**DOCUMENTO DE REQUERIMIENTOS**

**PRESENTADO POR:**

**GRUPO UMBRELLA**

**JOSE JAVIER VIRVIESCAS TOLEDO**

**ALVARO ANDRES SUAREZ ALFONSO**

**ERICA ESTEFANNY PRADO CASAÑEDA**

**OSCAR JAVIER MORENO REY**

**GABRIEL ALBERTO CASTILLO PRADA**

**PRESENTADO A:**

**LUIS DANIEL BENAVIDES**

**UNIVERSIDAD DE LOS ANDES**

**INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**

**ESPECIALIZACIÓN EN CONSTRUCCIÓN DE SOFTWARE**

**BOGOTÁ D.C.**

**ABRIL 04 DE 2015**

TABLA DE CONTENIDO

[TABLA DE CONTENIDO 2](#_Toc418026196)

[1. INTRODUCCIÓN 3](#_Toc418026197)

[2. DEFINICION DE REQUERIMIENTOS FUNCIONALES 4](#_Toc418026198)

[3. INTERFAZ DE USUARIO 6](#_Toc418026199)

[4. ATRIBUTOS DE CALIDAD 9](#_Toc418026200)

[5. RESTRICCIONES 10](#_Toc418026201)

[6. REFERENCIAS Y CONTEXTO 13](#_Toc418026202)

[7. DISEÑO 15](#_Toc418026203)

[**7.1.** **Diagrama de clases** 15](#_Toc418026204)

[**7.2.** **Diagrama de despliegue** 16](#_Toc418026205)

[**7.3.** **Diagrama de casos de uso** 16](#_Toc418026206)

[**7.4.** **Diagrama de componentes** 17](#_Toc418026207)

1. INTRODUCCIÓN

Se pretende establecer un marco general de los requerimientos obtenidos en la etapa de elicitación del proyecto TSP con el fin de definir el pilar del desarrollo de las funciones analizadas para cada ciclo. Adicional a esto contar con una base común para los miembros del equipo y al final de cada ciclo poder verificar más fácilmente el cumplimiento de cada requerimiento.

Para dirigir el trabajo y el desarrollo de las funciones seguiremos los procesos estructurados por TSP con el fin de ejecutar un trabajo en equipo de forma organizada y exitosa mejorando los niveles de calidad y la productividad del proyecto.

Nuestro grupo de trabajo Umbrella: Álvaro Suarez (Líder del equipo), Javier Virviescas (Líder de desarrollo), Oscar Moreno (Líder de Soporte), Gabriel Castillo (Líder de Planeación) y Erica Prado (Líder de Calidad) presentamos el siguiente levantamiento de requerimientos de software.

1. DEFINICION DE REQUERIMIENTOS FUNCIONALES

A continuación se presenta una descripción de cada uno de los requerimientos funcionales a implementar para analizar un programa de software ya existente para determinar sus características:

* 1. **CICLO 1**

**R1.** Contar las líneas lógicas de código de todo el programa y contar las líneas lógicas de código de cada una de sus funciones, procedimientos o rutinas

Esta funcionalidad se requiere para realizar un análisis después de modificar o desarrollar software en diferentes ciclos con el fin de poder conocer el número de líneas de código que han sido agregadas, eliminadas o modificadas a lo largo de las diferentes versiones del programa. Esto evita que dicha actividad se tenga que realizar manualmente.

* 1. **CICLO 2**

**R2.** Calcular la complejidad ciclomática McCabe del programa y la de cada una de sus funciones, procedimientos o rutinas

Esta funcionalidad se requiere para determinar la complejidad de un programa y a su vez predecir su nivel de calidad. Estos parámetros son útiles para los desarrolladores cuando diseñan y luego buscan modificar, ampliar o mejorar programas de software. Además contribuyen a detectar componentes con posibles problemas de diseño para que sean corregidos antes de realizar pruebas.

**R3.** Calcular los enlaces de datos del programa con su ambiente externo y los enlaces de datos del programa con cada una de sus funciones, procedimientos o rutinas.

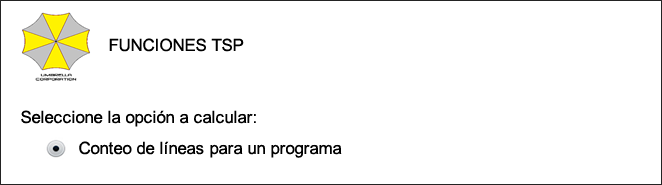
Esta funcionalidad contribuye a medir y analizar relaciones y dependencias de un programa de software con su ambiente externo. Determina el nivel de enlace o acoplamiento de sus componentes para detectar errores de implementación y poder corregirlos a tiempo.

**R4.** Calcular la fuerza de los enlaces de datos internos del programa y cada una de sus funciones, procedimientos o rutinas.

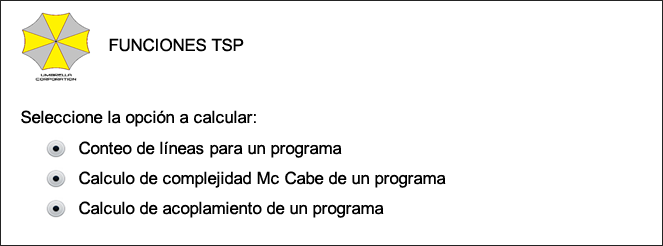
La fuerza de los enlaces de datos internos del programa se calcula contando la unión entre todas las rutinas dentro de dicho programa. El cálculo se basa en saber el número de variables o parámetros pasados entre rutinas o procedimientos internos.

1. INTERFAZ DE USUARIO

Como pantalla principal tenemos un menú con las funcionalidades disponibles para el usuario, el **ciclo 1** cubre el conteo de líneas de un programa determinado por el usuario, para acceder a este debe seleccionar la opción de “Conteo de líneas para un programa”:

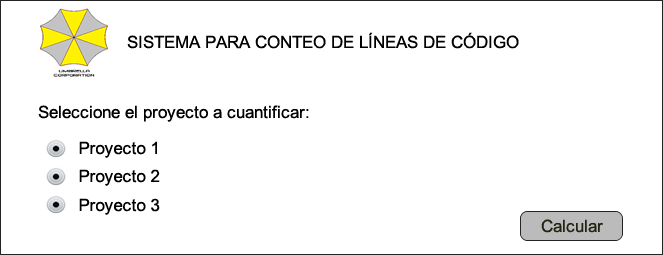


El **ciclo 2** cubre las funcionalidades de calcular la complejidad del programa y el acoplamiento:

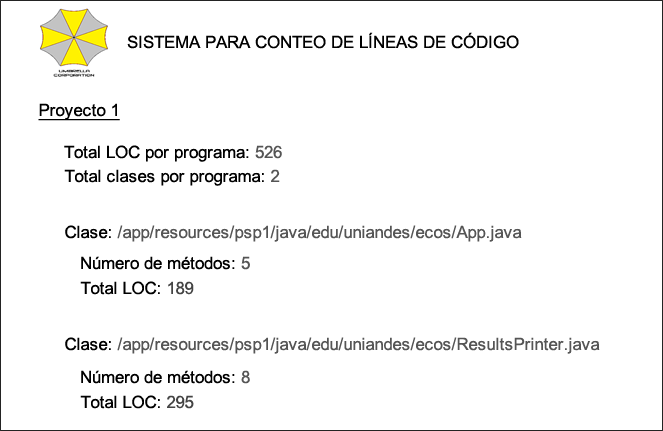


**Conteo de líneas para un programa**

El sistema tendrá precargados tres proyectos de prueba para realizar el conteo de líneas, por lo tanto el usuario selecciona el proyecto sobre el cual desea realizar el cálculo:

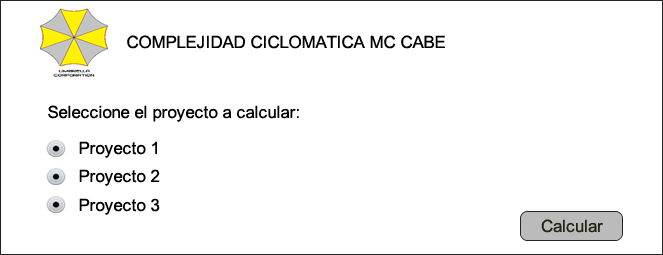


Los resultados se visualizan de la siguiente forma:

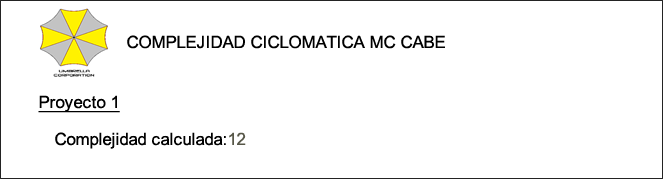


**Calculo de complejidad Mc Cabe**

El sistema tendrá precargados tres proyectos de prueba para realizar el cálculo de complejidad, por lo tanto el usuario selecciona el proyecto sobre el cual desea realizar el cálculo:

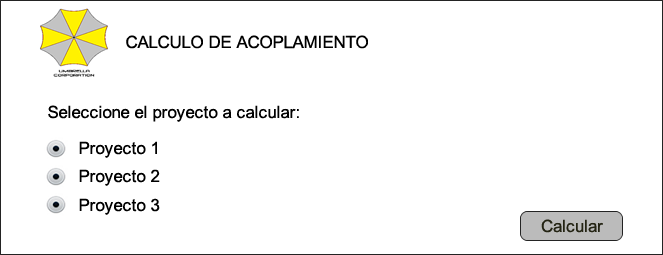


El resultado se visualiza de la siguiente forma:



**Calculo de acoplamiento**

El sistema tendrá precargados tres proyectos de prueba para realizar el cálculo de acoplamiento, por lo tanto el usuario selecciona el proyecto sobre el cual desea realizar el cálculo:

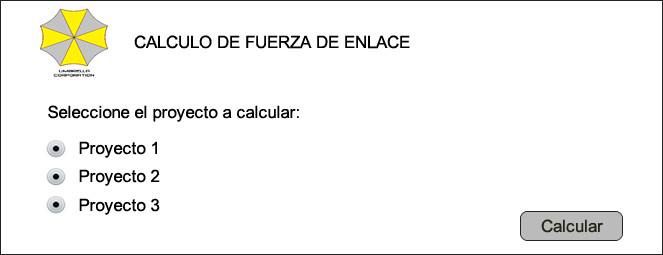


El resultado se visualiza de la siguiente forma:

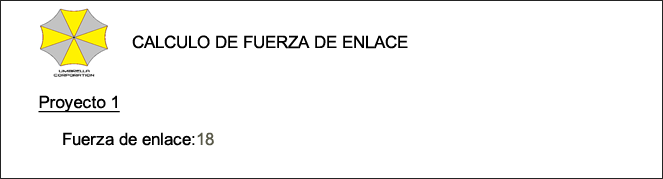


**Calculo de fuerza de enlace**

El sistema tendrá precargados tres proyectos de prueba para realizar el cálculo de la fuerza de enlace, por lo tanto el usuario selecciona el proyecto sobre el cual desea realizar el cálculo:



El resultado se visualiza de la siguiente forma:



1. ATRIBUTOS DE CALIDAD

A continuación se definen los atributos de calidad con los que debe cumplir el sistema:

* **Disponibilidad:** El programadebe brindar alta disponibilidad, 95%
* **Rendimiento:** Este indicador debe brindar una capacidad de respuesta del sistema rápida para ejecutar una acción dentro de un intervalo corto de tiempo dado.
* **Integridad:** El sistema debe asegurar consistencia y coherencia en su diseño.
* **Usabilidad:** El sistema debe cumplir con los requerimientos de los usuarios y los consumidores, debe ser intuitiva, fácil de localizar y globalizar.
* **Reusabilidad:** Las funcionalidades deben estar en capacidad para ser usadas por otras aplicaciones en otros escenarios.
* **Capacidad de ser probado:** Debe tener permitir crear un criterio de pruebas del sistema con el fin de lograr aislar fallas de forma rápida y efectiva.
* **Capacidad de Soporte:** Las funcionalidades deben ser fáciles de usar para cualquier tipo de usuario y a su vez los errores deben poder ser resueltos de forma rápida y efectiva

1. RESTRICCIONES

El siguiente listado corresponde a las restricciones de desarrollo que aplican para todos los ciclos:

* Todos los ejercicios desarrollados deben ser desarrollados en Java
* El aplicativo debe ser desplegado en el ambiente web de Heroku y debe ser posible ejecutarlo desde internet

La codificación se debe regir según el siguiente estándar:

|  |  |
| --- | --- |
| **Purpose** | Guía para desarrollar programas en JAVA |
| **ProgramHeaders** | Comenzar cada clase con un encabezado |
| **HeaderFormat** | /\*  \* Description  \*@author: your name  \*@version:  \*/ |
| **ListingContents** | Provide a summary of the listing contents. |
| **ReuseInstructions** | * Para describir cómo será usado un método, se empleara Javadoc con los tags que tiene la estructura que se ve en el ejemplo. |
| **ReuseExample** | /\*\*  \*Obtener la suma entre dos números enteros  \*@author Javier Virvescas  \*@param a: int  \*@param b: int  \*@return resultado: int  \*@throws e: Exception  \*\*/  publicintsumar(int a, int b)throw Exception{… |
| **Identifiers** | Utilizar nombres descriptivos, evitando abreviaturas e identificadores de una sola letra. |
| **IdentifierExample** | publicvoidcontarLineas(){… // Así está bien  publicvoidcontLin(){… // Así está mal |

|  |  |
| --- | --- |
| **Comments** | * Los comentarios deben proporcionar información clara y precisa de lo que está realizando en el código. * Utilizar “//” únicamente para comentarios de una sola línea. |
| **GoodComment** | //Variable que almacena el número de líneas del código |
| **BadComment** | //Variable que almacena  //un numero |
| **MajorSections** | * Proporcionar información más detallada sobre algún proceso que se está realizando. Debe ser igualmente clara y precisa. * Utilizar /\* … \*/ únicamente para comentarios de más de una línea. |
| **Example** | /\*Recorrer toda la clase java para conocer y determinar cuantas líneas de código se emplearon y cuantos métodos tiene\*/ // Así está bien  /\*Recorrer toda la clase\*/ // Así está mal |
| **BlankSpaces** | * Dejar un espacio en blanco entre cada uno de los bloques de paquete, imports, definición de clase, parámetros y métodos. * No dejar espacios en blanco dentro de la implementación de un método. |
| **Indenting** | * Cada línea de código deben estar alineada con su bloque de ejecución. * El primer corchete ({) debe ir en la misma línea de donde va a empezar el bloque de ejecución y el corchete de cierre debe sola en una línea. |
| **Indenting**  **Example** | for (inti = 0; i< 10; i++){  int j = i;  if (j%2 == 0){  System.out.println(“Es par”);  System.out.println(“Es par”);  }  else  System.out.println(“Esimpar”);  } |
| **Capitalization** | * Clases: La primera letra de cada palabra debe comenzar con mayúsculas seguido de minúsculas. * Variables y métodos: La primera palabra debe empezar con minúscula y si tiene más de una palabra su primera letra debe empezar con mayúscula seguido de minúsculas. * Constantes: Se deben definir en mayúscula y separando las palabras con raya al piso (\_) |
| **CapitalizationExample** | Ejemplo Clases:  publicclassProgramaPrincipal{… // Así está bien  publicclassprograma\_principal{… // Así está mal  Ejemplo Variables y Métodos:  intnumeroLineas; // Así está bien  publicvoidcontarLineas() // Así está bien  floatnumeroconteo; // Así está mal  Ejemplo Constantes:  publicstatic final int NUMERO\_PI = 3.1416 // Así está bien  publicstatic final stringconstante\_No = “NO” // Así está mal |

1. REFERENCIAS Y CONTEXTO

**Desarrollo:**

Para el desarrollo nos basamos en el patrón MVC (Modelo Vista Controlador) como modelo para la construcción del software, separando los datos, la lógica de negocio y la interfaz de usuario. Esto con el fin de desacoplar la vista del modelo y así lograr mayor reusabilidad, menos trabajo y menor riesgo al momento de modificar o agregar código.

* Modelo: Contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
* Vista o interfaz de usuario: Compone la información que se envía al cliente y los mecanismos interacción con éste.
* Controlador: Actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

En cuanto al lenguaje de programación haremos uso de Java por su orientación a objetos ya que es un lenguaje basado en clases y capaz de manejar los conceptos básicos de Búsqueda Dinámica, Herencia y Subtipos. Esto nos ofrece facilidad de diseño, reusabilidad, facilidad de mantenimiento, sistemas más complejos, abstracción, etc.

**Tecnologías:**

Para el desarrollo de las funciones definidas en cada ciclo utilizaremos los siguientes componentes tecnológicos:

**Java:** Lenguaje orientado a objetos desarrollado por Sun Microsystems. Permite el desarrollo de aplicaciones bajo el esquema de Cliente Servidor, y aplicaciones distribuidas.

Es distribuido, multiplataforma, compilado, seguro y posee una arquitectura neutral. Proporciona un conjunto de clases potente y flexible lo hace más manejable desde el punto de vista del programador.

**Netbeans:** Entorno de desarrollo gratuito y de código abierto que permite el uso de un amplio rango de tecnologías de desarrollo para aplicaciones Web. Brinda soporte a múltiples tecnologías y puede instalarse en varios sistemas operativos.

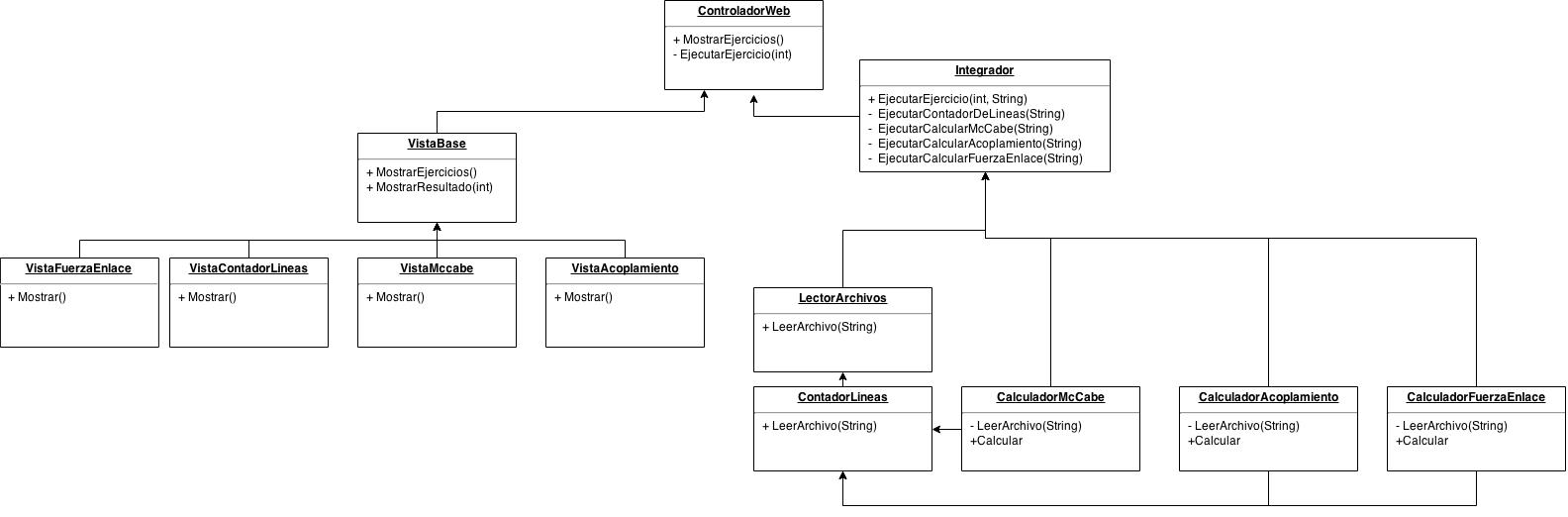
**Heroku:** Plataforma como servicio en la nube con una interfaz avanzada de gestión de recursos y con soporte a múltiples lenguajes de programación.

**Git:** Es un sistema distribuido. No existe un repositorio central. Todas las copias que existen de un proyecto son un branch del proyecto. Cada usuario que tiene una copia puede hacer cambios y cualquier otra operación sin afectar a ningún otro usuario.

1. DISEÑO
   1. Diagrama de clases

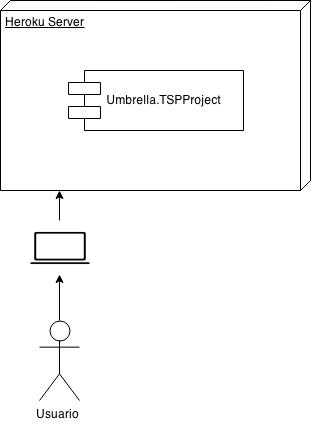
El siguiente diagrama muestra las 6 clases principales utilizadas para desarrollar el ciclo 1 de desarrollo:

* Controlador Web: Clase encargada de comunicarse con las vistas y el integrador para poder mostrar finalmente el contenido.
* Vista Base: Vista que muestra el listado de ejercicios
* VistaContadorLineas: Es la vista encargada de generar el HTML del ejercicio número 1 el cual debe contar las líneas de código un archivo
* VistaMccabe: Vista encargada de generar el HTML que muestra el resultado del cálculo Mccabe
* VistaFuerzaEnlace: Vista encargada de generar el HTHML que muestra el resultado del ejercicio de fuerza de enlace
* VistaAcoplamiento: Vista encargada de generar el HTML que muestra el resultado del cálculo de acoplamiento
* Integrador: Clase que se encarga de agrupar todas las subclases que contienen la lógica de cada una de los ejercicios. Adicionalmente le pasa el modelo al controlador.
* Lector de Archivos: Clase genérica que lee archivos y retorna las líneas del mismo
* Contador Líneas: Clase encargada de ejecutar la lógica del ejercicio 1 en el cual se deben contar las líneas de código un archivo.
* CalculadorMccabe: Clase encargada de realizar los cálculos Mccabe, se conecta a la clase que lee archivos para reutilizar funcionalidad
* CalculadorAcoplamiento: Clase encargada de realizar el cálculo de acoplamiento, utiliza la clase de lectura de archivos.
* CalculadorFuerzaEnlace: Clase encargada de realizar los cálculos relacionados con el ejercicio 4 de la fuerza de enlace.



* 1. Diagrama de despliegue

Ya que el sistema no contiene suficiente interacción con sistemas externos o acceso a Base de datos, solo se contará con un servidor web el cual será el encargado de almacenar la aplicación y exponerla en internet. Los usuarios podrán consumir el aplicativo desde sus dispositivos por medio de un navegador web.

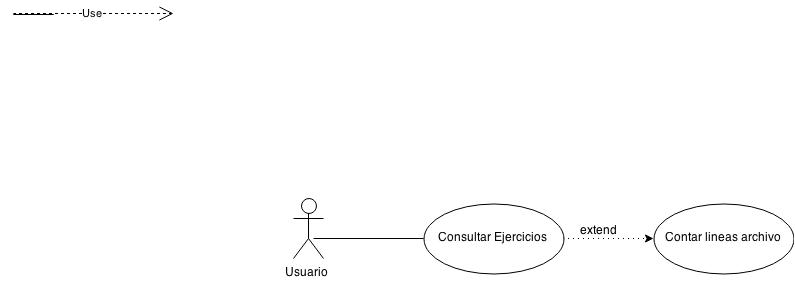


* 1. Diagrama y documentación de casos de uso
* Consultar Ejercicios (CU-01)



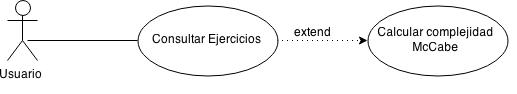
|  |  |  |
| --- | --- | --- |
| **RF- 01** | **Consultar ejercicios** | |
| **Versión** | 1.0 | |
| **Autores** | Gabriel Castillo, Erica Prado | |
| **Descripción** | El sistema permitirá a los usuarios consultar los ejercicios disponibles para probar | |
| **Precondición** | El usuario debe acceder al sistema | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El usuario ingresa a la URL donde el aplicativo está publicado |
| 2 | El sistema muestra los ejercicios disponibles para ejecutar:   * Contar líneas de un programa * Calcular Mc Cabe * Calcular acoplamiento |
| **Postcondición** |  | |
| Excepciones | **Paso** | **Acción** |
| 1 | No hay |

* Contar Líneas Archivo (CU-02): El usuario selecciona el ejercicio que contará las líneas de código de un archivo que el mismo cargue.



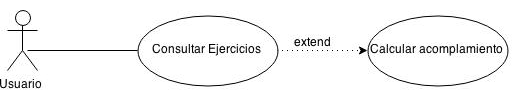
|  |  |  |
| --- | --- | --- |
| **RF- 02** | **Contar líneas archivo** | |
| **Versión** | 1.0 | |
| **Autores** | Gabriel Castillo, Erica Prado | |
| **Descripción** | El sistema permitirá contar las líneas de código un archivo que se ha cargado previamente en el directorio de recursos del aplicativo | |
| **Precondición** | RF-01 | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El usuario selecciona el ejercicio de conteo de líneas. Esta opción es mostrada en RF-01 |
| 2 | El sistema muestra el cálculo de las líneas de código encontradas en los archivos de la carpeta de recursos |
| **Postcondición** |  | |
| Excepciones | **Paso** | **Acción** |
| 1 | Si no hay archivos en la carpeta el sistema debe comunicarlo |

* Calcular complejidad Mc Cabe (CU-03):



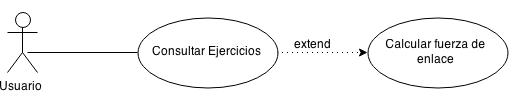
|  |  |  |
| --- | --- | --- |
| **RF- 03** | **Calcular complejidad McCabe** | |
| **Versión** | 1.0 | |
| **Autores** | Gabriel Castillo, Erica Prado | |
| **Descripción** | El sistema encargará de calcular la complejidad de un programa basado en las formulas McCabe proveídas por el libro | |
| **Precondición** | RF-01 | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El usuario selecciona el ejercicio de cálculo Mc Cabe. Esta opción es mostrada en RF-01 |
| 2 | El sistema muestra el listado de proyectos disponibles para calcular |
| 3 | El usuario selecciona el proyecto que desea calcular |
| 4 | El sistema realiza los cálculos necesarios y debe mostrar el número que surge como resultado de estas operaciones |
| **Postcondición** |  | |
| Excepciones | **Paso** | **Acción** |
| 1 | Si no hay archivos en la carpeta el sistema debe comunicarlo |

* Calcular acoplamiento (CU-04):



|  |  |  |
| --- | --- | --- |
| **RF- 04** | **Calcular acomplamiento** | |
| **Versión** | 1.0 | |
| **Autores** | Gabriel Castillo, Erica Prado | |
| **Descripción** | El sistema realiza el cálculo de acoplamiento, el cual consiste en validar el nivel de enlace de los componentes desarrollados y detectar posibles errores de implementación | |
| **Precondición** | RF-01 | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El usuario selecciona el ejercicio de cálculo de acomplamiento. Esta opción es mostrada en RF-01 |
| 2 | El sistema muestra el listado de proyectos disponibles para calcular |
| 3 | El usuario selecciona el proyecto que desea calcular |
| 4 | El sistema realiza los cálculos necesarios y debe mostrar el número que surge como resultado de estas operaciones |
| **Postcondición** |  | |
| Excepciones | **Paso** | **Acción** |
| 1 | Si no hay archivos en la carpeta el sistema debe comunicarlo |

* Calcular fuerza de enlace (CU-05):



|  |  |  |
| --- | --- | --- |
| **RF- 05** | **Calcular fuerza de enlace** | |
| **Versión** | 1.0 | |
| **Autores** | Gabriel Castillo, Erica Prado | |
| **Descripción** | El sistema realiza el cálculo de fuerza de enlace, el cual consiste en contar la unión entre todas las rutinas dentro de un programa. El cálculo se basa en saber el número de variables o parámetros pasados entre rutinas o procedimientos internos. | |
| **Precondición** | RF-01 | |
| **Secuencia**  **Normal** | **Paso** | **Acción** |
| 1 | El usuario selecciona el ejercicio de calcular la fuerza de enlace. Esta opción es mostrada en RF-01 |
| 2 | El sistema muestra el listado de proyectos disponibles para calcular |
| 3 | El usuario selecciona el proyecto que desea calcular |
| 4 | El sistema realiza los cálculos necesarios y debe mostrar el número que surge como resultado de estas operaciones |
| **Postcondición** |  | |
| Excepciones | **Paso** | **Acción** |
| 1 | Si no hay archivos en la carpeta el sistema debe comunicarlo |

* 1. Diagrama de componentes

El proyecto cuenta con cuatro componentes principales:

* Controlador: Encargado de generar la comunicación entre la vista y el modelo.
* Vista: Son las encargadas de generar el HTML que se mostrará en los navegadores de los usuarios.
* Integrador: Encargado de ejecutar toda la lógica de negocio referente a cada uno de los ejercicios
* Utilidades: Clases que se encargan de hacer tareas generales del aplicativo como leer archivos físicos.

