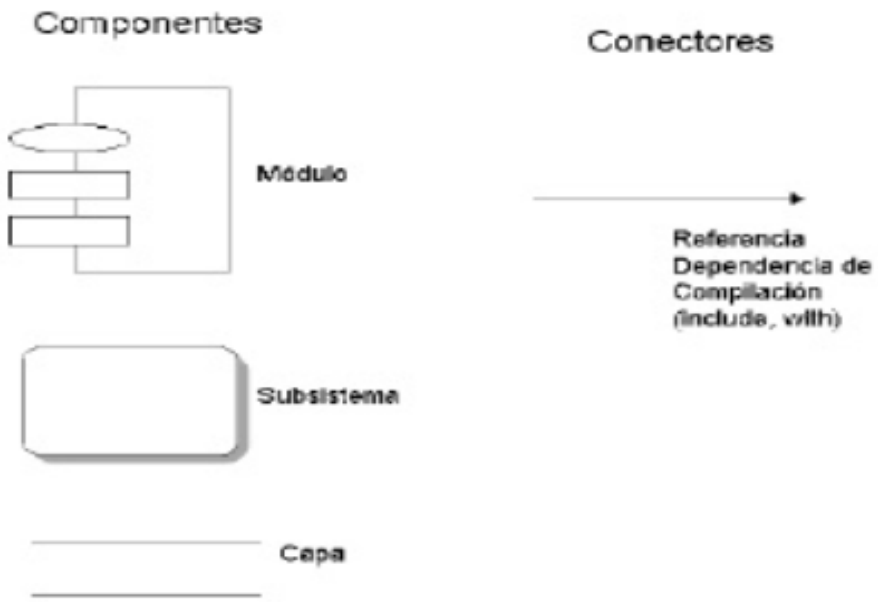


Figura 5: Notación para el diagrama de desarrollo

Arquitectura Física

Mapeando el software al hardware

La arquitectura física toma en cuenta primeramente los requisitos no funcionales del sistema tales como la disponibilidad, con fiabilidad (tolerancia a fallas), performance (throughput), y escalabilidad. El software ejecuta sobre una red de computadores o nodos de procesamiento (o tan solo nodos). Los variados elementos identificados redes, procesos, tareas y objetos requieren ser mapeados sobre los variados nodos. Esperamos que diferentes configuraciones puedan usarse: algunas para desarrollo y pruebas, otras para emplazar el sistema en varios sitios para distintos usuarios. Por lo tanto, el mapeo del software en los nodos requiere ser altamente flexible y tener un impacto mínimo sobre el código fuente en sí.

Notación.

Los diagramas físicos pueden tornarse muy confusos en grandes sistemas, y por lo tanto toman diversas formas, con o sin él mapeo de la vista de procesos.

MODELO 4+1

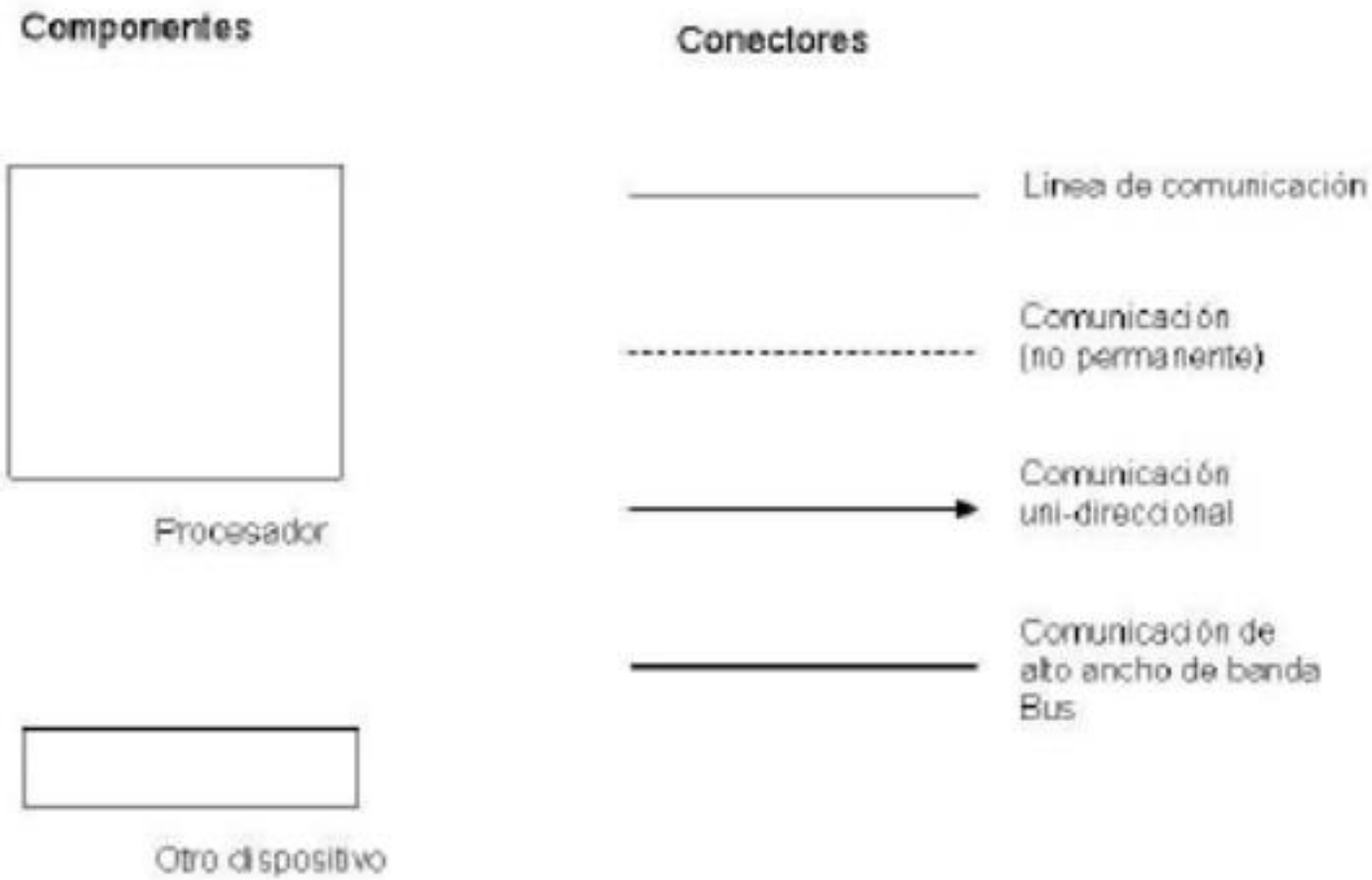


Figure 6: Notación para el diagrama físico

MODELO 4+1

Escenarios

Todas las partes juntas. Los elementos de las cuatro vistas trabajan conjuntamente en forma natural mediante el uso de un conjunto pequeño de escenarios relevantes instancias de casos de uso más generales para los cuales describimos sus

Scripts correspondientes (secuencias de interacciones entre objetos y entre procesos) tal como lo describen Rubín y Goldberg. Los escenarios son de alguna manera una abstracción de los requisitos más importantes. Su diseño se expresa mediante el uso de diagramas de escenarios y diagramas de interacción de objetos.

Notación.

La notación es muy similar a la vista lógica para los componentes (ver Figura 2), pero usa los conectores de la vista de procesos para la interacción entre objetos (ver Figura 3). Nótese que las instancias de objetos se denotan con líneas sólidas. Para el diagrama lógico, capturamos y administramos los diagramas de escenarios de objetos usando Rational Rose.

UML Y EL MODELADO

El Lenguaje Unificado de Modelado preescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas.

DIAGRAMAS

•Diagramas de Casos de Uso para modelar los procesos ’business’.
• Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
• Diagramas de Colaboración para modelar interacciones entre objetos.
• Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
• Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
• Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
• Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
• Diagramas de Componentes para modelar componentes.
• Diagramas de Implementación para modelar la distribución del sistema.

UML es una consolidación de muchas de las notaciones y conceptos más usados orientados a objetos.

IMPORTANCIA DE MODELAR

Para modelar los programas de forma gráfica, viendo las relaciones entre las clases y cómo colaboran unos objetos con otros, se prefiere utilizar una notación estándar. De esta manera, el alumno va conociendo parte de la notación, UML en este caso, lo cual le será útil por varias razones:

Le permite ir familiarizándose con un lenguaje de modelado estándar que verá en otras asignaturas de la titulación.

Le permite entender modelos con esta notación

Principios básicos del modelado

Existen cuatro principios básicos, estos principios son fruto de la experiencia en todas las ramas de la ingeniería.

a) La elección de qué modelos se creen influye directamente sobre cómo se acomete el problema. Hay que seleccionar el modelo adecuado para cada momento y dependiendo de que modelo se elija se obtendrán diferentes beneficios y diferentes costes. En la industria software ese ha comprobado que un modelado orientado a objetos proporciona unas arquitecturas más flexibles y readaptables que otros por ejemplo orientados a la funcionalidad o a los datos.

b) Todo modelo puede ser expresado a diferentes niveles de precisión. Esto es, es necesario poder seleccionar el nivel de detalle que se desea ya que en diferentes partes de un proyecto y en diferentes etapas se tendrán unas determinadas necesidades.

c) Los mejores modelos están ligados a la realidad. Lo principal es tener modelos que nos permitan representar la realidad lo más claramente posible, pero no sólo esto, tenemos que saber, exactamente cuando se apartan de la realidad para no caer en la ocultación de ningún detalle importante.

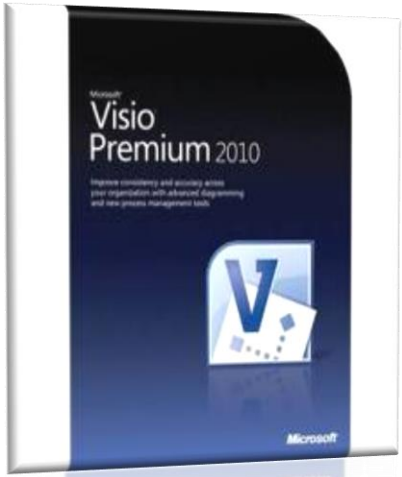
d) Un único modelo no es suficiente. Cualquier sistema que no sea trivial se afronta mejor desde pequeños modelos casi independientes, que los podamos construir y estudiar independientemente y que nos representen las partes más diferenciadas del sistema y sus interrelaciones.

¿Qué es UML?

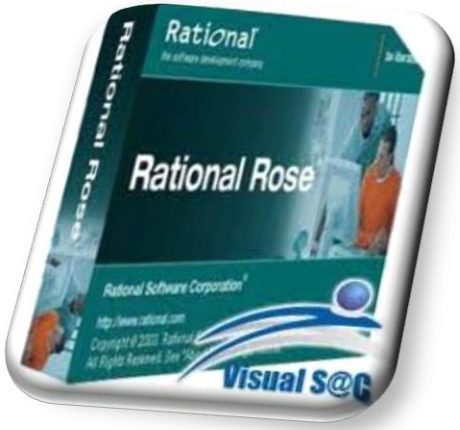
El Lenguaje Unificado de Modelado (UML, *Unified Modeling Language*) es un lenguaje estándar, basado en una notación gráfica, que se utiliza para modelar software.

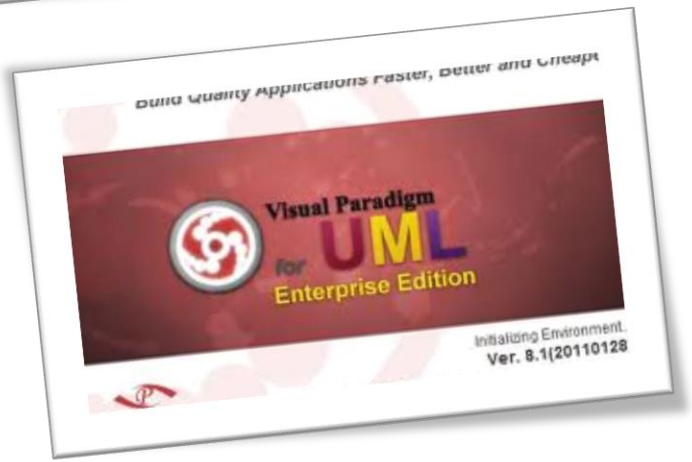
Es un lenguaje de propósito general que pretende ser un estándar mundial y se utiliza para visualizar, especificar, construir y documentar las diferentes “piezas” de un sistema.

HERRRAMIENTAS CASE



UMBRELLO





HERRAMIENTAS CASE

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia
a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software. Como es sabido, los estados en el Ciclo de Vida de desarrollo de un Software son: Investigación Preliminar, Análisis, Diseño, Implementación e Instalación.
CASE se define también como:

Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

La sigla genérica para una serie de programas y una filosofía de desarrollo de software que ayuda a automatizar el ciclo de vida de desarrollo de los sistemas.
Una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

HERRRAMIENTAS CASE

Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunas de ellas son:

- Verificar el uso de todos los elementos en el sistema diseñado.
- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la Base de Datos.
- Generar estructuras de código.

HERRRAMIENTAS CASE

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia herramienta.

Algunos de los diagramas y modelos utilizados con mayor frecuencia son:

– Diagrama de casos de uso
– Diagrama de clase
– Diagrama de secuencia
– Diagrama de colaboración.
– Diagrama de estados
– Diagrama de actividad.
– Diagrama de componentes
– Diagrama de despliegue.
– Diagrama de composición estructural

La Vista de Casos de Uso

Los modelos de objetos son apropiados para representar la estructura de los objetos, sus asociaciones, y como interactúan dinámicamente. Sin embargo no son modelos apropiados para capturar y comunicar requerimientos funcionales de la aplicación, ni para verificar si el modelo de objetos se corresponde con la realidad.
El primer modelo del sistema que se debe construir debe ser comprensible tanto para los usuarios como para los desarrolladores.

Los Casos de Uso fueron introducidos por Jacobson en 1992 [Jacobson92].

Existen algunas diferencias entre los casos de uso y los eventos. Las principales son:

1) Los eventos se centran en describir qué hace el sistema cuando el evento ocurre, mientras que los casos de uso se centran en describir cómo es el diálogo entre el usuario y el sistema.

2) Los eventos son “atómicos”: se recibe una entrada, se la procesa, y se genera una salida, mientras que los casos de uso se prolongan a lo largo del tiempo mientras dure la interacción del usuario con el sistema. De esta forma, un caso de uso puede agrupar a varios eventos.

3) Para los eventos, lo importante es qué datos ingresan al sistema o salen de él cuando ocurre el evento (estos datos se llaman datos esenciales), mientras que para los casos de uso la importancia del detalle sobre la información que se intercambia es secundaria. Según esta técnica, ya habrá tiempo más adelante en el desarrollo del sistema para ocuparse de este tema.

Los casos de uso combinan el concepto de evento del análisis estructurado con otra técnica de especificación de requerimientos bastante poco difundida: aquella que dice que una buena forma de expresar los requerimientos de un sistema es escribir su manual de usuario antes de construirlo. Esta técnica, si bien ganó pocos adeptos, se basa en un concepto muy interesante: al definir requerimientos, es importante describir al sistema desde el punto de vista de aquél que lo va a usar, y no desde el punto de vista del que lo va a construir. De esta forma, es más fácil validar que los requerimientos documentados son los verdaderos requerimientos de los usuarios, ya que éstos comprenderán fácilmente la forma en la que están expresados.

Descripción de los Casos de Uso

Los casos de uso se documentan con texto informal. En general, se usa una lista numerada de los pasos que sigue el actor para interactuar con el sistema. A continuación se muestra una parte simplificada de la descripción del caso de uso "Ingresando Pedido".

Caso de Uso: Ingresando Pedido.
Actor: Empleado de Ventas.
1) El cliente se comunica con la oficina de ventas, e informa su número de cliente
2) El oficial de ventas ingresa el número de cliente en el sistema
3) El sistema obtiene la información básica sobre el cliente
4) El cliente informa el producto que quiere comprar, indicando la cantidad
5) El sistema obtiene la información sobre el producto solicitado, y confirma su disponibilidad.
6) Se repite el paso 4) hasta que el cliente no informa más productos

- *Vista de procesos:* Muestra la concurrencia del sistema, comunicación y sincronización.

  Útil a desarrolladores e integradores.

MODELADO DEL PROCESO DE NEGOCIOS

Business Process Modelling (BPM)
_ BPM también (Business Process Management) [Ko 2009][Ko 2009b]
_ "Representar los procesos de negocio de una empresa u organización con objeto de que puedan ser analizados y mejorados":

- **Validación**: Se realizan todas la tareas, ciclos
- **Simulación**: Ahorro de costes antes de la implementación

Principales responsables:

Analistas, Arquitectos/ diseñadores, desarrolladores del sistema de información

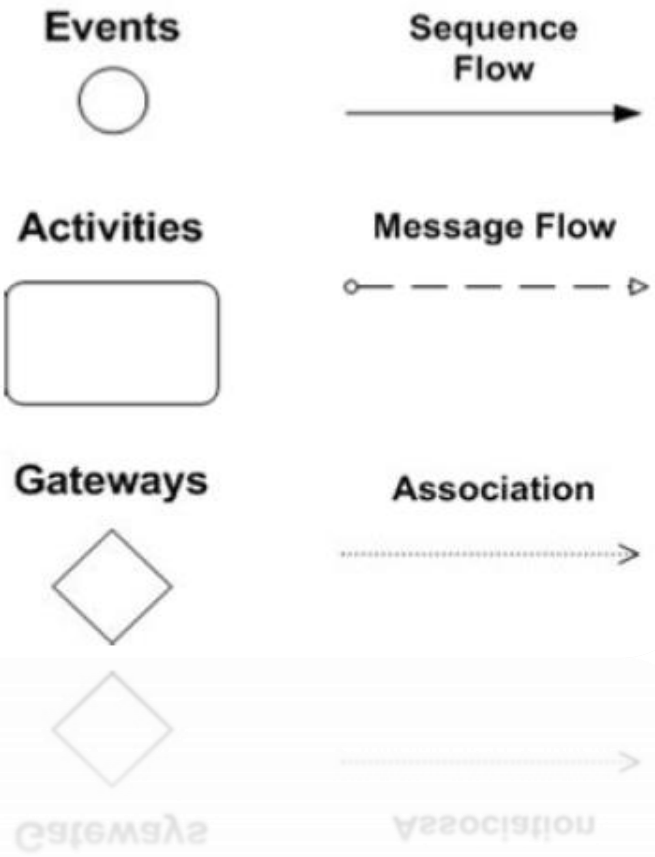
Elementos de un Modelo de
Proceso de Negocio

- Típicamente:

_ **Objetivo**(s) o motivo del proceso
_ **Entradas**
_ **Salidas**
_ **Recursos** utilizados
_ Secuencia de **Actividades**
_ **Eventos** que dirigen el proceso
_ **Roles/participantes** involucrados

BPMN. Elementos Básicos

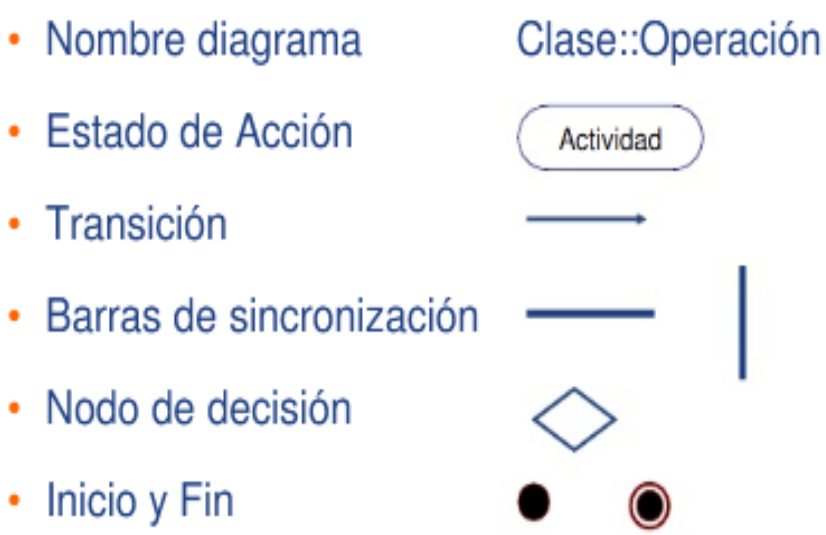
- Elementos de modelado para la creación de diagramas de procesos de negocio sencillos



ELEMENTOS DEL DIAGRAMA DE ACTIVIDADES

Representa el comportamiento interno de una operación o de un caso de uso, bajo la forma de un desarrollo por etapas, agrupadas secuencialmente. El propósito del diagrama de actividad es:

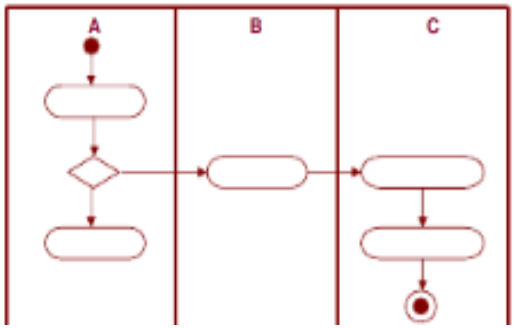
- Modelar el flujo de tareas
- Modelar las operaciones



- Muestra los aspectos dinámicos de un sistema
- Puede describir procesos o casos de uso.
- Permite elegir el orden en que pueden hacérselas cosas.
- Establece las reglas de secuencia a seguir.

Carriles (swimlanes) o Calles

- Franja de división vertical
- Muestra las actividades responsabilidad de un determinado objeto
- Puede representar a un actor o trabajador del negocio que participa en el proceso modelado por un caso de uso.



Nodo inicial (initial state).

- Indica el comienzo del flujo de actividades.
- Representa el inicio del flujo de trabajo del caso de uso del negocio.
- Se representa a través de un círculo de color negro.
- Se coloca dentro del swimlane correspondiente al rol que comienza el caso de uso.
- Es un estado único para el flujo de actividades

Nodo Final (end state)

- Indica el final del flujo de actividades del caso de uso.
- Se representa a través de un círculo de color negro dentro de un círculo transparente.
- Se coloca dentro del swimlane correspondiente al rol que termina el caso de uso.
- Puede haber más de un estado final en dependencia de las diferentes maneras de acabar el caso de uso.

Actividad (activity).

Actividad

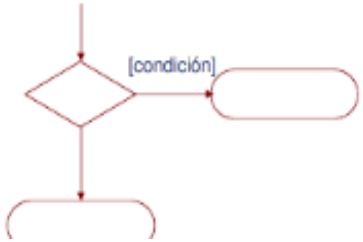
- Representa una tarea, actividad o paso dentro del flujo de trabajo del caso de uso del negocio.
- Se representa a través de un rectángulo ovalado en los extremos.
- El nombre de la actividad debe:
  - Ser simple y breve.
  - Ser un verbo o frase verbal en infinitivo.
  - Incluir el objeto de la actividad.
  - Colocarse dentro del símbolo de la actividad.

Flujo de Control (**Transición**)

- Señala la dirección en que fluyen las actividades.
- Representa la secuencia de cada elemento dentro del diagrama.
- Al completarse la ejecución de una actividad el flujo de control pasa a la siguiente
- Se representa por una línea dirigida

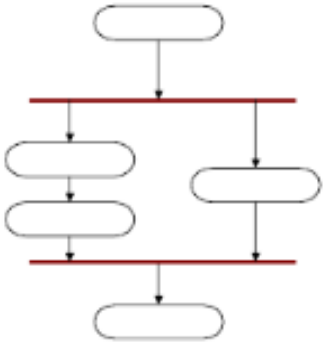
Nodo de Decisiones

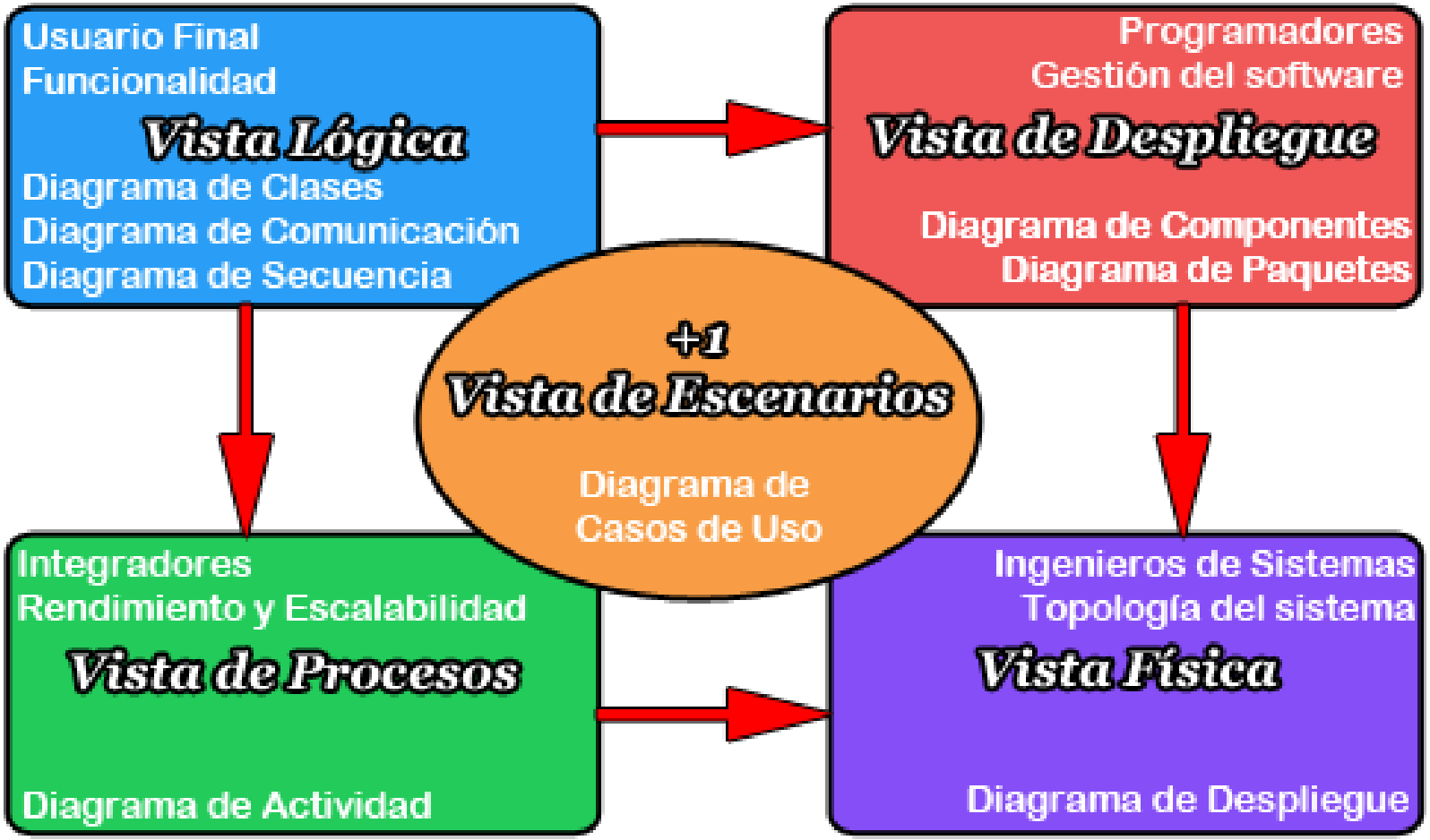
- Representa momentos para tomar caminos alternativos
- Se representa por un rombo
- Debe nombrarse tal y como se hace en el negocio.
- Se acompaña de la pregunta que debe hacerse el proceso para tomar la decisión.



Nodo Fork y Nodo Join

- Actividades a desarrollar simultáneamente
- Se representa por una línea horizontal o vertical gruesa




Usuario Final
Funcionalidad
Vista Lógica
Diagrama de Clases
Diagrama de Comunicación
Diagrama de Secuencia
Programadores
Gestión del software
Vista de Despliegue
Diagrama de Componentes
Diagrama de Paquetes
+1
Vista de Escenarios
Diagrama de
Casos de Uso
Integradores
Rendimiento y Escalabilidad
Vista de Procesos
Diagrama de Actividad
Ingenieros de Sistemas
Topología del sistema
Vista Física
Diagrama de Despliegue

VISTA LÓGICA

Muestra el diseño de la funcionalidad del sistema en sus dos aspectos esenciales: su estructura, es decir, los componentes que lo integran, y su comportamiento, expresado en términos de la dinámica de interacción de dichos componentes. Es utilizada fundamentalmente por los equipos de diseño y desarrollo, y consta de los siguientes diagramas.

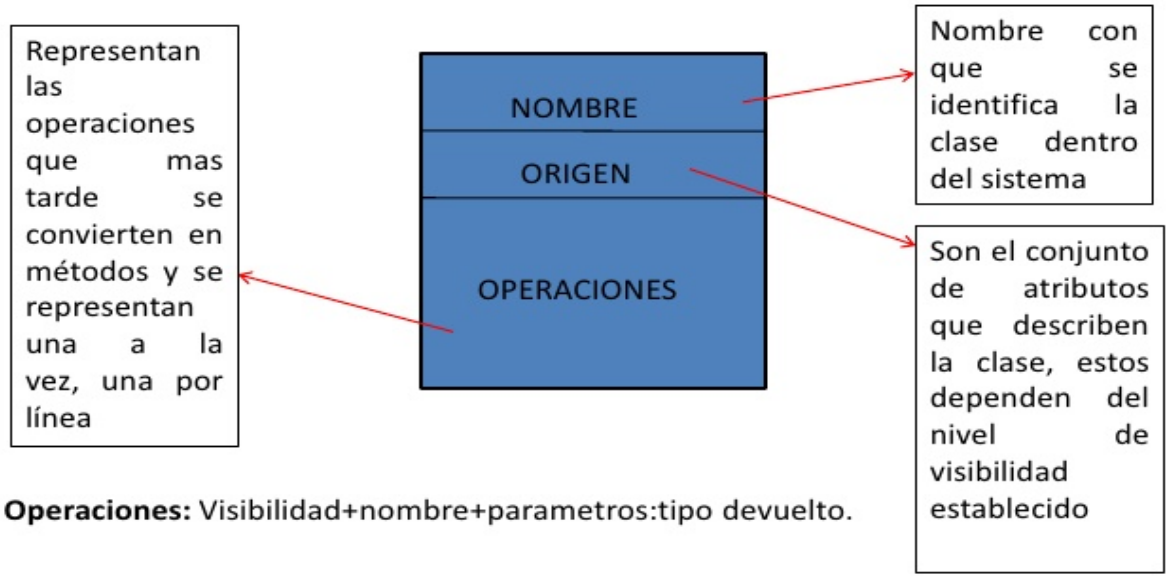
Para la descripción de estructura:

- Diagramas de Clases y de Objetos
- Para la descripción del comportamiento:
- Diagramas de Estado, Secuencia, Colaboración y Actividad.

Elementos del Diagrama de Clases

Los diagramas de clases se utilizan para modelar la visión estática de un sistema. Esta visión soporta los requisitos funcionales del sistema, en concreto, los servicios que el sistema debería proporcionar a sus usuarios finales. Normalmente contienen: clases, interfaces y relaciones entre ellas: de asociación, de dependencia y/o de generalización

Las clases tienen una representación para identificarlas dentro del enfoque O.O, y las debemos crear teniendo en cuenta los siguientes aspectos:

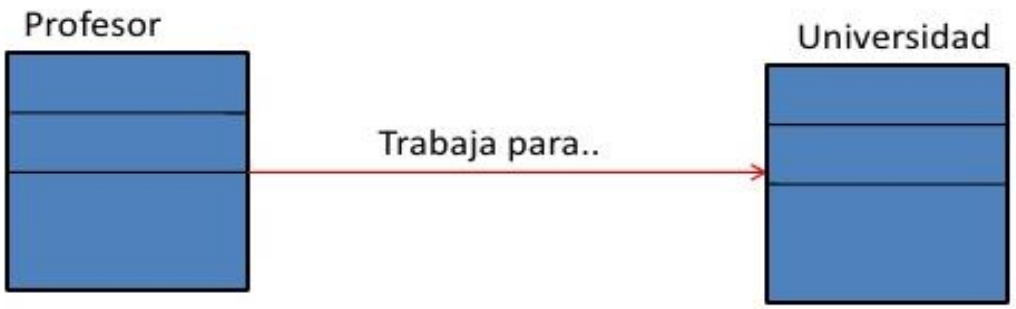


Operaciones: Visibilidad+nombre+parametros:tipo devuelto.

Hay tres tipos:
– Asociaciones: relaciones estructurales.
– Generalizaciones: herencias.
– Dependencias: relaciones de uso, especifican que un cambio en la especificación de un elemento puede afectar a otro que lo utiliza.

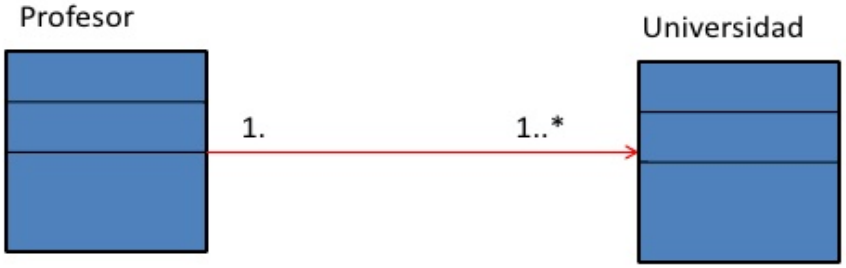
1.2.1 Asociaciones.

1.2.1.1 Nombre: Las asociaciones pueden tener un nombre que las identifiquen



Elementos del Diagrama de Clases

La multiplicidad: Indica el número de objetos involucrados en una asociación.

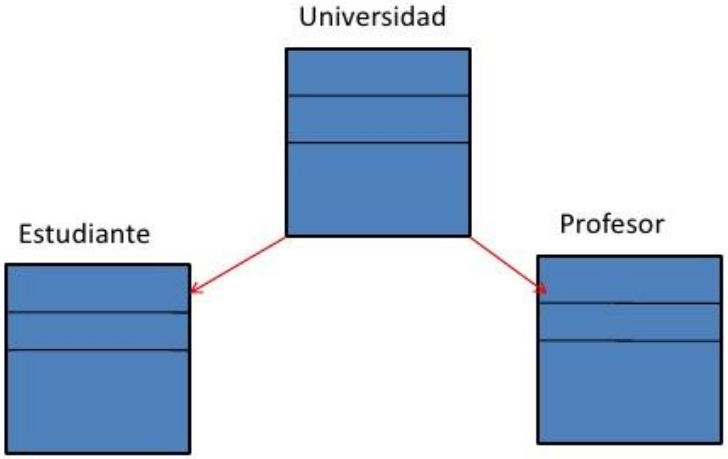


Esta misma relación se puede mirar como una relación n….n.

- **Tipos de multiplicidad habituales:**
  - * = Cero, uno ó n.
  - 0,1 = Cero o uno.
  - 1..* = Uno o más.
  - 1 = Exactamente uno (también podría ser otro número).
  - 1..5 = Entre uno y cinco.

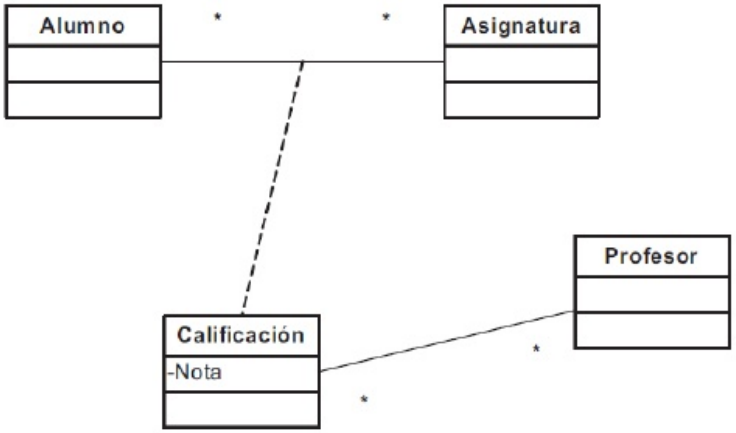
1.2 Relaciones de generalización

- Indica **una relación de "tipo/subtipo".**
- Permite a una clase (hijo o subclase) heredar los atributos de otra (padre o superclase), siendo posible añadir atributos y operaciones especificas.

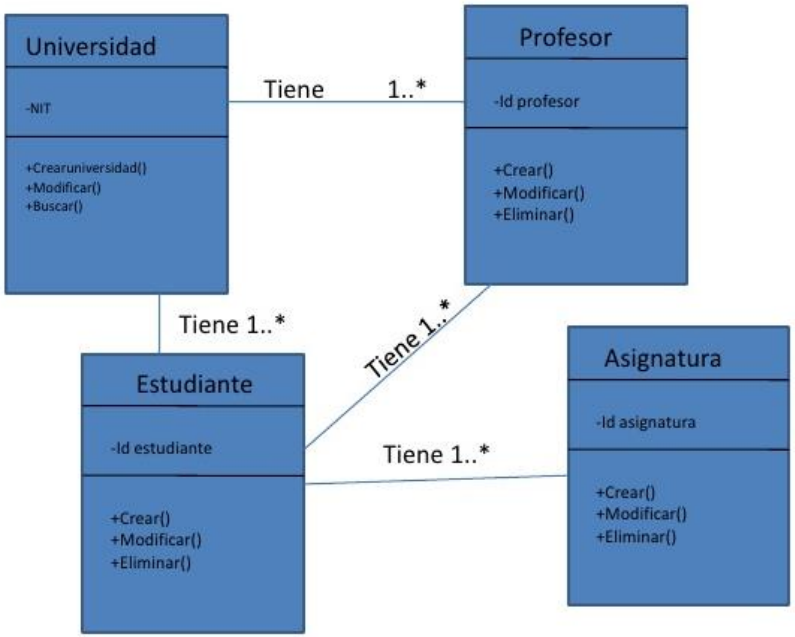


Clases de asociación

Las clases de asociación permiten añadir atributos y operaciones a las relaciones estructurales.



Ejemplo 1: Relación estudiante-Universidad



Tarjetas CRC

CRC (Class-Responsibility-Collaboration) es una técnica de diseño orientado a objetos propuesta por Kent Beck (introductor de la metodología de programación extrema) y Ward Cunningham (también muy conocido entre otras muchas materias, por sus aportaciones a dicha metodología).

El objetivo de la misma es hacer, mediante tarjetas, un inventario de las clases que vamos a necesitar para implementar el sistema y la forma en que van a interactuar, de esta forma se pretende facilitar el análisis y discusión.

Tarjetas CRC

Un esquema típico de tarjeta CRC puede ser aquel en el que se indiquen los siguientes datos:

- Nombre de la clase.
- Nombre de las superclases y subclases (si procede).
- Las responsabilidades de la clase.
- Las clases con las que va a colaborar para poder realizar las responsabilidades indicadas.
- Autor, fecha, etc…

Nombre de la Clase : Pedido	
Responsabilidad	Colaboración
Revisa si hay elementos en existencia	Línea de pedido
Determina precio	Línea de pedido
Revisa si el pago es válido	Cliente
Despacha a la dirección de entrega	

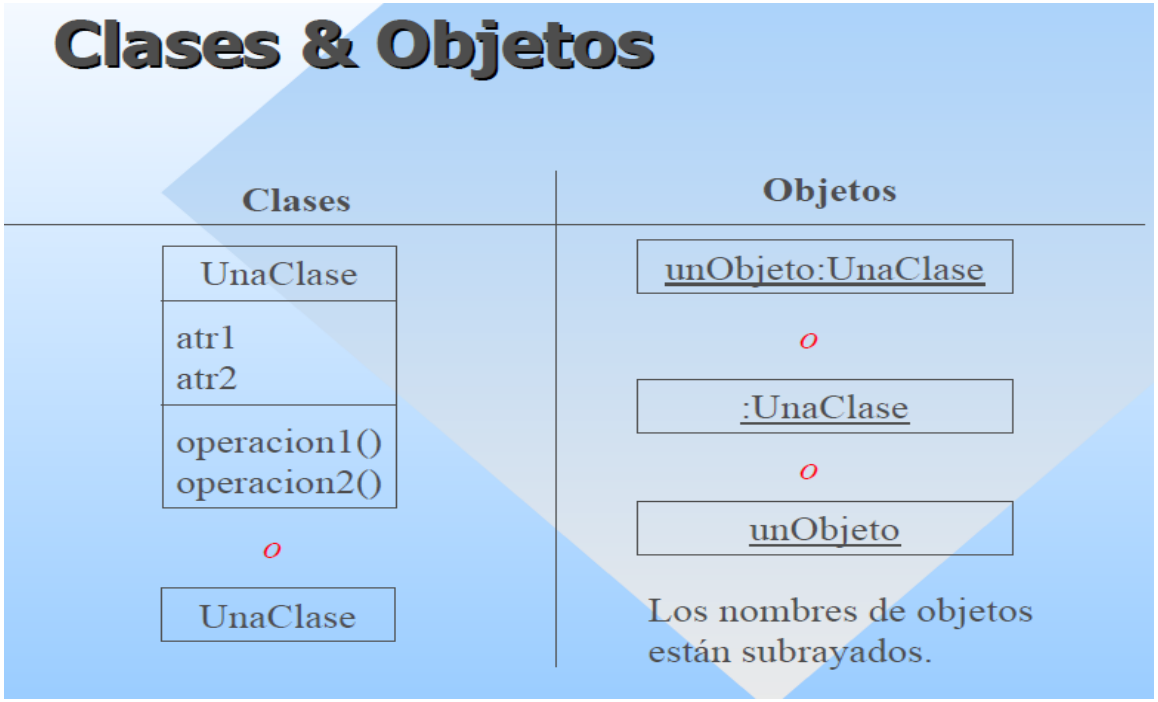
MODELADO DE CLASES, RESPONSABILIDADES Y COLABORACIONES

Una vez que se han desarrollado los escenarios de uso básicos para el sistema, es el momento de identificar las clases candidatas e indicar sus responsabilidades y colaboraciones. El modelado de clases-responsabilidades colaboraciones (CRC) aporta un medio sencillo de identificar y organizar las clases que resulten relevantes al sistema o requisitos del producto.

Se describe el modelado CRC de la siguiente manera:
Un modelo CRC es realmente una colección de tarjetas índice estándar que representan clases. Las tarjetas están divididas en tres secciones.

Elementos de Diagrama de Objetos

Un diagrama de Objeto se puede considerar un caso especial de un diagrama de clase. Los diagramas de objetos usan un sub conjunto de elementos de un diagrama de clase para enfatizar la relación entre las instancias de las clases en algún punto en el tiempo. Estos son útiles para entender los diagramas de clases. Estos no muestran nada diferente en su arquitectura a los diagramas de secuencia, pero reflejan multiplicidad y roles.



Elementos de Diagrama de Secuencia

El diagrama de secuencia de uml muestran la forma en que los objetos se comunican entre si al transcurrir el tiempo.

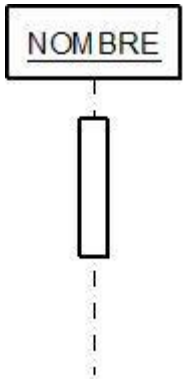
Los diagramas de secuencia, formalmente diagramas de traza de eventos o de interacción de objetos, se utilizan con frecuencia para validar los casos de uso.

El diagrama muestra:

- ✓Los objetos participando de la interacción
- ✓La secuencia de mensajes intercambiados
- ✓Un diagrama de secuencia contiene:
- ✓Objetos con su línea de vida
- ✓Mensajes intercambiados entre objetos de una secuencia ordenada
- ✓Línea de vida activa

OBJETOS

Los diagramas de secuencia constan de objetos que se representan de modo usual: rectángulo con nombre, mensajes entre los objetos representados por líneas continúas con una punta de flecha y el tiempo representado como una progresión vertical. Los objetos se colocan cerca de la parte superior del diagrama de izquierda a derecha y se acomodan de manera que simplifiquen el diagrama.



LINEA DE VIDA

Una línea de vida representa un participante individual en un diagrama de secuencia. Una línea de vida usualmente tiene un rectángulo que contiene el nombre del objeto.

MENSAJE

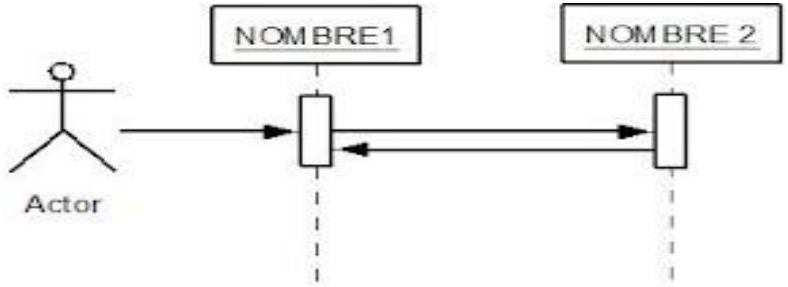
Un mensaje que va de un objeto a otro pasa de la línea de vida de un objeto al de otro. Un objeto puede enviarse un objeto a si mismo es decir de su línea de vida así propia línea de vida.

Un mensaje puede ser simple, síncrono y asíncrono

•Mensaje simple: es la transferencia del control de un objeto a otro.
•Mensaje síncrono: es cuando el objeto espera la respuesta a ese mensaje antes de continuar con su trabajo.
•Mensaje asíncrono: es cuando el objeto no espera la respuesta a ese mensaje antes de continuar.

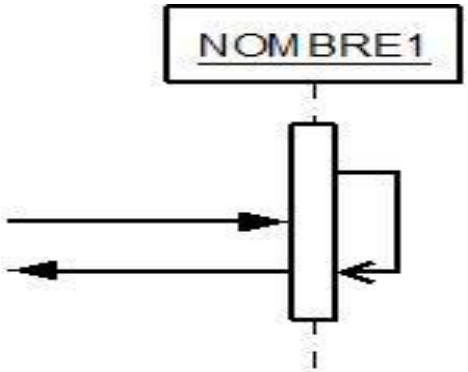
TIEMPO

El diagrama representa el tiempo en dirección vertical. El tiempo se inicia en la parte superior y avanza hacia la parte inferior. Un mensaje que esté más cerca de la parte superior ocurrirá antes que uno que esté cerca de la parte inferior.



RECURSIVIDAD

En ocasiones un objeto posee una operación que se invoca a si misma. A esto se le conoce como recursividad y es una característica fundamental de varios lenguajes de programación.

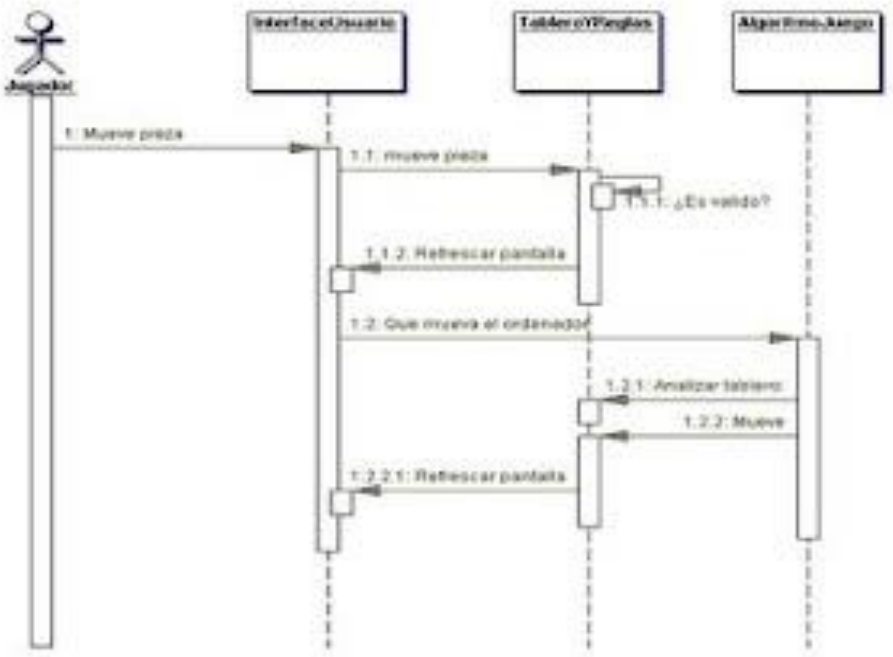


Elementos de Diagrama de Secuencia

OBJETIVO

Descubrir las interfaces requeridas para cada objeto y validar que cada interface se usa realmente.

El diagrama de Secuencias modela interacciones entre objetos. Ya que estas interacciones pueden ser muy complejas, se modelan un pequeño juego de interacciones como un solo escenario.

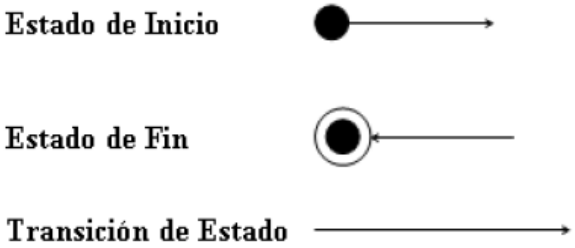


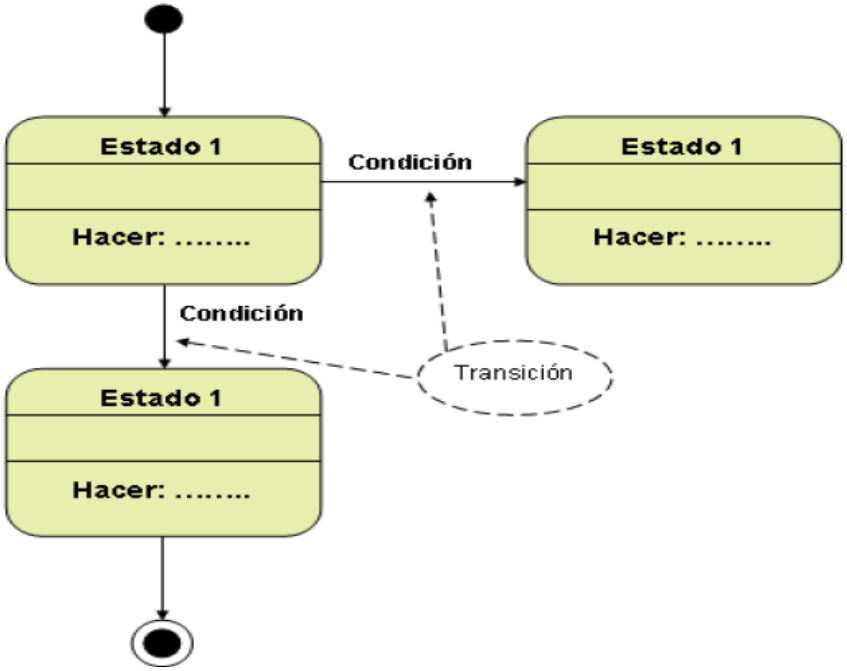
Elementos de Diagrama de Estados

Los diagramas de estado muestran el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación en respuesta a eventos (por ejemplo, mensajes recibidos, tiempo rebasado o errores), junto con sus respuestas y acciones.

También ilustran qué eventos pueden cambiar el estado de los objetos de la clase. Normalmente contienen: estados y transiciones. Como los estados y las transiciones incluyen, a su vez, eventos, acciones y actividades, vamos a ver primero sus definiciones.

Elementos de Diagrama de Estados





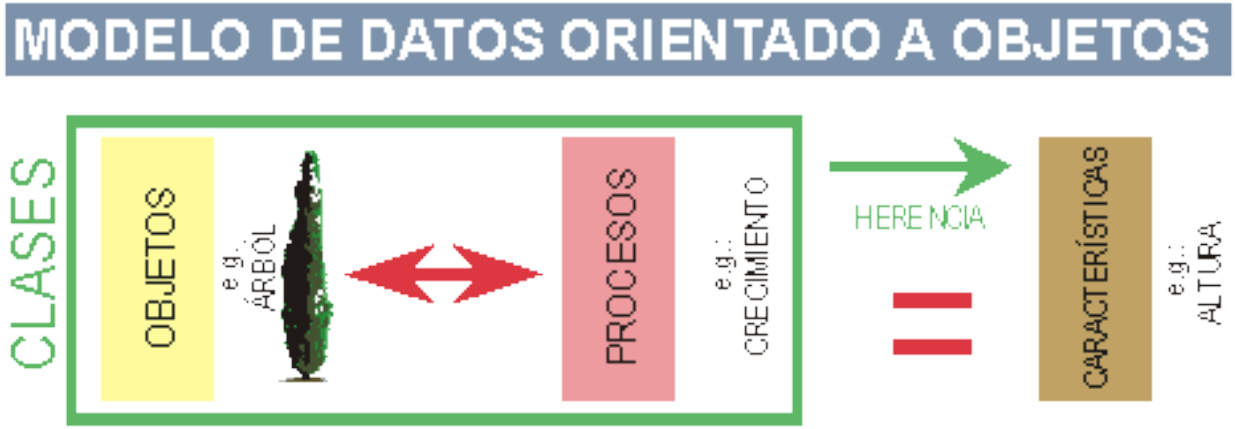
Diseño de la Interfaz gráfica del usuario

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

Diseño del modelado de datos

Está basado en una colección de objetos. Un objeto contiene valores almacenados en variables ejemplares dentro de este objeto. Contiene fragmentos de código que operan dentro del mismo y a éstos se les llama métodos. La única manera en que pueden acceder a la base de datos es a través del paso de mensajes a otro objeto.

Los objetos que contienen los mismos tipos de valores y los mismos métodos se agrupan en clases. Los objetos acceden a los datos de otros objetos mediante el envío de mensajes.



JERARQUERIZADA

Almacena su información en una forma ordenada (del grande al chico o inverso).

DE RED

Su diferencia fundamental es al modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico).

RELACIONAL

En este modelo, el lugar y la forma en que se almacenan los datos no tienen relevancia

MULTIDIMENSIONALES

Son bases para desarrollar aplicaciones muy concretas, como creación de cubos OLAP.

ORIENTADAS A OBJETOS

Este modelo es bastante reciente, y propio de los modelos informativos orientados a objetos, trata de almacenar en la base los objetos completos.

DOCUMENTALES

Permite la indexación a textos completos, y líneas generales realizar búsquedas mas potentes.

DEDUCTIVAS

Permite hacer deducciones a través de inferencias se basa en reglas y hechos que son almacenados.

Mapeo Objeto-Relación

El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional.

Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM

Lenguajes de programación orientada a objetos

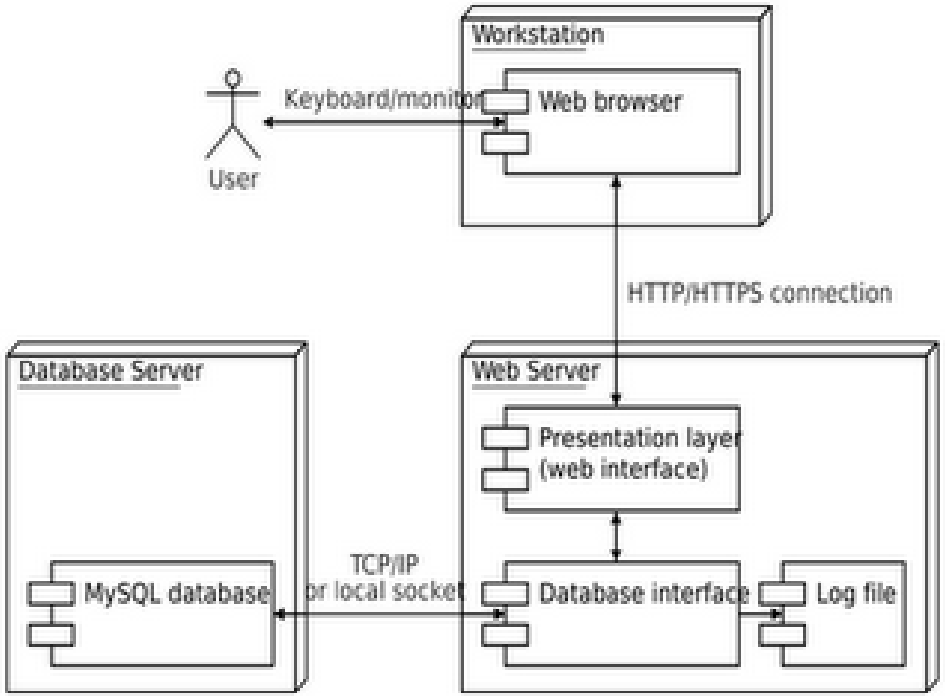
Los lenguajes de programación orientados a objetos tratan a los programas como conjuntos de objetos que se ayudan entre ellos para realizar acciones. Entendiendo como objeto a las entidades que contienen datos. Permitiendo que los programas sean más fáciles de escribir, mantener y reutilizar.

Entre los principales lenguajes de este tipo tenemos:

Ada, C++, C#, VB.NET, Clarion, Delphi, Eiffel, Java, Lexico (en castellano), Objective-C, Ocaml, Oz, PHP, PowerBuilder, Python, Ruby y Smalltalk.
No todos estos lenguajes son específicamente orientados a objetos. Sino que algunos de ellos se le han añadido extensiones orientadas a objetos.

Vista Física

Diagrama de Componentes a nivel de Hardware
Este diagrama modela los componentes de hardware que se utilizan en un sistema, en este diagrama cada cuadro representa un servidor o un cliente, también se deben modelar los usuarios y a través de que computadoras hacen uso del sistema.



Vista Física

El componente es la unidad de construcción elemental del diseño físico. Las **características de un componente** son:

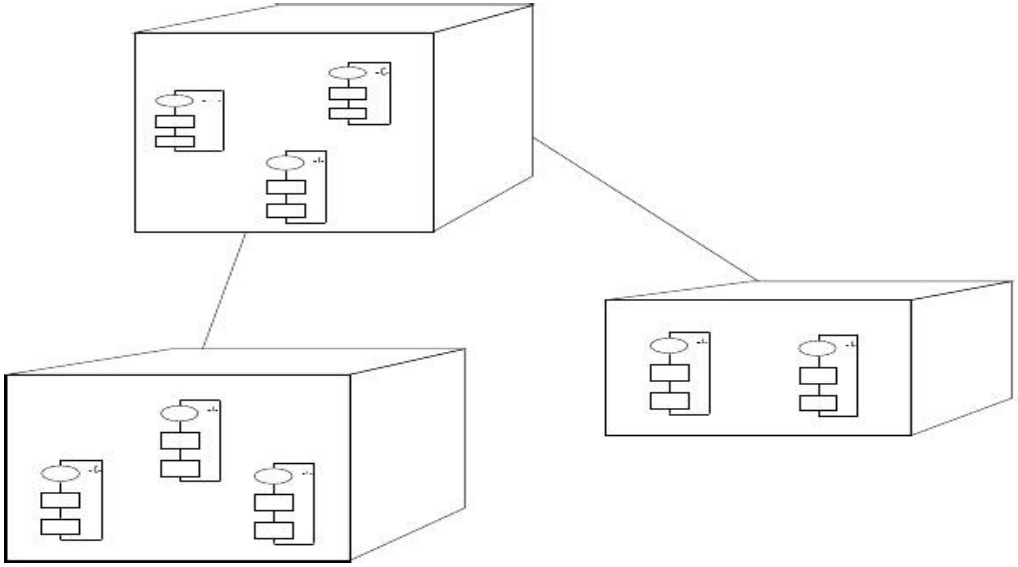
- Se define según cómo interactúa con otros
- Encapsula sus funciones y sus datos
- Es reusable a través de las aplicaciones
- Puede verse como una caja negra
- Puede contener otros componentes

El diseño físico comprende las siguientes tareas:

- ✓Definir los componentes
- ✓Refinar el empaquetamiento y distribución de componentes
- ✓Especificar las interfaces de los componentes
- ✓Distribuir los componentes en la red
- ✓Distribuir los repositorios físicos de datos
- ✓Examinar la tolerancia a fallas y la recuperación de errores
- ✓Validar el diseño físico

Diagrama de Distribución

Los Diagramas de Distribución muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene algo de memoria y, a menudo, capacidad de procesamiento. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Representa típicamente un procesador o un dispositivo sobre el que se pueden desplegar los componentes.



Vista de desarrollo

La vista de desarrollo o despliegue se enfoca en la organización de los módulos software en el entorno de desarrollo. El software es empaquetado en pequeños trozos (librerías de programa, subsistemas, componentes, etc.), los subsistemas se organizan en capas jerárquicas, y cada capa proporciona una interfaz bien definida a sus capas superiores

Vista de desarrollo

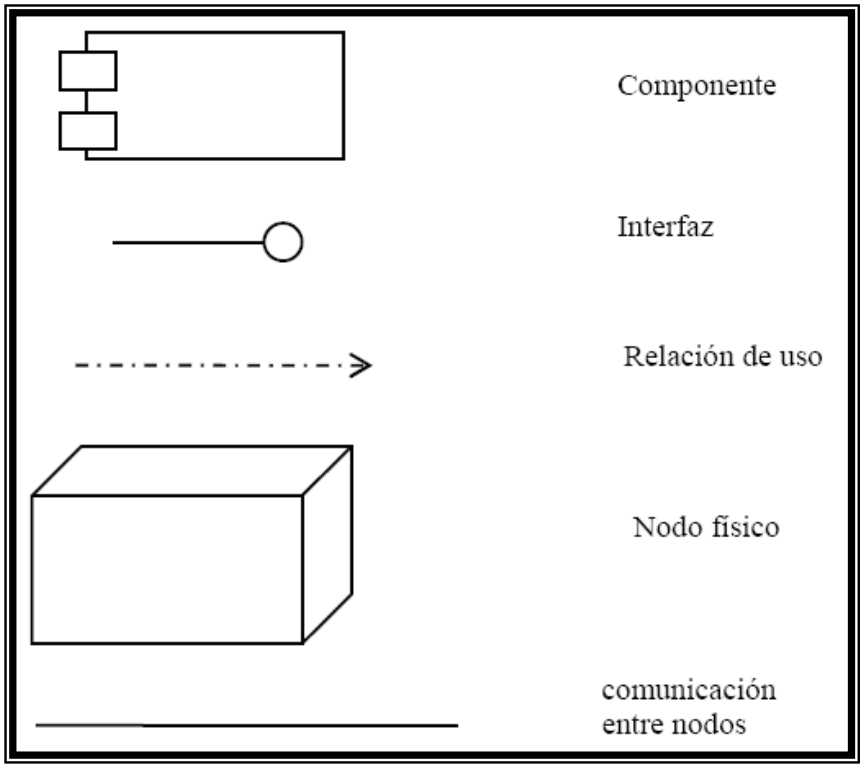
Y destacar que podrá describirse la vista de desarrollo por completo solamente después de haber identificado todos los elementos software.

Notación: La notación más usada es UML, y dentro de esta diagramas de componentes y paquetes.

Estilo: se recomienda definir de cuatro a seis capas de subsistemas en la vista de desarrollo. Una regla de diseño es que un subsistema puede solamente depender de subsistemas en la misma capa o en las menores. Esto minimiza las dependencias entre módulos a favor de una más simple estrategia capa – capa.

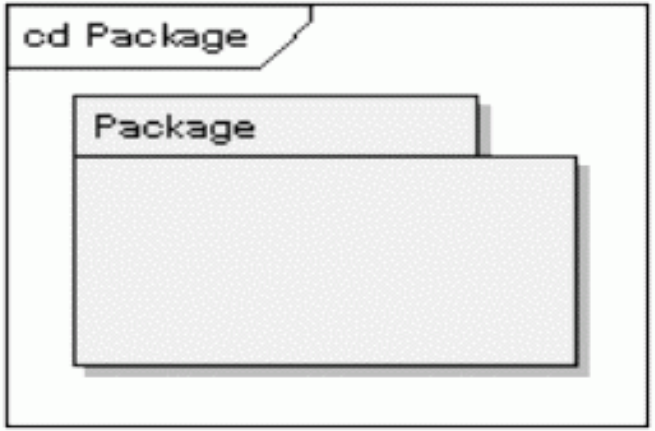
Diagrama de componentes

Los Diagramas de Componentes ilustran las piezas del software, controladores embebidos, etc. que conformarán un sistema. Un diagrama de Componentes tiene un nivel más alto de abstracción que un diagrama de clase – usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución. Estos son bloques de construcción, como eventualmente un componente puede comprender una gran porción de un sistema.



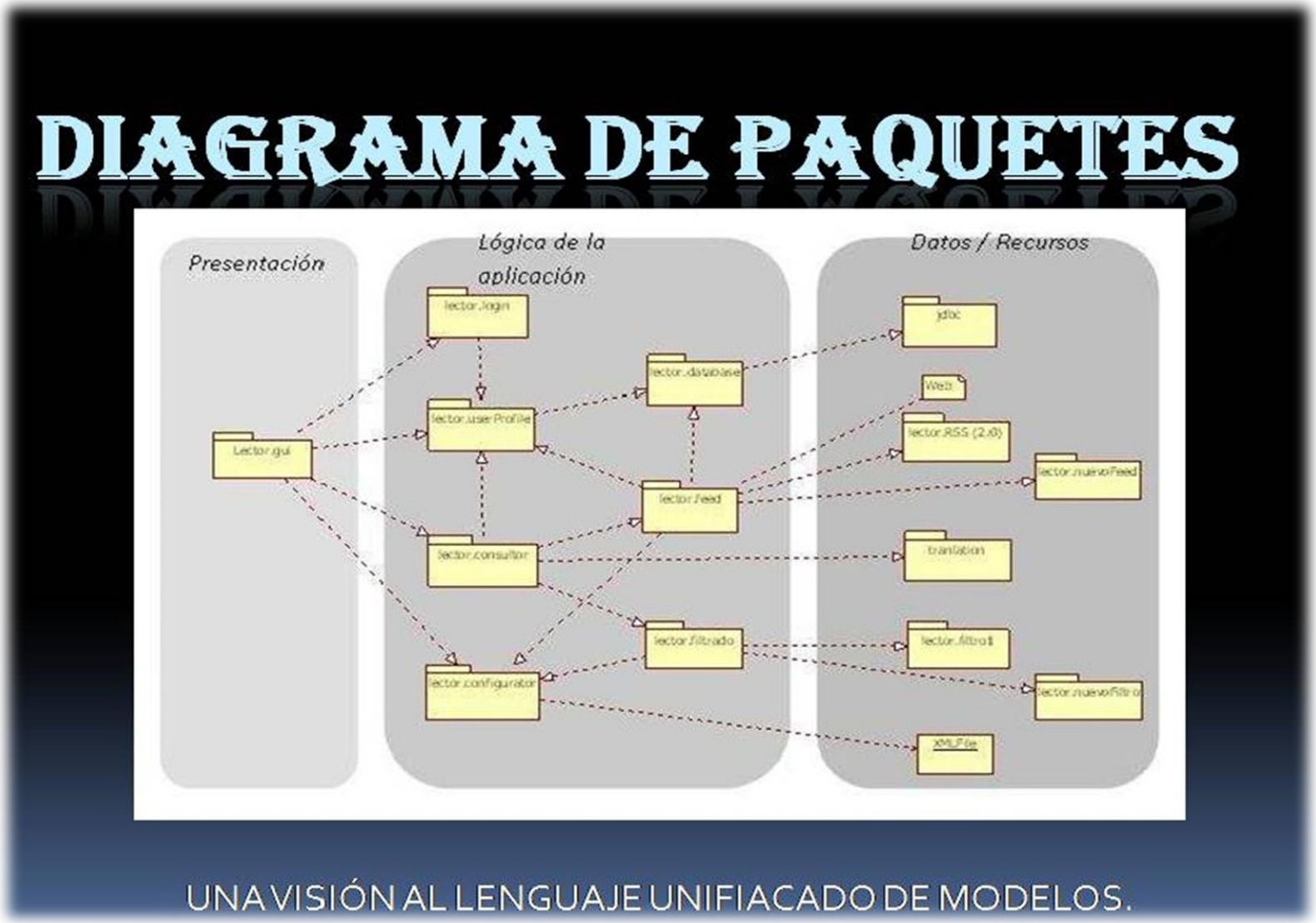
Diagramas de paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Cuando se usan para representaciones, los diagramas de paquete de los elementos de clase se usan para proveer una visualización de los espacios de nombres. Los usos más comunes para los diagramas de paquete son para organizar diagramas de casos de uso y diagramas de clase, a pesar de que el uso de los diagramas de paquete no es limitado a estos elementos UML.



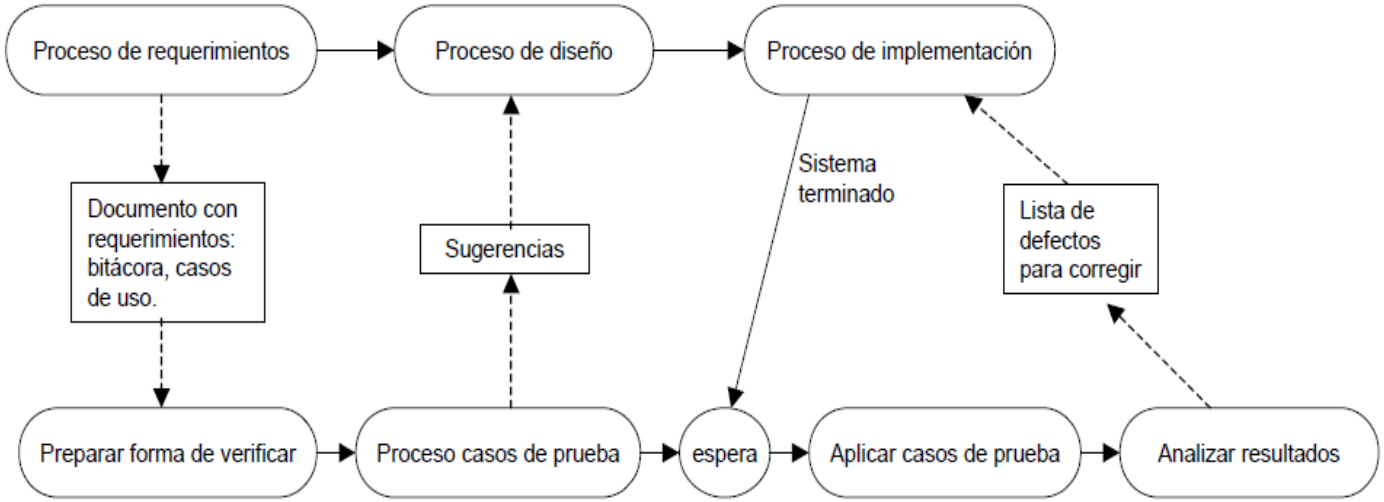
Diagramas de paquetes

EJEMPLO



Pruebas del Sistema

Cualquier pieza de software completo, desarrollado o adquirido, puede verse como un sistema que debe probarse, ya sea para decidir acerca de su aceptación, para analizar defectos globales o para estudiar aspectos específicos de su comportamiento, tales como seguridad o rendimiento. A éste tipo de pruebas donde se estudia el producto completo se les llama ***Pruebas de Sistema***.



Patrones de diseño

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo.(Alexander(arquitecto/urbanista))

Un patrón de diseño es:

- una solución estándar para un problema común de programación
- una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios
- un proyecto o estructura de implementación que logra una finalidad determinada
- un lenguaje de programación de alto nivel
- una manera más práctica de describir ciertos aspectos de la organización de un programa

conexiones entre componentes de programas la forma de un diagrama de objeto o de un modelo de objeto.

Elementos de un patrón:

- **Nombre**: describe el problema de diseño.
- **El problema**: describe cuándo aplicar el patrón.
- **La solución**: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.

Clasificación de los patrones

- Según su propósito:

– **De creación**: conciernen al proceso de creación de objetos.
– **De estructura**: tratan la composición de clases y/o objetos.
– **De comportamiento**: caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos.

Clasificación de los patrones

GoF (gang of Four) [Gamma]

Propósito / Ámbito	Creación	Estructural	Comportamiento
Clase	✓ **Factory Method**	✓ **Adapter**	Interpreter Template Method
Objeto	Abstract Factory Builder Prototype ✓ **Singleton**	✓ **Adapter** Bridge ✓ **Composite** Decorator ✓ **Facade** Flyweight Proxy	Chain of Responsability Command ✓ **Iterator** Mediator Memento ✓ **Observer** State ✓ **Strategy** Visitor

Lenguaje de Restricción de objetos (OCL)

OCL (*Object Constraint Language*, lenguaje de restricciones de objetos) es un nuevo lenguaje notacional un subconjunto del UML estándar industrial, que permite a los desarrolladores de software escribir restricciones sobre modelos de objetos (pre y postcondiciones, guardas, invariantes, valores derivados, restricciones sobre operaciones, etc.).

Estas restricciones son particularmente útiles, en la medida en que permiten a los desarrolladores crear un amplio conjunto de reglas que rigen el aspecto de un objeto individual.

Lenguaje de expresiones

OCL es un lenguaje de expresiones puro. Una expresión OCL garantiza que quedará sin efecto. Esto no puede cambiar nada en el modelo. Esto significa que un estado del sistema nunca cambiará debido a una expresión OCL, incluso una expresión OCL podría usarse para describir tal cambio de estado (p.e. en una postcondición).

Todos los valores de todos los objectos, incluidos los enlaces, no cambiarán. En cualquier momento en que se evalúa una expresión OCL, simplemente devuelve un valor.

Lenguaje de modelos

OCL es un lenguaje de modelos y no un lenguaje de programación. No se puede escribir un programa lógico o un flujo de control en OCL. Especialmente, no se puede invocar procesos o activar operaciones no de consulta en OCL. Debido a que OCL es en primer lugar un lenguaje de modelos, no se puede asegurar que todo sea directamente ejecutable.

Lenguaje formal

OCL es un lenguaje formal donde todos los constructores tienen un significado formal definido. La especificación de OCL es parte de la especificación de UML. OCL no tiene la intención de reemplazar los lenguajes formales existentes.