Diseño de Aplicaciones

En este apartado vamos a tratar lo concerniente al **Diseño de Aplicaciones Informáticas**. Es un tema extremadamente complejo que se trata de forma sencilla en la Formación Profesional, pero en profundidad en la Ingeniería Informática. Normalmente, el diseño de una aplicación es tarea de los **Analistas** y **Arquitectos**, no de los **Programadores**; pero un programador tiene que conocer los rudimentos del diseño e interpretar la Documentación del Proyecto creada por los Analistas.

Vamos a trabajar un poco el Diseño de Aplicaciones. Como veréis, este documento hará referencia a contenidos de diferentes asignaturas, como Entornos de Desarrollo [EDE], Programación [Prog] o Base de Datos [BBDD]. No repetiremos temario. Si tienes alguna duda, recurre a tus apuntes de la materia correspondiente.

Fases del ciclo de vida del software [EDE]

El **Ciclo de Vida del Software** es el conjunto de fases, procesos y actividades requeridas para ofertar, desarrollar, probar, integrar, explotar y mantener un producto software. Abarca toda la vida del sistema, comenzando con su concepción y finalizando cuando ya no se utiliza o se retira. En general, lo más importante que tenemos que entender del Ciclo de Vida del Software es que está dividido en **fases**, y que en cada fase se realiza una serie de tareas concretas.

Tradicionalmente, estas fases iban una detrás de la otra, de forma que al concluir una se pasaba a la siguiente. En el momento en el que se encontraba un problema en una fase, se debía detener el trabajo y valorar si era necesario retroceder a una fase anterior. No podías proseguir con las tareas hasta que el problema no se arreglase, por lo que, retroceder más de una fase implicaba retrasos importantes en el proyecto.

Las fases son:

- **Análisis**: En esta fase se definen los **requisitos** del proyecto que hay que desarrollar. Entendemos por requisito algo que tiene o no tiene que hacer el software.

- **Diseño**: En esta fase se define la arquitectura del sistema. En esta fase trabajan los Analistas y generan la documentación del proyecto. En dicha documentación, además de texto explicativo, se usan diagramas UML, de Base de Datos, se planifican las Pruebas Unitarias, etc.

- **Codificación**: En esta fase entran en juego los Programadores. Siguen las indicaciones de la documentación generada por los Analistas para crear el software necesario. Si la documentación está muy bien detallada, programar resulta casi una tarea mecánica.

- **Pruebas**: En esta fase, personal diferente al que ha programado el software testea la aplicación en busca de errores. Cualquier problema o inconsistencia será reportado para que se subsane.

- **Implantación**: En esta fase, se realiza la entrega y su puesta en marcha del software. Esto no significa que se haya acabado el proyecto…

- **Mantenimiento**: Normalmente, siempre hay detalles pendientes una vez el proyecto está en marcha, desde errores sencillos a correcciones menores. Es habitual que una empresa extienda un contrato de mantenimiento adicional por el software que ha creado, de forma que con los años se vayan añadiendo funcionalidades o modificando partes del mismo.

Hoy en día existen formas más modernas y dinámicas de gestionar el desarrollo de un proyecto. Lo que hemos visto es simplemente la forma 'de libro', o lo que se llama, **Ciclo de Vida en Cascada**.

Diseño de Ejemplo: Agenda de Música

Con el fin de entender las Fases de Diseño y aplicarlas después a nuestros propios desarrollos, vamos a suponer que una empresa nos contrata para realizar un proyecto que consiste en una **Agenda de Música**. Básicamente lo que desea es una aplicación que utilice una Base de Datos para almacenar usuarios y canciones.

No vamos a trabajar todos los puntos del Diseño al completo, pero sí vamos a profundizar en una serie de técnicas que nos permitan sentar unas bases sobre las que programar nuestra Agenda. Recordad que no sois Analistas; sois Programadores. Pero cuanto mejor entendáis estas bases, más sencillo os resultará después manejaros con la documentación del proyecto.

Fase de Análisis [Proyecto]

Durante esta fase, se mantienen una serie de entrevistas con el cliente y se recoge lo que quiere. Normalmente, el cliente no tiene ni idea de lo que implica el desarrollo de un proyecto de software, por lo que es importante que los Analistas sean capaces de ir detectando los problemas poco a poco, solicitando información y por supuesto, buscando las tecnologías que deberán utilizarse.

Para nosotros, estudiantes de programación, esta fase se resume muchas veces a un **enunciado de un ejercicio o reto**. El profesor nos dice lo que hay que hacer, y lo hacemos. Y ya está.

Obviamente, cuando nos enfrentamos a un **proyecto real** la cosa cambia. Ya no tenemos un párrafo con un par de indicaciones vagas, sino una pila de hojas con un montón de movidas que detallan el 100% del proyecto de forma no ambigua. Normalmente es un aburrimiento de leer esta parte de la documentación. Veamos un par de ejemplos para el caso de nuestra Agenda de Música.

Requisitos Funcionales

2. Login de Usuario

2.1. El usuario se identificará por su login y su password

2.2. El login debe ser único en nuestro sistema para cada usuario

2.3. Introducir erróneamente el login y/o password un mínimo de tres veces bloqueará dicho usuario.

2.4. Un usuario bloqueado no puede acceder al sistema mientras permanezca en ese estado.

2.5. Todo usuario identificado con éxito accederá al menú principal.

Extracto de la Documentación de Análisis

Requisitos No Funcionales

11. Login de Usuario

11.1. El password debe tener una longitud mínima de 8 caracteres alfanuméricos

11.2. El password debe contener caracteres en mayúscula y números

11.3. El password deberá cambiarse cada dos meses

11.4. El password no podrá repetirse anualmente.

Extracto de la Documentación de Análisis

Todos estos requisitos vienen siempre acompañados de más información, desde detalles técnicos sobre el lenguaje de programación a utilizar (y su versión), a explicaciones sobre quiénes son las personas responsables de cada cosa en el proyecto. Por supuesto, siempre se incluyen extensas descripciones sobre los motivos por los que se realiza el proyecto, descripciones de quién es el cliente, cómo trabaja, etc. Esto es lo que se llama normalmente **Contextualización del Proyecto**.

Fase de Diseño [EDE, BBDD]

Durante esta fase, se describe cómo va a ser la aplicación que se va a desarrollar. Existen diversas técnicas para conseguirlo, algunas de las cuales ya las habéis trabajado en otras asignaturas. Por ejemplo, los **Diagramas de Entidad-Relación** que aprendisteis en Base de Datos se utilizan para explicar cómo van a ser las tablas que contendrán la información del proyecto.

La **Documentación de Diseño** se genera siempre basándose en la **Documentación de Análisis**. Es decir, si tú has establecido que un requisito para el login es que debe ser único en nuestro sistema para cada usuario, entonces tendrás que reflejarlo de alguna manera en el Diseño. La forma concreta puede variar. Lo que NO puede suceder es que no tengas en cuenta un requisito.

La **Documentación de Diseño** también suele ser muy extensa y estar dividida en partes. Por ejemplo, puede ser que el proyecto tenga dos componentes separados: la gestión de usuarios y la gestión de las canciones. Para explicar correctamente cada una de las partes puede hacerse lo siguiente:

1.- Gestión de Usuarios

Descripción de lo que implica la gestión de usuarios en nuestro programa.

1.1. Diagrama de Clases

Con las clases implicadas en el login, registro, etc.

1.2. Diagrama de Clases (POJO)

Con las clases de los POJO de los usuarios (solamente)

1.3. Diagrama de E-R

Con las tablas de la Base de Datos implicadas

1.4 Interfaces de Usuario

Con pantallazos de las ventanas de usuario (maquetas)

Extracto de la Documentación de Diseño

Obviamente, habrá muchas más explicaciones y tablas con descripciones, no solamente un diagrama UML suelto. El detalle de esta documentación puede llegar incluso a describir hasta el tipo de datos de cada columna de cada tabla de base de datos.

A tener en cuenta por los programadores:

- Diseñar ignorando los requisitos **es un error**.
- Programar ignorando el diseño **es un error**.
- Ante la duda, la documentación **siempre gana**. **Siempre**.
- Está **igual de mal** que tu programa no cumpla los requisitos de diseño como que haga cosas que no vienen especificadas en la documentación.
- Cuando encuentres una inconsistencia**, lo correcto es** recurrir al Analista. Él decidirá lo que hay que hacer. A veces incluso puede cambiar el Diseño.

Ejemplo práctico: Diseño inicial de Clases [Prog]

Los primero que se hace al comenzar el diseño de una aplicación, antes incluso de empezar a pensar en clases, es pensar en la **arquitectura** de nuestra aplicación. Lo más habitual es pensar en que usaremos un patrón **Modelo-Vista-Controlador**, y por tanto tendremos:

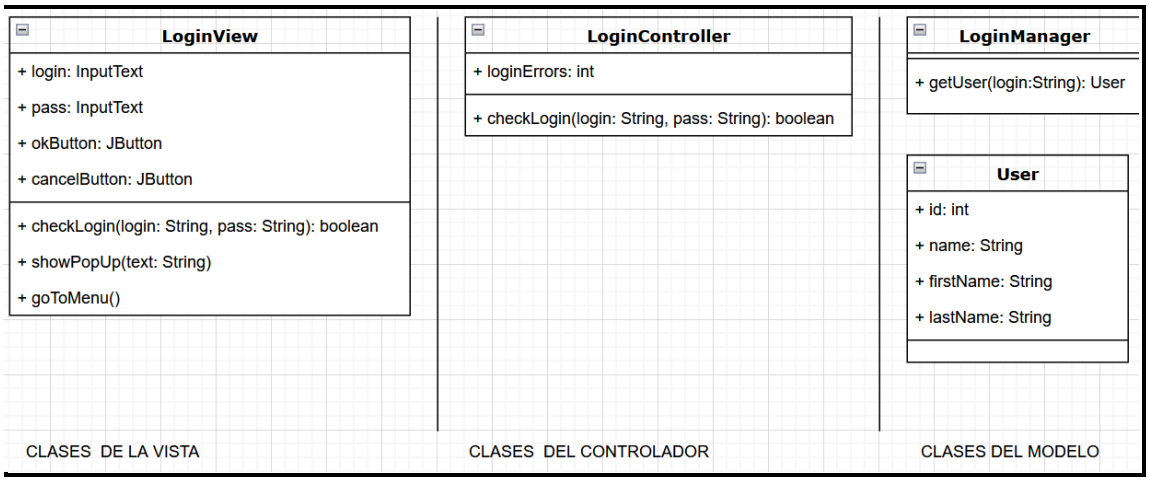
- **Vista**: una serie de clases que se encargarán de interactuar con el usuario, recibiendo solicitudes y mostrando resultados.

- **Controlador**: las clases que 'toman decisiones' y las que 'hacen cosas'.

- **Modelo**: las clases que tienen que ver con la persistencia de la información, en general, las que tienen que ver con la Base de Datos.

Una vez escogida la arquitectura, para realizar un **diseño preliminar** se suele utilizar alguna clase de técnica de brainstorming. Uno o más Analistas estudian por separado una misma parte de la documentación y van generando un borrador con las clases, atributos y métodos que creen que van a formar parte de la aplicación. Van colocando cada clase en cada una de las columnas, Vista, Modelo o Controlador, y lo completan con toda la información que cada uno cree conveniente. Después, los Analistas se reúnen y ponen en común sus diseños preliminares. Si una clase coincide en todos los casos, pasa al diseño definitivo. Si no, los Analistas estudian el asunto y alcanzan una decisión en consenso. Cuando han completado el diseño definitivo, se revisa varias veces más hasta que todo el mundo está de acuerdo que se ha completado con éxito. Los diferentes patrones de diseño estudiados en Programación suelen ser añadidos al proyecto en este momento.

Por ejemplo, supongamos la **Agenda de Música**. Al estudiar la Documentación de Análisis se han ido identificando diferentes clases, atributos y métodos. Para el ejemplo mostrado en el apartado de Requisitos Funcionales y No Funcionales, uno de los Analistas propone la siguiente solución:



Brainstorming inicial

El Analista ha decidido que cuando el usuario le pulsa a Ok en la ventana, se llama al método checkLogin () de LoginView. **No se ha molestado** en pensar cómo se programa ese método (eso es cosa del Programador), pero ha decidido que para que funcione tiene que llamar al método checkLogin () de LoginController. De alguna manera, este método sabe cuántos reintentos se han hecho, llamará según lo necesite a getUser () de Login-Manager, y decidirá si el login es correcto. Responderá true o false a LoginView, que según lo que sea, usará o showPopUp () con un mensaje de error o goToMenu () para cambiar a la ventana del menú.

Evidentemente, la cosa no queda aquí, ya que la documentación es mucho más extensa y puede implicar más de un centenar de clases diferentes. Lo que sí tiene que hacerse es diseñarlo todo de forma homogénea. Si todo es igual, es más fácil de arreglar cuando falle. Por tanto, según este Analista:

- **Vista**: las clases de aquí nunca hacen nada, solo recogen y muestran datos.

- **Controlador**: todo lo que sea 'hacer algo' por tonto que sea se tiene que hacer en una clase de aquí.

- **Modelo**: aquí sólo hay POJOs y Gestores de cada tabla.

Una vez completada su tarea, sólo queda poner en común su diseño con sus compañeros y realizar las adaptaciones necesarias.

Antes de que lo preguntes:

No, este diagrama brainstorming no es un Diagrama de Clases al uso con relaciones y tal y tal. De hecho, está mal hecho porque no las tiene. Pero es muy útil para los que están aprendiendo las bases de diseño hacer una chapuza como esta para tener las bases sobre las que construir tu software.

Ejemplo práctico: Diseño de Base de Datos [BBDD]

Cuando tienes que describir cómo son las tablas, normalmente se hace lo mismo que en los ejercicios de la asignatura de Base de Datos. Obviamente, el 'enunciado' del ejercicio es en nuestro caso es la **Documentación de Análisis**. Estos apartados suelen ser algo parecido a lo siguiente:

1.6- Base de Datos

Se definen dos usuarios diferentes: Usuario y Administrador.

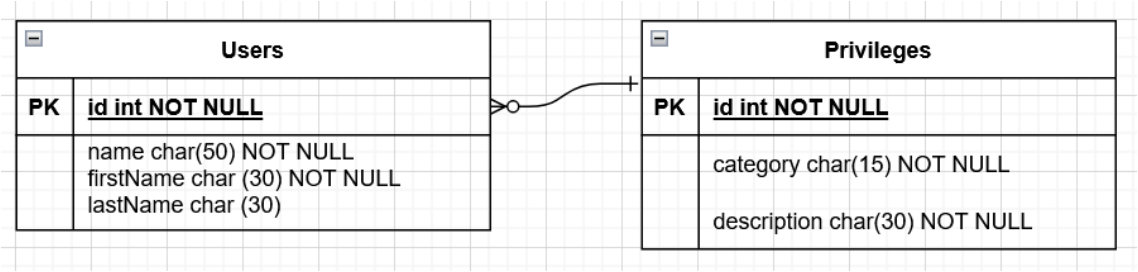
1.6.1 Usuarios

Se define Usuario como la persona que accede a nuestra aplicación. Sus datos son los siguientes:

- Id: Tipo int, no nulo, auto incremental.
- Name: Tipo char (50), no nulo.
- …

Un usuario dispone de una Privilegio, definido en la tabla…

Extracto de la Documentación de Diseño



Extracto del Diagrama de E-R de la Documentación de Diseño

Fase de Desarrollo [Prog]

A esta fase hay que llegar con los deberes hechos. Y por deberes me refiero a la **Documentación de Diseño** 100% terminada, revisada y aprobada por los Analistas y el Cliente.

Poco hay que decir de esta fase que no te imagines ya. El Analista suele dividir el trabajo en **subtareas** sencillas que va asignando a cada Programador. Por ejemplo, uno puede dedicarse a implementar la Base de Datos y otro a implementar las ventanas. Si el Analista ha hecho bien su trabajo, todos los Programadores podrán trabajar a la vez en distintas partes del proyecto. Para coordinar este tipo de cosas se usan aplicaciones como el Trello, pero en versiones mucho más sofisticadas.

La gente en general tiene ideas preconcebidas sobre las responsabilidades de los Analistas y los Programadores.

- Un Analista es un experto en diseño. Aunque han programado antes, en general llevan mucho tiempo sin dedicarse a ello profesionalmente, por lo que están 'oxidados' y no les suele gustar que les mareen con movidas de programadores. Tienen una visión del conjunto de la aplicación que los programadores no tienen.

- Un Programador es un experto en un lenguaje de programación en concreto. Sabe más que el Analista sobre ello, por lo que debe de entender que a éste se le escapan ciertos detalles de la implementación. Pero hay que entender que un Programador tiene una visión parcial del conjunto de la aplicación.

Fase de Pruebas [EDE]

El diseño de las pruebas de usuario, caja negra, caja blanca, etc. se realiza durante la **Fase de Diseño**, inmediatamente después de terminar el diseño completo, pero antes de programar una sola línea de código. Normalmente quienes realizan las pruebas son los propios Programadores, pero nunca sobre tu propio código. Esto se debe a que inconscientemente las personas inconscientemente tendemos a evitar las cosas que sabemos pueden fallar, lo que impide que probemos a conciencia nuestro propio código.

En general, las pruebas se realizar en sucesivas etapas tras completar la implementación de una parte del código. Cada vez que se encuentra un error se reporta la incidencia al Programador al cargo. Ningún programa puede ser puesto en producción si no ha superado todas las pruebas diseñadas.

Fase de Implantación y Mantenimiento [EDE]

Normalmente la implantación la hace el Analista. En cuanto al mantenimiento, la mayor parte de las veces consiste en dar soporte a errores durante un tiempo, tarea para la que se destinó un paquete de horas para un Programador que ha participado en el proyecto.