Capitulo 6. Conclusión

En el desarrollo de software tal como lo afirma Freeman, lo único constante es el cambio [Freeman, 2004], sin importar que tan bien esté diseñada una aplicación, a través del tiempo, puede crecer y cambiar o convertirse en un sistema que se vuelva obsoleto.

Algunas de las razones por las que el software está en constante cambio son:

- Cambios en los requerimientos del cliente.
- Creación de nuevas funcionalidades al software existente.
- Cambios en el software con el que interactúa el sistema.

Los patrones de diseño permiten que una parte del sistema varié de forma independiente sin afectar a las demás partes.

Los patrones de diseño ayudan a poder estructurar clases y objetos para poder resolver ciertos problemas y con ello poder adaptarlos al contexto de la aplicación. Tal es la importancia del uso de patrones de diseño que los principales ambientes de desarrollo como NetBeans ya contienen “plug-in’s” para aplicar a diseños UML de diagramas de clases patrones de diseño.

Una de las etapas durante el desarrollo de software en las que se debe poner mayor atención durante el desarrollo de software es el Diseño, ya que un sistema bien diseñado será menos propenso a fallos, además del costo que implica encontrar un fallo en diseño en

lugar de la etapa de las pruebas. Además sí se realiza una buena arquitectura y estructura de objetos ó clases será mucho más fácil su modificación en el futuro.

Algunas de las bases del diseño orientado a objetos son abstracción, encapsulamiento, polimorfismo y la herencia, sin embargo para que un diseño sea bueno, éste debe ser reusable, extensible y de fácil mantenimiento. Los patrones ayudan a construir sistemas con buena calidad en el diseño orientado a objetos.

Los patrones de diseño, permiten crear sistemas con un acoplamiento débil, es decir permiten construir sistemas orientados a objetos flexibles, que puedan cambiar, ya que minimizan la interdependencia entre objetos. En el sistema implementado en ésta tesis, cabe mencionar que el diseño alcanzado permite que los cambios dentro de una capa, no afecten a las demás capas, así que si se desea agregar alguna nueva funcionalidad, ésta no afectara a todo el sistema, solo se deberán crear nuevas clases u objetos con los nuevos comportamientos.

Actualmente la mayoría de los desarrolladores gastan demasiado tiempo encontrando o aislando fallas, el “debugger” es la principal herramienta que permite saber si un sistema funciona de manera correcta o no en tiempo real, y con ello poder aislar el fallo presentado, sin embargo es un proceso tedioso y cansado. Con el uso de JUnit, el uso del “debug” queda relegado.

Usando JUnit se puede saber de forma inmediata que clase u objeto, así como el método en el cual se presenta la falla, además se observa de manera inmediata si la falla está afectando algunas otras funcionalidades del sistema. Esta es una de las ventajas de uso

de “Test Driven Development”, ya que en la mayoría de los sistemas de las grandes empresas, cuando se presenta algún fallo, la única forma de encontrarlo es mediante “debugs”, sin embargo esto se complica cuando los sistemas son grandes y resulta imposible realizar el “debug” de toda la aplicación.

Tal como lo menciona Beck con el uso de “Test Driven Development” se debe pasar más tiempo diseñando que usando el “debugger” [Beck, 2004]. Un uso constante del “debugger” significa que hay algo mal en el diseño del sistema. Un punto importante de ésta tesis es que el uso del “debugger” no es necesario para detectar la causa de algún fallo, JUnit los detecta y aísla de forma automática, sin necesidad de ejecutar toda la aplicación, solo se necesita correr la “suite” de pruebas unitarias.

La combinación de ambas técnicas usadas durante la presente tesis permitió tener una arquitectura en la que los cambios afectan lo menos posible al sistema, y permiten de manera sencilla encontrar y aislar los fallos, así como asegurar que los requerimientos del cliente son los deseados ya que al pensar primero en las pruebas antes de escribir el código funcional, se asegura que los requerimientos del cliente sean cubiertos completamente.

6.1 Trabajo a Futuro

Después de haber desarrollado la presente tesis, algunas de las extensiones que se le pueden hacer tanto al sistema como al trabajo de investigación realizado se enlistan a continuación:

- JUnit es un marco de desarrollo de software que permite automatizar las pruebas unitarias y se encuentra en proceso de crecimiento. Una extensión a esta tesis, es el

la integración de JUnit con Ant, como herramienta para la implementación de pruebas unitarias.

- Creación de una tercera capa del sistema que servirá como “driver”, mediante el cual se puedan conectar otros sistemas para almacenar datos en el software, con ello se podrán usar más patrones de diseño.

- Creación de un importador de documentos XML con el cual se puedan crear tablas mediante el sistema usado.

- Creación de una interfaz Web para el sistema y con ello hacer uso de “HttpUnit” para la realización de las pruebas unitarias.

- Creación de un “plug-in” de “HttpUnit” para NetBeans.

- Creación de un “plug-in” de creación de patrones de diseño para el ambiente de desarrollo Eclipse.

- Uso de la versión 4 de JUnit en el” SQL Interpreter”.