INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

“Sistema Web Para La Evaluación De Métricas De Software”

TRABAJO TERMINAL

2021-B012

PRESENTAN

Cuevas Olvera Ian Axel

Ortega Estrada Ismael

DIRECTORES

Dra. Lorena Chavarría Báez

M. en C. Nancy Ocotitla Rojas



Ciudad de México Mes 2022

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

No. de TT: 2021-B012 dd de Mes de 2022

DOCUMENTO TÉCNICO

“Sistema Web Para La Evaluación De Métricas De Software”

PRESENTAN

Cuevas Olvera Ian Axel 1

Ortega Estrada Ismael 2

DIRECTORES

Dra. Lorena Chavarría Báez

M. en C. Nancy Ocotitla Rojas

RESUMEN

Las métricas de software son una medida cualitativa de atributos del software, están relacionadas con la calidad de este ya que cuantifican una serie de propiedades y atributos en los programas. Aunque las ventajas de utilizar métricas de software son significativas, a saber: selección de mejores alternativas, ahorro de recursos y mantenibilidad del proyecto, no se aplican con la frecuencia que deberían debido a distintos factores, entre los que destacan el tiempo y el esfuerzo que requieren y la falta de conocimiento sobre las mismas. Por lo tanto, la propuesta aquí planteada tiene como objetivo realizar un sistema capaz de examinar programas, y desplegar un análisis de sus características, utilizando como base algunas métricas de software para código fuente estructurado y orientado a objetos.

Palabras clave – Departamento de Ingeniería en Sistemas Computacionales, análisis, código, métricas, programación estructurada, programación orientada a objetos.

*1* [*icuevas1801@alumno.ipn.mx*](mailto:icuevas1801@alumno.ipn.mx)

*2* [*iortega1800@alumno.ipn.mx*](mailto:iortega1800@alumno.ipn.mx)

Índice

[CAPÍTULO I INTRODUCCIÓN 6](#_Toc98359983)

[1.1 Planteamiento del problema 7](#_Toc98359984)

[1.2 Objetivo 7](#_Toc98359985)

[1.2.1 Objetivos específicos 8](#_Toc98359986)

[1.3 Justificación 8](#_Toc98359987)

[1.4 Estado del arte 9](#_Toc98359988)

[1.5 Descripción del documento 10](#_Toc98359989)

[CAPÍTULO II PARADIGMAS DE PROGRAMACIÓN 11](#_Toc98359990)

[2.1 Paradigmas de programación 12](#_Toc98359991)

[2.1.1 ¿Qué es un lenguaje de programación? 12](#_Toc98359992)

[2.1.2 ¿Qué es un programa? 12](#_Toc98359993)

[2.1.3 Compilador 13](#_Toc98359994)

[2.1.4 ¿Qué es la programación estructurada? 13](#_Toc98359995)

[2.1.5 ¿Qué es la programación orientada a objetos? 14](#_Toc98359996)

[2.2 Ingeniería de software 14](#_Toc98359997)

[2.2.1 ¿Qué es la ingeniería de software? 14](#_Toc98359998)

[2.2.2 Objetivo de la ingeniería de software 14](#_Toc98359999)

[2.3 Normas y estándares de la calidad en la ingeniería de software 14](#_Toc98360000)

[2.3.1 ¿Qué es la calidad? 14](#_Toc98360001)

[2.3.2 ¿Qué es una norma en la ingeniería de software? 15](#_Toc98360002)

[2.3.4 Normas ISO 15](#_Toc98360003)

[2.4 Sistema Web 15](#_Toc98360004)

[2.4.1 ¿Qué es un sistema web? 15](#_Toc98360005)

[2.4.2 Características de un sistema web 16](#_Toc98360006)

[2.4.3 Ventajas y Desventajas de un sistema web 16](#_Toc98360007)

[3.1 Concepto de medida 17](#_Toc98360008)

[3.2 ¿Qué son las métricas de Software? 17](#_Toc98360009)

[3.3 ¿Qué es un Indicador? 17](#_Toc98360010)

[3.4 ¿Para qué se utilizan las métricas de software? 17](#_Toc98360011)

[3.5 ¿Cuáles son las características de las métricas de Software? 17](#_Toc98360012)

[3.6 ¿Cuáles son las ventajas de implementar métricas de software? 17](#_Toc98360013)

[3.7 Tipos de Métricas de Software 17](#_Toc98360014)

[3.7.1 De acoplamiento 18](#_Toc98360015)

[3.7.2 De cohesión 18](#_Toc98360016)

[3.7.3 De complejidad 19](#_Toc98360017)

[3.7.4 De herencia 20](#_Toc98360018)

[3.7.5 De encapsulamiento 21](#_Toc98360019)

[3.7.6 De tamaño 22](#_Toc98360020)

[CAPÍTULO IV METODOLOGÍA 23](#_Toc98360021)

[4.1 ¿Qué es una metodología? 24](#_Toc98360022)

[4.2 Tipos de metodología 24](#_Toc98360023)

[4.2.1 Metodologías clásicas 24](#_Toc98360024)

[4.2.1.1 Modelo en cascada 24](#_Toc98360025)

[4.2.2 Metodologías ágiles 25](#_Toc98360026)

[CAPÍTULO V ESTUDIO DE FACTIBILIDAD 27](#_Toc98360027)

[5.1 Factibilidad Técnica 28](#_Toc98360028)

[5.2 Factibilidad Económica 28](#_Toc98360029)

[5.3 Facilidad Operacional 28](#_Toc98360030)

[5.4 Interfaz gráfica de usuario 28](#_Toc98360031)

[5.4.1 Menús 28](#_Toc98360032)

[5.4.2 Formulario 28](#_Toc98360033)

[5.4.3 Logotipo 28](#_Toc98360034)

[5.4.4 Colores de diseño 28](#_Toc98360035)

[5.4.5 Pantallas del sistema 28](#_Toc98360036)

[5.5 Archivos 28](#_Toc98360037)

[5.5.1 Diseño de la base de datos (Relacional) 28](#_Toc98360038)

[5.6 Procesamiento de captura 28](#_Toc98360039)

[CAPÍTULO VI ANÁLISIS 29](#_Toc98360040)

[6.1 Análisis de requerimientos 30](#_Toc98360044)

[6.1.1 Requerimientos funcionales 31](#_Toc98360045)

[6.1.2 Requerimientos no funcionales 32](#_Toc98360046)

[6.2 Modelado UML 33](#_Toc98360047)

[6.2.1 Diagrama de casos de uso 34](#_Toc98360048)

[6.2.2 Diagrama de clases 35](#_Toc98360049)

[6.2.3 Diagrama de estados 35](#_Toc98360050)

[CAPÍTULO VII DISEÑO 39](#_Toc98360051)

[7.1 Entradas 40](#_Toc98360052)

[7.2 Salidas 40](#_Toc98360053)

[7.3 Arquitectura del sistema 41](#_Toc98360054)

[7.4 Interfaz gráfica de usuario 41](#_Toc98360055)

[7.4.1 Menús 41](#_Toc98360056)

[7.4.2 Formularios 41](#_Toc98360057)

[7.4.3 Logotipo 41](#_Toc98360058)

[7.4.4 Colores en el diseño 41](#_Toc98360059)

[7.4.5 Pantallas del sistema 41](#_Toc98360060)

[7.5 Archivos 41](#_Toc98360061)

[7.5.1 Diseño de la base de datos 41](#_Toc98360062)

[7.6 Procesamiento de captura 41](#_Toc98360063)

[REFERENCIAS 42](#_Toc98360064)

CAPÍTULO I  
INTRODUCCIÓN

## 1.1 Planteamiento del problema

La medición es una herramienta fundamental en cualquier tipo de ingeniería y la ingeniería de software no está exenta de ello. Para el ingeniero de software es indispensable poder evaluar el código que produce con el propósito de tomar mejores decisiones y aportar mayor valor al funcionamiento y desempeño del programa.

Las métricas de software se definen como una medida cualitativa de la forma en que un programa o sistema posee determinados atributos. Si se desea tener calidad en el software es esencial recurrir a ellas para que sirvan de guía en su evaluación, mejoramiento y clasificación [1] y así lograr dicho objetivo.

De manera general, la finalidad de las métricas de software es proporcionar un conjunto de indicadores que describan, detalladamente, cómo está escrito y cómo funciona un programa. Dentro de sus características, se puede decir que son [2]:

* Cuantificables, ya que están basadas en hechos, no en opiniones.
* Independientes, porque no pueden ser alteradas por las personas que las apliquen.
* Explicables, debido a que ellas mismas y su uso deben estar documentadas.
* Precisas, puesto que no se debe perder información.

La importancia de las métricas del software radica en que, a través de ellas, los desarrolladores pueden estimar los recursos necesarios en la realización de programas, el tiempo aproximado de entrega, el costo que tendrá el software, el mantenimiento de este y, lo más relevante, determinar si el programa es apto para una etapa de producción [2].

Aunque el desarrollo de programas se puede beneficiar de la medición, como se mencionó anteriormente, esta práctica ha perdido relevancia en los últimos años debido a que la mayoría de los desarrolladores, incluso desde que son estudiantes de ingeniería, no la toman en cuenta como una herramienta fundamental para el desarrollo de software. Un estudio interno realizado a 60 participantes de la industria del software en el año 2014 en Sri Lanka [5], país localizado al sur de Asia, reveló que las personas que desarrollan proyectos justifican que no hacen uso de las métricas de software debido a que es tedioso, no tienen los conocimientos necesarios o requieren mucho tiempo y esfuerzo. Según el estudio, el 62% de todos los participantes han realizado proyectos de software sin hacer uso de métricas; sin embargo, el 76% conoce las ventajas de aplicarlas durante el desarrollo de estos.

Esto indica que, si bien un porcentaje alto de los participantes no han utilizado métricas de software en sus proyectos, un porcentaje aún mayor es consciente de las ventajas que implica su uso, por lo cual es valioso hacer énfasis en ponerlas en práctica.

## 1.2 Objetivo

A continuación, se enuncia el objetivo general de este Trabajo Terminal.

Implementar un sistema de software para evaluar programas orientados a objetos y estructurados, escritos en los lenguajes de programación Java y C, respectivamente, usando métricas de software para código fuente.

### 1.2.1 Objetivos específicos

Una vez establecido el objetivo general, es necesario fragmentarlo en objetivos específicos para tener claro los puntos que se deben desarrollar para alcanzarlo. A continuación, se listan los objetivos específicos de este trabajo.

* Implementar un subsistema gestor de usuarios.
* Desarrollar un subsistema gestor de archivos.
* Implementar un subsistema de análisis de métricas de programación orientada a objetos.
* Implementar un subsistema de análisis de métricas de programación estructurada.
* Desarrollar un subsistema de despliegue y comparación de resultados.

## 1.3 Justificación

El desarrollo de software a través de la aplicación de métricas es una práctica muy importante en el ámbito de la ingeniería, tan es así que existe una diferencia abismal entre un software que es desarrollado haciendo uso de dichas métricas y uno que no lo es. En primera instancia, un software que está desarrollado aplicando métricas permite tener un mejor panorama del programa que se pretende implementar. Su uso permite tener un estimado de los recursos que serán necesarios, debido a que, al tener parámetros de las características del código, se puede aproximar el costo y la eficiencia [12]. El uso de las métricas hace que el desarrollo de software se realice de manera más organizada ya que permite hacer un cálculo del tiempo en el que se planea culminar un proyecto, así como también, de la manera en que se reparten las tareas entre todos los miembros de un equipo de trabajo. Otro aspecto es la mantenibilidad, que hace referencia a la capacidad de que un programa pueda ser modificado, y que, gracias a las métricas, se ve beneficiado al tener un código limpio y ordenado. Las métricas de software deberían ser aplicadas a todos los proyectos, pues estas pueden ser la clave para la realización de un software que sea apto para salir al mercado y tener éxito. Sin embargo, en la actualidad, las métricas de software no son aplicadas a la gran mayoría de proyectos ya que existen falsas creencias alrededor de su uso, tales como asumir que puede resultar tedioso al representar una tarea extra cuando se desarrolla software, lo que trae consigo un mayor tiempo de desarrollo. Otra de las razones surge de la falta de conocimientos acerca de las buenas prácticas de programación.

Con este proyecto se pretende que los usuarios puedan tener una visión más clara de las características en sus programas escritos en lenguaje C o Java, mediante una evaluación de métricas estructuradas y orientadas a objetos. La intención es incentivar y recalcar las ventajas que tienen en el desarrollo de programas para que la comunidad de programadores, principalmente estudiantes de ingeniería, las tengan presentes cuando desarrollen proyectos, logrando así que el producto de software generado sea costeable, mantenible y, sobre todo, de calidad.

Para la realización de este Trabajo Terminal se pondrán en práctica los conocimientos adquiridos en las distintas unidades de aprendizaje cursadas en la carrera de Ingeniería en Sistemas Computacionales, entre las que destacan: Algoritmia y Programación Estructurada, Teoría Computacional, Análisis y Diseño Orientado a Objetos, Compiladores y Análisis de Algoritmos. Finalmente, se hará uso de los conocimientos de Ingeniería de Software para la organización del proyecto.

## 1.4 Estado del arte

En el mercado actual, existen pocas herramientas que realizan un análisis de métricas de programación estructurada y orientada a objetos, además, estos sistemas no son aptos para todos los usuarios debido a que sólo despliegan valores y no su significado, por lo cual sólo serán útiles para los usuarios que tengan conocimientos previos de las métricas y su interpretación. Por lo tanto, este es el problema que se abordará en este trabajo: la falta de herramientas computacionales para evaluar métricas de software para código fuente estructurado y orientado a objetos.

En la Tabla 1 se presentan distintas herramientas que fueron implementadas con el fin de evaluar métricas de software.

*Tabla 1 Resumen de productos similares*

|  |  |  |
| --- | --- | --- |
| **SOFTWARE** | **CARACTERÍSTCAS** | **PRECIO EN EL MERCADO** |
| SONARQUBE [6] | Herramienta de gestión de la calidad del código fuente. Permite recopilar, analizar y visualizar métricas del código fuente. Está formado por *Programming Mistake Detector* (PMD), *Check Style*, *Findbugs*, *Clover* y Cobertura. Se usa principalmente con Java, pero da soporte a otros lenguajes. Encuentra vulnerabilidades de seguridad de código, está actualizado y cuenta con soporte a diferentes lenguajes. No ofrece retroalimentación de los  resultados, no cuenta con soporte para programación estructurada. | Software gratuito |
| GOOGLE CODEPRO ANALYTIX [7] | Es una herramienta de gestión de la calidad del software. Ofrece un entorno para evaluación de  código, métricas, análisis de dependencias, cobertura de código, generación de pruebas unitarias, etc. | Ya no ofrece soporte (2010) |
| METRICS PLUGIN [8] | Es un *plugin* para el Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés *Integrated Development Environment*) de Eclipse. Es necesario tener instalada la versión de Eclipse 3.1 (2004). El *plugin* analiza métricas de software orientado a objetos. Sin embargo, sólo ofrece los resultados de las métricas y no da una explicación de estos. La  ventaja que tiene este *plugin* es que ofrece un diagrama de dependencias. | Ya no ofrece soporte (2004) |
| JARCHITECT [9] | Es una herramienta que brinda un análisis de métricas de software orientado a objetos para el lenguaje  JAVA. Ofrece una interpretación de los resultados en forma de gráficos, pero es una herramienta de paga. | Software de paga |
| JAVA MEASUREMENT TOOL (JMT) [10] | JMT es una herramienta que analiza las clases Java y  las relaciones entre ellas. Utiliza métricas de software orientadas a objetos. | Ya no está disponible (2002) |
| JAVA NON COMMENTING SOURCE STATEMENTS (JavaNCSS) [11] | Es una herramienta que permite realizar mediciones sobre el código fuente Java, obteniendo los valores de dichas mediciones agrupados a nivel global, clase y función. Utiliza métricas de software orientado a objetos. Su documentación aún se encuentra disponible, sin embargo, los enlaces para su descarga  no funcionan. | Ya no está disponible (2019) |

|  |  |  |
| --- | --- | --- |
| **Solución propuesta** | Sistema web para la evaluación de métricas de código fuente en programas que siguen el paradigma estructurado u orientado a objetos. Otorgará los resultados e interpretación de cada métrica. Se encontrará al alcance de cualquier usuario y, principalmente, cualquier estudiante de ingeniería en la rama de la computación debido a que será un  sistema web gratuito. | Software gratuito |

Como puede observarse, la gran mayoría de herramientas presentadas, si bien son gratuitas, no ofrecen una interpretación de las distintas métricas aplicadas. Debido a esto, el software propuesto proporcionará una descripción de la medición en los programas que se analicen con la finalidad de que se tenga un mayor entendimiento de las características que describe cada una de ellas. Lo anterior tiene como propósito incentivar el uso de buenas prácticas en la comunidad de desarrolladores de software.

## 1.5 Descripción del documento

Este trabajo se desarrolla en X capítulos, comenzando con el capítulo 1, en el cual se establece el planteamiento del problema, justificación, objetivo general, objetivos específicos y el estado del arte.

En el capítulo 2 se enuncian todos los conceptos indispensables en torno a los paradigmas de programación: programas, compiladores, tipos de programación, temas de la ingeniería de software como la calidad, las normas, los estándares, etc. A continuación, en el capítulo 3, se profundiza en el tema de las métricas de software, objeto de estudio principal de este trabajo, se menciona su definición, propósito, características, ventajas y desventajas, así como su clasificación, funcionamiento y significado.

# CAPÍTULO II PARADIGMAS DE PROGRAMACIÓN

## 2.1 Paradigmas de programación

Un paradigma de programación [3] “provee (y determina) la visión y métodos que un programador utiliza en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de los problemas”. Pérez y López (2007).

Un lenguaje de programación se guía en uno o más paradigmas, por ejemplo: Lenguaje C está basado en el paradigma estructurado. El lenguaje JAVA está basado en el paradigma orientado a objetos.

URL:

<http://fcaenlinea.unam.mx/apuntes/interiores/docs/20182/informatica/6/LI_1656_21117_A_Informatica_VI_Plan2016.pdf>

<https://labsys.frc.utn.edu.ar/ppr-2009/Unidad%20I%20-%20POO%20Avanzada/Apunte/Unidad%20I%20-%20POO%20Avanzada.pdf>

### 2.1.1 ¿Qué es un lenguaje de programación?

Un lenguaje de programación es la forma mediante la cual los humanos le dan instrucciones específicas a una computadora para poder generar un programa, es el medio de comunicación.

Un lenguaje de programación es un conjunto de símbolos y reglas semánticas que definen un programa, existen lenguajes de programación denominados de bajo nivel y lenguajes de programación de alto nivel.

Un lenguaje de bajo nivel contiene instrucciones que una computadora puede leer directamente dado que sus símbolos son 1 y 0, es decir, modo binario. También es conocido como lenguaje máquina.

Un lenguaje de alto nivel contine instrucciones que son más fáciles de entender por un programador dado que sus instrucciones están más apegadas a la forma en que los humanos nos comunicamos. Ejemplos de los lenguajes de alto nivel son: C, JAVA, Python, Pascal.

URL:

<https://www.infor.uva.es/~cevp/FI_I/fichs_pdf_teo/FI_I_Tema1.pdf>

<https://cursos.aiu.edu/Programacion%20de%20Computadoras/PDF/Tema%201.pdf>

### 2.1.2 ¿Qué es un programa?

Un programa informático se define como “un conjunto de instrucciones que, una vez ejecutadas, realizan una o varias tareas en una computadora”, el programa se debe encontrar almacenado en memoria. Si en la computadora se encuentra un conjunto de programas a esto se le llama software.

Un programa se escribe mediante instrucciones en un lenguaje de programación.

URL:

<https://www.editorialpatria.com.mx/pdffiles/9786074384154.pdf>

<https://www.mheducation.es/bcv/guide/capitulo/844814645X.pdf>

#### 2.1.2.1 Propiedades y atributos de un programa

Lorem ipsum

#### 2.1.2.2 Diferencias entre pseudocódigo y código

El pseudocódigo se puede visualizar como un borrador para la creación del código, debido a que el pseudocódigo no está vinculado a ningún lenguaje de programación y se puede interpretar para el que se desee.

Un código es la representación de instrucciones que una computadora va a recibir e interpretar para la realización de una tarea, el código debe de estar escrito en un lenguaje de programación.

Por lo tanto, la diferencia entre un pseudocódigo y código es que el pseudocódigo esta hecho para que sea fácil de entender para el ser humano y una computadora no lo puede entender, mientras que el código esta vinculado a un lenguaje de programación y la computadora puede interpretar las instrucciones.

URL:

<https://answers.mx/diferencia-entre-algoritmo-pseudocodigo-y-codigo.html>

<http://www.algoritmos.cedetec.cl/images/pdf/Pseudocodigo.pdf>

### 2.1.3 Compilador

#### 2.1.3.1 ¿Qué es un compilador?

Un compilador es “un programa que lee un programa escrito en lenguaje fuente, y lo traduce a un programa equivalente en lenguaje objeto”. (Libro del Dragón)

Para la compilación se realizan dos actividades: análisis y síntesis.

En el análisis se divide al programa fuente en sus elementos componentes y crea una representación intermedia del programa fuente.

En la síntesis se construye el programa objeto deseado a partir de la representación intermedia.

URL:

<https://www.untumbes.edu.pe/vcs/biblioteca/document/varioslibros/0623.%20Compiladores,%20principios,%20t%C3%A9cnicas%20y%20herramientas.pdf>

<http://www.lcc.uma.es/~galvez/ftp/libros/Compiladores.pdf>

#### 2.1.3.1 Errores de compilación

En un compilador se pueden encontrar errores semánticos, el encargado de buscar estos errores es el analizador semántico, estos errores se producen desde el momento en que se está escribiendo el código, ya que puede ser omitir algún símbolo de puntuación, no respetar alguna regla de sintaxis, escribir mal las instrucciones del lenguaje, etc.

URL:

<http://www.frlp.utn.edu.ar/materias/algoritmos/errores2010.pdf>

<https://www.untumbes.edu.pe/vcs/biblioteca/document/varioslibros/0623.%20Compiladores,%20principios,%20t%C3%A9cnicas%20y%20herramientas.pdf>

### 2.1.4 ¿Qué es la programación estructurada?

La programación estructurada fue propuesta en 1960 por Böhm y Jacopini, con el motivo de mejorar la calidad, entendimiento y tiempo de programación. Böhm y Jacopini demostraron que todo programa puede escribirse utilizando únicamente tres instrucciones de control: Secuencia, Instrucción condicional y Bucles.

Lenguaje C es conocido como un procedimental, es decir, cada una de las instrucciones le indican al compilador que tiene que hacer una tarea, estas instrucciones se van ejecutando una a una, y no simultáneamente, cada instrucción para ser ejecutada tiene que esperar a que la anterior termine.

URL:

<https://www.mheducation.es/bcv/guide/capitulo/844814645X.pdf>

<https://www.ecured.cu/Programacion_estructurada>

### 2.1.5 ¿Qué es la programación orientada a objetos?

La programación orientada objetos surge a partir de problemas de la vida real que debemos plasmar en código. A partir de esto los problemas los debemos ver en forma de Objetos. La programación orientada a objetos consta de cuatro elementos: clases, propiedades, métodos y objetos. Se puede decir que la programación orientada se sustenta en el encapsulamiento, la abstracción, la herencia y el polimorfismo.

Un objeto se crea a partir de un molde llamado clase, cada objeto es la instancia de una clase. Un objeto tiene datos conocidos como atributos y funcionalidades conocidas como métodos.

El lenguaje JAVA es conocido por utilizar el paradigma orientado a objetos.

URL:

<https://kataix.umag.cl/~ruribe/Utilidades/Introduccion%20a%20la%20Programacion%20Orientada%20a%20Objetos.pdf>

<https://platzi.com/cursos/oop/>

## 2.2 Ingeniería de software

### 2.2.1 ¿Qué es la ingeniería de software?

La ingeniería de software es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar software confiable y eficaz. Consta del desarrollo, operación y mantenimiento del software con un enfoque sistemático, disciplinado y cuantificable, así como también del estudio de estos enfoques.

URL:

<http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF> pp. 43

### 2.2.2 Objetivo de la ingeniería de software

Su principal objetivo es el desarrollo y producción del software de alta calidad. Con el fin de lograr este objetivo, es primordial aplicar métodos y herramientas efectivos en el contexto de desarrollo de software.

URL:

<http://www.usfx.bo/nueva/vicerrectorado/citas/TECNOLOGICAS_20/Ingenieria%20Sistemas/37.pdf> pp.53

## 2.3 Normas y estándares de la calidad en la ingeniería de software

### 2.3.1 ¿Qué es la calidad?

La calidad del software es la necesidad de contar con parámetros que permitan establecer estándares mínimos que un producto de este tipo debe alcanzar. En el software, la gran mayoría de estos parámetros no se pueden medir de manera cuantitativa, por lo que se establecen de manera cualitativa.

La calidad en la ingeniería de software se puede definir en términos de seis características básicas (ISO-9126), las cuales son la funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad que pueda tener un software, y a su vez, cada una de estas características tiene un conjunto de subcaracterísticas que permiten profundizar en la evaluación de la calidad.

URL:

[https://www.nacionmulticultural.unam.mx/empresasindigenas/docs/2094.pdf pp. 1](https://www.nacionmulticultural.unam.mx/empresasindigenas/docs/2094.pdf%20pp.%201)

### 2.3.2 ¿Qué es una norma en la ingeniería de software?

Las normas en la ingeniería son documentos creados por instituciones reconocidas y que pueden ser aplicados a un proyecto en desarrollo, estas contienen especificaciones técnicas basadas en resultados del desarrollo tecnológico. Su objetivo es garantizar la calidad y seguridad del producto.

Algunas de las instituciones normalizadoras son:

* Instituto de Ingenieros Eléctricos y Electrónicos (IEEE)
* Organización Internacional de Estandarización (ISO)
* Comité Europeo de Normalización (CEN)

URL:

<https://d1wqtxts1xzle7.cloudfront.net/50812180/Establishing_software_product_quality_re20161210-6766-1vbimav-with-cover-page-v2.pdf?Expires=1645724181&Signature=JIvPC6YZECdd3Tz~pmKtNTvUmzrTu4PBjo~Iu1lcN~PRRKYwBnqdluUi0y50ktb3MQ33Ocdl2R4EakhjkYe4SkNsyzXUR9bHnOo5WESHuTavNEE78XlbioQeIFcAZAae8ay3IsrGMzRrqTueluLW20D9yCrYRjuQZFLEAsYgOOHwYdOTRQ59WOgKvN0ZgELXSoR1eKq2TPfwrYLr4Cqk-U9J1zyDfBZEQLNJ-a1C0MPGoMsusBnzlWuZOiGfjO82CXii3kHXwEHuhNsOeBzqOaz3FBVX9KpArq5-CeeHYclu5t67ZHBKorZSWE0sH6nqR9wwcqNOfLhdBRNDu~th-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>

### 2.3.4 Normas ISO

La Organización Internacional de Estandarización es una organización independiente y no gubernamental que aplica en una numerosa cantidad de países, con más de 20 mil normas internacionales que abarcan casi todos los aspectos de la tecnología y la manufactura. Estas normas cubren los siguientes rubros:

* Calidad de los productos
* Impacto ambiental
* Salud
* Energía
* Alimentación
* Seguridad de la información tecnológica

URL:

<https://www.iso.org/standards.html>

## 2.4 Sistema Web

### 2.4.1 ¿Qué es un sistema web?

Se define un sistema web aquellos sistemas a aquellos programas de software que pueden utilizarse accediendo a un servidor web a través de Internet o de una intranet mediante un navegador, es decir, un sistema web no necesita estar instalado en la computadora del cliente, sino que está cargado en un servidor en otro equipo y que se accede al sistema por medio de la red.

Los sistemas web son muy utilizados en la actualidad, debido a que toda computadora cuenta con un navegador web y el sistema de esta forma se vuelve independiente al sistema operativo.

### 2.4.2 Características de un sistema web

Un sistema web, consta como anteriormente se mencionó de un cliente y un servidor, el cliente tiene que enviar una petición al servidor del sistema, el servidor del sistema devuelve una interfaz con la información solicitada.

El cliente puede acceder al sistema empleando un navegador web y por medio de internet, esto hace que se pueda acceder al sistema desde cualquier parte del mundo.

De acuerdo con la lógica de programación utilizada en el sistema, pueden existir miles de usuarios accediendo al sistema instalado en el servidor, esto hace que, si se desea implementar una actualización en el sistema, todos los usuarios la vean inmediatamente.

### 2.4.3 Ventajas y Desventajas de un sistema web

Ventajas

* Ahorra tiempo
* Acceso inmediato
* Usuarios concurrentes
* Actualizaciones inmediatas
* Multiplataforma
* Portables
* Centralización de datos

Desventajas

* Se necesita pagar un servidor.
* Se necesita

## 3.1 Concepto de medida

Una medida es el resultado de observar con atención una cosa o a una persona con el fin de establecer semejanzas o diferencias con otra, tomando como referencia a la primera y con el fin de averiguar cuántas veces la segunda está contenida en esta.

URL:

<https://dle.rae.es/medida>

<https://dle.rae.es/medir>

<https://dle.rae.es/comparar>

## 3.2 ¿Qué son las métricas de Software?

Las métricas de software son medidas que pueden ser aplicadas a un sistema, componente o proceso, estas abarcan algunos aspectos del software como lo pueden ser la calidad, la fiabilidad, la efectividad y la eficiencia.

Las métricas están basadas en técnicas de ingeniería que son aplicados directamente al software y que tienen el objetivo de asegurar el funcionamiento óptimo del mismo.

URL:

<https://d1wqtxts1xzle7.cloudfront.net/55797299/metricas_1-with-cover-page-v2.pdf?Expires=1646268871&Signature=enGsOOXz~OKyTim6fGSgqqu-4TD2u0~Bl7JWlD17fJeLgJ-t9iZU1N3JSrXfZXZgAjO0OChHvBShasHUcP2oa3nlLsRumFore8q~nCRmq~pJaznKNHjVO58B05f1zLx9ttGwtH-YC2wvRlLxJjb5D~X6lji2kstoeTDRQMstegnnrcd6b9vgB6R8sYAF5wAeMyi5ogllojj7uRaTckpBnyVHTREDtdSd7-sryUR-NS2CvwUX32czH9avC6fKJ5sAHyvXmquL5iBCCoNAdxQTlyHAegSrtTbBLlvmwVsblseeuIvzL3jZS3gkaymEATyb05IHcGEYhTMoHb1hLnjf2g__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>

## 3.3 ¿Qué es un Indicador?

Un indicador es el uso de una métrica o de un conjunto de métricas que proveen de una visión detallada del software, componente o proceso. El uso de indicadores permite evaluar el estado y la calidad del proyecto, así como detectar áreas de riesgo potencial antes de que sean críticas.

URL:

<http://avellano.usal.es/~mmoreno/APITema2.pdf>

## 3.4 ¿Para qué se utilizan las métricas de software?

Las métricas de software abarcan distintos rubros como lo son la estimación de coste y esfuerzo, modelos de productividad, calidad y fiabilidad, métricas estructurales y de complejidad, análisis basados en métodos, evaluación de la capacidad de una organización, evaluación del rendimiento, entre otros.

URL:

<http://avellano.usal.es/~mmoreno/APITema2.pdf>

## 3.5 ¿Cuáles son las características de las métricas de Software?

Lorem ipsum

## 3.6 ¿Cuáles son las ventajas de implementar métricas de software?

Lorem ipsum

## 3.7 Tipos de Métricas de Software

### 3.7.1 De acoplamiento

El acoplamiento se refiere a los de métodos o atributos definidos en una clase que son usados por otra, y, debido a que las clases interactúan entre ellas para que los sistemas puedan funcionar, esta interacción puede resultar en un indicador de la complejidad.

Existen dos tipos de acoplamiento, el acoplamiento poco acoplado o débil y el acoplamiento muy acoplado o fuerte. El acoplamiento débil es aquel en el que todas las clases tienen funciones distintas y definidas, por otro lado, el acoplamiento fuerte es aquel en el que todas las clases participan en todas las tareas.

URL:

<http://www.cc.uah.es/drg/b/RodHarRama00.pdf>

<http://laurel.datsi.fi.upm.es/_media/docencia/cursos/cplusplus/poo_dia2_bn.pdf>

#### 3.7.1.1 Acoplamiento entre objetos (CBO)

El acoplamiento entre objetos (por sus siglas en inglés *Coupling Between Objects*) es el número de clases con las que una clase se relaciona, de esta manera se puede medir la dependencia que existe entre clases. Dos clases están relacionadas o acopladas siempre que una actúa sobre los atributos de la otra.

URL:

<https://d1wqtxts1xzle7.cloudfront.net/63849831/Succi.C72.DefiningMetricsforSoftwareComponents20200707-98053-1i461gc-with-cover-page-v2.pdf?Expires=1646800383&Signature=Hkwj5fcUlE35p6jkV2j-xtEPOdNT1KBP0BcF1-~W-KFtwLj9JwOrL3Wtwe4TeQnJo5IC-Y5U2O~gQ5BAdbENPVO-Dihn8GW~FhTEfwoUxEDwrX3AzfVFUxHi7vQh9FTcMiJuZPEOd2qIAvyXYCj8SSt6AtzORqxllUtXY2gES0fvMXM7UWMIsdl8GZzmwbfXFhqK6zPfau2mzGETNnoU-Bh7zh6FGpYS62zKpfnyDShMaMcH5vRRpu3tm8v~4Nno~sEYWWmqaiYxiWEilQDZqbn0BNMxTkVzdnYSBUV924gOXiRBNVDnGmDWLwN0OWdyFUeVuRkhAkP-0i7cSZ6t-A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>

#### 3.7.1.2 Proporción de acoplamiento (CF)

El factor de acoplamiento (por sus siglas en inglés *Coupling Factor*) se refiere a la relación que existe entre el máximo número posible de acoplamientos en un sistema y el número real de acoplamientos (casos en los que no se le atribuye a la herencia), este indicador describe la comunicación que existe entre clases. En la siguiente ecuación se muestra la forma de calcular el CF:

Donde *es\_cliente* es una función que da 1 si la clase cliente (C) está acoplada a la clase proveedor y que en este acoplamiento exista al menos una referencia que no sea heredada de la clase proveedor.

### 3.7.2 De cohesión

La cohesión mide el grado de relación funcional que tienen los elementos o características de un objeto, básicamente, la cohesión es lo que mantiene unido a un módulo. Los objetos con una función muy bien definida son objetos muy cohesionados y, por otro lado, los objetos con muchas funciones no relacionadas son poco cohesionados.

A su vez, la cohesión puede ser categorizada desde la forma más débil hasta la más fuerte siguiendo el siguiente orden: coincidente, lógico, temporal, procesal, comunicativo, secuencial y funcional.

URL:

<https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/>

<https://arxiv.org/ftp/arxiv/papers/1004/1004.3277.pdf>

#### 3.7.2.1 Falta de cohesión en los métodos (LCOM)\*

Esta métrica está basada en la cohesión de una clase considerando la misma cantidad de atributos que son usados por diferentes métodos, tomando en cuenta la calidad de abstracción que se utiliza en la clase.

Cuando el indicador LCOM es alto, significa falta de cohesión o que los métodos tienen poca similitud, lo que puede estar dado por la falta de relación entre los elementos, y esto a su vez aumenta la probabilidad de tener errores. Cuando el indicador LCOM es bajo, significa que hay una alta cohesión, lo que es deseable ya que esta no puede ser fragmentada y por consecuente llevar a la encapsulación.

Considérese una clase C1 con n métodos el conjunto de variables instanciadas por el método .

Hay *n* conjuntos tales que ; Sea , y

\*Incluir la versión de Henderson-Sellers

### 3.7.3 De complejidad

#### 3.7.3.1 Complejidad Ciclomática

La complejidad ciclomática es probablemente la métrica de software más utilizada en cuanto a ingeniería de software se refiere. Esta métrica fue creada y definida por Thomas McCabe en el año de 1976 y está basada en el control estructurado del flujo de un programa. Está basada en la teoría de grafos ya que los distintos caminos que puede recorrer un programa son convertidos en un grafo.

Esta métrica se calcula mediante la medición del patrón linealmente independiente en el grafo y está representada por un simple número. Se define de la siguiente manera:

Donde:

* CC es la complejidad ciclomática del grafo G basado en el diagrama de flujo del programa.
* e es el número de aristas en G
* n es el número de nodos en G

URL:

<https://d1wqtxts1xzle7.cloudfront.net/35359323/6-with-cover-page-v2.pdf?Expires=1646883651&Signature=DRgmvtTHyfz6X8-5cDj12fNOWUBOmG5PISxyaiWtzfMlI01T2fj~jsRH4CrqewkCwrd15LtFtsWbpUOBpK-lfz9g~boLU~nTlh76bozueG-PalS8wd5Gw4MMKt08~EJF08eY0lo6x~WZvRzjVJojjIKYoYdQFdd1cNBVZkOrvF5DutE6bMOF-rvE45ejxPxKfeOQkMhsgZXxJGCrRuOsvg2wju3RUGA~9ql0xe5O4DgYGU60n5JozbvwanqB6sFSo~nF-6jN2vvW0KwCY1wCR3R6XzT9EMgHF2wZXG3q3at-2KSuARNBk51CSd-3ZmxLsF7VQBUIOcZycAWHPgL6Ww__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>

#### 3.7.3.2 Respuesta para una clase (RFC)

La RFC (por sus siglas en inglés *Response For a Class*) se refiere a la cantidad de métodos que son llamados desde una clase en particular, estos métodos tienen que pertenecer a otra clase y se incluyen a todos aquellos a los que se puede acceder.

Chidamber y Kemerer consideraban a la RFC como una medida de complejidad de una clase, ya que toma en cuenta el número de métodos llamados desde fuera de la clase y la comunicación con otras. A continuación se enuncia la obtención de RS, que refiere al conjunto respuesta de la clase, cuya cardinalidad es el RFC, por lo que .

Donde:

* es el conjunto de métodos que son llamados por el método i.
* es el conjunto de métodos llamados en la clase.

### 3.7.4 De herencia

Las métricas de herencia proporcionan información acerca del árbol de herencia del sistema, es una característica clave de el paradigma Orientado a Objetos. Distintas métricas acerca de la herencia han sido propuestas y sus reseñas están disponibles en la literatura.

El uso de herencia reduce el mantenimiento que se tiene que dar a un software, facilita la fase de pruebas, y promueve la reutilización de clases, lo que permite generar software mantenible, entendible y confiable.

URL: <http://ischolar.info/index.php/IJCSIT/article/view/138232/126453>

#### 3.7.4.1 Índice de especialización por clase (SIX)

El SIX (por sus siglas en inglés *Specialisation Index per Class*) refiere a la medida en que las subclases determinan el funcionamiento de sus superclases.

Esta métrica indica si existe un gran número de métodos redefinidos, provocando que las abstracciones puedan no ser adecuadas y esto altere el comportamiento. En pocas palabras, una subclase, más que redefinir o borrar el comportamiento de sus superclases, debe extenderlo. Según Lorenz y Kidd, un valor cercano al 15% ayuda a identificar a superclases que son distintas a sus subclases en funcionamiento.

URL: <http://www.cc.uah.es/drg/b/RodHarRama00.pdf>

#### 3.7.4.2 Profundidad del árbol de herencia (DIT)

La DIT (por sus siglas en inglés *Depth of Inheritance Tree*) se refiere al número máximo de pasos desde la clase nodo hasta la raíz del árbol de herencia y es medido por el número de clases padre. En ocasiones, el utilizar distintas herencias, el DIT será la longitud máxima desde el nodo hasta la raíz del árbol.

URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.9833&rep=rep1&type=pdf>

#### 3.7.4.3 Proporción de métodos heredados (MIF)

El MIF (por sus siglas en inglés *Method Inheritance Factor*) se define como la relación que existe entre la cantidad de métodos heredados en todas las clases y el número total de métodos en todas las clases (locales y heredados)

Donde:

#### 3.7.4.4 Proporción de atributos heredados (AIF)

El AIF (por sus siglas en inglés *Attribute Inheritance Factor*) se refiere a la relación que existe entre el número de atributos heredados entre el número total de atributos (Heredados o no heredados).

Donde:

### 3.7.5 De encapsulamiento

El encapsulamiento es una técnica utilizada para minimizar las interdependencias entre módulos, ya que define métodos a los cuáles sólo se puede acceder mediante una interfaz, así los detalles de la implementación están “ocultos” dentro del mismo módulo.

URL: <https://academiccommons.columbia.edu/doi/10.7916/D8TT4ZZD>

#### 3.7.5.1 Proporción de métodos ocultos (MHF)

La MHF (por sus siglas en inglés *Method Hiding Factor*) se define como la relación que existe entre la cantidad de métodos ocultos en todas las clases y la cantidad de métodos que existen en total, es el porcentaje de métodos ocultos en una clase. Se define de la siguiente manera:

Donde:



Si el valor del MHF es alto, significa que todos los métodos son privados y por lo tanto indicaría escasa o nula funcionalidad. No es posible reusar métodos con un MHF alto. Por el contrario, si el MHF es bajo, indicaría que la gran mayoría de métodos son públicos y por lo tanto que están desprotegidos.

#### 3.7.5.2 Proporción de atributos ocultos (AHF)

La AHF (por sus siglas en inglés *Attribute Hiding Factor*) se define como la relación que existe entre la cantidad de atributos ocultos en todas las clases y la cantidad de atributos que existen en total, es el porcentaje de atributos ocultos en una clase. Se define de la siguiente manera:

Donde:



Si el valor del AHF es alto, significa que la mayoría de los atributos son privados, por el contrario, si el valor es bajo, significa que una gran parte de los atributos son públicos.

URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.188&rep=rep1&type=pdf>

#### 3.7.5.3 Número de hijos (NOC)

El número de hijos o NOC (por sus siglas en inglés *Number of Children*) es el número de subclases inmediatas de una clase en una jerarquía.

### 3.7.6 De tamaño

#### 3.7.6.1 Líneas de código (LOC)

El LOC (por sus siglas en inglés *Lines Of Code*) refiere al número de líneas de código que se encuentran activas (líneas que pueden ser ejecutadas), dentro de un método, clase, sistema, etc.

#### 3.7.6.2 Métricas de Halstead

# CAPÍTULO IV METODOLOGÍA

## 4.1 ¿Qué es una metodología?

En software, una metodología está definida como un proceso detallado mediante una serie de técnicas y métodos que abordan las actividades a realizar dentro del ciclo de vida de un proyecto, se basan en una combinación de modelos y definen a los involucrados, roles y actividades.

Una metodología permite el desarrollo, gestión y administración del software de manera sistemática, esta comprende los pasos a seguir para diseñar, implementar y mantener un proyecto, y tiene la finalidad de generar productos útiles, eficientes y eficaces, para asegurar su éxito dentro del mercado.

## 4.2 Tipos de metodología

### 4.2.1 Metodologías clásicas

Las metodologías clásicas o tradicionales se basan en una serie de pasos secuencial, tales como definición de requerimientos, la implementación de la solución, pruebas y despliegue. Los requerimientos son definidos y documentados desde el principio del proceso de desarrollo. Algunos ejemplos de metodologías clásicas son Cascada, Modelo en Espiral y Proceso Unificado.

URL: <https://d1wqtxts1xzle7.cloudfront.net/58993716/10.1.1.464.609020190422-13963-j0ju8a-with-cover-page-v2.pdf?Expires=1647368350&Signature=MF9LXAI5Dai4w93GR-GdhtjrZJWXiAoxbRj6TUzPpUVI~JlWfTJO2bfa7hJC5SyjqlYRBe~73n~6lvy6I3d8A5t9MOedIYEg3JTfc-bTEI8NCBSRm7jJx~QC7hBFYIHFmwaaNSpsvqPcBiBKswrOMmUQq-ZiFRdv1633sOJBD8qK-YCOmRRy-keL2AzmxNP9ilyBp15lMF0zC6V12B0yzMP17ZMbkbZ57rK3vkvvfn4X5rLheUoubkrLdpw282LYo~Hl0CuAupG7lYxde21tZ6tk1TqEC5Fg6IKIbjOfzmu8b1Rl6SgbGsFwphNm2SVnUNDL0TNeeUznPjYbaY0QlA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>

#### 4.2.1.1 Modelo en cascada

Para realizar este Trabajo Terminal se hará uso de la metodología del modelo en Cascada, ya que es una metodología que permite ordenar de forma secuencial las distintas etapas y actividades a seguir al momento de desarrollar el sistema. En el modelo en cascada las fases se van ejecutando de forma escalonada, de manera que solo se puede hacer otra actividad hasta finalizar la actividad anterior y así hasta llegar a la última etapa.

El modelo en casaca es la mejor opción para el desarrollo del proyecto, ya que el sistema consta de diferentes subsistemas a los cuales se les debe hacer un análisis y comprobación del funcionamiento de cada uno de ellos, para detectar errores y corregirlos antes de seguir avanzando. Otro punto importante es que se eligió está metodología debido al número de integrantes en el equipo, ya que necesitamos llevar un orden en la realización de las fases. Este modelo es apropiado para el sistema web debido a que no es un proyecto grande y se pueden tener bien definidas las fases del sistema.

En la figura “X” podemos observar el modelo en casada.



De acuerdo con el diagrama anterior para cada una de las fases se tomará en cuenta lo siguiente:

* **Análisis**: En esta etapa se determinarán cuáles son los requisitos y objetivos por cumplir del sistema, en esta se llevará acabo toda la investigación y recolección de información necesaria para precisar los requisitos del sistema y se realizará el modelado UML pertinente.
* **Diseño**: Para la etapa del diseño se establecerá cuales son las entradas y salidas del sistema, se diseñarán las interfaces del sistema web, se establecerá los archivos con los que se trabajarán, así como se plantearán los primeros esbozos del sistema web.
* **Implementación**: En la implementación se llevará acabo toda la parte de lógica de programación y codificación vista anteriormente, así como seguir con los prototipos de diseño planteados anteriormente. En esta etapa se codificará cada uno de los subsistemas del sistema web.
* **Verificación**: Se realizarán todas las pruebas individuales y la ejecución del sistema web final para poder testear su funcionamiento. Se hará una comparativa de los resultados obtenidos con los objetivos iniciales.
* **Mantenimiento**: Por último, para la etapa del mantenimiento se llevarán a cabo los cambios necesarios para dar solución a los ajustes que se consideren en la etapa de la verificación del sistema.

Se debe de tomar en cuenta que para cada una de las etapas se realizará una comprobación de las tareas designadas para cada fase, en caso de no estar completas o de necesitar corrección, se llevará a cabo un retroceso hasta concluir exitosamente con esa fase. Es importante cumplir con los tiempos designados para no tener sobrecarga de trabajo y contratiempos.

### 4.2.2 Metodologías ágiles

Las metodologías ágiles representan una alternativa para satisfacer el desarrollo de software que requiere métodos rápidos, principalmente dentro de la comunidad empresarial. Se caracterizan por priorizar el cumplimiento de metas en un periodo de tiempo corto, por dividir las tareas para reducir su tamaño y por tener la habilidad de responder al cambio.

URL: <https://d1wqtxts1xzle7.cloudfront.net/58993716/10.1.1.464.609020190422-13963-j0ju8a-with-cover-page-v2.pdf?Expires=1647368350&Signature=MF9LXAI5Dai4w93GR-GdhtjrZJWXiAoxbRj6TUzPpUVI~JlWfTJO2bfa7hJC5SyjqlYRBe~73n~6lvy6I3d8A5t9MOedIYEg3JTfc-bTEI8NCBSRm7jJx~QC7hBFYIHFmwaaNSpsvqPcBiBKswrOMmUQq-ZiFRdv1633sOJBD8qK-YCOmRRy-keL2AzmxNP9ilyBp15lMF0zC6V12B0yzMP17ZMbkbZ57rK3vkvvfn4X5rLheUoubkrLdpw282LYo~Hl0CuAupG7lYxde21tZ6tk1TqEC5Fg6IKIbjOfzmu8b1Rl6SgbGsFwphNm2SVnUNDL0TNeeUznPjYbaY0QlA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>

#### 4.2.2.1 SCRUM

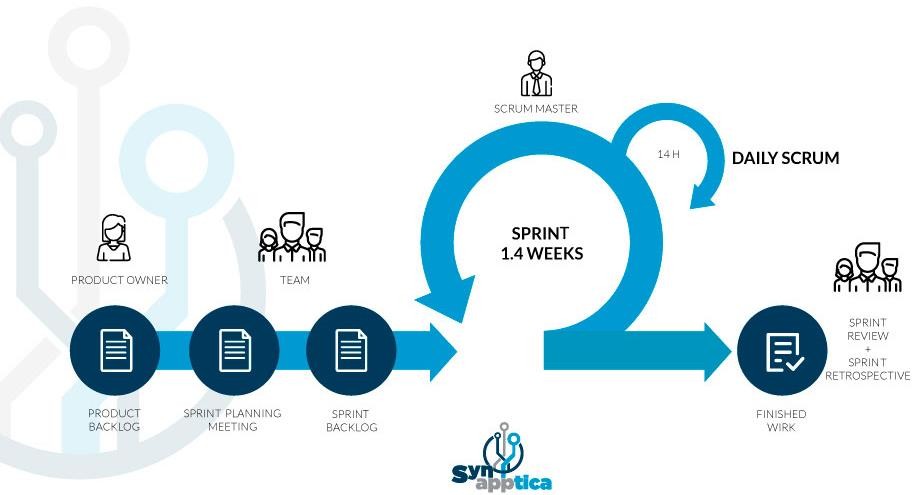
Para desarrollar este Trabajo Terminal se hará uso de la metodología SCRUM, ya que es una metodología ágil para el desarrollo del software o para la gestión de proyectos. Se basa en aspectos como la flexibilidad para la adopción de cambios y nuevos requisitos, la comunicación constante entre los integrantes y, debido a su desarrollo iterativo, se pueden asegurar buenos resultados [15].

SCRUM es la mejor opción para el desarrollo del proyecto, ya que el sistema se puede dividir en subsistemas, cada uno de los cuales tendrá una cantidad determinada de *sprints*, para la etapa de desarrollo, implementación, revisión y pruebas. Con esto se espera una entrega en tiempo y forma del proyecto.

Como se puede ver en la Figura 3, dentro de SCRUM existen dos roles importantes, el *Product Owner* que, para este proyecto será Ian Axel Cuevas Olvera, cuyo trabajo es transformar y elegir las ideas propuestas del equipo para realizar el *Product Backlog* (Lista del Producto), y, el *Scrum Master*, del cual se encargará Ismael Ortega Estrada, quien es el responsable de que las técnicas de scrum sean aplicadas durante la organización del proyecto y realizará el *Sprint Backlog* que contiene la lista de tareas en el *Product Backlog* que se realizarán en cada *sprint* [16].

Como se observa en la Figura 3, el ciclo de SCRUM se compone de 4 eventos principales, que son:

* **El *Sprint*:** En el cual se puede tomar un tiempo entre 2 a 4 semanas para desarrollar la lista de tareas en el *Sprint Backlog*.
* ***Daily Scrum* (Reunión Scrum diaria):** Se realiza una reunión de no más de 15 minutos para hablar de los avances del día anterior y sobre algún problema que se tenga.
* ***Sprint review* (Revisión del sprint):** Se lleva a cabo 1 vez por semana y no dura más de 1 hora para analizar las tareas terminadas del *sprint.*
* ***Sprint retrospective* (Retrospectiva del Sprint):** Se lleva a cabo cuando un *sprint* dura más de 4 semanas y la intención es ver que cambios se pueden realizar para mejorar la productividad y tiempo. Este bloque no puede durar más de 3 horas.

 *Figura 3 Diagrama de la metodología SCRUM [15]*

SCRUM consta de cinco fases para el desarrollo de un proyecto. A continuación, se detallan cada una de ellas aplicadas a el sistema de evaluación de métricas de software [17]:

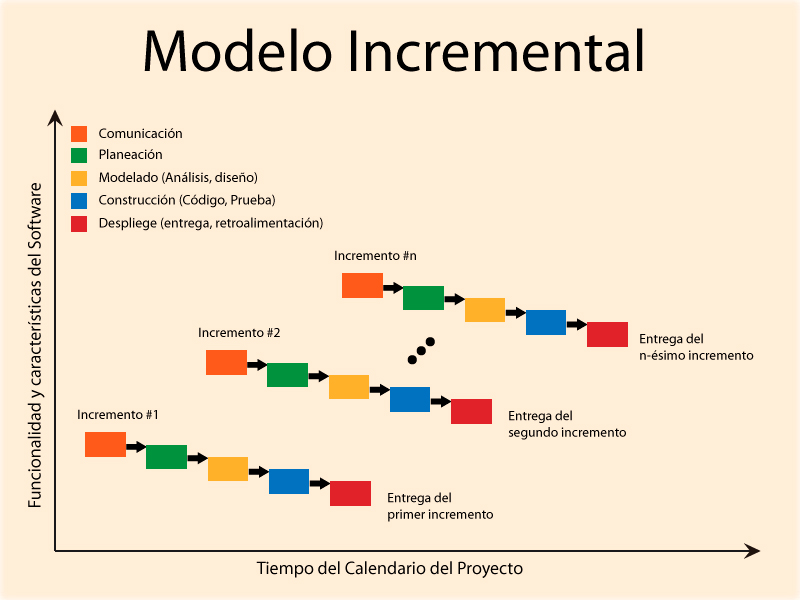
* **Inicio:** Se investigará y se analizará cada una de las métricas de software propuestas a profundidad.
* **Planificación y estimación:** Se asignará el número exacto de *sprints* para cada subsistema en la arquitectura del sistema, tomando en cuenta un aproximado de 3 a 4 *sprints* por subsistema.
* **Implementación**: Se desarrollarán los subsistemas uno a uno y se crearán los entregables.
* **Revisión y retrospectiva**: Se realizarán juntas para hacer una evaluación de los entregables.
* **Lanzamiento**: El *Product Manager* tendrá que aprobar los entregables y los enviará.

#### 4.2.1.2 Modelo Incremental

Para desarrollar este Trabajo Terminal se hará uso de la metodología del modelo incremental, ya que pertenece a las metodologías ágiles para el desarrollo de software o gestión de proyectos. Algunas de las ventajas de este modelo es su flexibilidad a cambios o la adopción de nuevos requisitos y, debido a su desarrollo iterativo, se pueden asegurar avances continuos y buenos resultados.

El modelo incremental es la mejor opción para el desarrollo del proyecto, ya que el sistema se puede dividir en subsistemas, cada uno de los cuales se puede designar a un incremento, para la etapa de análisis, construcción, revisión y pruebas. Con esto se espera una entrega en tiempo y forma del proyecto.

Como se puede ver en la Figura “X” dentro del modelo incremental es importante saber cuales son las funcionales y características del software, ya que estas serán el punto de partida para determinar el número de incrementos para el desarrollo del proyecto, de igual forma cada incremento debe de tener una fecha en calendario es decir una fecha de inicio del incremento y una fecha de fin del incremento para poder continuar con los siguientes.



Siguiendo la estructura del modelo incremental, se proponen los siguientes incrementos para el desarrollo del sistema:

* Incremento #1: Base de Datos de Usuarios y Archivos.
* Incremento #2: Subsistema Gestor de Usuarios.
* Incremento #3: Subsistema Gestor de Archivos.
* Incremento #4: Subsistema de Análisis de Métricas de Programación Estructurada.
* Incremento #5: Subsistema de Análisis de Métricas de Programación Orientada a Objetos.
* Incremento #6: Subsistema de Despliegue de Resultados

Para cada incremento se tiene previsto un tiempo de tres a cuatro semanas, de tal forma que se irán desarrollando uno a uno los subsistemas, haciendo las pruebas y revisiones correspondientes para poder crear entregables en cada incremento.

# CAPÍTULO V ESTUDIO DE FACTIBILIDAD

## 5.1 Factibilidad Técnica

Lorem ipsum

## 5.2 Factibilidad Económica

Lorem ipsum

## 5.3 Facilidad Operacional

Lorem ipsum

## 5.4 Interfaz gráfica de usuario

Lorem ipsum

### 5.4.1 Menús

Lorem ipsum

### 5.4.2 Formulario

Lorem ipsum

### 5.4.3 Logotipo

Lorem ipsum

### 5.4.4 Colores de diseño

Lorem ipsum

### 5.4.5 Pantallas del sistema

Lorem ipsum

## 5.5 Archivos

Lorem ipsum

### 5.5.1 Diseño de la base de datos (Relacional)

Lorem ipsum

## 5.6 Procesamiento de captura

Lorem ipsum

# CAPÍTULO VI ANÁLISIS



## 6.1 Análisis de requerimientos

El análisis de los requerimientos es una sección que nos ayuda a recabar la información necesaria para la definición del sistema.

Para un correcto análisis de los requerimientos tanto funcionales como no funcionales, se hizo uso de la norma IEEE 830. Esta plantea los siguientes elementos:

* Contemplar los requisitos del usuario: Necesidades que los usuarios expresan verbalmente.
* Contemplar los requisitos del sistema: Son los componentes que el sistema debe tener para realizar determinadas tareas.
* Describir los requerimientos funcionales: Servicios que el sistema debe proporcionar al finalizar el sistema.
* Describir los requerimientos no funcionales: Definen propiedades o restricciones de los servicios o funciones ofrecidos por el sistema.

**La especificación de requisitos de software es una descripción completa del comportamiento del sistema a desarrolla. Incluye la descripción de todas las interacciones que se prevén que los usuarios tendrán con el software. También contiene requisitos no funcionales (o suplementarios). Los requisitos no funcionales son los requisitos que imponen restricciones al diseño o funcionamiento del sistema (tal como requisitos de funcionamiento, estándares de calidad, o requisitos de diseño.)**

FUNCIONALES: Son los que el usuario necesita que efectúe el sistema.

NO FUNCIONALES: Son los “recursos” para que trabaje el sistema de información (redes, tecnología.)

URL:

<https://www.espacios.media/analisis-de-requerimientos/>

<https://repositorio.continental.edu.pe/handle/20.500.12394/4281>

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=https%3A%2F%2Frepositorio.continental.edu.pe%2Fbitstream%2F20.500.12394%2F4281%2F1%2FDO\_FIN\_103\_MAI\_UC0939\_2018.pdf&clen=5440886

<https://scholar.google.com/scholar?hl=es&as_sdt=0%2C5&q=requerimientos+funcionales&btnG=>

<file:///D:/Descargas/142a0fcb4308d882f6a2d4fa9913b7af.pdf>

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=http%3A%2F%2Fzeus.inf.ucv.cl%2F~bcrawford%2FAULA\_ICI\_3242%2FERS\_IEEE830.pdf&clen=49586&chunk=true

<https://scholar.google.com/scholar?hl=es&as_sdt=0,5&q=norma+ieee+830>

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=http%3A%2F%2Fsedici.unlp.edu.ar%2Fbitstream%2Fhandle%2F10915%2F20070%2FDocumento\_completo.pdf%3Fsequence%3D1%26isAllowed%3Dy&clen=38639&chunk=true

### 6.1.1 Requisitos del sistema

\* ¿Qué métricas se van a tomar en cuenta?

\* ¿Cuáles son las interpretaciones para cada una de estas métricas? (Puntajes)

\* ¿Qué tipos de programas se van a evaluar? (Simples, Recursivos, Funcionales, etc).

\* ¿Qué limitantes tienen los programas que se van a evaluar?

\* ¿Qué características debe de tener el programa?

### 6.1.2 Requerimientos funcionales

Los requerimientos funcionales son especificaciones de los servicios que proveerá el sistema, sobre la manera en que éste reaccionará a entradas particulares. A continuación, se muestra una tabla que especifica cada requerimiento funcional con un identificador, el nombre del requerimiento y su descripción.

|  |  |  |
| --- | --- | --- |
| ID | Nombre | Descripción |
| Rf 01 | Registro de Usuarios | El sistema permitirá crear una cuenta a los usuarios. |
| Rf 02 | Validación de Usuarios | El sistema permitirá ingresar a los usuarios que coloquen correctamente sus credenciales. |
| Rf 03 | Agregar Archivos | El sistema permitirá que el usuario agregue archivos, solo archivos con extensión .c o .java |
| Rf 04 | Selección de Métricas | El usuario podrá elegir que métricas desea evaluar en su programa |
| Rf 05 | Compilación | El sistema ofrecerá un compilador para Lenguaje C y JAVA. |
| Rf 06 | Evaluación de Métricas | El sistema evaluará el programa mediante las métricas seleccionadas. Basándose en la lógica de programación establecida. |
| Rf 07 | Interpretación de Resultados | El sistema deberá mostrar una interpretación de los resultados, mediante gráficos. |
| Rf 08 | Historial de Resultados | El sistema permitirá al usuario revisar su historial de resultados. |

### 6.1.3 Requerimientos no funcionales

Los requerimientos no funcionales representan características generales y restricciones del sistema que se está desarrollando. A continuación, se muestra una tabla que especifica cada requerimiento no funcional con un identificador, el nombre del requerimiento y su descripción.

|  |  |  |
| --- | --- | --- |
| ID | Nombre | Descripción |
| RNf 01 | Seguridad | El sistema debe de proporcionar seguridad de los datos en la base de datos. |
| RNf 02 | Alertas de error | El sistema deberá mostrar alertas de error que sean informativos para el usuario. |
| RNf 03 | Alertas exitosas | El sistema deberá mostrar alertas de éxito informativas para el usuario. |
| RNf 04 | Manual de Usuario | El sistema debe contar con el manual de usuario. |
| RNf 05 | Disponibilidad | El sistema estará disponible para el usuario en cualquier mes, día y hora, de igual forma cuenta con tiempo ilimitado dentro del sistema. |
| RNf 06 | Usabilidad | El uso y manejo del sistema deberá ser guiado. |
| RNf 07 | Lista de Métricas | El sistema proporcionará el listado de métricas de software que puede evaluar. |

## 6.2 Modelado UML

El modelado UML está compuesto por diversos elementos gráficos que se combinan para formar diagramas. Debido a que UML es un lenguaje, cuenta con reglas para combinar estos elementos. La finalidad de los diagramas es presentar diversas perspectivas del sistema, a las cuales se les conoce como modelado.

En esta sección los diagramas UML que se desarrollaron para representar los componentes del sistema son.

* Diagrama de casos de uso.
* Diagrama de clases
* Diagramas de estados.

Es importante mencionar que los diagramas se realizaron considerando la norma ISO/IEC 9126 con el propósito de cumplir con los aspectos de calidad de un modelado del sistema. Por lo que en la siguiente tabla se describen tales aspectos y cómo influyen en el proceso.

|  |  |  |
| --- | --- | --- |
| Proceso | ISO/IEC 9126 | Sistema Web |
| Funcionalidad | Adecuación, exactitud, interoperabilidad | El uso de Python como el lenguaje base del sistema permite un desarrollo funcional. |
| Fiabilidad | Capacidad de recuperación y tolerancia a fallas. | El sistema estará dentro del servicio de Nube de Azure por lo que se cuenta con tolerancia a fallas. |
| Usabilidad | Capacidad de ser aprendido u operado | La interfaz gráfica será sencilla de tal forma que el sistema sea intuitivo de utilizar.  De forma extra se contará con un asistente que guiará paso a paso el uso del sistema |
| Eficiencia | Comportamiento temporal y utilización de recursos |  |
| Mantenibilidad | Capacidad de ser probado y cambiado. | La documentación del sistema facilitará la comprensión del programa, para un futuro mantenimiento. |
| Portabilidad | Capacidad de ser utilizado en cualquier entorno. | Al ser un sistema web nos olvidamos del sistema operativo, por lo que se podrá ejecutar en cualquier entorno siempre y cuando se tenga un navegador. |

### 6.2.1 Diagrama de casos de uso

**Módulo de Registro/Inicio de Sesión**

Diagrama

Descripción generada automáticamente

**Casos de Uso Usuario**

Diagrama

Descripción generada automáticamente

### 6.2.2 Diagrama de clases

Diagrama

Descripción generada automáticamente

### 6.2.3 Diagrama de estados

Diagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

# CAPÍTULO VII DISEÑO

## 7.1 Entradas

El sistema web para la evaluación de métricas de software para poder funcionar necesita como materia prima el archivo del programa a evaluar ya sea con extensión .c o .java. Este archivo se solicitará y se cargará al sistema para posteriormente ser procesado.

Para lograr el ingreso de entradas precisas y no recibir archivos basura, el diseño de las entradas o generalizando formularios (donde se reciben datos, archivos, etc.) dentro del sistema se basa en cuatro principales lineamientos, los cuales se enlistan a continuación:

1. Mantener la pantalla simple. Cada módulo está creado para realizar funciones específicas, por lo tanto, cada pantalla muestra las solicitudes exactas para recabar los archivos necesarios del usuario.
2. Mantener la presentación de la pantalla consistente. Todos los formularios tienen un estándar establecido, por lo que cada vez que se accede a ellos en el mismo módulo o en uno diferente, su diseño se mantendrá intacto.
3. Facilitar el movimiento del usuario entre los formularios. El sistema cuenta con entradas intuitivas y fáciles de llenar para minimizar errores y acelerar la velocidad en la que se llenan los campos.
4. Crear una pantalla atractiva y agradable. La creación de formularios atractivos y útiles es indispensable para que el sistema sea completo y eficaz.

Los datos de entrada que son indispensables para que el sistema funcione y que se emplearán formularios para obteneros son:

* Datos personales del usuario para su registro:
  + Nombre
  + Edad
  + Ocupación
  + Institución/Compañía
  + Correo
  + Contraseña
* Datos de inicio de sesión (Correo y Contraseña).
* Proyecto:
  + 1 archivo: Extensión .c, .java, .zip.
  + 2 o más archivos: Extensión .zip.

**Nota:** Dentro del archivo zip se debe mantener la siguiente estructura (\*Investigar estructura de proyectos):

Source -> main.java

Controllers ->

Models ->

## 7.2 Salidas

El diseño de una salida es indispensable para garantizar la aceptación del sistema. Se deben de cumplir con las expectativas del usuario para poder satisfacerlo. Se tiene que organizar la información de tal manera que no sea excesiva para el usuario y sea lo más clara posible y esté pueda entenderla fácilmente.

En el caso del sistema web para evaluación de métricas, se tienen como salidas alertas de error y alertas de éxito que se muestran al usuario cada vez que lleva a cabo una actividad, como es la carga de un archivo, la compilación del archivo, la obtención de los resultados de las métricas y la interpretación de estos.

Los datos del usuario y el archivo de programa después de ser procesado pasarán a ser una salida mediante las siguientes salidas:

* Verificación de los datos de usuario: Se indicará mediante una alerta si existe algún error o si se realizó satisfactoriamente el proceso.
* Alertas de Actividad: Se mostrará una alerta con el propósito de informar al usuario que fue lo que sucedió.
* Presentación de Resultados: Los resultados obtenidos se mostrarán por medio de puntajes y se dará una interpretación de estos mediante gráficos y una explicación textual del significado de la métrica.

## 7.3 Arquitectura del sistema

Diagrama, Esquemático

Descripción generada automáticamente

## 7.4 Interfaz gráfica de usuario

### 7.4.1 Menús

### 7.4.2 Formularios

### 7.4.3 Logotipo

### 7.4.4 Colores en el diseño

### 7.4.5 Pantallas del sistema

## 7.5 Archivos

Por la naturaleza del sistema para la evaluación de métricas, se necesitan requerirá recibir y almacenar muchos archivos, por lo que es necesario una base de datos que permita tener un control y gestión de los datos de cada usuario con sus respectivos archivos y resultados.

De esta manera se busca implementar una base de datos que nos permita escalar conforme se aumentan las necesidades del sistema y de los usuarios, sin olvidar la velocidad de acceso a los datos.

La base de datos que se ocupará para este sistema es de tipo Relacional o SQL, la cual esta diseñada para modelos de datos específicos y con esquemas estrictos.

Las ventas que nos ofrece implementar una base de datos relaciones se describen a continuación:

* Es simple pero muy poderosa para una amplia variedad de necesidades de información.
* Sirven para administrar grandes cantidades de información de usuario.
* Tener un control, orden y relación para el seguimiento de los datos y archivos.
* Es uno de los modelos más aceptado aún para base de datos.

Es importante tomar en cuenta que para poder implementar una base de datos relacional los datos a almacenar se deben relacionar entre sí y se deben administrar de manera segura, consistente y basada en reglas.

En el siguiente subapartado se mostrará el diseño de la base de datos relacional propuesta.

### 7.5.1 Diseño de la base de datos relacional

Como se menciono anteriormente la base de datos que se implementará para nuestro sistema es una base de datos SQL.

Una base de datos relacional es un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Las bases de datos relacionales se basan en un modelo relacional, una forma intuitiva y directa de representar datos en tablas. En una base de datos relacional, cada fila de la tabla es un registro con un identificador único llamado llave. Las columnas de la tabla contienen atributos, y cada registro generalmente tiene un valor para cada atributo, lo que facilita el establecimiento de las relaciones entre los puntos de datos.

Los datos de los atributos de un registro pueden se pueden codificar de varias formas, entre las que se incluyen XML, JSON y BSON, o incluso se pueden almacenar como texto sin formato, para el sistema los datos se almacenaran en formato JSON.

El diagrama entidad-relación que representa la base de datos es el siguiente:

Diagrama

Descripción generada automáticamente

## 7.6 Procesamiento de captura

# REFERENCIAS

**Referencias**

|  |  |
| --- | --- |
| [1] | S. M. a. L. Atymtayeva, de *Definition of software metrics project development by using fuzzy sets and logic*, The 6th International Conference of Soft Computing and Intelligent Systems, 2012, pp. 272-276. |
| [2] | H. S. a. M. D. Ince, Introduction to software management and quality assurance, London: McGraw-Hill Publishing, 1993. |
| [3] | L. E. T. Rangel, Informática VI (Programación e implementación de sistemas), Ciudad de México: UNAM, 2016. |

Cronograma

Nombre del alumno: Ortega Estrada Ismael

*Tabla 2 Cronograma de actividades Ismael Ortega Estrada*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Etapa | Actividad | FEB | MAR | ABR | MAY | JUN | AGO | SEP | OCT | NOV | DIC |
|  | Documentación | X | X |  |  |  |  |  |  |  |  |
| 1 | Investigación de Métricas de  Programación Estructurada | X |  |  |  |  |  |  |  |  |  |
| 2 | Análisis de los requerimientos de la Base  de Datos |  | X |  |  |  |  |  |  |  |  |
| 3 | Diseño y Creación de la Base de Datos |  |  |  |  |  |  |  |  |  |  |
| 4 | Diseño y Creación del Sistema Gestor  de Usuarios |  |  |  |  |  |  |  |  |  |  |
|  | **Evaluación TT1** |  |  |  |  |  |  |  |  |  |  |
| 5 | Diseño y Creación del Sistema Gestor  de Archivos |  |  |  |  |  |  |  |  |  |  |
| 6 | Conexión de la Base de Datos |  |  |  |  |  |  |  |  |  |  |
| 7 | Pruebas de la Base de Datos |  |  |  |  |  |  |  |  |  |  |
| 8 | Reajuste y Pruebas |  |  |  |  |  |  |  |  |  |  |
| 9 | Manual técnico |  |  |  |  |  |  |  |  |  |  |
|  | **Evaluación TT2** |  |  |  |  |  |  |  |  |  |  |

Nombre del alumno: Cuevas Olvera Ian Axel

*Tabla 3 Cronograma de actividades Ian Axel Cuevas Olvera*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Etapa | Actividad | FEB | MAR | ABR | MAY | JUN | AGO | SEP | OCT | NOV | DIC |
|  | Documentación | X | X |  |  |  |  |  |  |  |  |
| 1 | Investigación de Métricas de  Programación Orientada a Objetos | X |  |  |  |  |  |  |  |  |  |
| 2 | Análisis de los requerimientos de la  interfaz |  | X |  |  |  |  |  |  |  |  |
| 3 | Diseño y Creación del Módulo de Detección del Lenguaje de  Programación |  |  |  |  |  |  |  |  |  |  |
| 4 | Diseño y Creación del Sistema de Análisis de Métricas de Programación  Orientada a Objetos |  |  |  |  |  |  |  |  |  |  |
|  | **Evaluación TT1** |  |  |  |  |  |  |  |  |  |  |
| 5 | Diseño y Creación del Sistema de  Análisis de Métricas de Programación Estructurada |  |  |  |  |  |  |  |  |  |  |
| 6 | Diseño y Creación del Sistema de Despliegue y Comparación de  Resultados |  |  |  |  |  |  |  |  |  |  |
| 7 | Desarrollo de la Interfaz Web |  |  |  |  |  |  |  |  |  |  |
| 8 | Reajuste y Pruebas |  |  |  |  |  |  |  |  |  |  |
| 9 | Manual de usuario |  |  |  |  |  |  |  |  |  |  |
|  | **Evaluación TT2** |  |  |  |  |  |  |  |  |  |  |

Referencias

1. S. Mirseidova and L. Atymtayeva, “Definition of software metrics for software project development by using fuzzy sets and logic,” in The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems, 2012, pp. 272–276.
2. D. Ince, H. Sharp, and Mark, Introduction to software project management and quality assurance. London, England: McGraw-Hill Publishing, 1993.
3. D. Rodríguez y Rachel Harrison, “MEDICIÓN EN LA ORIENTACIÓN A OBJETOS,” Uah.es. [Online]. Available: [http://www.cc.uah.es/drg/b/RodHarRama00.pdf.](http://www.cc.uah.es/drg/b/RodHarRama00.pdf) [Accessed: 07-Oct-2021].
4. P. Escudero, M. Moreno, and F. J. García, “MÉTRICAS ORIENTADAS A OBJETOS,”, Universidad de Salamanca (España), nov. 2001.
5. W. K. S. Fernando, D. G. S. Wijayarathne, J. S. D. Fernando, M. P. L. Mendis, y I. Guruge, “The Importance of Software Metrics: Perspective of a software development projects in Sri Lanka”, SAITM Research Symposium on Engineering Advancements, pp. 91–95, abr. 2014.
6. “SonarQube Documentation,” Sonarqube.org. [Online]. Available: https://docs.sonarqube.org/latest/. [Accessed: 07-Oct-2021].
7. L. Gracia, “Google CodePro Analytix”, Unpocodejava.com, 20-sep-2010. [En línea]. Disponible en: https://unpocodejava.com/2010/09/20/google-codepro-analytix/. [Consultado: 17-oct-2021].
8. “Metrics 1.3.6”, Sourceforge.net. [En línea]. Disponible en: [http://metrics.sourceforge.net/.](http://metrics.sourceforge.net/) [Consultado: 17- oct-2021].
9. CoderGears, https://[www.](http://www/) codergears.com/home, “JArchitect :: Achieve Higher Java code quality,”

Jarchitect.com. [Online]. Available: https://[www.jarchitect.com/.](http://www.jarchitect.com/) [Accessed: 19-Oct-2021].

1. C. Kolbe, “JMT - Java Measurement Tool”, Hu-berlin.de. [En línea]. Disponible en: https://www2.informatik.hu-berlin.de/swt/intkoop/jcse/tools/jmt.html. [Consultado: 17-oct-2021].
2. “JavaNCSS - A Source Measurement Suite for Java”, Github.io. [En línea]. Disponible en: https://javancss.github.io/. [Consultado: 17-oct-2021].
3. T. J. Mccabe, “A Complexity Measure,” IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
4. C. Blanco, “Ingeniería del Sofware II,” Unican.es. [Online]. Available: https://ocw.unican.es/pluginfile.php/1408/course/section/1803/tema1-pruebasSistemasSoftware.pdf. [Accessed: 07-Oct-2021].
5. T. Honglei, S. Wei y Z. Yanan, "La investigación sobre métricas de software y métricas de complejidad del software", *Foro internacional de 2009 sobre tecnología y aplicaciones informáticas*, 2009, págs. 131-136, doi: 10.1109 / IFCSTA.2009.39
6. “Primeros pasos scrum”, Synapptica.net. [En línea]. Disponible en: https://synapptica.net/metodologia- scrum.html. [Consultado: 17-oct-2021].
7. M. Trigas, “Metodología SCRUM.” [Online]. Available:

<http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf>

1. A. Salazar, “Procesos de Scrum”, Prozessgroup.com, 16-oct-2016. [En línea]. Disponible en: [http://www.prozessgroup.com/procesos-de-scrum/.](http://www.prozessgroup.com/procesos-de-scrum/) [Consultado: 17-oct-2021]

**ANEXOS**

**CUESTIONARIO:**

1.- ¿Cuál es su ocupación actual?

a) Estudiante

b) Maestro

c) Programador

d) Empleado

2.- ¿Cuál es el nombre de su compañía u empresa a la que trabaja? (Se puede omitir esta pregunta.

R.- \_\_\_\_\_\_\_\_\_\_\_\_

3.- ¿Cuál de los siguientes lenguajes utiliza con más frecuencia para desarrollar sus programas?

a) Lenguaje C

b) JAVA

c) Python

d) JavaScript

4.- ¿Cuenta con algún programa para evaluar sus programas?

a) Si. ¿Cuál? \_\_\_\_\_\_

b) No.

5.- ¿Sabes que son las métricas de software?

a) Si. R.- \_\_\_\_\_\_\_\_\_\_

b) No.

6.- ¿Pará que utilizarías un sistema que evalué tus programas en base a métricas?

a) Reducir errores y aumentar la eficiencia.

b) Comparar mi programa respecto a otros.

c) Mejorar mi programa

d) Hacer mi programa mantenible a un futuro.

7.- ¿Qué tipo de asistente te gustaría que tuviera el sistema?

a) Un video donde se explique como utilizar el sistema.

b) Un asistente que te lleve paso a paso como evaluar un programa.

c) Con la documentación del sistema es suficiente.

8.- ¿Consideras que el sistema debe contener un compilador interno de los lenguajes a evaluar?

a) Si

b) No

9.- ¿Cómo te gustaría recibir la interpretación de los resultados de las métricas?

a) Mediante gráficos.

b) Comparando con los resultados promedio

c) Explicación textual del significado de los resultados.

10.- El sistema debería tener un gestor de usuarios.

a) Si, me gustaría poder tener un perfil

b) No, considero que no es necesario

11.- ¿Te gustaría mantener un historial de los programas evaluados?

a) Sí, poder recibir mis resultados tiempo después.

b) No, borrar instantáneamente los resultados.

12.- ¿Cómo mides la calidad de tus programas?

R.- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

13.-

Éxito

------------------ SISTEMA WEB PARA LA EVALUACIÓN DE MÉTRICAS --------------------

--------------------------- INICIAR SESIÓN ---------------------------------------------------------------

Correo: \_\_\_\_\_\_\_\_\_\_\_\_

Contraseña: \_\_\_\_\_\_\_\_\_\_

------------------------- REGISTRO -------------------------------------------------------------------------

Nombre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Edad: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ocupación: Estudiante o Empleado

Escuela o Compañía: \_\_\_\_\_\_\_\_\_\_\_\_

Correo: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Contraseña: \_\_\_\_\_\_\_\_\_\_\_\_\_

Confirmar contraseña: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

------------------------ INSERTE PROGRAMA -----------------------------------------------------------  
Agregar archivos: [Seleccionar un archivo] (.c o .java)

Botón: Evaluar Programa.

------------------------- FASE DE COMPILACIÓN ------------------------------------------------------------

* Si existe un error el programa no puede ser procesado
* Si el programa contiene warnings, se le muestran al usuario.
* Si el programa compila correctamente, se envía a la FASE DE EVALUACIÓN

------------------------- FASE DE EVALUACIÓN DE MÉTRICAS ---------------------------------------

* Se evalúan todas las métricas de acuerdo con el lenguaje en que está programado el archivo.

----------------------------- PANTALLA DE RESULTADOS --------------------------------------------

* Mostrar el resultado de cada una de las métricas.
* Interpretación de los resultados

------------------------------- GUARDAR RESULTADOS ------------------------------------------

* Enviar los resultados y el archivo a la base de datos.

**Índice**

1. Introducción
   1. Planteamiento del problema
   2. Objetivos
   3. Justificación
   4. Estado de Arte
   5. Descripción del Documento (¿Qué va en cada capítulo?)
2. Paradigmas de Programación
   1. Programación (Paradigmas de Programación)
      1. ¿Qué es un lenguaje de programación?
      2. ¿Qué es un Programa?
         1. Propiedades y Atributos de un Programa
         2. Diferencias entre pseucódigo y código.
      3. Compilador
         1. ¿Qué es un Compilador?
         2. Errores de compilación
      4. ¿Qué es la programación estructurada?
      5. ¿Qué es la programación Orientada a Objetos?
   2. Ingeniería de Software
      1. ¿Qué es la Ingeniería de Software?
      2. Objetivos de la Ingeniería de Software
   3. Normas y Estándares de Calidad en la Ingeniería de Software
      1. ¿Qué es la calidad?
      2. ¿Qué es una Norma en la Ingeniería de Software?
      3. ¿Qué es un Estándar en la Ingeniería de Software?
      4. Normas ISO
   4. Sistema Informático
      1. ¿Qué es un Sistema Informático?
      2. Tipos de sistemas informáticos
      3. Componentes de un sistema informático
3. Métricas de Software.
   * 1. Concepto de medida
        1. ¿Qué es una medida cuantitativa?
        2. ¿Qué es una medida cualitativa?
     2. ¿Qué son las métricas de Software?
     3. ¿Qué es un Indicador?
     4. ¿Para qué se utilizan las métricas de software?
     5. ¿Cuáles son las características de las métricas de Software?
     6. ¿Cuáles son las ventajas de implementar métricas de software?
     7. Tipos de Métricas de Software
        1. De acoplamiento
           1. Acoplamiento entre objetos (CBO)
           2. Proporción de acoplamiento (CF)
        2. De cohesión
           1. Falta de cohesión en los métodos (LCOM)
        3. De complejidad
           1. Complejidad Ciclomática
           2. Respuesta para una clase (RFC)
        4. De herencia
           1. Índice de especialización por clase (SIX)
           2. Profundidad del árbol de herencia (DIT)
           3. Proporción de métodos heredados (MIF)
           4. Proporción de atributos heredados (AIF)
        5. De encapsulamiento
           1. Proporción de métodos ocultos (MHF)
           2. Proporción de atributos ocultos (AHF)
        6. De reutilización
           1. Número de hijos (NOC)
        7. De tamaño
           1. Líneas de código (LOC)
           2. Métricas de Halstead
4. Metodología
   1. ¿Qué es una metodología?
   2. Tipos de Metodologías
      1. Metodologías clásicas
      2. Metodologías agiles
      3. SCRUM
5. Estudio de Factibilidad (Técnica, Económica y Operacional)
   1. Front-end
      1. ¿Qué es el front-end?
      2. Herramientas para front-end
         1. Lenguajes
            1. HTML5
            2. CSS
            3. JavaScript
         2. Frameworks
            1. React.js
         3. Librerías
            1. Bootstrap
   2. Back-end
      1. ¿Qué es el back-end?
      2. Herramientas para back-end
         1. Lenguajes
            1. Python
            2. JavaScript
         2. Frameworks
            1. Flask
         3. Librerías
            1. Pylint
            2. Python Suprocess
            3. JavaBridge
6. Métricas de Software.
   * 1. Concepto de medida
        1. ¿Qué es una medida cuantitativa?
        2. ¿Qué es una medida cualitativa?
     2. ¿Qué son las métricas de Software?
     3. ¿Qué es un Indicador?
     4. ¿Para qué se utilizan las métricas de software?
     5. ¿Cuáles son las características de las métricas de Software?
     6. ¿Cuáles son las ventajas de implementar métricas de software?
     7. Tipos de Métricas de Software
        1. De acoplamiento
           1. Acoplamiento entre objetos (CBO)
           2. Proporción de acoplamiento (CF)
        2. De cohesión
           1. Falta de cohesión en los métodos (LCOM)
        3. De complejidad
           1. Complejidad Ciclomática
           2. Respuesta para una clase (RFC)
        4. De herencia
           1. Índice de especialización por clase (SIX)
           2. Profundidad del árbol de herencia (DIT)
           3. Proporción de métodos heredados (MIF)
           4. Proporción de atributos heredados (AIF)
        5. De encapsulamiento
           1. Proporción de métodos ocultos (MHF)
           2. Proporción de atributos ocultos (AHF)
        6. De reutilización
           1. Número de hijos (NOC)
        7. De tamaño
           1. Líneas de código (LOC)
           2. Métricas de Halstead