# Métricas Orientadas a Objetos

## Ejercicio 1:

|  |  |
| --- | --- |
| Problema | Medir ayuda a |
| Requisitos incorrectos | Describir los requisitos en términos expresados de manera que puedan ser verificables, medibles, no ambiguos. |
| Toma de decisiones | Cuantificar la aceptación de los usuarios, por ejemplo, en Lean, se está midiendo todo el tiempo el comportamiento de los usuarios, esto nos ayuda mucho para saber si estamos haciendo las cosas bien, o no. En caso de que no, entonces *pivotamos*, pero, si sí, entonces continuamos. |
| Falta de control | Analizar el progreso del equipo de desarrollo, tecnologías desde *monday.com* o *Trello*, nos ayudan a ver qué hace falta, qué ya se hizo y qué se está haciendo. Gracias a esto, podemos tomar las decisiones necesarias para reducir la falta de control y hacer que las áreas trabajen bien. |
| Exceso de gastos | Estimar el gasto futuro en un determinado proyecto, por ejemplo, al usar una API como las de Google Cloud, en sus métricas, nos dice que, si seguimos al mismo ritmo, entonces se factura cierta cantidad de dinero. Con esto, podemos comparar con otros servicios y cambiar de proveedor, o bien, ver el código y optimizar las llamadas a la API, con el fin de reducir gastos, aunque, todo depende del proyecto. |
| Costos de mantenimiento | Recabar datos como el costo de energía de un servidor propio o la eficiencia con la que se usa un servidor, por ejemplo, *Azure*, que brinda métricas para ver cómo se usan los recursos y las horas en las que se usan, con lo que podemos hacer presupuestos de lo que costaría apagar el servidor durante un tiempo o tener servidores paralelos, al igual que en el exceso de gastos, depende del proyecto. |

## Cuestionario a partir de los documentos de Métricas

**¿Por qué considera que puede ser útil estimar y medir el software?**

Porque las métricas nos dan un *mapa* del software, con las métricas podemos saber qué está ocurriendo, porque no sólo es un mapa *estático*, sino que cada vez que medimos va cambiando, esto es muy útil para el equipo de desarrollo (y por consiguiente para todos los *stakeholders*), porque pueden saber qué está bien, qué está mal, qué se debe mejorar, qué se debe hacer hoy y qué mañana, qué quiere el usuario, qué quiere el cliente, cuánto va a costar al mes, cuánto puede soportar, cuánto puede fallar, qué tanto cumple con lo que se pide. Así que, prácticamente es *como darle a un botón de Mi Ubicación* estando en el ciclo de desarrollo. ***Si conocemos dónde estamos y dónde debemos estar, entonces, podemos definir cómo ir del punto A al B.***

**¿Qué relación guarda con la calidad del producto y del proceso de desarrollo de un sistema de SW?**

Antes de responder, es necesario tener un contexto de la calidad. Entendamos por calidad, a la satisfacción de lo que fue pedido, de la manera esperada, tanto explícita como implícitamente. Entonces, la relación entre la calidad del producto y el proceso de desarrollo con las métricas se encuentra en que las personas encargadas del proyecto puede tomar decisiones al observar las métricas, por ejemplo, los desarrolladores pueden decidir una nueva implementación si las métricas les dice que el tiempo de espera es grande, o el coordinador del equipo de desarrollo puede decidir si primero se debe hacer la función Agregar a guardados o la función de Crear PDF, siendo que las métricas le indican que los usuarios se pasan mucho tiempo buscando documentos que enviaron en una conversación, o bien, el equipo de pruebas, puede decidir si medir primero una métrica dependiente de otra, o bien una independiente. En pocas palabras, ***cuando podemos contar lo intangible (el software) \*, podemos saber si los estamos haciendo bien o si debemos de hacerlo diferente.***

## Ejercicio: Cálculo de métricas orientadas a objetos

El objetivo del programa es realizar una medición rudimentaria de los métodos ponderados de una clase, la profundidad del árbol de herencia, y el número de subclases que heredan de esta clase.

El anterior programa fue realizado en C++. Como entradas, se recibieron los siguientes archivos:

-A.java: clase

import java.io.\*;

public class A implements D

{

public static void main (String args[])

{

System.out.println ("Hola, mundo");

for (int i = 0; i < n; i++)

if (a == b)

a;

else

{

while (a < b)

a;

}

}

}

-B.java: clase

import java.io.\*;

public class B extends A

{

do

System.out.println ("a");

while (j != 0);

}

-C.java: clase

import java.io.\*;

public class C extends B

{

switch (var1)

{

case 1:

break;

case 2:

break;

default:

break;

}

}

-D.java: interfaz

import java.io.\*;

public interface D

{

}

Se puede observar que A tiene una profundidad de herencia (DIT) de tres (contándose a sí mismo), con una clase hija directa. Hay tres instrucciones ponderadas en la clase (WMC).

-WMC: 3

-DIT: 3

-NOC: 1

El mismo análisis se puede realizar para el resto de las clases, cuyos resultados se pueden ver en la tabla de abajo. D, por ser una interfaz, no es apta para el análisis.

|  |  |  |  |
| --- | --- | --- | --- |
| **Clase** | **WMC** | **DIT** | **NOC** |
| A | 3 | 3 | 1 |
| B | 1 | 2 | 1 |
| C | 1 | 1 | 0 |

Notas: Para poder realizar el programa, fue necesario utilizar un archivo de texto con los nombres y las extensiones de los archivos por leer (cada uno en una línea distinta), y un directorio "files" que contiene las clases y las interfaces en cuestión.

El código del programa principal es el siguiente:

#include <bits/stdc++.h>

using namespace std;

void checkInstructions (string &, int (&)[5], bool &);

int

main (void)

{

bool doWhileFlag = false;

int instructionCount[5]; //Conteo de instrucciones

int numFiles = 0;

int res = 0;

/\*

\* [0]: switch

\* [1]: if

\* [2]: for

\* [3]: while

\* [4]: do-while

\*/

memset (instructionCount, 0, sizeof (instructionCount));

map < string, int >classNames;

map < string, int >interfaceNames;

string fileName;

string fileNameTemp;

string fileData;

numFiles = 3;

ifstream file;

file.open ("directorio.txt");

for (int i = 0; i < numFiles; i++)

{

getline (file, fileName);

cout << fileName << "\n";

fileNameTemp = fileName.substr (0, fileName.length () - 5);

classNames.insert (make\_pair (fileNameTemp, 1));

ifstream classFile;

string a = "files/" + fileName;

classFile.open (a);

while (getline (classFile, fileData))

{

cout << "\t" + fileData << "\n";

checkInstructions (fileData, instructionCount, doWhileFlag);

checkHierarchy (fileData, classNames, interfaceNames);

classFile.close ();

}

}

return 0;

}

void

checkHierarchy (string & fileData, map <string, int> &classNames, map <string, int> &interfaceNames)

{

int index = 0;

auto it;

string fileDataTemp;

index = fileData.find (" class ");

if (index != -1)

{

index = fileData.find (" class ");

index = index + 7;

while (fileData[index] != " " || fileData[index] != "\n")

fileDataTemp.append (fileData[index]);

it = classNames.find (fileDataTemp);

if (it == classNames.end ())

classNames.insert (make\_pair (fileDataTemp, 1));

else

it->second++;

}

index = fileData.find (" interface ");

if (index != -1)

{

index = fileData.find (" interface ");

index = index + 11;

fileDataTemp = fileData.substr (index, fileData.length () - index);

it = classNames.find (fileDataTemp);

if (it == classNames.end ())

classNames.insert (make\_pair (fileDataTemp, 1));

else

it->second++;

}

}

void

checkInstructions (string & fileData, int (&instructionCount)[5],

bool & doWhileFlag)

{

if (fileData.find ("switch") != -1)

instructionCount[0]++;

else if (fileData.find ("if") != -1)

{

if (fileData.find ("else if") == -1)

instructionCount[1]++;

}

else if (fileData.find ("for") != -1)

instructionCount[2]++;

else if (fileData.find ("while") != -1)

{

if (doWhileFlag == true)

doWhileFlag = false;

else

instructionCount[3]++;

}

else if (fileData.find ("do") != -1)

{

instructionCount[4]++;

doWhileFlag = true;

}

}