**Respuesta a Preguntas Adicionales**

1. **¿Cuáles serían las cualidades para un código limpio?**

Un código limpio es un código que se ve claramente lo que hace:

* A partir del nombramiento de variables y de métodos cualquiera puede entender sin mucho esfuerzo lo que el programador original quiso hacer.
* Está estructurado con métodos de soporte de tal manera que un método tiene una función puntual que puede establecerse claramente.
* No requiere comentarios ni documentación adicional en el código fuente.
* Cada método tiene un número limitado de líneas.
* El número de parámetros máximo es dos.

Por supuesto, estas son reglas generales y pueden aplicarse excepciones, pero en los casos de excepciones, el programador debe tener clara la razón. La idea es permitir que el código sea muy legible y, por tanto, mantenible.

1. **¿Cuáles serían los estándares según tú para un buen proyecto?**

Creo que los estándares principales dependen de un acuerdo entre el equipo, líderes y las personas que usan o reciben el resultado del proyecto. Es muy importante que este acuerdo exista y si es posible, esté estipulado explícitamente. El acuerdo generalmente consiste en:

* Determinar cómo se mide el resultado del proyecto
* Determinar cómo se va a evaluar el producto generado en términos de calidad. ¿Qué métricas van a utilizarse y cuáles van a ser sus límites inferiores y superiores?
* Determinar las fechas y límites de tiempo para entrega en producción y en los diferentes ambientes de cada uno de los entregables.
* Determinar qué condiciones tiene una entrega, por ejemplo: una entrega en producción aceptable es una que ha contemplado pruebas de usuario previamente al despliegue.
* Determinar reglas de comportamiento de equipo (humanas) y de interacción, que no son para nada menos importantes: ¿Qué es aceptable y que no? Ejemplos: Cuando alguien llega tarde al daily, debe reponer una hora, o llevar comida a sus compañeros; la interacción con los usuarios debe ser a través del PM; la comunicación para cambios debe hacerse por slack, y así sucesivamente.

En la medida en que este acuerdo esté claro, y los miembros del proyecto se comprometan a cumplirlo, se puede hablar de un proyecto exitoso,

1. **¿Qué patrones conoce y utiliza?**

En cuanto a patrones de diseño de software, creo que todos conocemos y utilizamos los patrones descritos por The Gang of Four:

1. **Patrones creacionales:** He usado el **Abstract Factory** para establecer un tipo de Conexión a base de datos (hace mucho tiempo) para establecer conexión con varias bases de datos y sólo crear los elementos de conexión necesarios, de acuerdo con configuración de usuario. Entonces sólo los elementos de configuración determinarían qué implementación real se realizaría dependiendo de la base de datos utilizada en un momento dado. Para esto se creaban componentes abstractos y se instanciaban las implementaciones de acuerdo al requerimiento establecido en la configuración.
2. **Patrones de comportamiento:** Uno de los más utilizados por mi parte es el **Observer,** y es utilizado más que todo en la parte Web, gráfica, pero también en otros contextos, por ejemplo, actualización de datos o de estado de acuerdo con un cambio en una parte del sistema. Este patrón permite evitar que exista una centralización de la responsabilidad para efectuar todos los cambios en los objetos relacionados en un sistema, y poner esta responsabilidad en los objetos que deben cambiar. Básicamente el objeto Observado, envía una notificación de que cambió y los objetos Observer ( que se han suscrito a esta notificación) tienen como responsabilidad determinar qué cambios implica para sí mismo y aplicarlos. De tal manera, que no hay un ente central que recorra los objetos y deba saber qué cambios aplicar en cada caso, lo cual sería inadecuado en términos de mantenibilidad y performance.
3. **Patrones de estructura:** Uno de mis favoritos es el **Decorator,** ya que permite agregar funcionalidad a un objeto dependiendo de una implementación concreta. Este concepto se usa en muchos niveles, especialmente cuando se quiere agregar comportamiento dinámicamente, sin afectar la implementación básica del objeto, ni alterar su compatibilidad con implementaciones anteriores.

En cuanto a patrones de arquitectura, más que todo he usado aquellos que trata de comunicación e integración entre sistemas, así como procesamiento de datos:

* **Publish/suscribe:** In the use of systems like Kafka for sharing data
* **Map Reduce:** Optimized processing of data systems to perform calculations
* **CQRS:** Arquitectura optimizada para poder al mismo tiempo hacer consultas demandantes y, por otra parte, tener un sistema vivo en el cual los usuarios puedan seguir actualizando y creando nuevos datos. En arquitecturas como esta, debe haber consistencia eventual como compromiso para poder tener lo mejor de ambos mundos.
* **Pipeline pattern :** Patrón especializado para realizar procesamiento de procesos de negocio o de datos cuando intervienen muchos pasos en la ejecución de un proceso.