Copiar & Pegar

Siempre que a la hora de crear una nueva clase o método copiemos y peguemos código para ello, debemos tener en cuenta que, sin excepción, estamos haciendo algo mal. Duplicar el código en vez de reutilizarlo nunca es una práctica adecuada, la necesidad de duplicar código es síntoma de que nuestro código necesita ser refactorizado, por ejemplo con una refactorización de "Extraer método" o pensar un modo de hacer y modificar el método para que sea accesible desde los sitios donde necesitas invocarlo.

ACTIVIDADES 1.2

- ¿Qué patrones podrían usarse para construir una aplicación que se encargase de administrar y crear los extractos de un banco que tiene varias sucursales en una misma ciudad?
- Relacione los patrones y antipatrones vistos especificando qué patrones podrían sustituir al uso de algún antipatrón.

1.6.2 DESARROLLO EN TRES CAPAS

El desarrollo en capas nace de la necesidad de separar la lógica de la aplicación del diseño, separando a su vez los datos de la presentación al usuario.

Para solventar esa necesidad, se ideó el desarrollo en 3 capas, que separa la lógica de negocio, el acceso a datos y la presentación al usuario en tres capas que pueden tener cada una tantos niveles como se quiera o necesite.

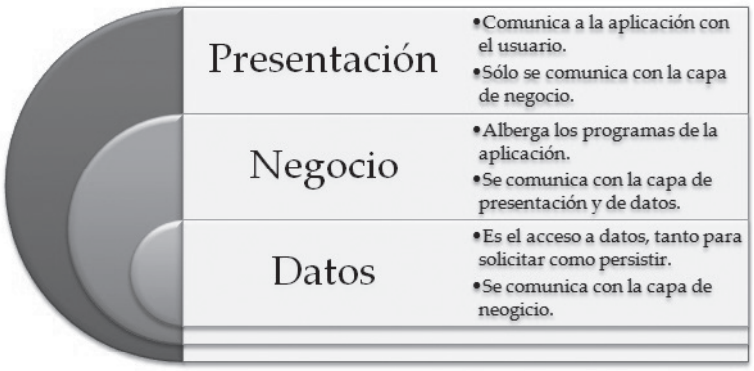

Figura 1.4. *Desarrollo en 3 capas*

El desarrollo por capas no solo nos mejora y facilita la estructura de nuestro propio software, sino que nos aporta la posibilidad de interoperar con otros sistemas ajenos a nuestra aplicación. Por ejemplo, podríamos necesitar acceder y utilizar datos contenidos tanto en nuestra propia base de datos, como en la base de datos de un banco, para ello utilizaríamos la capa de datos, en donde accederíamos a nuestro gestor de base de datos y al servicio que nos ofrezca

el banco para solicitar dichos datos. Esto podría hacerse sin necesitar un desarrollo en tres capas, pero la principal ventaja (aparte de la encapsulación y ocultación de código y datos entre las capas) que nos aporta reside en evitar modificar la lógica de negocio por necesitar acceder a diferentes datos, todo está perfectamente estructurado y nos permite modificar las fuentes y el modo en que accedemos a los datos de los programas que trabajan con ellos.

Modelo vista controlador

Dentro del desarrollo por capas, encontramos diferentes modelos de software, uno de ellos es el MVC (Modelo Vista Controlador).

El MVC define tres componentes para las capas del desarrollo del software, organiza el código mediante unas directrices específicas utilizando un criterio basado en la funcionalidad y no en las características del componente en sí mismo.

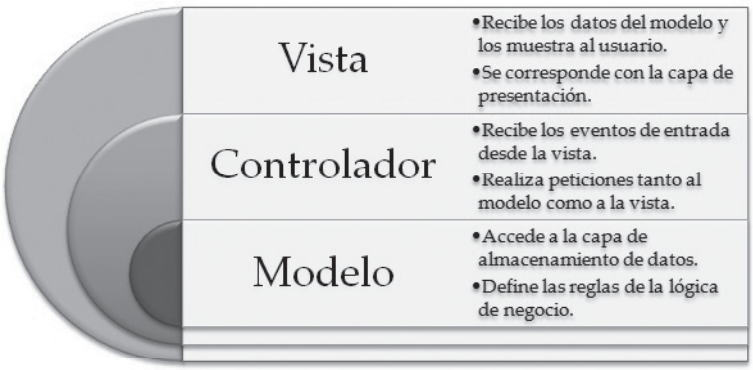

Figura 1.5. *Modelo Vista Controlador*

A la hora de mostrar los datos de modelo en las vistas, se suele establecer una serie de "bindings" que enlacen diferentes componentes de la vista a propiedades y campos de las entidades de los datos a los que tiene acceso el modelo.

Modelo vista vistamodelo

EL MVVM parte de un concepto muy similar al modelo MVC, tanto, que no resulta extraño pensar que es una ampliación o modificación al MVC. De hecho, muchas *frameworks* actuales ofrecen el uso del MVVM a través del MVC *framework* añadiéndole un ViewModel. Siendo objetivos, no es una visión muy apartada de la realidad, ya que en esencia parten de la misma necesidad y concepto.

A diferencia del MVC, la vista del MVVM es un observador que se actualiza cuando cambia la información contenida en el VistaModelo. El componente VistaModelo en MVVM contiene a su vez un controlador al igual que en el MVC, y además utiliza un VistaModelo que actúa como un modelo virtual y personalizado que contiene la información necesaria para mostrarla en la vista. Es decir, al igual que en el MVC, los eventos de la vista son recogidos por el controlador, pero a diferencia del MVC los datos de la vista son obtenidos y actualizados a través del VistaModelo, ocultando así al modelo de la vista, dejando al modelo como una mera representación de las entidades para la persistencia de los datos.

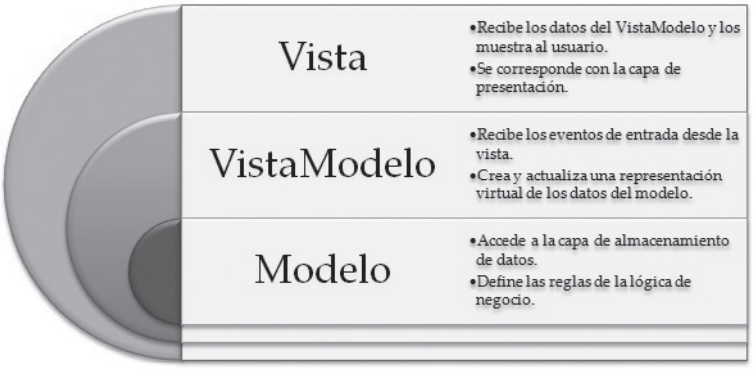

Figura 1.6. *Modelo Vista VistaModelo*

Mediante el uso del VistaModelo, hemos logrado ocultar el modelo a la vista, además de evitar la necesidad de crear "bindings" manuales entre la vista y el modelo, ya que el propio VistaModelo se puede "bindear" directamente.

ACTIVIDADES 1.3

- Relacione los dos modelos de desarrollo en 3 capas con los diferentes patrones, especificando qué patrones podrían incluirse en las diferentes capas de los modelos.