# Implementación de Nuevos Requerimiento

# sobre el Trabajo Práctico del Grupo Nº10

## Análisis previo

Hemos realizado un análisis del sistema previo a realizar cualquier tipo de implementación. Luego de observar el diagrama UML de clases, leimos y ejecutamos los tests de aceptación. Los tests pasaban exitosamente y a priori las pruebas parecían completas. Luego continuamos ejecutando los tests unitarios de las clases. Los mismos también pasaron exitosamente.

## Evaluación de la construcción del sistema

Con los tests corriendo exitosamente, y una noción del diseño, decidimos realizar pequeñas variantes a los tests de aceptación para poder comprobar la robustez del diseño y su implementación.

El primer cambio que realizamos fue sobre el test de aceptación “PruebaIntegración7” que se muestra a continuación. El test originalmente creaba promociones del tipo 2X1 sobre un producto, y luego aplicaba esa promociones sobre una compra de 10 productos. Observamos que si variamos el número de productos de la compra a un número distinto de 10 el test de aceptación falla. Más aún, para algunos valores obtuvimos excepciones de casteo en la clase Producto.

A partir de este test fuimos recorriendo las clases y métodos involucrados hasta la excepción. Descubrimos en ese camino no solo una implementación que “fuertemente atada” a los tests que se diseñaron, sino también varios “code smells” que dificultan el entendimiento de la implementación.

## Code Smells y otros problemas

1) No modela correctamente el dominio del problema. En lugar de un modelo basado en objetos claro y orientado al problema, nos encontramos con un diseño más bien “relacional”

2) Type Cast Exceptions similares a la que se encontró cuando se analizaba el sistema

3) Inconsistencias en descuentos. Al variar la cantidad de productos encontrabamos que a veces se aplicaban descuentos más grandes cuando menor era la cantidad de productos que tenia la venta

4) Nombres de métodos poco descriptivos.  Por ejemplo, el método “procesarFactura” de la clase Factura

```
public void procesarFactura() {
        this.generarMontoTotalConDescuentos();
        descontarMontosDescuentosFactura();
        this.generarMontoTotalSinDescuentos();
}
```

5) El método mostrado anteriormente también peca de otros errores, por ejemplo, es un super método. Concentra toda la responsabilidad del procedimiento en lugar de delagarla en otros objetos.

## Resolución

Encontramos que la implementación de los nuevos requerimientos se volvia muy costosa con el diseño actual. No solo por lo poco simple y no orientado a objetos del diseño, sino también por los errores en la implementación que llevaban a que modificaciones controladas en un test unitario generen excepciones. Hemos realizado algunas modificaciones en el código en post de solucionar esos problemas y poder continuar con la implementación, pero no hemos llegado a buen puerto.