PROYECTO #1 Analizador Funcional

Conceptos Avanzados de Ingenieria de Software

TSPi

|  |  |
| --- | --- |
| ALEJANDRA MARÍA CHICA RIVERA | Cód. 201522947 |
| DANIEL FELIPE RENTERÍA MARTÍNEZ | Cód. 201522630 |
| DEIVID ALEXANDER OSORIO BARRERA | Cód. 201523018 |
| DIEGO ANDRES MONTEALEGRE GARCIA | Cód. 201510727 |
| SEBASTIAN CARDONA CORREA | Cód. 201510311 |

Contenido

[Introducción 3](#_Toc418519743)

[Propósito del documento 3](#_Toc418519744)

[Descripción 3](#_Toc418519745)

[Información del equipo 3](#_Toc418519746)

[Requerimientos Funcionales Ciclo 1 3](#_Toc418519747)

[Interfaz de Usuario 5](#_Toc418519748)

[Atributos de calidad 5](#_Toc418519749)

[Restricciones 5](#_Toc418519750)

[Referencias y Contexto 5](#_Toc418519751)

[Desarrollo 5](#_Toc418519752)

[Tecnologías 5](#_Toc418519753)

# Introducción

## Propósito del documento

Este documento tiene como propósito especificar los requerimientos contemplados en la planeación del ciclo 1, del proyecto Analizador funcional, para el equipo Minmose Software.

## Descripción

El programa que se desarrollará por el equipo de trabajo, es un analizador funcional. Se trata de una herramienta de software para ayudar a los programadores a determinar la complejidad de los programas que han desarrollado.

## Información del equipo

Líder de equipo: Deivid Alexander Osorio.

Líder de planeación: Diego Montealegre.

Líder de desarrollo: Alejandra Chica.

Líder de calidad: Daniel Rentería.

Líder de soporte: Sebastian Cardona.

# Requerimientos Funcionales Ciclo 1

Caso de uso 1: Realizar conteo de LOC

|  |  |
| --- | --- |
| **Nombre**:  Realizar conteo de LOC. | |
| **Descripción**:  Se debe realizar el conteo de LOC del programa a analizar. | |
| **Complejidad:**  Alta | **Documentos asociados:** |
| **Actores:**  Usuario final. | |
| **Precondiciones:**  El programa a analizar debe estar dentro de la carpeta src/site/resources del analizador funcional. | |
| **Plataformas / Servicios involucrados:**  Consola del SO  Heroku (ciclo 2) | |
| **Flujo básico:**  El usuario ejecuta el programa.  El sistema automáticamente realizar el conteo de LOC, de todas las clases que hagan parte del programa a analizar, basándose en el estándar de conteo de líneas, distinguiendo entre líneas totales del programa, y líneas efectivas (consideradas como código).  Se muestra el resultado en pantalla. | |
| **Manejos de situaciones anormales:**  Si ocurre un error al realizar el conteo, se mostrará un mensaje con esta información. | |

Caso de uso 2: Calcular Data Binding Externo

|  |  |
| --- | --- |
| **Nombre**:  Calcular Data Binding Externo. | |
| **Descripción**:  Se debe realizar el cálculo del data binding externo del programa. | |
| **Complejidad:**  Alta | **Documentos asociados:** |
| **Actores:**  Usuario final. | |
| **Precondiciones:**  El programa a analizar debe estar dentro de la carpeta src/site/resources del analizador funcional. | |
| **Plataformas / Servicios involucrados:**  Consola del SO  Heroku (ciclo 2) | |
| **Flujo básico:**  El usuario ejecuta el programa.  El sistema automáticamente realiza el cálculo del data binding externo del programa, de la siguiente manera:   * Para los métodos: se cuenta el número de parámetros, el número de variables que se modifican dentro de métodos, y si tiene retorno o no. * Para el total del programa: Se suma el resultado del data binding externo por método.   Se muestra el resultado en pantalla. | |
| **Manejos de situaciones anormales:**  Si ocurre un error al realizar el análisis, se mostrará un mensaje con esta información. | |

Caso de uso 3: Calcular Data Binding Interno

|  |  |
| --- | --- |
| **Nombre**:  Calcular Data Binding Interno. | |
| **Descripción**:  Se debe realizar el cálculo del data binding interno del programa. | |
| **Complejidad:**  Alta | **Documentos asociados:** |
| **Actores:**  Usuario final. | |
| **Precondiciones:**  El programa a analizar debe estar dentro de la carpeta src/site/resources del analizador funcional. | |
| **Plataformas / Servicios involucrados:**  Consola del SO  Heroku (ciclo 2) | |
| **Flujo básico:**  El usuario ejecuta el programa.  El sistema automáticamente realiza el cálculo del data binding interno del programa, de la siguiente manera:   * Para los métodos: se cuenta el número de parámetros. * Para el total del programa: Se suma el resultado del data binding interno por método, mas el numero de variables globales de todas las clases del programa.   Se muestra el resultado en pantalla. | |
| **Manejos de situaciones anormales:**  Si ocurre un error al realizar el análisis, se mostrará un mensaje con esta información. | |

Caso de uso 4: Calcular relacion de fuerza de acoplamiento.

|  |  |
| --- | --- |
| **Nombre**:  Calcular relaciòn de fuerza de acoplamiento. | |
| **Descripción**:  Se debe realizar el cálculo de la relaciòn de fuerza de acoplamiento del programa. | |
| **Complejidad:**  Alta | **Documentos asociados:** |
| **Actores:**  Usuario final. | |
| **Precondiciones:**  El programa a analizar debe estar dentro de la carpeta src/site/resources del analizador funcional. | |
| **Plataformas / Servicios involucrados:**  Consola del SO  Heroku (ciclo 2) | |
| **Flujo básico:**  El usuario ejecuta el programa.  El sistema automáticamente realiza el cálculo de la relaciòn de fuerza de acoplamiento del programa., de la siguiente manera:   * Calcula el data binding externo del programa * Calcula el data binding interno del programa * El resultado es la division del total del data binding interno, sobre el total del data binding externo del programa.   Se muestra el resultado en pantalla. | |
| **Manejos de situaciones anormales:**  Si ocurre un error al realizar el análisis, se mostrará un mensaje con esta información. | |

# Interfaz de Usuario

La interfaz de usuario, consistirá en una salida de texto por consola. Para el ciclo dos, se incluirá la salida de texto, a través de una interfaz web, sobre la plataforma Heroku.

# Atributos de calidad

* Escalabilidad: El programa debe ser diseñado de forma, que al ser desarrollado, las funcionalidades sean fáciles de implementar, y si más adelante se hace necesario incorporar nuevas funcionalidades, modificar o eliminar funcionalidades ya existentes, la afectación al código existente debe ser de mínimo impacto. Para lograr cumplir con este atributo, el código deberá contar con componentes reutilizables o clases de utilidades.
* Facilidad de Mantenimiento: El código debe ser legible, para que permita identificar y corregir posibles defectos.
* Facilidad de probar el sistema (testability): Para cumplir con este atributo, se hará necesario el uso de JUnit, como framework de pruebas, y por cada clase del modelo, deberá existir un caso de prueba.
* Portabilidad: El programa podrá ser ejecutado en cualquier computador siempre que se tenga instalado Maven (si es clonado desde GitHub) y el Java Runtime Environment.

# Restricciones

El programa a analizar debe estar dentro de la carpeta src/site/resources del analizador funcional.

# Referencias y Contexto

## Desarrollo

Cada miembro del equipo tendrá una responsabilidad como desarrollador.

## Tecnologías

Maven

Git

Java

Heroku