# Ingeniería del Software: Laboratorio

### Cuaderno de prácticas. Métricas de Proceso y calidad

## CONTENIDOS

El presente cuaderno de prácticas debe servir como guía al alumno para la asimilación y estudio de procesos y su mejora.

## EJERCICIO

El objetivo de esta práctica es que el estudiante comprenda la importancia de la medición de ciertos procesos de ingeniería críticos en proyectos con un ciclo de desarrollo a largo plazo.

Se tomará como código de ejemplo una pequeña aplicación que permita practicar algunas métricas de proceso. En un proyecto con un gran número de requisitos se podrían implementar métricas más complejas pero consumiría un tiempo del que no se dispone en este laboratorio.

Se supone que se parte de una línea base de la aplicación que implementa los siguientes requisitos:

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Descripción** | **Probado** |
| 001 | El prototipo permitirá crear cuentas bancarias de dos tipos: Cuenta corriente (CC) y Cuenta a plazo | X |
| 002 | Un cliente del sistema debe proveer como información obligatoria:   * DNI * Nombre * Teléfono * Fecha de nacimiento * Dirección | X |
| 003 | Para poder dar de alta una nueva cuenta (independientemente del tipo CC o cuenta a plazo) se debe proporcionar la siguiente información:   * Saldo * Cliente * Fecha de creación | X |
| 004 | Sobre una cuenta general se podrán realizar las siguientes operaciones:   * Retirar dinero * Ingresar dinero | X |
| 005 | Para crear una cuenta de tipo Cuenta Corriente se deberá proporcionar el número de tarjeta de crédito asociada |  |
| 006 | Para crear una cuenta de tipo Cuenta a Plazo se deberá proporcionar el interés y el plazo ya que son variables negociables según el tipo de cliente. |  |
| 007 | Como operación crítica del sistema se deben poder aplicar intereses en función del tipo de cuenta según los siguientes criterios:   * Si es una cuenta corriente (CC) se aplicara un 0,5% de interés cada vez que se invoque la operación de aplicar intereses * Si es una cuenta a plazo se creará con la información del interés y el plazo y los intereses se calcularán según la fórmula:  SaldoActual x(Interés/100)x Plazo |  |

El diseño de la aplicación se muestra en el siguiente diagrama de clases:



El código que implementa la línea base se adjunta a continuación (léase detenidamente):

**Cuenta.java**

public interface Cuenta

{

public double saldo();

public void ingresar(double cantidad);

public void retirar(double cantidad);

public Cliente getCliente();

public Fecha2 getFecApe();

public String toString();

public void aplicaIntereses();

}

**Cliente.java**

public class Cliente

{

private String dni;

private String nombre;

private Fecha2 fechaNac;

private String direccion;

private String tfno;

//constructor

public Cliente(String p\_dni, String p\_nombre, Fecha2 p\_fecnac, String p\_direccion, String p\_tfno)

{

dni = p\_dni;

nombre = p\_nombre;

fechaNac = p\_fecnac;

direccion = p\_direccion;

tfno = p\_tfno;

}

//métodos

public String getDni()

{

return this.dni;

}

public void setDni(String dni)

{

this.dni = dni;

}

public String getNombre()

{

return this.nombre;

}

public void setNombre(String nombre)

{

this.nombre = nombre;

}

public Fecha2 getFechaNac()

{

return this.fechaNac;

}

public void setFechaNac(Fecha2 fechaNac)

{

this.fechaNac = fechaNac;

}

public String getDireccion()

{

returnthis.direccion;

}

public void setDireccion(String direccion)

{

this.direccion = direccion;

}

public String getTfno()

{

return this.tfno;

}

publicvoid setTfnos(String tfno)

{

this.tfno = tfno;

}

public String toString()

{

return dni + " " + nombre;

}

}

**Fecha2.java**

//Clase para manejar fechas (DD/MM/AAAA)

import java.util.\*;

classFecha2

{

//atributos

private int año;

private int mes;

private int dia;

//constructores

public Fecha2(int dia, int mes, int año)

{

this.dia=dia;

this.mes=mes;

this.año=año;

}

public Fecha2(String cadenaFecha)

{

StringTokenizer st = new StringTokenizer( cadenaFecha,"/" );

String dd = st.nextToken();

String mm = st.nextToken();

String aa = st.nextToken();

this.dia=Integer.parseInt(dd);

this.mes=Integer.parseInt(mm);

this.año=Integer.parseInt(aa);

}

public Fecha2()

{

Calendar hoy = Calendar.getInstance();

año = hoy.get(Calendar.YEAR) ;

mes = hoy.get(Calendar.MONTH) + 1 ;

dia = hoy.get(Calendar.DAY\_OF\_MONTH) ;

}

//métodos

public String toString()

{

return dia + "/" + mes+ "/" + año;

}

public int getAño()

{

return this.año;

}

public void setAño(int año)

{

this.año = año;

}

public int getDia()

{

return this.dia;

}

public void setDia(int dia)

{

this.dia = dia;

}

public int getMes()

{

returnthis.mes;

}

public void setMes(int mes)

{

this.mes = mes;

}

}

**Decorador.java**

public abstract class Decorador implements Cuenta

{

private Cuenta cuenta;

public Decorador(Cuenta cuenta)

{

this.cuenta = cuenta;

}

public void setCuenta( Cuenta cuenta )

{

this.cuenta = cuenta;

}

public Cuenta getCuenta()

{

return cuenta;

}

public double saldo()

{

return cuenta.saldo();

}

public void ingresar(double cantidad)

{

cuenta.ingresar(cantidad);

}

public void retirar(double cantidad)

{

cuenta.retirar(cantidad);

}

public Cliente getCliente()

{

return cuenta.getCliente();

}

public Fecha2 getFecApe()

{

return cuenta.getFecApe();

}

public String toString()

{

return cuenta.toString();

}

public abstract void aplicaIntereses();

}

**Cuentageneral.java**

public class CuentaGeneral implements Cuenta

{

private double saldo;

private Cliente cliente;

private Fecha2 fecApertura;

public CuentaGeneral(double saldoini, Cliente cliente)

{

this.saldo = saldoini;

this.cliente = cliente;

this.fecApertura = new Fecha2();

}

public double saldo()

{

return this.saldo;

}

public void ingresar(double cantidad)

{

this.saldo += cantidad;

}

public void retirar(double cantidad)

{

this.saldo -= cantidad;

}

public Cliente getCliente()

{

returnthis.cliente;

}

public Fecha2 getFecApe()

{

returnthis.fecApertura;

}

public String toString()

{

return"\n### Información cuenta:\n Cliente: \n" + cliente.toString()

+ "\n Fecha apertura: " + this.fecApertura;

}

public void aplicaIntereses()

{

}

}**CuentaPlazo.java**

public class CuentaPlazo extends Decorador

{

//interés aplicado a la cuenta a plazo

private double interes;

//plazo en años

private int plazo;

public CuentaPlazo(Cuenta c, double interes, int plazo)

{

super(c);

this.interes = interes;

this.plazo = plazo;

}

public void aplicaIntereses()

{

double saldoActual = super.saldo();

super.getCuenta().ingresar(saldoActual\*(interes/100)\*plazo);

}

public double getInteres()

{

return this.interes;

}

public int getPlazo()

{

return this.plazo;

}

public String toString()

{

return super.toString() + "\nInterés: " + interes + " - Plazo: " + plazo;

}

}

**CuentaCorriente.java**

public class CuentaCorriente extends Decorador

{

//tarjeta de crédito asociada a la cuenta corriente

private String tarjetaCredito;

public CuentaCorriente(Cuenta c, String tc)

{

super(c);

this.tarjetaCredito = tc;

}

public String getTarjeta()

{

return this.tarjetaCredito;

}

publicvoid aplicaIntereses()

{

double interes=0.5; //0.5% de interés para todas las cuentas corrientes

double saldoActual = super.saldo();

super.getCuenta().ingresar(saldoActual\*(interes/100));

}

public String toString()

{

return super.toString() + "\nTarjeta de Crédito: " + tarjetaCredito;

}

}

**PruebaPatron.java**

public class PruebaPatron

{

public static void main( String args[] )

{

Fecha2 f1 = new Fecha2(15, 3, 1985);

Cliente juan = new Cliente( "15664386T", "Juan", f1, "calle1", "915554499" );

Fecha2 f2 = new Fecha2(10, 10, 1975);

Cliente maria = new Cliente( "89773374F", "María", f2, "calle2", "915558811" );

//creamos las cuentas generales

Cuenta c1 = new CuentaGeneral(1000,juan);

Cuenta c2 = new CuentaGeneral(20000,maria);

//ahora las envolvemos para que sean del tipo cuenta que necesitemos

//cuenta corriente con el número de la tarjeta de crédito

Decorador cc = new CuentaCorriente(c1, "1111222233334444");

System.out.println ("\n\nCuenta corriente: " + cc.toString());

System.out.println ("Su saldo actual es: " + cc.saldo());

//cuenta a plazo con un interés del 3% y una duración de 2 años

Decorador cp = new CuentaPlazo(c2, 3, 2);

System.out.println ("\n\nCuenta a plazo: " + cp.toString());

System.out.println ("Su saldo actual es: " + cp.saldo());

cp.aplicaIntereses();

System.out.println ("Su saldo después del plazo será de: " + cp.saldo());

}

}

Tras el desarrollo de la primera iteración del proyecto se solicitan cambios en el alcance de la aplicación, quedando una especificación de requisitos como la mostrada en la siguiente tabla (los cambios sobre la línea base se muestran en rojo):

|  |  |  |
| --- | --- | --- |
| Identificador | Descripción | Probado |
| 001 | El prototipo permitirá crear cuentas bancarias de dos tipos: Cuenta corriente (CC), Cuenta a plazo **y Cuenta Nómina**. |  |
| 002 | Un cliente del sistema debe proveer como información obligatoria:   * DNI * Nombre * Teléfono * Fecha de nacimiento * Dirección | X |
| 003 | Para poder dar de alta una nueva cuenta (independientemente del tipo CC o cuenta a plazo) se debe proporcionar la siguiente información obligatoria:   * Saldo * Cliente * Fecha de creación | X |
| 004 | Sobre una cuenta general se podrán realizar las siguientes operaciones:   * Retirar dinero * Ingresar dinero | X |
| 005 | Para crear una cuenta de tipo Cuenta Corriente se deberá proporcionar el número de tarjeta de crédito asociada |  |
| 006 | Para crear una cuenta de tipo Cuenta a Plazo se deberá proporcionar el interés y el plazo ya que son variables negociables según el tipo de cliente. |  |
| 065 | **Para crear una cuenta de tipo Cuenta Nómina se deberá proporcionar el salario del cliente** |  |
| 007 | Como operación crítica del sistema se deben poder aplicar intereses en función del tipo de cuenta según los siguientes criterios:   * Si es una cuenta corriente (CC) se aplicara un 0,5% de interés cada vez que se invoque la operación de aplicar intereses * Si es una cuenta a plazo se creará con la información del interés y el plazo y los intereses se calcularán según la fórmula: SaldoActual x(Interés/100)x Plazo * **Si es una Cuenta Nómina se deberá aplicar el interés sobre el salario del cliente siguiendo la fórmula: Salario x (0.3/100)** |  |
| 008 | **Se deberá crear un conjunto de procesos programables en función del tipo de cuenta que disparen las operaciones de cálculo de intereses. La lógica de negocio seguirá las siguientes reglas: TBD** |  |

A partir de la información de la nueva especificación de requisitos, se pide al alumno practicar las siguientes métricas de proceso:

1. Porcentaje de requisitos nuevos.
2. Cobertura de requisitos.
3. En la práctica no se calculará la tasa de requisitos derivados, pero ¿Qué interpretación puede darse a que esta tasa sea elevada?
4. Tasa de requisitos incompletos.
5. Tasa de requisitos probados (teniendo en cuenta la tercera columna de la tabla). En los casos de
   * Línea Base.
   * Nueva especificación
6. Modifique el código para cubrir las nuevas necesidades y calcule el porcentaje de reutilización.
7. A partir de las modificaciones calcúlese el índice de madurez CONSIDERANDO UNA CLASE COMO EL EQUIVALENTE A UN MÓDULO.

## Apendice A – Algunas métricas del proceso

Número de requisitos nuevos (o modificados) por mes. Se calcula como el cociente entre el número de cambios realizados y el número total de requisitos. El porcentaje de nuevos requisitos refleja la estabilidad del desarrollo, ya que los cambios en los requisitos tiene un fuerte impacto en la estabilidad del producto final.

*% de Requisitos Nuevos = Cambios en los requisitos/Requisitos totales*

La cobertura de los requisitos mide el porcentaje de requisitos actualmente incluidos en alguna línea base que han sido soportados por un componente incluido en el diseño detallado del sistema. Para computar este valor, es necesario comprobar, por ejemplo mediante una matriz de seguimiento, que existe una correlación entre el requisito y el componente y que dicha correlación está actualizada.

*Cobertura de requisitos = Requisitos soportados / Requisitos en la linea base*

El número de requisitos derivados a partir de cada requisito inicial es un indicador de la explosión de requisitos que se produce como consecuencia de una incorrecta especificación o de su descripción a un nivel de detalle incorrecto. En el caso medio, cada requisito inicial produce un cierto número de requisitos derivados. Sin embargo, la existencia de requisitos incorrectamente descritos conlleva una expansión e interpretación a menudo excesivas. De igual manera, la obtención de un número de requisitos derivados por cada requisito individual demasiado bajo, indica que los requisitos pueden estar a un nivel de detalle incorrecto (demasiado bajo, en este caso), por lo que deberá evaluarse el proceso de análisis de requisitos.

*Tasa de requisitos derivados = requisitos derivados/Requisitos iniciales*

La compleción de los requisitos mide cuántos requisitos han sido completamente definidos en un determinado punto del desarrollo. Algunos requisitos no pueden ser completamente definidos hasta etapas tardías del ciclo de desarrollo, sin embargo, postergar innecesariamente la definición de ciertos requisitos puede causar efectos negativos, especialmente siestos requisitos influyen en el diseño del sistema. A menudo los requisitos no descritos completamente han sido identificados como TBD (*to be determined*):*Tasa de requisitos incompletos = Requisitos TDB / Requisitos totales*

Dado que el estado de los requisitos cambia constantemente durante el desarrollo, es posible obtener medidas detallada sobre el estado actual de los requisitos no completados, computando un valor según su estado actual (en codificación, codificación completada, validado mediante pruebas, etc.). Así por ejemplo tendríamos:

*Tasa de requisitos probados = Requisitos probados / Requisitos totales*

Otras medidas también interesantes en el proceso de requisitos, son el número de requisitos asignados a una persona (útil para medir las cargas de trabajo en el proyecto), la frecuencia de cambios en el conjunto completo de requisitos con respecto al tiempo, el número de cambios con respecto a una línea base, el porcentaje de defecto cuya causa es un error en la especificación de requisitos o el número de solicitudes de cambio en los requisitos.

## Apendice B - Índice de madurez del software

El índice de madurez del software proporciona una indicación de la estabilidad de un producto software. Se define como:

*IMS = (MT − (Fa + Fm + Fe )) / MT*

Donde:

* *MT* = número de módulos en la versión actual.
* *Fm* = número de módulos en la versión actual que han sido modiﬁcados.
* *Fa* = número de módulos en la versión actual que han sido añadidos.
* *Fe* = número de módulos de la versión anterior eliminados de la versión actual.

Si el IMS se aproxima al valor 1, se dice que el producto comienza a estabilizarse, por lo que el esfuerzo de mantenimiento requerido será menor.