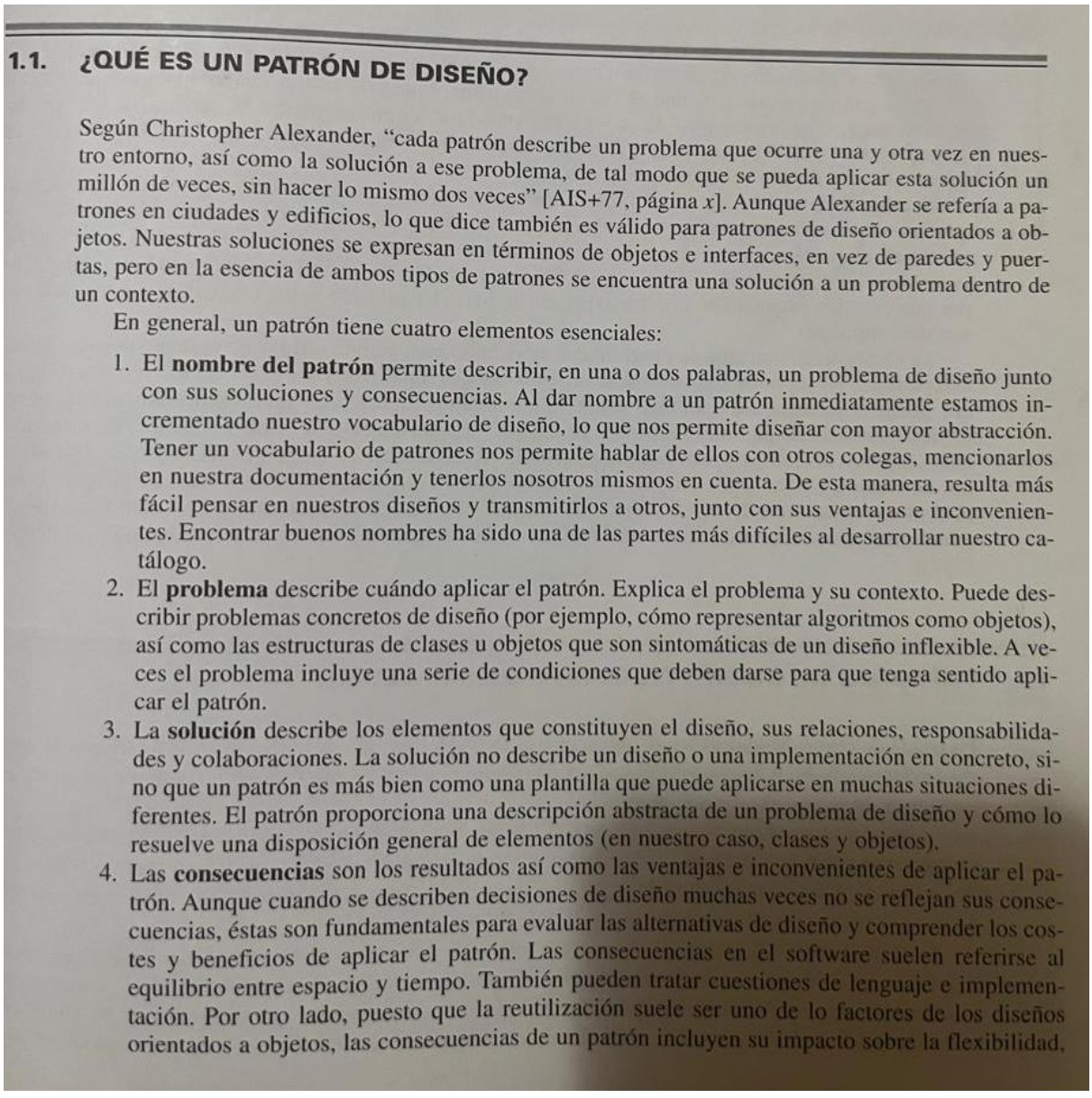
UNIDAD 3: INTRODUCCION AL DISEÑO DE SOFTWARE

Que es diseño. Conceptos. Características. Diferencia entre análisis y Diseño.

Principios Generales de Diseño: cohesión, acoplamiento y reusabilidad

Apunte Patrones de Diseño



1.1. ¿QUÉ ES UN PATRÓN DE DISEÑO?

Según Christopher Alexander, "cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces" [AIS+77, página *x*]. Aunque Alexander se refería a patrones en ciudades y edificios, lo que dice también es válido para patrones de diseño orientados a objetos. Nuestras soluciones se expresan en términos de objetos e interfaces, en vez de paredes y puertas, pero en la esencia de ambos tipos de patrones se encuentra una solución a un problema dentro de un contexto.

En general, un patrón tiene cuatro elementos esenciales:

1. El **nombre del patrón** permite describir, en una o dos palabras, un problema de diseño junto con sus soluciones y consecuencias. Al dar nombre a un patrón inmediatamente estamos incrementado nuestro vocabulario de diseño, lo que nos permite diseñar con mayor abstracción. Tener un vocabulario de patrones nos permite hablar de ellos con otros colegas, mencionarlos en nuestra documentación y tenerlos nosotros mismos en cuenta. De esta manera, resulta más fácil pensar en nuestros diseños y transmitirlos a otros, junto con sus ventajas e inconvenientes. Encontrar buenos nombres ha sido una de las partes más difíciles al desarrollar nuestro catálogo.
2. El **problema** describe cuándo aplicar el patrón. Explica el problema y su contexto. Puede describir problemas concretos de diseño (por ejemplo, cómo representar algoritmos como objetos), así como las estructuras de clases u objetos que son sintomáticas de un diseño inflexible. A veces el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón.
3. La **solución** describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o una implementación en concreto, sino que un patrón es más bien como una plantilla que puede aplicarse en muchas situaciones diferentes. El patrón proporciona una descripción abstracta de un problema de diseño y cómo lo resuelve una disposición general de elementos (en nuestro caso, clases y objetos).
4. Las **consecuencias** son los resultados así como las ventajas e inconvenientes de aplicar el patrón. Aunque cuando se describen decisiones de diseño muchas veces no se reflejan sus consecuencias, éstas son fundamentales para evaluar las alternativas de diseño y comprender los costes y beneficios de aplicar el patrón. Las consecuencias en el software suelen referirse al equilibrio entre espacio y tiempo. También pueden tratar cuestiones de lenguaje e implementación. Por otro lado, puesto que la reutilización suele ser uno de lo factores de los diseños orientados a objetos, las consecuencias de un patrón incluyen su impacto sobre la flexibilidad,

extensibilidad y portabilidad de un sistema. Incluir estas consecuencias de un modo explícito nos ayudará a comprenderlas y evaluarlas.

Qué es y qué no es un patrón de diseño es una cuestión que depende del punto de vista de cada uno. Lo que para una persona es un patrón puede ser un bloque primitivo de construcción para otra. En este libro nos hemos centrado en patrones situados en un cierto nivel de abstracción. *Patrones de Diseño* no se ocupa de diseños como listas enlazadas y tablas de dispersión (*hash*) que pueden codificarse en clases y reutilizarse como tales. Tampoco se trata de complicados diseños específicos de un dominio para una aplicación o subsistema completo. Los patrones de diseño de este libro son *descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en un determinado contexto.*

Un patrón de diseño nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se centra en un problema concreto, describiendo cuándo aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como las consecuencias y las ventajas e inconvenientes de su uso. Por otro lado, como normalmente tendremos que implementar nuestros diseños, un patrón también proporciona código de ejemplo en C++, y a veces en Smalltalk, para ilustrar una implementación.

Aunque los patrones describen diseños orientados a objetos, están basados en soluciones prácticas que han sido implementadas en los lenguajes de programación orientados a objetos más usuales, como Smalltalk y C++, en vez de mediante lenguajes procedimentales (Pascal, C, Ada) u otros lenguajes orientados a objetos más dinámicos (CLOS, Dylan, Self). Nosotros hemos elegido Smalltalk y C++ por una cuestión pragmática: nuestra experiencia diaria ha sido con estos lenguajes, y éstos cada vez son más populares.

La elección del lenguaje de programación es importante, ya que influye en el punto de vista. Nuestros patrones presuponen características de los lenguajes Smalltalk y C++, y esa elección determina lo que puede implementarse o no fácilmente. Si hubiéramos supuesto lenguajes procedimentales, tal vez hubiéramos incluido patrones llamados "Herencia", "Encapsulación" y "Polimorfismo". De manera similar, algunos de nuestros patrones están incluidos directamente en lenguajes orientados a objetos menos corrientes. CLOS, por ejemplo, tiene multi-métodos que reducen la necesidad de patrones como el Visitor (página 305). De hecho, hay suficientes diferencias entre Smalltalk y C++ como para que algunos patrones puedan expresarse más fácilmente en un lenguaje que en otro (por ejemplo, el Iterator (237)).

1.2. PATRONES DE DISEÑO EN E...

1.2. PATRONES DE DISEÑO EN EL MVC DE SMALLTALK

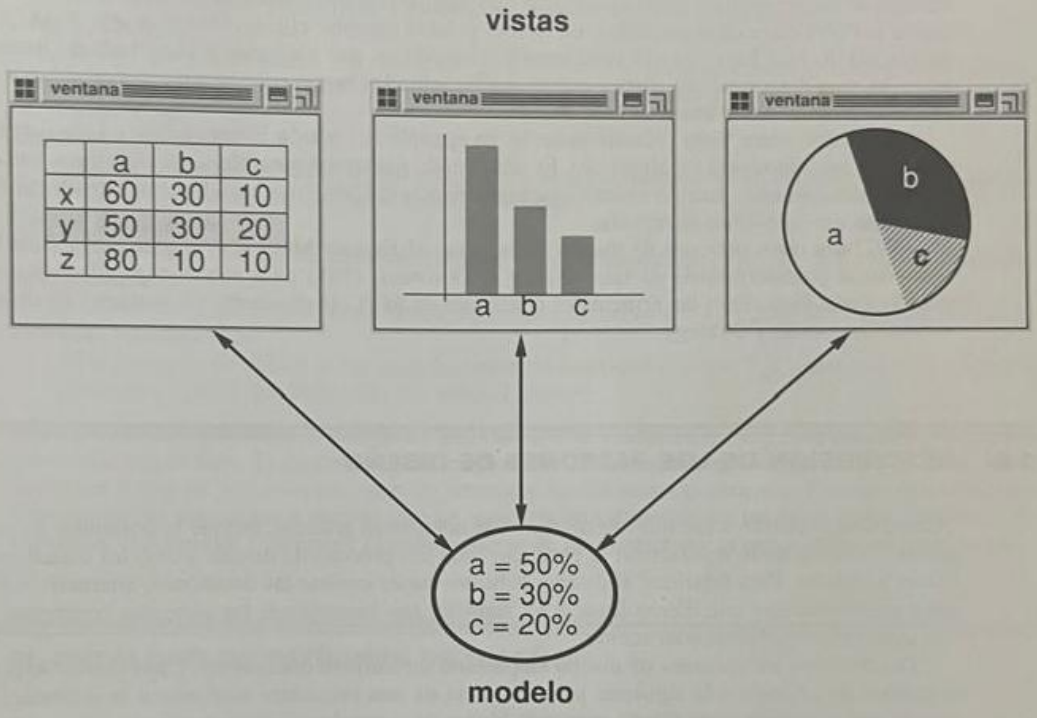
La tríada de clases Modelo/Vista/Controlador (MVC) [KP88] se usa para construir interfaces de usuario en Smalltalk-80. Observar los patrones de diseño que hay en MVC debería ayudar a entender qué queremos decir con el término "patrón".

MVC consiste en tres tipos de objetos. El Modelo es el objeto de aplicación, la Vista es su representación en pantalla y el Controlador define el modo en que la interfaz reacciona a la entrada del usuario. Antes de MVC, los diseños de interfaces de usuario tendían a agrupar estos objetos en uno solo. MVC los separa para incrementar la flexibilidad y reutilización.

MVC desacopla las vistas de los modelos estableciendo entre ellos un protocolo de suscripción/notificación. Una vista debe asegurarse de que su apariencia refleja el estado del modelo. Cada vez que cambian los datos del modelo, éste se encarga de avisar a las vistas que dependen de él. Como respuesta a dicha notificación, cada vista tiene la oportunidad de actualizarse a sí misma. Este enfoque

permite asignar varias vistas a un modelo para ofrecer diferentes presentaciones. También se pueden crear nuevas vistas de un modelo sin necesidad de volver a escribir éste.

El siguiente diagrama muestra un modelo y tres vistas (hemos dejado fuera los controladores para simplificar). El modelo contiene algunos valores de datos y las vistas, consistentes en una hoja de cálculo, un histograma y un gráfico de tarta que muestran estos datos de varias formas. El modelo se comunica con sus vistas cuando cambian sus valores, y las vistas se comunican con el modelo para acceder a éstos.



Si nos fiamos de él, este ejemplo refleja un diseño que desacopla las vistas de los modelos. Pero el diseño es aplicable a un problema más general: desacoplar objetos de manera que los cambios en uno puedan afectar a otros sin necesidad de que el objeto que cambia conozca detalles de los otros. Dicho diseño más general se describe en el patrón Observer (página 269).

Otra característica de MVC es que las vistas se pueden anidar. Por ejemplo, un panel de control con botones se puede implementar como una vista compleja que contiene varias vistas de botones anidadas. La interfaz de usuario de un inspector de objetos puede consistir en vistas anidadas que pueden ser reutilizadas en un depurador. MVC permite vistas anidadas gracias a la clase VistaCompuesta, una subclase de Vista. Los objetos VistaCompuesta pueden actuar simplemente como objetos Vista, es decir, una vista compuesta puede usarse en cualquier lugar donde pudiera usarse una vista, pero también contiene y gestiona vistas anidadas.

De nuevo, podríamos pensar en él como un diseño que nos permite tratar a una vista compuesta exactamente igual que a uno de sus componentes. Pero el diseño es aplicable a un problema más general, que ocurre cada vez que queremos agrupar objetos y tratar al grupo como a un objeto individual. El patrón Composite (151) describe este diseño más general. Dicho patrón permite crear una jerarquía

Tomando como base estas definiciones, investigar y presentar un informe que describa los siguientes patrones:

- Singleton
- Factory
- Observer
- Strategy

- Repository