



Indizen Labs

iMade

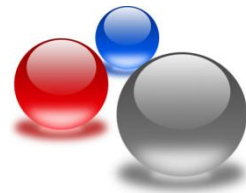
Marco de Desarrollo Aplicaciones de Indizen

Índice de contenidos

- Indizen Labs
- Introducción a iMade
- Metodología iMade
- Arquitectura iMade
- Herramientas iMade

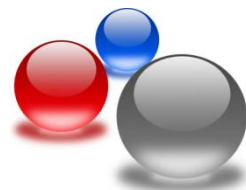
Indizen Labs

- Son **estructuras organizativas** dotadas de recursos humanos y materiales que tienen como **objetivo la investigación y el desarrollo de las ideas innovadoras** de nuestros ingenieros e investigadores, dentro de las líneas de trabajo que consideramos de interés para la compañía.
- Los laboratorios tienen como **misión probar las tecnologías y desarrollar los conceptos, las ideas y los proyectos** que puedan servir de base o como germen para el desarrollo de futuros productos y servicios propios de Indizen o en colaboración con otros organismos, empresas y/o centros de investigación.



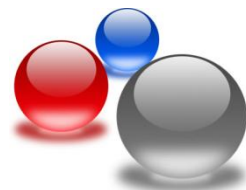
Estructura de los Laboratorios

- **Laboratorios de Tecnologías**, donde se prueban y validan nuevas tecnologías, arquitectura, herramientas, frameworks, etc., que puedan ser de utilidad para la compañía.
- **Laboratorio de Ideas**, donde se crean prototipos de nuevos productos y servicios.
- **Laboratorio de proyectos**, donde se ponen en marcha proyectos propios o de los clientes en siguiendo esta modalidad



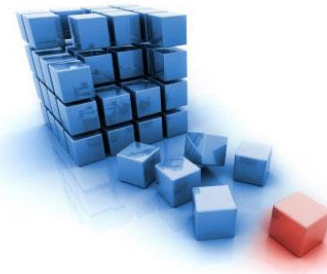
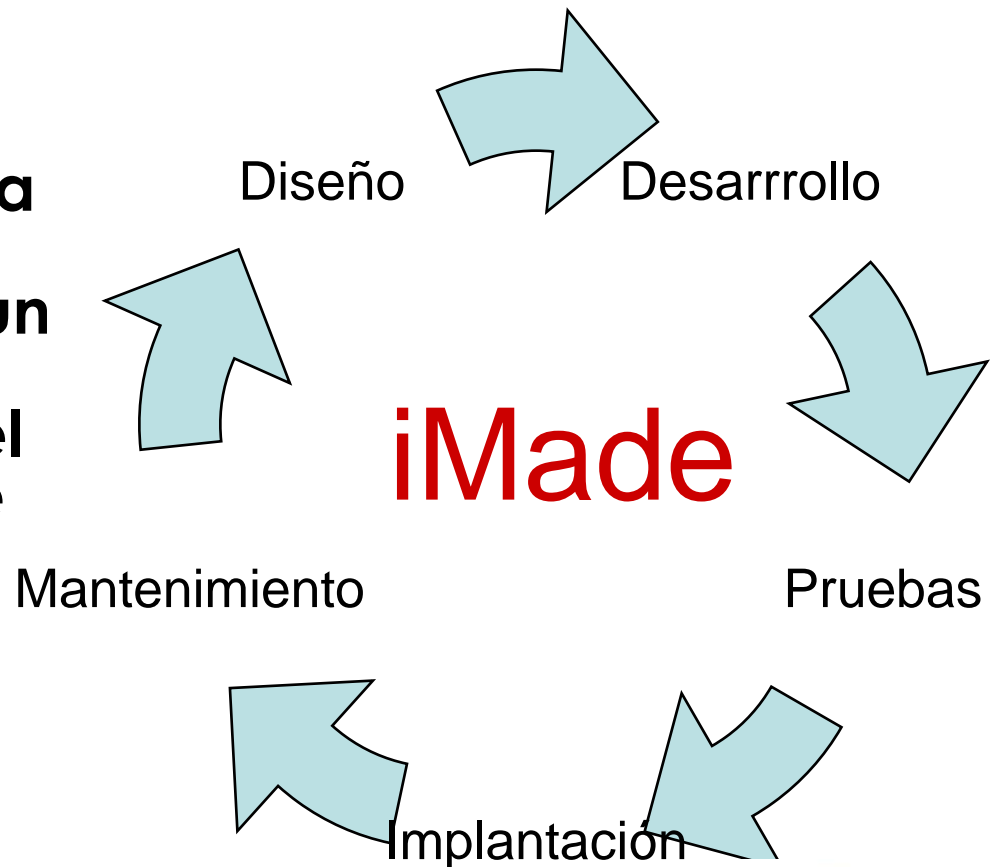
Líneas de trabajo de los laboratorios

- Desarrollo de nuevas herramientas de Gestión de Riesgos.
- Desarrollo e investigación de metodologías de integración de Simuladores.
- Integración de estudios Deterministas (Simulación) y Probabilistas (PSA) en Análisis de Seguridad y Simulación de Sistemas Complejos.
- Nuevas Herramientas de computación de Procedimientos y Fiabilidad Humana (HRA).
- Herramientas para la clasificación y normalización estadística en las áreas de biomedicina.
- Tecnologías de Web Semántica, computación lingüística, etc.
- Herramientas inteligentes de búsqueda
- Tecnologías para la distribución de procesos, Grid Computing, etc.
- Generación automática de código fuente para aplicaciones Web tipo CRUD.
- Algorítmica matemática.



Introducción iMade

Plasmar la experiencia de Indizen en proyectos reales, en un marco de trabajo (iMade) que mejore el ciclo de desarrollo de aplicaciones.



Objetivos iMade

- **Buscar la simplicidad** huyendo de modelos complejos, y buscando soluciones adaptadas a los distintos proyectos.
- **Fomentar el uso de estándares abiertos** que nos den una cierta independencia respecto a tecnologías propietarias o productos específicos de fabricantes.
- **Mejorar la productividad** dotando a los desarrolladores de la mayor cantidad de herramientas para poder realizar su trabajo de forma eficiente.
- **Intentar reducir los costes** de desarrollo y explotación en lo posible.
- **Basado en la experiencia** no se trata de seguir "recomendaciones de fabricante", sino que está contrastado con aplicaciones reales en clientes.
- **Facilitar las tareas de soporte y mantenimiento**, no solo es un conjunto de herramientas, sino que se incluyen procedimientos de soporte necesarias para apoyar a los equipos de desarrollo y explotación.
- **Uso de metodologías ágiles de desarrollo**, tipo Extreme Programming que impulsen y agilicen el desarrollo de aplicaciones.
- **Fomentar la calidad del software.**



Metodología de trabajo

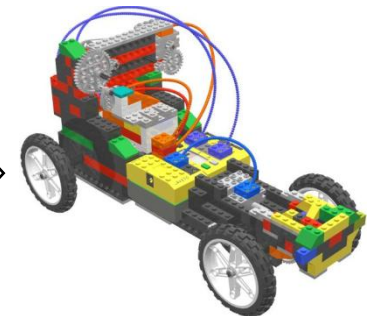
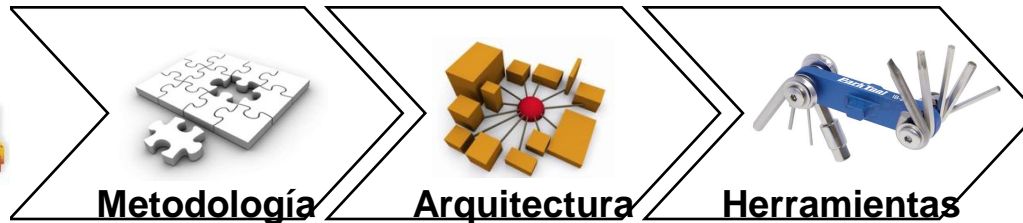
- Procedimientos de diseño, desarrollo, pruebas, implantación y mantenimiento.
- Guía de estilos de programación
- Gestión de software
- Documentación

Herramientas predefinidas

- Herramientas de desarrollo
- Frameworks
- Librerías
- ...



Ideas
Conceptos
Proyectos



Soluciones
Herramientas
Productos

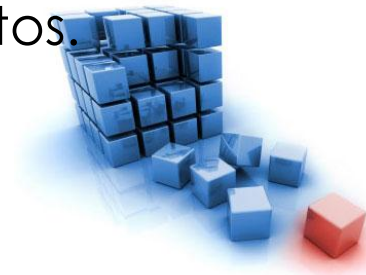
Arquitectura de aplicación

- Modelo de componentes y servicios
- Implementación base (esqueleto de aplicación)
- ...



¿Características principales de iMade?

- Permite **construir aplicaciones de diferente complejidad funcional y tecnológica**, de una forma sencilla y rápida.
- **Base de los laboratorios de Indizen** y sirve tanto para desarrollos con Java o .Net.
- Recoge las **mejores prácticas** de desarrollo de aplicaciones.
- **Arquitectura Orientada a Servicios probada** en distintas aplicaciones (InnoVaR, WebScan, AutoCoder, ...)
- Emplea **herramientas software líderes** (Eclipse, MyEclipse, Subversion, ...)
- **Integra librerías y frameworks probados** que se han convertido en estándares (Struts, JSF, iBatis, Junit, Log4J, Ant, ...).
- Implementa **patrones de diseño** (Factory, Proxy, DAO, etc.).
- Dispone de un **generador de código** para genera de forma automática partes de la aplicación.
- Es una **esqueleto de aplicación** con los componentes necesarios para la puesta en marcha de los proyectos.

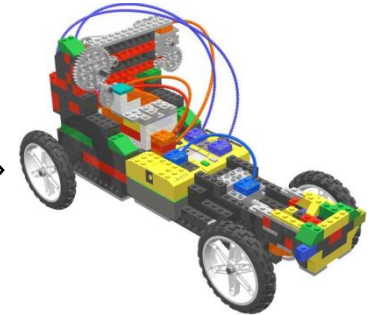
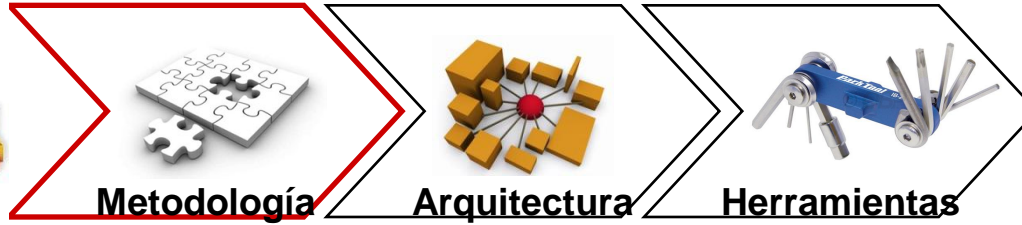


Lo deseable para una arquitectura de aplicación

- Clara **separación de capas** (presentación, negocio, acceso a datos).
- Arquitectura **orientación a servicios**.
- Facilidad para ejecutar **pruebas unitaria**.
- Generación de **código automático** (dentro de lo posible).
- **Automatización del procesos** de compilación, construcción, pruebas y despliegue.
- **Gestión de errores y excepciones**.
- **Soporte de logs avanzados** para depuración y gestión de incidencias.
- Seguridad de acceso (autenticación y autorización)
- **Facilidades de integración, instalación, configuración y mantenimiento**.



Metodología iMade



Metodología iMade



Extreme Programming

- La programación extrema trata de adaptar los procesos de desarrollo los continuos cambios en los requisitos de los proyectos, poniendo el énfasis en la adaptación.
- Los cambios de requisitos son algo natural, inevitable e incluso deseable del desarrollo de proyectos.
- La adaptación a los cambios es una formula más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.
- Es una metodología orientada a gestionar los cambios de forma dinámica durante el ciclo de vida del software.

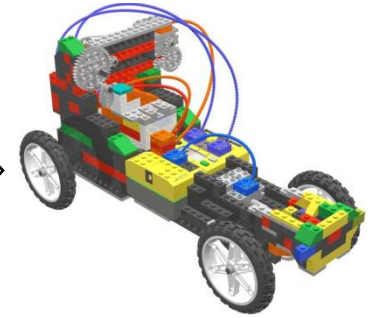
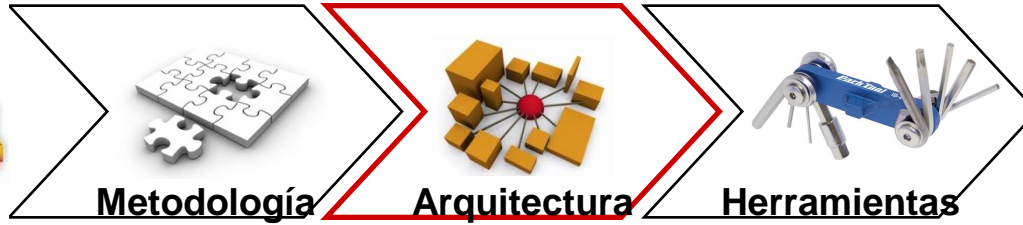
Son características de esta metodología

- Desarrollo iterativo e incremental
- Pruebas unitarias continuas
- Interacción con el cliente
- Propiedad común del código (del equipo)
- Refactorización del código
- Simplicidad
- ...



- Documentos base (plantillas)
 - Visión y alcance de proyecto
 - Especificación de requisitos
 - Reglas de negocio
 - Casos de uso
 - ...
- Guía de desarrollo con iMade
- Guía de estilos de programación
- ...

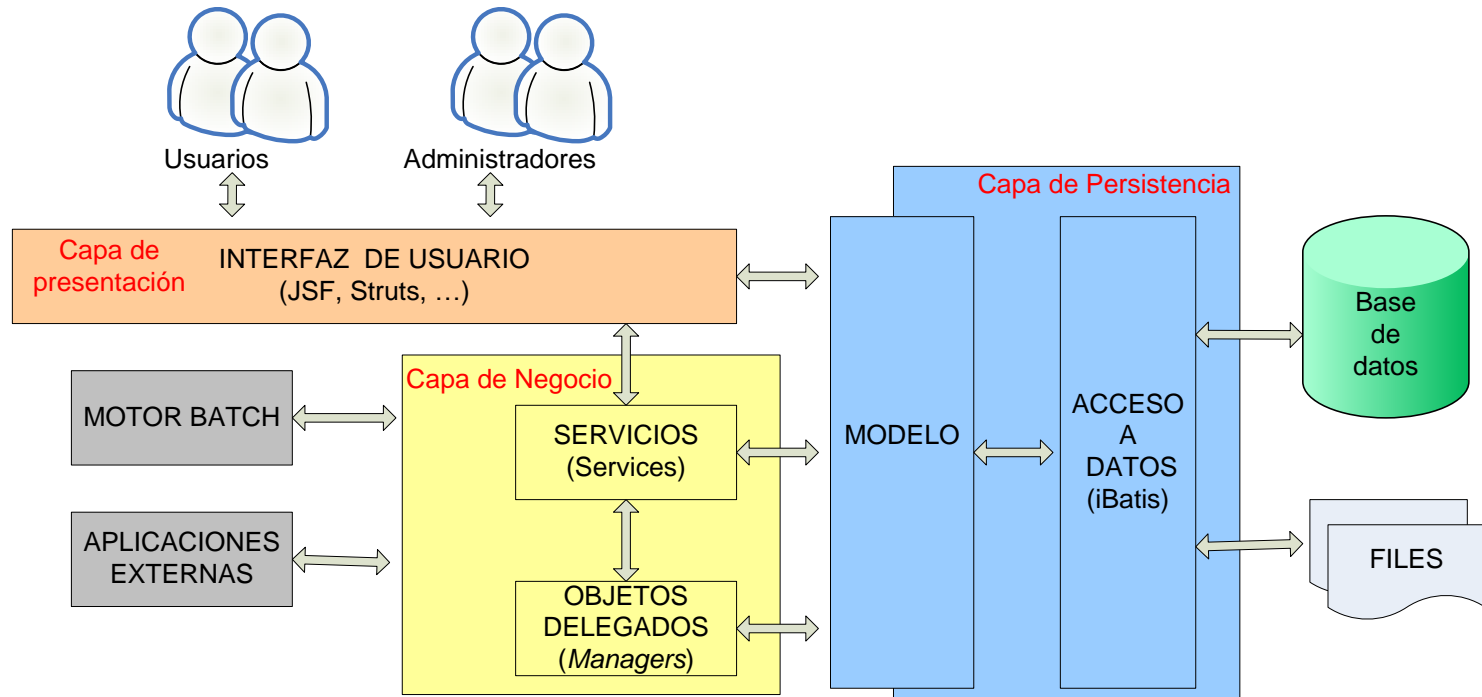




Arquitectura de iMade



- Componentes de la arquitectura iMade

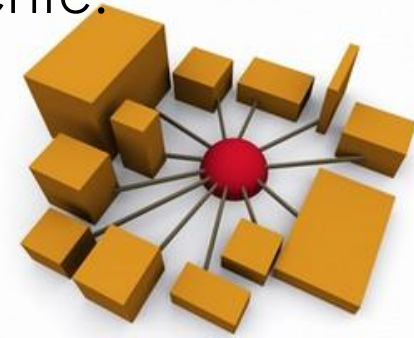


- **Capa de presentación-control**
 - Atiende a las peticiones de interface de usuario.
 - Control de sesiones
- **Capa de negocio-servicios.**
 - Orientada a servicios
 - Cada servicio hace de proxy entre la aplicación y el interface de usuario o con otras aplicaciones.
 - Se puede desarrollar con Web Services (SOAP, WSDL) o de forma adhoc.
- **Capa de persistencia**
 - Control de transacciones.
 - Su función es la de realizar operaciones CRUD
- **Capa transversal de modelo**
 - Implementada en objetos JavaBeans
 - Su función es la de transferencia de datos (DTO).



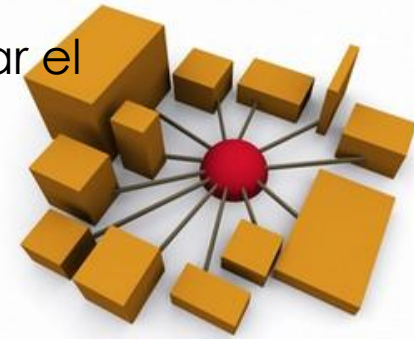
Presentación-control

- **Intercepta las solicitudes del cliente.** Examina el request, lee sus cabeceras HTTP, su *locale*, valida las entradas, controla seguridad,..., y actúa en consecuencia.
- **Invoca el proceso acciones y procesos de negocio correspondientes.** A partir de la solicitud esta capa debe ser capaz de discriminar cual es el proceso que subyace.
- **Gestiona las variables de estado**, para el mantenimiento de sesiones.
- **Recolecta los resultados**, tanto los obtenidos de la acción realizada como los guardados en los diversos niveles de memoria (sesión o aplicación), y se los dispone al cliente.
- Selecciona la vista y, finalmente, se la envía al cliente.
- Implementación con JSF, STRUTS, Ajax, ...



Capa de Negocio-Servicios

- **Contiene la lógica de la aplicación:** reglas de negocio, workflows, operaciones no persistentes...
- **Aislada y encapsulada en Servicios** y objetos de apoyo Managers que tienen la siguiente funcionalidad y características:
- **Separa la aplicación del interfaz de usuario**, que sirva de “contrato” entre las capas.
- La comunicación entre capas se establece mediante objetos de Modelo.
- Establece un **modelo de excepciones** que se propaga a la capa de presentación.
- Para la capa de negocio se contemplan dos implementaciones:
 - Servicios Ad-hoc. Se trata de objetos de servicio diseñados para contener la lógica de negocio de forma sencilla.
 - Spring. Se trata de un framework que permite realizar el desarrollo de esta capa de forma “estandarizada”.



Persistencia y Acceso a Datos

- Encargada de la funcionalidad transaccional y de acceso a la base de datos.
- Las principales **características y funcionalidades** de esta capa son:
 - Se utiliza el **patrón DAO** para abstraer el acceso a datos. Esto consigue:
 - Ser **independientes de la implementación**: jdbc, Ibatis, Hibernate.
 - Ser **independientes de los base de datos**: Oracle, DB2, Host...
 - **Agrupar la funcionalidad de persistencia** en clases y paquetes siguiendo criterios de diseño. Son Clases sencillas: POJOS.
 - Simplificar el código de los servicios.
 - El patrón DAO se apoya en un interfaz que sirve de contrato entre capas, que exige:
 - Que los datos de entrada y salida sean objetos del Modelo.
 - Que las excepciones sean genéricas, independientes de la implementación.
- Para acceso a base de datos relacionales (RDBMS), iMade sugiere dos alternativas: iBatis, Hibernate.



Modelo

- Las clases del *Modelo* son una **representación Java de las entidades de negocio** y tienen una cierta correspondencia el esquema de base de datos.
- **Describen la información contenida en la base de datos a nivel conceptual.** Se implementan como objetos Java tradicionales (POJOs), JavaBeans puros. Pueden existir diferentes alternativas para su definición:
 - Modelos sencillos, muy vinculados al modelo de datos de la aplicación, se podría decir que son contenedores de registros de base de datos.
 - Modelos ricos, distintos al modelo de datos, empleando patrones avanzados de OO (composición, jerarquías, patrones...).
- La funcionalidad y propiedades de esta capa se pueden resumir en las siguientes:
 - Participan en la construcción de las Interfaces de Usuario, Servicio y DAO.
 - Transferencia a través de la red, en configuraciones con separación física de capas.
 - Ocupar el menor espacio en memoria posible, para agilizar el tráfico.
 - Deben implementar la interfaz Serializable.



- **Gestión de errores y excepciones**

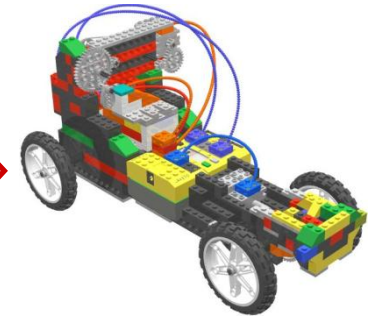
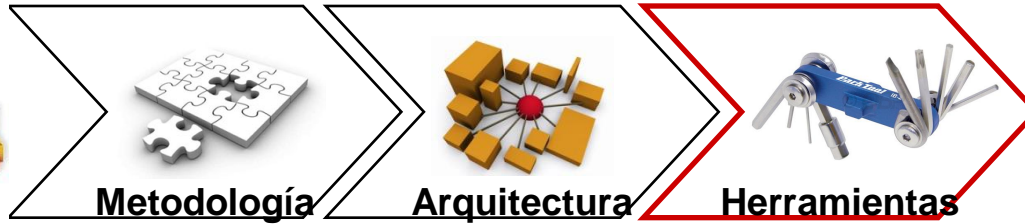
- Cada capa (Servicios, Persistencia) define su propio conjunto de excepciones (de tipo `BaseException`), que son las únicas que deben transmitirse a la capa inmediatamente superior.
- Para cada capa se define una jerarquía de excepciones, vinculada a las clases que constituyen dicha capa.
- Estas excepciones “capturan” las excepciones derivadas de la implementación específica.
- En la capa de presentación, Struts define una serie de mecanismos para el tratamiento de errores (`ActionMessage`, `GlobalException`, Errores de Validación...)



Resumen de características arquitectura iMade

- Clara separación en capas con roles definidos para cada una de ellas.
- Existe un exhaustivo control de errores que se transmite de una capa a otra.
- Es una arquitectura base que permite mejorar todo el ciclo de vida
- Cada capa integra frameworks (Struts, iBatis,...) que cumplen con la funcionalidad encomendada.





Herramientas de iMade



Herramientas de iMade

- Los objetivos perseguidos en la selección herramientas usadas en iMade son:
 - Evitar reinventar la rueda
 - Integrar herramientas probadas y contrastadas
 - Mejorar los procesos que van desde el diseño, desarrollo, pruebas e implantación hasta el soporte y mantenimiento
 - Aumentar la productividad (generación automática de código).
 - Mejorar la calidad del software



Librerías

- **Ant.** Es una herramienta para la realización de tareas mecánicas y repetitivas durante la fase de compilación y construcción y despliegue.
- **JUnit.** Es un conjunto de librerías creadas por que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.
- **Log4J.** Librería open source desarrollada en Java por la ASF que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o “logs” en tiempo de ejecución, mediante el uso de archivos de configuración externos.
- **JAAS.** que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso.



Frameworks

- **Struts.** es una herramienta de soporte para el desarrollo de aplicaciones web que implementa el modelo MVC en plataformas.
- **JSF.** Estándar MVC de J2EE.
- **Ajax.** script.aculo.us, prototype, dojo, etc.
- **iBatis**
 - Framework de código abierto basado en capas desarrollado por ASF, que se ocupa de la capa de Persistencia (se sitúa entre la lógica de Negocio y la capa de la Base de Datos). Java y .Net.
 - Asocia objetos de modelo (POJO) con sentencias SQL o procedimientos almacenados mediante ficheros descriptores XML, simplificando la utilización de bases de datos.
- **Hibernate:**
 - Framework ORM. Persistencia de Objetos Java. Un modelo más orientado a POO, más avanzado, permite portabilidad de base de datos, y mayor transparencia ante cambios del modelo de objetos. Modelo ideal para nuevos proyectos, que deban ser portables entre BBDD



Herramientas de productividad

- MyEclipse. Herramienta para desarrollo web, despliegue y depuración.
- Subversion. Software de control de versiones
- Generación de código automático de modelo y acceso a datos (DAO) tanto en iBatis como en Hibernate.

